OPTIMASI RUTE PENDISTRIBUSIAN KOMODITI BERDASARKAN JARAK DAN KONDISI JALAN UNTUK MEMINIMALKAN BIAYA BAHAN BAKAR DENGAN ALGORITMA GENETIKA (STUDI KASUS DI PT. XYZ)

SKRIPSI

oleh:
POPI APRILIA AYU SARI
0610962004-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2008

OPTIMASI RUTE PENDISTRIBUSIAN KOMODITI BERDASARKAN JARAK DAN KONDISI JALAN UNTUK MEMINIMALKAN BIAYA BAHAN BAKAR DENGAN ALGORITMA GENETIKA (STUDI KASUS DI PT.XYZ)

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dalam bidang Ilmu Komputer

Oleh:
POPI APRILIA AYU SARI
0610962004-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2008

LEMBAR PENGESAHAN TUGAS AKHIR

OPTIMASI RUTE PENDISTRIBUSIAN KOMODITI BERDASARKAN JARAK DAN KONDISI JALAN UNTUK MEMINIMALKAN BIAYA BAHAN BAKAR DENGAN ALGORITMA GENETIKA (STUDI KASUS DI PT.XYZ)

Oleh :
POPI APRILIA AYU SARI
0610962004-96

Setelah dipertahankan di depan Majelis Penguji pada tanggal 4 Agustus 2008 dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Wayan Firdaus M., S.Si, M.T	Bayu Rahayudi, S.T, M.T
NIP. 132 158 724	NIP. 132 318 424

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc.
NIP. 132 126 049

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Popi Aprilia Ayu Sari
NIM : 0610962004-96
Jurusan : Matematika
Program Studi : Ilmu Komputer

Penulis tugas akhir berjudul : Optimasi Rute Pendistribusian Komoditi Berdasarkan Jarak dan Kondisi Jalan untuk Meminimalkan Biaya Bahan Bakar dengan Algoritma Genetika (Studi Kasus di PT.XYZ)

Dengan ini menyatakan bahwa :

1. **Isi dari tugas akhir yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Tugas Akhir ini.**
2. **Apabila dikemudian hari ternyata Tugas Akhir yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran

Malang, 4 Agustus 2008
Yang menyatakan,

Popi Aprilia Ayu Sari
NIM. 0610962004-96

OPTIMASI RUTE PENDISTRIBUSIAN KOMODITI BERDASARKAN JARAK DAN KONDISI JALAN UNTUK MEMINIMALKAN BIAYA BAHAN BAKAR DENGAN ALGORITMA GENETIKA (STUDI KASUS DI PT.XYZ)

ABSTRAK

Optimasi rute pendistribusian komoditi berdasarkan jarak dan kondisi jalan diperlukan untuk memperoleh rute yang efisien dan pemakaian bahan bakar yang minimal. Algoritma genetika merupakan alternatif solusi untuk menentukan rute optimal. Pada algoritma genetika teknik pencarian solusi menggunakan prinsip seleksi alam, individu yang memiliki tingkat *fitness* yang lebih baik memiliki tingkat ketahanan hidup yang lebih baik pula. Pada kasus ini, representasi kromosom yang digunakan adalah pengkodean permutasi. Sedangkan metode seleksi yang digunakan adalah roda *roulette.* Untuk *crossover* menggunakan metode *one cut point crossover.* Sedangkan untuk mutasi, menggunakan metode *swap,* yaitu menukar langsung nilai gen dari 2 titik. Menurut uji coba yang dilakukan, model inisialisasi kromosom yang diterapkan secara *random* dan menggunakan *crossover rate* yang berbeda pada optimasi rute pendistribusian komoditi ini, tidak menimbulkan perbedaan yang signifikan terhadap nilai *fitness* yang dihasilkan. Namun, penggunaan *mutation rate* yang berbeda menyebabkan perubahan waktu komputasi untuk mendapatkan generasi yang konvergen. Dalam kasus ini algoritma genetika dapat memecahkan dan menyelesaikannya sebaik mungkin dengan menggunakan parameter genetika yang tepat.

(halaman ini sengaja dikosongkan)

OPTIMIZATION OF COMMODITY DISTRIBUTION ROUTE BASED ON THE CONDITION AND DISTANCE ROAD TO MINIMIZE THE COST OF FUEL WITH GENETIC ALGORITHM (CASE STUDY IN PT.XYZ)

ABSTRACT

Optimization of commodity distribution route based on the condition and distance road needed to obtain; get efficient route and the minimum fuel usage. Genetic algorithm represent solution alternative to determine optimal route. The solution searching technique of genetic algorithm using nature selection principles, which individuals who own better fitness condition will own better survival condition too. In this case, the model of chromosome initialization used permutation code. For crossover use method of one cut point crossover. While for the mutation, using method swap, that is direct exchange assess gene from 2 spot. According to the experiments, the model of chromosome initialization which were applied in random and use different crossover rate in this optimize of commodity distribution route, showed that there's no significant difference in this fitness result. But, use different mutation rate cause change of time of computing to get generation which convergent. In this case genetic algorithm can solve and finish it as good as possible by using a correct genetic parameter.

(halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yesus Kristus atas segala berkat dan karunia-Nya, sehingga penulis dapat menyelesaikan tugas akhir ini tepat waktu. Penulis Tugas Akhir ini mengambil judul "OPTIMASI RUTE PENDISTRIBUSIAN KOMODITI BERDASARKAN JARAK DAN KONDISI JALAN UNTUK MEMINIMALKAN BIAYA BAHAN BAKAR DENGAN ALGORITMA GENETIKA" dengan mengambil studi kasus pada PT.XYZ.

Tugas Akhir ini bertujuan untuk menerapkan algoritma genetika sebagai alternatif solusi masalah pencarian rute distribusi komoditi berdasarkan jarak dan kondisi jalan untuk meminimalkan biaya BBM dengan menggunakan perangkat lunak.

Dalam penyusunan Tugas Akhir ini penulis banyak menerima bantuan dari berbagai pihak. Oleh karena itu pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si., MT., selaku pembimbing utama penulisan Tugas Akhir dan selaku Ketua Program Studi Ilmu Komputer, Jurusan Matematika, FMIPA Universitas Brawijaya.
2. Bapak Bayu Rahayudi, S.T, MT., selaku pembimbing pendamping dalam penulisan Tugas Akhir.
3. Bapak Aditya Wiyasa ST, selaku pembimbing lapangan dalam pencarian data untuk Tugas Akhir ini.
4. Bapak Drs. Marji, MT, selaku penasehat akademik.
5. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
6. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan Tugas Akhir ini.
7. Orang tua dan keluarga penulis yang tak pernah berhenti memberikan doa dan dukungannya kepada penulis.
8. Rekan-rekan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan Tugas Akhir ini.
9. Dan semua pihak yang telah membantu dalam penyusunan tugas akhir ini yang tidak dapat Penulis sebutkan satu per satu.

Penulis sadari bahwa masih banyak kekurangan dalam laporan ini, oleh karena itu penulis sangat menghargai saran dan kritik yang sifatnya membangun demi perbaikan penulisan dan mutu isi Tugas Akhir ini untuk kelanjutan penelitian serupa di masa mendatang.
Semoga laporan Tugas Akhir ini dapat bermanfaat.

Malang, Agustus 2008

Penulis

DAFTAR ISI

(halaman ini sengaja dikosongkan)

DAFTAR TABEL

(halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR *SOURCECODE*

(halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Negara Kesatuan Republik Indonesia sebagai negara kepulauan dengan jumlah pulau lebih dari 15.000 buah, yang dihuni tak kurang dari 220 juta penduduk mempunyai permasalahan yang kompleks. Dan permasalahan yang paling mendasar adalah tuntutan pemenuhan kebutuhan primer (bahan pangan, sandang dan papan) dalam keseharian. Melihat keadaan geografis negara ini yang begitu luas, maka permasalahan yang timbul adalah ketersediaan barang-barang secara merata di semua daerah.

Dalam rangka menjawab pemerataan ketersediaan barang, banyak perusahaan distribusi dan logistik bermunculan. Namun tidak lama setelah itu tidak sedikit pula perusahaan-perusahaan tersebut yang gulung tikar. Adapun salah satu penyebabnya adalah ketidakmampuan perusahaan menemukan formula optimasi distribusi.

Optimasi pada hakikatnya adalah pencarian nilai-nilai variabel yang dianggap optimal, efektif dan efisien untuk mencapai hasil yang diinginkan. Dalam dunia distribusi dan logistik, optimasi berarti proses menemukan suatu formula yang bisa menekan biaya transportasi dengan tetap mempertahankan tujuan utama menyediakan barang sesuai permintaan dengan ketentuan tepat tempat dan tepat pada waktunya.

Transportasi merupakan persoalan yang menarik untuk dibahas, karena transportasi memberikan manfaat geografis pada sistem logistik dengan menghubungkan fasilitas-fasilitas dengan pasar. Biaya yang dikeluarkan untuk transportasi pada umumnya berkisar antara sepertiga sampai dua per tiga dari total biaya logistik (Ballou, 1992). Biaya transportasi tersebut terdiri dari pembayaran sesungguhnya untuk pengangkutan di antara sumber dan tujuan, serta biaya yang dikeluarkan selama dalam perjalanan. Pada umumnya perusahaan akan berusaha semaksimal mungkin untuk mengurangi atau meminimalkan biaya transportasi yang dikeluarkan untuk pendistribusian komoditinya agar keuntungan yang didapatkan meningkat. Salah satu dari biaya yang dibutuhkan adalah biaya bahan bakar.

Sebagai perusahaan yang bergerak di bidang transportasi dengan 175 armada yang terdiri dari 3 jenis kendaraan dengan kapasitas dan konsumsi bahan bakar yang berbeda, PT.XYZ mengharapkan adanya optimasi pendistribusian komoditinya untuk meminimalkan biaya transportasi (biaya

bahan bakar). Salah satu metode optimasi adalah dengan algoritma genetika.

Algoritma genetika adalah suatu teknik pencarian solusi dengan menggunakan prinsip seleksi alami (Gen dan Cheng, 1997). Ide awal algoritma ini adalah teori evolusi dalam konsep biologi yang dikemukakan oleh Charles Darwin. Algoritma genetika dimulai dengan memilih himpunan penyelesaian, yang direpresentasikan dengan kromosom, yang disebut dengan populasi. Solusi dari suatu populasi diambil untuk membentuk populasi baru, dimana pemilihannya tergantung dari nilai *fitness*. Hal ini diharapkan agar populasi baru yang terbentuk akan lebih baik dari populasi terdahulu. Proses ini dilakukan berulang-ulang sampai terpenuhi kondisi tertentu (Kurnia, 2006).

Dalam tugas akhir ini masalah optimasi distribusi untuk menekan biaya transportasi akan didasarkan pada pencarian rute terpendek, waktu tercepat dan kondisi-kondisi yang timbul di dalamnya untuk sebuah jalur perjalanan dari posisi awal menuju ke beberapa posisi tujuan dan berakhir pada posisi akhir. Berdasarkan latar belakang yang telah dipaparkan, maka judul yang diambil dalam tugas akhir ini adalah **"Optimasi Rute Pendistribusian Komoditi Berdasarkan Jarak dan Kondisi Jalan Untuk Meminimalkan Biaya Bahan Bakar Dengan Menggunakan Algoritma Genetika (Studi Kasus di PT.XYZ) ".**

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah di atas, maka dalam tugas akhir ini dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana menerapkan algoritma genetika pada persoalan pencarian rute terpendek dan waktu tercepat serta meminimalkan biaya bahan bakar dalam distribusi komoditi?
2. Bagaimana efisiensi metode *crossover* dan mutasi yang digunakan?

1.3 Batasan Masalah

Adapun batasan masalah dalam penulisan tugas akhir ini adalah:

1. Studi kasus yang dipakai pada aplikasi ini adalah data dari PT.XYZ yang bergerak di bidang transportasi.

2. Persoalan transportasi yang dibahas merupakan persoalan transportasi distribusi komoditi dari satu sumber ke satu tujuan dan satu sumber ke beberapa tujuan.
3. Setiap perjalanan kendaraan, dapat memiliki titik awal dan titik tujuan yang berbeda.
4. Diasumsikan bahwa pada setiap perjalanan jumlah komoditi yang diangkut sesuai dengan ketentuan kapasitas kendaraan.
5. Kondisi jalan yang diperhitungkan adalah persentase dari kepadatan lalu lintas, jalan berlubang yang akan mengakibatkan kenaikan waktu tempuh kendaraan.
6. Dasar perhitungan biaya BBM adalah nilai jarak tempuh dibagi degan rasio BBM sesuai dengan tipe kendaraan dikalikan dengan harga BBM yang berlaku.

1.4 Tujuan

Tujuan yang hendak dicapai di dalam pembuatan tugas akhir ini adalah untuk mendapatkan hasil yang optimal di dalam masalah transportasi dari segi jarak dan kondisi jalan serta biaya BBM dalam distribusi komoditi dengan menggunakan algoritma genetika pada PT.XYZ.

1.5 Manfaat

Formula optimasi yang didapatkan diharapkan dapat membantu para praktisi distribusi untuk menemukan skenario perjalanan distribusi yang optimal.

1.6 Metodologi Pemecahan Masalah

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah:

1. Studi literatur
   Mempelajari teori-teori yang berhubungan dengan konsep pencarian rute terpendek dan distribusi komoditi dengan algoritma genetika dari berbagai referensi.
2. Pendefinisian dan analisa masalah
   Mendefinisikan dan menganalisa masalah untuk mencari solusi yang tepat.
3. Perancangan dan implementasi sistem
   Membuat perancangan perangkat lunak dengan analisa terstruktur dan mengimplementasikan hasil rancangan tersebut yaitu membuat sistem

pencarian solusi masalah transportasi dengan menggunakan algoritma genetika.

4. Uji coba dan analisa hasil implementasi
   Menguji perangkat lunak dengan data yang sebenarnya, dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievaluasi dan disempurnakan.

1.7 Sistematika Penulisan

Tugas akhir ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN
   Berisi tentang latar belakang masalah, perumusan masalah, batasan masalah, tujuan penulisan, manfaat dan metode pemecahan masalah serta sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Menguraikan teori-teori yang berhubungan dengan algoritma genetika dan pencarian rute terpendek, waktu tercepat dalam distribusi komoditi.
3. BAB III METODOLOGI DAN PERANCANGAN SISTEM
   Pada bab ini akan dijelaskan mengenai metode-metode yang digunakan dalam menyelesaikan masalah pencarian rute terpendek, waktu tercepat dalam distribusi komoditi menggunakan algoritma genetika.
4. BAB IV IMPLEMENTASI DAN PEMBAHASAN
   Bab ini berisi tentang penjelasan implementasi sistem dan hasil pengujian yang dilakukan.
5. BAB V PENUTUP
   Berisi tentang kesimpulan dari seluruh rangkaian penulisan tugas akhir ini serta kemungkinan saran pengembangannya.

BAB II
TINJAUAN PUSTAKA

2.1 Algoritma Genetika

2.1.1 Pengertian Algoritma Genetika

Algoritma genetika merupakan teknik pencarian yang menirukan mekanisme seleksi dan genetika natural (Gen dan Cheng, 1997).

Konsep dasar yang mengilhami timbulnya algoritma genetika adalah teori evolusi alam yang dikemukakan oleh Charles Darwin. Dalam teori tersebut dijelaskan bahwa pada proses evolusi alami, setiap individu harus melakukan adaptasi terhadap lingkungan disekitarnya agar dapat bertahan hidup. Empat kondisi yang sangat mempengaruhi proses evolusi yaitu:

- Kemampuan organisme untuk melakukan reproduksi
- Keberadaan populasi organisme yang bisa melakukan reproduksi
- Keberagaman organisme dalam suatu populasi
- Perbedaan kemampuan untuk bertahan

Individu yang lebih kuat akan memiliki tingkat ketahanan dan tingkat reproduksi yang lebih tinggi jika dibandingkan dengan individu yang kurang fit. Sehingga populasi secara keseluruhannya akan lebih banyak memuat organisme yang fit (Kusumadewi, 2005).

Pertama kali algoritma ini dikemukakan oleh John Holland di Universitas Michigan pada tahun 1960an dengan dua tujuan yaitu:

- Menjelaskan proses adaptasi dari suatu sistem alami
- Mendesain suatu perangkat lunak kecerdasan buatan yang dapat mengerti mekanisme penting dari sistem alami.

Kemudian pada pertengahan tahun 1980-an, algoritma genetika ini menjadi teori yang sangat popular ketika diadakan konferensi internasional algoritma genetika pertama di Universitas Illionis.

2.1.2 Manfaat Algoritma Genetika

Keuntungan penggunaan algoritma genetika sangat jelas terlihat dari kemudahan implementasi dan kemampuannya untuk menemukan solusi yang ‘bagus’ (bisa diterima) secara cepat untuk masalah-masalah

berdimensi tinggi. Algoritma genetika sangat berguna dan efisien untuk masalah dengan karakteristik sebagai berikut (Suyanto, 2005):

a. Ruang masalah sangat besar, kompleks, dan sulit dipahami.
b. Kurang atau bahkan tidak ada pengetahuan yang memadai untuk merepresentasikan masalah ke dalam ruang pencarian yang lebih sempit.
c. Tidak tersedianya analisis matematika yang memadai.
d. Ketika metode-metode konvensional sudah tidak mampu menyelesaikan masalah yang dihadapi.
e. Solusi yang diharapkan tidak harus paling optimal, tetapi cukup 'bagus' atau bisa diterima.

Terdapat batasan waktu, misalnya dalam *real time systems* atau sistem waktu nyata.

2.1.3 Struktur Umum Algoritma Genetika

Algoritma genetika merupakan teknik pencarian yang didasarkan pada mekanisme seleksi dan genetika alami. Berbeda dengan teknik pencarian konvensional, algoritma genetika berangkat dari himpunan solusi yang dihasilkan secara acak. Himpunan ini dinamakan populasi (*population*). Masing-masing individu dalam populasi disebut dengan kromosom (*chromosome*). Kromosom terdiri atas gen yang tersusun secara linier. Posisi yang ditempati gen dalam kromosom disebut *loci*, sedangkan nilai yang terdapat dalam gen disebut *allele.* Istilah yang terdapat pada algoritma genetika dapat dilihat pada Tabel 2.1.

Tabel 2.1 Istilah dalam algoritma genetika

Istilah Biologi	Keterangan
Chromosom	Individu yang berupa segmen string yang sudah ditentukan.
Gen	Bagian dari string.
Loci	Posisi dari gen.
Allele	Nilai yang dimasukkan dalam gen.
Phenotype	String yang merupakan solusi akhir.
Genotype	Sejumlah string hasil perkawinan yang berpotensi sebagai solusi.

Populasi awal dibangun secara acak, sedangkan populasi berikutnya merupakan hasil evolusi kromosom-kromosom melalui iterasi yang disebut generasi. Pada setiap generasi, kromosom akan melalui proses evaluasi

dengan menggunakan alat ukur yang disebut dengan fungsi *fitness*. Nilai *fitness* dari suatu kromosom akan menunjukkan kualitas kromosom dalam populasi tersebut (Kusumadewi, 2005).

Generasi berikutnya dikenal dengan istilah anak (*offspring*), yang terbentuk dari gabungan 2 kromosom generasi sekarang yang bertindak sebagai induk (*parent*) dengan menggunakan operator penyilangan (*crossover*). Selain operator penyilangan, suatu kromosom dapat juga dimodifikasi dengan cara mutasi. Kondisi berhenti dari algoritma genetika terjadi apabila solusi yang diberikan telah konvergen atau jumlah generasi yang diminta telah tercapai (Kusumadewi, 2003).

Secara garis besar, algoritma genetika dapat dijelaskan dengan algoritma berikut (Ivan, 2003):

1. [**Mulai**] Membangun populasi secara random sebanyak n kromosom (sesuai dengan masalahnya)
2. **[Fitness**] Evaluasi nilai *fitness* f(x) dari setiap kromosom x pada populasi
3. [**Populasi baru**] Membuat populasi baru dengan mengulang langkah-langkah berikut sampai populasi baru lengkap :
   a. [**Seleksi**] Pilih dua kromosom induk dari populasi berdasarkan *fitness*nya (semakin besar *fitness*nya semakin besar kemungkinannya untuk terpilih)
   b. [**Perkawinan silang**] Sesuai dengan besarnya kemungkinan perkawinan silang, induk terpilih disilangkan untuk membentuk anak. Jika tidak ada perkawinan silang, maka anak merupakan salinan dari induknya.
   c. [**Mutasi**] Sesuai dengan besarnya kemungkinan mutasi, anak dimutasi pada *loci* (posisi pada kromosom).
   d. [**Penerimaan**] tempatkan anak baru pada 'populasi baru'.
4. [**Ganti**] Gunakan populasi yang terbentuk pada generasi selanjutnya untuk proses algoritma selanjutnya.
5. [**Tes**] jika kondisi akhir terpenuhi, berhenti, dan hasilnya adalah solusi terbaik dari populasi saat itu.
6. [**Ulangi**] Ke nomer 2.

Apabila *P*(t) dan *C*(t) merupakan *parent* dan *offspring* pada generasi *t*, maka *pseudo code* dari algoritma genetika dapat dituliskan seperti pada Gambar 2.1.(Michalewicz, 1996):

```
1  procedure Algoritma Genetika
2    begin
```

```
3    t  ← 0;
4    initialize P(t);
5    evaluate P(t);
6    while (not termination condition) do
7      recombine P(t) to yield C(t);
8      evaluate P(t);
9
10   select P(t+1) from P(t) and C(t);
11   t ← t+1;
12   end
13 End
```

Gambar 2.1 *Pseudo code* algoritma genetika

2.2 Pengkodean

Langkah pertama yang dilakukan pada algoritma genetika adalah menerjemahkan atau merepresentasikan masalah riil menjadi terminologi biologi. Cara untuk merepresentasikan masalah ke dalam bentuk kromosom disebut pengkodean. Ada beberapa cara pengkodean dan pemilihannya berdasarkan masalah yang dihadapi. Berikut adalah beberapa cara pengkodean yang umum digunakan:

a. Pengkodean Biner

Merupakan pengkodean yang paling umum dan paling sederhana dalam merepresentasikan masalah pada algoritma genetika. Pada pengkodean biner setiap kromosom terdiri atas barisan string bit 0 atau 1. Contoh pengkodean biner dapat dilihat pada Gambar 2.2.

Kromosom A	0101101100010011
Kromosom B	1011010110110101

Gambar 2.2 Contoh kromosom pada pengkodean biner.

Contoh masalah yang dapat menggunakan pengkodean biner adalah masalah nilai *maximize* atau *minimize* pada sebuah fungsi matematika.

b. Pengkodean Permutasi

Pengkodean ini dapat digunakan pada masalah pengurutan data (*ordering problems*), seperti wiriniaga (*Travelling Salesman*

Problem) atau masalah pengurutan tugas (*Task Ordering Problem*). Pada pengkodean permutasi, setiap kromosom terdiri dari barisan angka, yang merepresentasikan angka pada urutan. Contoh pengkodean permutasi dapat dilihat pada Gambar 2.3.

Kromosom A	8549102367
Kromosom B	9102438576

Gambar 2.3 Contoh kromosom pada pengkodean permutasi

c. Pengkodean Nilai

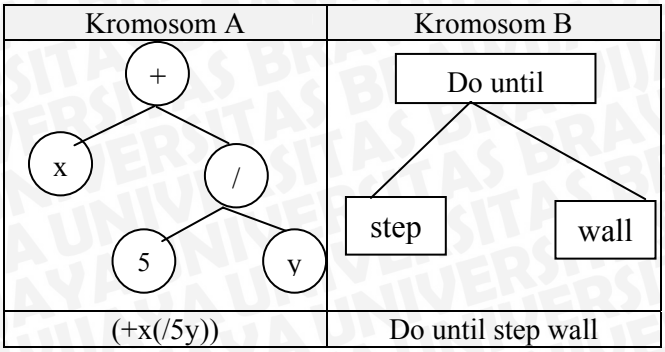
Pengkodean ini dapat digunakan untuk masalah yang sangat kompleks, dimana nilai yang dikodekan langsung merupakan representasi dari masalah. Pada pengkodean nilai, setiap kromosom adalah barisan dari beberapa nilai. Nilai dapat berupa apa saja, seperti bilangan biasa, bilangan riil, karakter sampai dengan obyek-obyek yang rumit. Contoh pengkodean nilai dapat dilihat pada Gambar 2.4.

Kromosom A	[red], [black], [blue], [yellow]
Kromosom B	1.875, 3.9821, 9.1283, 6.8344
Kromosom C	A, B, C, D, E, F, G, H, I, J, K

Gambar 2.4 Contoh kromosom pada pengkodean nilai

d. Pengkodean Pohon

Pengkodean ini lebih banyak digunakan untuk menyusun program atau ekspresi, bagi pemrograman genetika (*genetic programming*). Pada pengkodean pohon, setiap kromosom merupakan pohon dari sejumlah obyek, seperti fungsi atau perintah pada bahasa pemrograman. Contoh pengkodean pohon dapat dilihat pada Gambar 2.5.



Gambar 2.5 Contoh kromosom dengan pengkodean pohon

2.3 Fungsi Evaluasi

Suatu individu dievaluasi berdasarkan fungsi tertentu sebagai ukuran performansinya. Di dalam evolusi alam, individu yang bernilai *fitness* rendah akan mati. Pada masalah optimasi, jika solusi yang dicari adalah memaksimalkan sebuah fungsi *h* (dikenal sebagai masalah maksimasi), maka nilai *fitness* yang digunakan adalah nilai dari fungsi *h* tersebut, yakni *fitness f = h*. Tetapi jika masalahnya adalah minimalkan fungsi *h* (masalah minimasi), maka fungsi *h* tidak bisa digunakan secara langsung. Hal ini disebabkan adanya aturan bahwa individu yang memiliki nilai *fitness* tinggi lebih mampu bertahan hidup pada generasi berikutnya. Oleh karena itu nilai *fitness* yang bisa digunakan adalah *f = 1/h*, yang artinya semakin kecil *h*, semakin besar nilai *f*. Tetapi hal ini akan menjadi masalah jika *h* bernilai 0, yang mengakibatkan *f* bisa bernilai tak hingga. Untuk mengatasinya, *h* perlu ditambah sebuah bilangan yang dianggap sangat kecil sehingga nilai *fitness* menjadi seperti pada Persamaan 2.1.

$$f = \frac{1}{(h+a)} \qquad (2.1)$$

Dimana *a* adalah bilangan yang dianggap sangat kecil dan bervariasi sesuai dengan masalah yang akan diselesaikan (Suyanto, 2005).

2.4 Seleksi

Proses seleksi bertanggung jawab untuk melakukan pemilihan terhadap individu yang hendak diikutkan dalam proses reproduksi. Langkah pertama yang dilakukan dalam seleksi ini adalah pencarian nilai *fitness*. Seleksi

mempunyai tujuan untuk memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang mempunyai nilai *fitness* terbaik.

Ada beberapa definisi yang biasa digunakan untuk melakukan perbandingan terhadap beberapa metode yang akan digunakan, antara lain (Kusumadewi, 2003):

- *Selective pressure*: probabilitas dari individu terbaik yang akan diseleksi dibandingkan dengan rata-rata probabilitas dari semua individu yang diseleksi.
- *Bias*: perbedaan absolute antara *fitness* ternormalisasi dari suatu individu dan probabilitas reproduksi yang diharapkan.
- *Spread*: *range* nilai kemungkinan untuk sejumlah *offspring* dari suatu individu.
- *Loss of diversity*: proporsi dari individu-individu dalam suatu populasi yang tidak terseleksi selama fase seleksi.
- *Selection intensity*: nilai *fitness* rata-rata yang diharapkan dalam suatu populasi setelah dilakukan seleksi (menggunakan distribusi Gauss ternormalisasi).
- *Selection variance*: variansi yang diharapkan dari ditribusi *fitness* dalam populasi setelah dilakukan seleksi (menggunakan distribusi Gauss ternormalisasi).

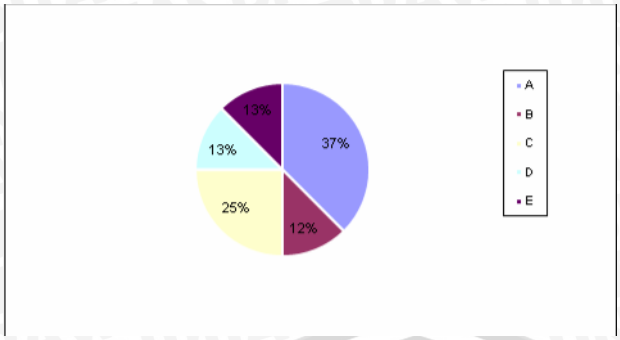
Ada beberapa metode seleksi dari induk, antara lain:

1. Seleksi Roda *Roulette*

   Metode seleksi ini juga sering dikenal dengan istilah *stochastic sampling with replacement*. Pada metode ini, individu-individu dipetakan dalam suatu segmen garis secara berurutan sedemikian hingga tiap-tiap segemen individu memiliki ukuran yang sama dengan ukuran *fitness* yang dimiliki (Kusumadewi, 2005). Maka semakin besar nilai *fitness* yang dimiliki maka semakin besar kemungkinan untuk terpilih menjadi induk. Diandaikan semua kromosom diletakkan pada sebuah roda *roulette*, besarnya kemungkinan bagi setiap kromosom adalah tergantung dari nilai *fitness*nya seperti pada Gambar 2.6 dan Gambar 2.7.

Kromosom	*Fitness*
A	15
B	5
C	10
D	5
E	5

Gambar 2.6 Contoh populasi dengan 5 kromosom



Gambar 2.7 Probabilitas suatu kromosom dalam roda *roulette*

Pada Gambar 2.6 merupakan contoh dimana dalam satu populasi terdiri dari lima kromosom. Pada tiap kromosom memiliki nilai *fitness* yang berbeda-beda. Pada Gambar 2.7 dapat diketahui probabilitas terpilihnya masing-masing kromosom untuk menjadi induk. Kromosom A memiliki nilai *fitness* 15 dan nilai tersebut adalah nilai *fitness* tertinggi pada populasi. Sehingga kromosom A memiliki probabilitas terbesar untuk terpilih menjadi induk.

Metode seleksi roda *roulette* akan bermasalah saat terdapat perbedaan *fitness* yang besar. Sebagai contoh, jika *fitness* kromosom yang terbaik adalah 90% dari semua roda *roulette* dapat menyebabkan kromosom lain memiliki kesempatan yang kecil untuk dapat terpilih.

2. Seleksi Rangking

   Metode seleksi ini dimulai dengan merangking atau mengurutkan kromosom di dalam populasi berdasarkan *fitness*nya kemudian memberi nilai *fitness* baru berdasarkan urutannya. Kromosom dengan nilai *fitness* terburuk akan memiliki *fitness* baru bernilai 1, terburuk kedua bernilai 2 dan begitu seterusnya, sehingga kromosom yang memiliki *fitness* terbaik akan memiliki nilai *fitness* N, dimana N adalah jumlah kromosom di dalam populasi. Contoh seleksi rangking dapat dilihat pada Gambar 2.8 dan Gambar 2.9

Kromosom	*Fitness*
A	15
B	5
C	10
D	5
E	5

Gambar 2.8 Keadaan sebelum dirangking

Kromosom	*Fitness*	*Fitness* baru
B	5	1
D	5	2
E	5	3
C	10	4
A	15	5

Gambar 2.9 Keadaan setelah dirangking

Pada Gambar 2.9 dapat dilihat hasil seleksi tersebut. Maka saat ini seluruh kromosom memiliki kesempatan untuk dipilh. Akan tetapi, metode ini dapat menyebabkan konvergensi menjadi lambat, karena kromosom terbaik tidak terlalu berbeda dengan yang lainnya.

3. Seleksi Turnamen
   Metode ini merupakan variasi antara metode roda *roulette* dan metode rangking. Sejumlah *k* kromosom tertentu dari populasi beranggota *n* kromosom $(k \leq n)$ dipilih secara acak dengan probabilitas yang sama. Dari *k* kromosom yang terpilih kemudian akan dipilih satu kromosom dengan *fitness* terbaik, yang diperoleh dari hasil pengurutan rangking *fitness* semua kromosom terpilih. Perbedaan dengan seleksi roda *roulette* adalah pemilihan kromosom yang akan digunakan untuk berkembang biak tidak berdasarkan skala *fitness* dari populasi (Kurnia, 2006).

4. Seleksi Pengambilan *Sampling Stochastic*

Metode ini disebut juga *stochastic universal sampling*. Pada metode ini, individu-individu dipetakan dalam suatu segmen garis secara berurutan sedemikian hingga tiap-tiap segmen individu memiliki ukuran yang sama dengan ukuran *fitness*nya seperti halnya pada seleksi roda *roulette*. Kemudian diberikan pointer sebanyak individu yang ingin diseleksi pada garis tersebut (Kusumadewi, 2005).

2.5 Operator Genetika

Setelah proses seleksi dilakukan, proses selanjutnya adalah operasi genetika. Terdapat 2 operator genetika, yaitu perkawinan silang dan mutasi.

2.5.1 Perkawinan Silang

Proses kawin silang (*crossover*) *gen* merupakan suatu mekanisme yang penting untuk terjadinya suatu evolusi. Proses kawin silang berfungsi untuk menghasilkan keturunan dari dua buah kromosom induk yang terpilih. Kromosom anak yang dihasilkan merupakan kombinasi dari gen-gen yang dimiliki kromosom kedua induknya.
Secara umum, mekanisme tukar silang adalah sebagai berikut:

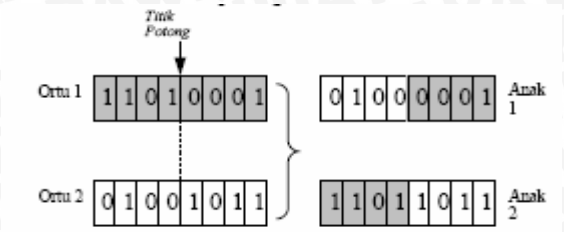
1. Memilih dua buah kromosom sebagai induk.
2. Memilih secara acak posisi dalam kromosom, biasa disebut titik perkawinan silang, sehingga masing-masing kromosom induk terpecah menjadi dua segmen.
3. Lakukan pertukaran antar segmen kromosom induk untuk menghasilkan kromosom anak.

2.5.1.1 Perkawinan Silang untuk Pengkodean Biner

Pada pengkodean biner terdapat 4 metode penyilangan yang sering digunakan. Metode tersebut adalah:

a. Perkawinan silang satu titik

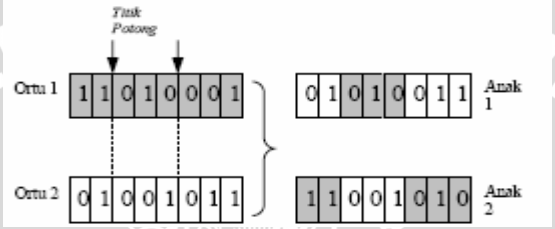
Proses ini dimulai dengan memilih satu titik pada barisan bit kromosom secara acak sebagai titik perkawinan silang. Kromosom baru akan dibentuk dengan cara menyalin barisan bit induk pertama dari bit pertama sampai titik perkawinan silang, sedangkan sisanya disalin dari induk kedua. Contoh dari perkawinan silang satu titik dapat dilihat pada Gambar 2.10.



Gambar 2.10 Pindah silang satu titik

b. Perkawinan silang dua titik

Proses ini dimulai dengan menentukan secara acak dua titik yang akan disilangkan. Kromosom baru akan dibentuk dengan cara menyalin barisan bit kromosom induk pertama dari bit pertama sampai dengan titik perkawinan silang pertama dari titik perkawinan silang kedua sampai dengan bit terakhir, sedangkan sisanya, yaitu titik perkawinan silang pertama sampai titik perkawinan silang kedua disalin dari induk kedua. Contoh perkawinan silang dua titik dapat dilihat pada Gambar 2.11.



Gambar 2.11 Pindah silang dua titik

c. Perkawinan silang seragam

Sebuah *mask* penyilangan dibuat sepanjang panjang kromosom secara random yang menunjukkna bit-bit dalam *mask* yang mana induk akan mensuplai anak dengan bit-bit yang ada. Disini, anak_1 akan dihasilkan dari induk_1 jika bit *mask* bernilai 1, dan anak_1 akan dihasilkan dari induk_2 jika bit *mask* bernilai 0. Sedangkan anak_2 dihasilkan dari kebalikan *mask* (Kusumadewi, 2003). Contoh dari perkawinan silang seragam adalah berikut ini:

Misalkan ada 2 kromosom dengan panjang 12:

Induk 1: 011100101110

Induk 2: 110100001101

Mask bit:

sampel 1: 100111001101
sampel 2: 011000110010

Setelah penyilangan, diperoleh kromosom baru:

Anak 1: 010100001100
Anak 2: 111100101111

2.5.1.2 Perkawinan Silang untuk Pengkodean Permutasi

Pada pengkodean permutasi, perkawinan silang yang sering digunakan adalah perkawinan silang satu titik, karena selain prosesnya yang sederhana juga dapat menjaga konsistensi urutan nilai pada kromosom.

Proses perkawinan silang satu titik dimulai dengan pemilihan satu titik perkawinan silang. Dari permutasi pertama sampai dengan titik perkawinan silang disalin dari induk pertama, sedangkan sisanya didapatkan dengan cara melihat satu persatu nilai pada orang tua kedua, jika belum ada pada kromosom keturunan, maka nilai tersebut ditambahkan. Contoh dari perkawinan silang untuk pengkodean permutasi adalah berikut ini:

Induk 1 : **1 2 3 4 5** 6 7 8 9
Induk 2 : **4 5 3 6 8** 9 7 2 1
Anak (AND) : **1 2 3 4 5** 6 8 7 9

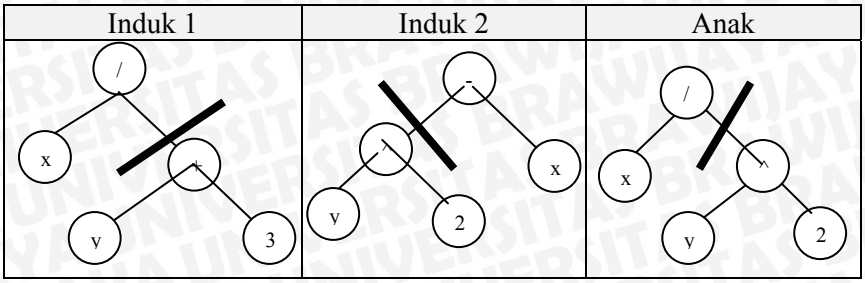
2.5.1.3 Perkawinan Silang untuk Pengkodean Nilai

Semua metode perkawinan silang yang terdapat pada pengkodean biner dapat diterapkan pada pengkodean nilai. Hal tersebut dikarenakan pada pengkodean nilai tidak perlu diperhatikan urutan seperti pada pengkodean permutasi. Berikut contoh perkawinan silang pada pengkodean nilai:

Induk1 : **10 39 45 12** 9 65 12 76
Induk 2 : 45 86 75 12 **3 85 64 58**
Anak : **10 39 45 12 3 85 64 58**

2.5.1.4 Perkawinan Silang untuk Pengkodean Pohon

Perkawinan silang untuk pengkodean pohon dimulai dengan pemilihan satu titik perkawinan silang pada induk, kemudian bagian dari orang tua yang berada di bawah titik perkawinan silang dipertukarkan untuk menghasilkan anak baru. Contoh perkawinan silang untuk pengkodean pohon dapat dilihat pada Gambar 2.12.



Gambar 2.12 Contoh perkawinan silang pada pengkodean pohon

2.5.2 Mutasi

Setelah melalui proses perkawinan silang, pada *offspring* dapat dilakukan proses mutasi. Mutasi dilakukan dengan cara melakukan perubahan pada sebuah gen atau lebih dari sebuah individu.

Tujuan dari mutasi adalah untuk membentuk individu-individu dengan fitur superior atau memiliki kualitas di atas rata-rata. Selain itu mutasi dipergunakan untuk mengembalikan kerusakan materi genetik akibat proses *crossover*. Mutasi akan sangat berperan jika pada populasi awal hanya ada sedikit solusi yang mungkin terpilih. Sehingga, operasi ini sangat berguna dalam mempertahankan keanekaragaman individu dalam populasi meskipun dengan mutasi tidak dapat diketahui apa yang terjadi pada individu baru.

Peluang mutasi menunjukkan jumlah total gen pada populasi yang akan mengalami mutasi. Untuk melakukan mutasi, harus dihitung jumlah total gen pada populasi tersebut terlebih dahulu. Kemudian dibangkitkan bilangan random yang akan menentukan posisi yang akan dimutasi.

Mutasi nilai gen dari suatu kromosom dapat dilakukan dengan dua cara (Novitasari, 2007), yaitu

- Cara random yaitu dengan menentukan dua gen yang akan dimutasi. Setelah itu nilai kedua gen tersebut dirandom ulang untuk mendapatkan nilai yang baru.
- Cara *swap* atau penukaran yaitu dengan menukar langsung nilai dari gen. Pemilihan gen yang akan ditukar dilakukan secara random.

Jika peluang mutasi terlalu kecil, banyak gen yang mungkin berguna tidak pernah dievaluasi. Tetapi bila peluang mutasi terlalu besar, maka akan terlalu banyak gangguan acak, sehingga anak akan kehilangan kemiripan dari induknya (Kurnia, 2006).

Misalkan ukuran populasi (popsize = 100), setiap kromosom memiliki panjang 20 *gen*, maka total *gen* adalah 100x20=2000 *gen*. Jika peluang mutasi (p_m=0,01), berarti bahwa diharapkan ada (1/100) x 2000 = 20 *gen* akan mengalami mutasi (Kusumadewi, 2003).

2.6 Transportasi Distribusi Komoditi PT.XYZ

2.6.1 Sistem Pendistribusian Komoditi

Sistem pendistribusian komoditi pada PT.XYZ, meliputi:

a. Kegiatan pendistribusian bahan baku ke pabrik dan pendistribusian barang jadi ke gudang yang telah ditentukan.
b. Kegiatan pendistribusian bahan baku dimulai dengan perjalanan kendaraan dari garasi dengan muatan kosong ke *plant* untuk mengambil bahan baku. Kemudian dengan muatan bahan baku, kendaraan bergerak menuju ke satu pabrik atau beberapa pabrik yang telah ditentukan.
c. Kegiatan penditribusian barang jadi dapat dibagi menjadi dua siatem, yang pertama dimulai dengan perjalanan kendaraan dari garasi langsung menuju ke pabrik untuk menangkut barang jadi dan mendistribusikan ke gudang yang telah ditentukan. Yang kedua diawali dengan kegiatan pendistribusian bahan baku, kemudian kendaraan memuat barang jadi untuk didistribusikan ke satu gudang atau beberapa gudang yang telah ditentukan.

2.6.2 Pemilihan Rute

Hal utama dalam proses pembebanan rute adalah memperkirakan pemilihan penggunaan jalan yang terbaik. Terdapat beberapa faktor yang mempengaruhi rute pada saat melakukan perjalanan. Beberapa diantaranya adalah waktu tempuh, jarak, biaya (bahan bakar dan lainnya), kemacetan dan antrian, kelengkapan rambu dan marka jalan, serta kebiasaan (Sulistiyaningtias, 2003).

Salah satu pendekatan yang paling sering digunakan adalah pertimbangan dua faktor utama dalam pemilihan rute, yaitu biaya pergerakan dan nilai waktu pergerakan yang dianggap proporsional dengan jarak tempuh. Dalam beberapa model pemilihan rute dimungkinkan penggunaan bobot yang berbeda bagi faktor waktu tempuh dan faktor jarak tempuh untuk menggambarkan persepsi pengendara dalam kedua faktor

tersebut. Dalam sistem pendistribusian PT.XYZ, jarak tempuh mempunyai bobot lebih dominan daripada waktu tempuh yang pada akhirnya akan menimbulkan nilai biaya.

2.6.3 Spesifikasi Kendaraan

PT.XYZ memiliki 175 armada yang terdiri dari 3 jenis *truck*, dengan kapasitas dan rasio BBM yang berbeda dan tersebar di 5 pangkalan armada (*basetown*) yang ada. Spesifikasi kapasitas kendaraan adalah sebagai berikut:

- Tronton : 12.640 Kg (12,64 ton)
- Engkel : 7.370 Kg (7,37 ton)
- *Light-Truck* : 3.740 Kg (3,74 ton)

Sedangkan pedoman rasio penggunaan BBM kendaraan adalah sebagai berikut:

- Rasio BBM standar tipe Tronton adalah 1:3,0 (1 liter solar dapat digunakan untuk menempuh jarak 3,0 km).
- Rasio BBM standar tipe Engkel adalah 1:3,6 (1 liter solar dapat digunakan untuk menempuh jarak 3,6 km).
- Rasio BBM standar tipe *Light-Truck* adalah 1:4,1 (1 liter solar dapat digunakan untuk menempuh jarak 4,1 km).

Sedangkan jumlah unit yang berada di tiap-tiap *basetown* dapat dilihat pada Tabel 2.2.

Tabel 2.2 Penyebaran kendaraan pada tiap *base town*

Base town	*Truck* Tronton	*Truck* Engkel	*Light-Truck*
Surabaya	44	3	-
Pandaan	61	-	-
Semarang	18	-	-
Jakarta	29	-	7
Bandung	11	-	2

2.6.4 Ketentuan Kecepatan Standar Kendaraan

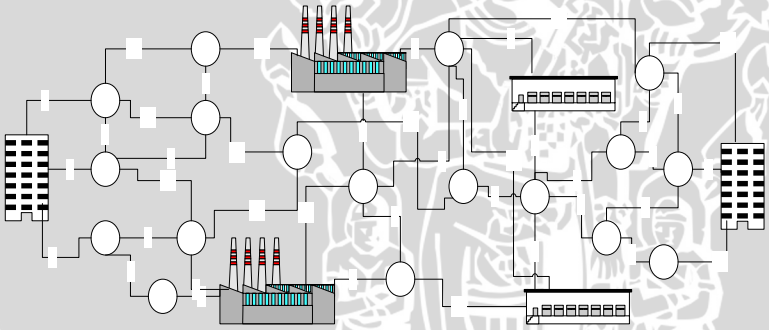
PT.XYZ memiliki ketentuan kecepatan standar kendaraan dan lama waktu istirahat perjalanan berdasarkan jarak tempuh. Ketentuan kecepatan standar kendaraan dan waktu istirahat yang diperbolehkan dapat dilihat pada Tabel 2.3.

Tabel 2.3 Ketentuan kecepatan standar dan waktu istirahat

Jarak Tempuh (Km)	Kecepatan Standar (Km/jam)	Waktu Istirahat (jam)
0-100	55	1
101-300	55	2
301-500	50	3
501-600	50	3,5
601-700	47	4
701-800	47	5
801-900	45	5
901-1000	45	5,5
1001-1300	45	6
1301-1600	45	6
1601-2000	45	6
2001-2400	42	6
> 2400	42	6

2.7 Contoh Perhitungan Manual

Misal terdapat 5 kendaraan (3 *truck* tronton dan 2 *light-truck*) dengan 2 jenis kendaraan yang berbeda, dan berada di 2 tempat garasi yang berbeda (*garage1*:2 *truck* tronton dan 1 *light-truck; garage2*:1 *truck* tronton dan 1 *light-truck*), dan mengunjungi 4 titik tujuan (*factory1*, *factory2*, *warehaouse1*, *warehouse2*). Penggambaran peta pembobotan jarak untuk contoh perhitungan manual dapat dilihat pada Gambar 2.13.



Gambar 2.13 Penggambaran peta pembobotan jarak

Alur perjalanan masing-masing *truck* dapat dilihat pada Tabel 2.4

Tabel 2.4 Contoh perjalanan kendaraan

Base town awal	Kendaraan	Titik muat barang	Titik bongkar muatan	*Base town* akhir
Garage 1	*Truck* Tronton1	*Factory* 1	*Warehouse* 2	*Garage* 2
Garage 1	*Truck* Tronton2	*Factory* 1 *Factory 2*	*Warehouse* 1	*Garage* 2
Garage 1	*Light-Truck* 1	*Factory* 1	*Warehouse* 2 *Warehouse 1*	*Garage* 2
Garage 2	*Light-Truck* 2	*Factory* 2	*Warehouse* 1	*Garage* 2
Garage 2	*Truck* Tronton3	*Factory* 2	*Warehouse* 2	*Garage* 1

Kapasitas yang diangkut oleh *truck* dianggap sesuai dengan kapasitas *truck* yang digunakan. Sebab barang yang diangkut berorientasi pada volume bukan pada berat. Sehingga tiap *truck* hanya melakukan satu kali perjalanan untuk rute yang sama.

2.7.1 Perhitungan Jarak Tempuh

Untuk perhitungan manual dapat dilakukan sebagai berikut ini:

1. Dari *garage* 1
   - *Truck* Tronton 1, melakukan perjalanan sebagai berikut:
     *Garage*1→1→4→*Factory*1→10→*Warehouse*2→13→15→18→*Garage2*
     Total jarak: 9+15+14+3+8+6+7+5+6 = 73km
   - *Truck* Tronton 2, melakukan perjalanan sebagai berikut:
     *Garage*1→2→1→4→*Factory*1→9→*Factory*2→11→*Warehouse*1→13→*Warehouse2*→13→15→18→*Garage2*
     Total jarak: 2+3+15+14+8+10+4+18+6+6+6+7+5+6=110
   - *Light-Truck* 1, melakukan perjalanan sebagai berikut:
     *Garage*1→2→5→4→*Factory*1→10→*Warehouse*2→13→*Warehouse*1→13→14→17→*Garage2*

Total jarak: 2+8+4+14+3+8+6+6+6+9+3+12=81km

2. Dari *garage* 2
   - *Light-Truck* 2, melakukan perjalanan sebagai berikut:
     *Garage*2→16→10→*Factory*1→9→*Factory*2→11→
     *Warehouse*1→13→15→18→*Garage2*
     Total jarak: 16+18+3+8+10+4+18+6+7+5+6=101km
   - *Truck* Tronton 3, melakukan perjalanan sebagai berikut:
     *Garage*2→17→14→13→*Warehouse*1→11→*Factory*2→11→*War*
     *ehouse*1→11→*Factory*2→6→3→*Garage*1
     Total jarak: 12+3+9+6+18+4+4+18+4+9+7+5=99km

2.7.2 Perhitungan Waktu

Untuk perhitungan waktu normal dengan anggapan kondisi jalan tidak padat atau tingkat keramaian lalu lintas dalam kondisi normal, dapat dilihat pada Persamaan 2.2.

$$Waktu\ tempuh\ normal\ (WTN) = \left(\frac{Jarak\ perjalanan\ rute}{kecepatan\ standar}\right) + waktu\ istirahat \quad (2.2)$$

- *Truck* Tronton1:
  $WTN = \left(\frac{73}{55}\right) + 1 = 2{,}33\ jam$
- *Truck* Tronton2:
  $WTN = \left(\frac{110}{55}\right) + 2 = 3\ jam$
- *Light-Truck* 1:
  $WTN = \left(\frac{81}{55}\right) + 1 = 2{,}47\ jam$
- *Light-Truck* 2:
  $WTN = \left(\frac{101}{55}\right) + 2 = 3{,}84\ jam$
- *Truck* Tronton3:
  $WTN = \left(\frac{99}{55}\right) + 1 = 2{,}8\ jam$

Untuk perhitungan waktu dengan kondisi jalan padat atau keramaian lalu lintas meningkat, dapat dilihat pada Persamaan 2.3.

$$Waktu\ tempuh = (\%D * WTN) + WTN \quad (2.3)$$

Dengan %D adalah persentase kenaikan kondisi jalan dari kondisi jalan normal dan WTN adalah waktu tempuh normal.

Sebagai contoh, *truck* Tronton1 melakukan perjalanan dengan kondisi kepadatan jalan meningkat sebesar 10%, maka perhitungan waktu tempuh dengan menggunakan Persamaan 2.3 adalah sebagai berikut:

$$Waktu\ tempuh = (10\% * 2{,}33) + 2{,}33 = 2{,}56\ jam$$

2.7.3 Perhitungan Biaya BBM

Untuk perhitungan BBM dalam kondisi jalan tidak padat atau tingkat keramaian lalu lintas dalam kondisi normal, memiliki rumusan yang dapat dilihat pada Persamaan 2.4.

$$Biaya\ BBM\ Normal = \frac{Jarak\ perjalanan\ rute}{Rasio\ BBM\ sesuai\ tipe} * Harga\ Solar \quad (2.4)$$

- *Truck* Tronton1:

$$Biaya\ BBM\ Normal = \frac{73}{3.0} * Rp.5.500{,}00 = Rp.133.833{,}33$$

- *Truck* Tronton2:

$$Biaya\ BBM\ Normal = \frac{110}{3.0} * Rp.5.500{,}00 = Rp.201.666{,}67$$

- *Light-Truck* 1:

$$Biaya\ BBM\ Normal = \frac{81}{4.1} * Rp.5.500{,}00 = Rp.108.658{,}54$$

- *Light-Truck* 2:

$$Biaya\ BBM\ Normal = \frac{101}{4.1} * Rp.5.500{,}00 = Rp.135.487{,}80$$

- *Truck* Tronton3:

$$Biaya\ BBM\ Normal = \frac{99}{3.0} * Rp.5.500{,}00 = Rp.181.500{,}00$$

Kepadatan lalu lintas, tidak hanya mempengaruhi waktu tempuh perjalanan, tetapi juga mempengaruhi biaya BBM. Untuk itu ada perhitungan konversi waktu tempuh ke BBM. Perbandingan kenaikan waktu tempuh dengan kenaikan konsumsi BBM adalah 3:1. Kenaikan waktu tempuh maksimal adalah 15% dari waktu tempuh normal, sehingga kenaikan konsumsi bahan BBM maksimal adalah 5% dari biaya BBM normal. Apabila waktu tempuh melebihi 15% dalam arti kondisi jalan macet total, maka *driver* harus berhenti sampai kondisi jalan memungkinkan. Hal ini digunakan untuk menghindari kenaikan pemakaian BBM yang berlebih dan melebihi anggaran yang disediakan.

Perhitungan konversi waktu tempuh ke konsumsi BBM adalah dengan menghitung penambahan waktu tempuh (dev waktu) yang akan dipergunakan untuk menghitung kenaikan konsumsi BBM (dev BBM). Rumus perhitungannya dapat dilihat pada Persamaan 2.5 dan Persamaan 2.6.

$$Dev\ waktu = \frac{(waktu\ tempuh - WTN)}{WTN} * 100\% \quad (2.5)$$

$$Dev\ BBM = \frac{Dev\ waktu}{3} \quad (2.6)$$

Dengan dev waktu adalah persentase kenaikan waktu tempuh perjalanan dan dev BBM adalah persentase kenaikan BBM. Rumus perhitungan biaya BBM dapat dilihat pada Persamaan 2.7.

$$Biaya\ BBM = Biaya\ BBM\ normal * (1 + dev\ BBM) \quad (2.7)$$

Sebagai contoh, *truck* Tronton1 melakukan perjalanan dengan kondisi kepadatan jalan meningkat sebesar 10%, maka perhitungan konversi waktu tempuh ke konsumsi BBM adalah sebagai berikut:

$$Dev\ waktu = \frac{(2{,}56 - 2{,}33)}{2{,}33} * 100\% = 9{,}87\%$$

$$Dev\ BBM = \frac{(9{,}87\%)}{3} = 3{,}29\%$$

$$Biaya\ BBM = Rp.\ 133.833{,}33 * (1 + 0{,}0329) = Rp.\ 138.236{,}45$$

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas lebih detail mengenai metodologi dan rancangan yang digunakan dalam menentukan rute tercepat pada tugas akhir ini. Tujuan dari pembahasan ini adalah untuk memberikan gambaran dan penjelasan kepada *user* mengenai sistem serta memberikan gambaran yang jelas tentang program yang akan dibuat berdasarkan tinjauan pustaka pada bab dua.

3.1 Deskripsi Masalah

Sistem transportasi pada PT.XYZ masih menggunakan sistem transportasi secara manual, yang artinya rute transportasi ditentukan dengan menerka jalur yang akan dilalui untuk mencapai titik-titik tujuan yang telah ditentukan tanpa mengetahui apakah rute tersebut merupakan rute yang optimal. Sehingga pemakaian BBM sulit untuk dikontrol.

Alur perjalanan yang dilakukan oleh masing-masing *truck* dalam satu kali perjalanan dimulai dari berangkat dari garasi dan menuju ke *plant* untuk mengambil bahan baku. Kemudian menuju ke pabrik untuk mengantar bahan baku dan mengambil barang jadi untuk dikirim ke gudang. Setelah bongkar muatan di gudang, *truck* menuju garasi yang telah ditentukan. Sehingga titik keberangkatan (*garage1*) tidak selalu sama dengan titik akhir dari perjalanan (*garage2*). Dalam satu kali perjalanan, masing-masing *truck* minimal menuju ke dua titik tujuan yaitu tempat mengisi muatan dan tempat bongkar muatan.

Dalam penulisan tugas akhir ini, sistem yang akan dibuat adalah pencarian rute terpendek berdasarkan jarak dan kondisi jalan untuk meminimalkan biaya BBM. Sistem ini dibuat untuk memudahkan perusahaan dalam hal pengambilan keputusan untuk distribusi komoditi dengan syarat *cost* minimal dan semua tujuan tercapai. Diharapkan dengan adanya sistem ini akan sangat membantu perusahaan sehingga segala sesuatu yang akan dilakukan menjadi lebih efektif dan efisien.

Algoritma yang digunakan pada sistem ini adalah algoritma genetika dimana algoritma genetika dapat digunakan untuk menyelesaikan masalah optimasi yang kompleks seperti mencari rute paling optimum dengan memperhatikan kondisi jalan seperti kepadatan lalu lintas yang akan mempengaruhi waktu tempuh perjalanan dan konsumsi BBM.

3.2 Faktor-Faktor yang Mempengaruhi Pencarian Rute Berdasarkan Jarak dan Kondisi Jalan

Beberapa komponen yang mempengaruhi pencarian rute, waktu optimal dan pemilihan kendaraan dalam distribusi komoditi antara lain:

1. Banyaknya titik tujuan
   Banyaknya titik tujuan ditentukan oleh *user*.
2. Urutan titik tujuan
   Titik tujuan yang diinputkan terlebih dahulu yang akan menjadi tujuan pertama dan demikian seterusnya.
3. *Truck*
   Jenis *truck* yang digunakan.
4. Kondisi jalan
   Tingkat kepadatan lalu lintas diinputkan oleh *user*.
5. Waktu tempuh perjalanan
   Waktu tempuh perjalanan dihitung oleh sistem berdasarkan jarak dan kondisi jalan.
6. Biaya BBM
   Biaya BBM dihitung oleh sistem berdasarkan jenis *truck*, jarak tempuh dan kondisi jalan.

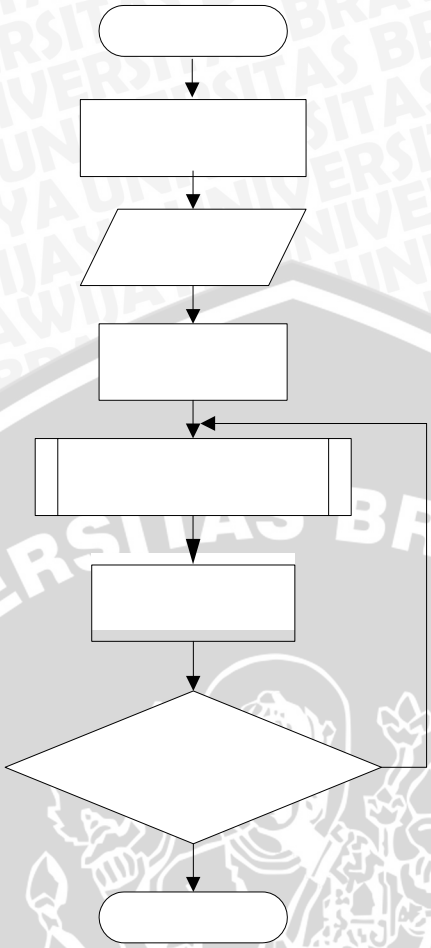
3.3 Aturan-aturan Pencarian Rute Optimal Berdasarkan Jarak dan Kondisi Jalan

1. Setiap perjalanan dapat memiliki titik awal dan titik akhir yang berbeda.
2. Urutan titik tujuan ditentukan oleh *user* sesuai dengan urutan inputan.
3. Perbandingan kenaikan waktu tempuh perjalanan dan kenaikan konsumsi BBM 3:1.
4. Kenaikan waktu tempuh perjalanan maksimal 15% dari waktu tempuh normal.
5. Kenaikan konsumsi BBM maksimal sebesar 5%.
6. Apabila tingkat kepadatan lalu lintas meningkat melebihi 15%, maka *driver* harus berhenti sampai kondisi jalan memungkinkan.

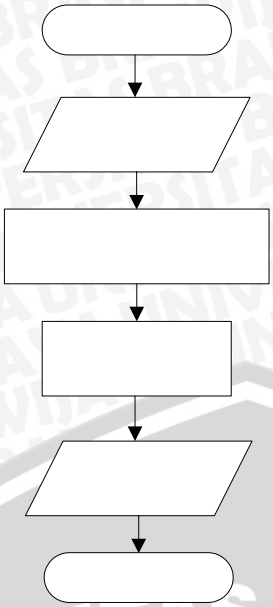
3.4 Model Genetika

Pada dasarnya algoritma genetika adalah algoritma pencarian solusi yang terbaik atau yang mendekati terbaik dari begitu banyak solusi yang ada. Agar algoritma genetika menjadi lebih jelas dalam susunan atau

urutan prosesnya, maka perlu dibuat *flowchart* (diagram alir). *Flowchart* dari cara kerja algoritma genetika dapat dilihat pada Gambar 3.1.



Gambar 3.1 *Flowchart* algoritma genetika



Gambar 3.2 *Flowchart* proses reproduksi

Flowchart algoritma genetika pada Gambar 3.1 merupakan suatu garis besar dari pembuatan *software* dari algoritma genetika. *Flowchart* proses produksi adalah penggambaran dari proses yang ada di dalam proses reproduksi dalam algoritma genetika. *Flowchart* tersebut akan dijelaskan secara umum sebagai berikut, yaitu:

1. Penentuan nilai awal
   Penentuan nilai awal ini dilakukan pada beberapa bagian yaitu penentuan nilai awal dari panjang krosmosom, penentuan cara teknik dekode dan juga penentuan fungsi obyektif.
2. Input
   Pada bagian input ini merupakan keinginan dari *user* dalam menentukan hal-hal yang akan dicari optimasinya. Input yang akan dimasukkan dalam *software* ini meliputi bermacam-macam hal, yaitu:
   - Memasukkan titik awal keberangkatan
   - Memasukkan titik tujuan dan titik akhir perjalanan
   - Memasukkan kondisi jalan
   - Memasukkan jenis *truck* yang digunakan
   - Memasukkan nilai jumlah populasi
   - Memasukkan *crossover rate* (nilai kawin silang)

- Memasukkan *mutation rate* (nilai mutasi)
- Memasukkan jumlah generasi

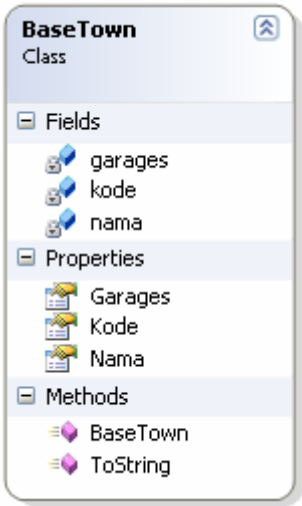
3. Populasi awal
   Populasi awal yang dipakai dapat bermacam-macam dan besarnya juga dapat diatur sesuai kebutuhan. Jumlah kromosom tiap populasi mengikuti nilai populasi awal.
4. Proses seleksi
   Setelah terjadi populasi awal, maka selanjutnya hasil populasi awal itu akan diseleksi. Metode seleksi yang digunakana di dalam algoritma genetika ini adalah metode seleksi *roulette-wheel*.
5. Proses reproduksi
   Proses reproduksi merupakan suatu proses yang terdiri dari proses perkawinan silang dan proses mutasi. Proses reproduksi bertujuan untuk menghasilkan *child* dari dua *parent,* yang diharapkan memiliki nilai *fitness* yang lebih baik.
6. Proses mutasi
   Proses mutasi pada tugas akhir ini nilainya berkisar antara nol sampai satu. Proses mutasi adalah proses mengubah nilai *gen* pada kromosom *child* dengan tujuan memperbaiki nilai *fitness.*
7. Proses kawin silang
   Proses kawin silang pada tugas akhir ini nilainya berkisar antara nol sampai satu. Proses perkawinan silang disini adalah melakukan *crossover* antara dua *parent* untuk menghasilkan *child.*
8. Kriteria penghentian regenerasi
   Kriteria penghentian regenerasi ini dilakukan jika tidak terdapat nilai optimal yang baru setelah terjadi sebanyak beberapa generasi. Nilai kriteria ini dapat dibuat besar dan dapat dibuat kecil sesuai keperluan. Besar dan kecilnya nilai tersebut sangatlah berpengaruh terhadap hasil dan juga lamanya proses regenerasi. Semakin kecil nilainya maka hasilnya belum tentu hasil yang paling optimal namun proses tersebut terjadi dengan singkat. Sebaliknya jika semakin besar nilanya maka hasil yang didapat diharapkan sudah yang paling maksimal namun proses tersebut memakan waktu yang lebih lama.

3.5 Rancangan *Class* Diagram

Kelas-kelas yang diperlukan dalam optimasi rute pendistribusian komoditi ini adalah sebagai berikut :

1. Kelas *Base Town*

Kelas ini merupakan kelas yang menggambarkan obyek *base town*. Diagram kelas *base town* dapat dilihat pada Gambar 3.3.



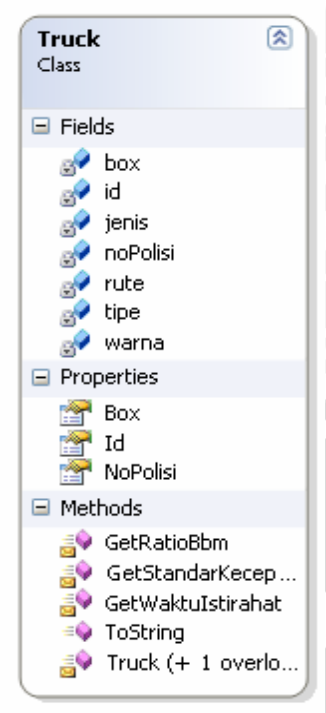
Gambar 3.3 Diagram kelas *base town*

2. Kelas *Truck*
   Kelas ini merupakan kelas yang menggambarkan obyek *truck*. Diagram kelas *truck* dapat dilihat pada Gambar 3.4.

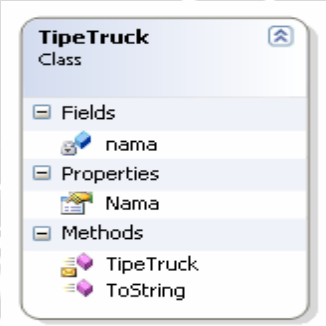
3. Kelas Tipe *Truck*
   Kelas ini merupakan kelas yang menggambarkan obyek tipe *truck*. Diagram kelas tipe *truck* dapat dilihat pada Gambar 3.5.

4. Kelas Warna *Truck*
   Kelas ini merupakan kelas yang menggambarkan obyek warna *truck*. Diagram kelas warna *truck* dapat dilihat pada Gambar 3.6.

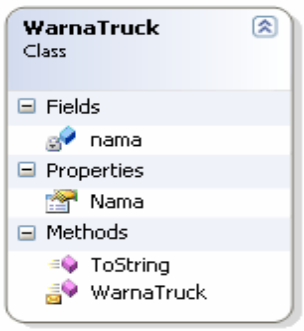
5. Kelas *Garage*
   Kelas ini merupakan kelas yang menggambarkan obyek *garage*. Diagram kelas *garage* dapat dilihat pada Gambar 3.7.
6. Kelas *Spot*
   Kelas ini merupakan kelas yang menggambarkan obyek *spot*. Diagram kelas *spot* dapat dilihat pada Gambar 3.8.



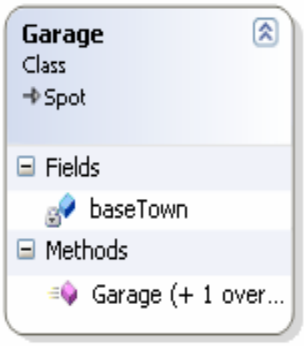
Gambar 3.4 Diagram kelas *truck*



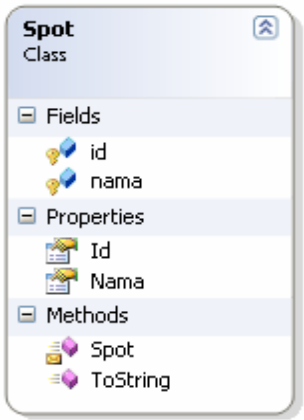
Gambar 3.5 Diagram kelas tipe *truck*



Gambar 3.6 Diagram kelas warna *truck*



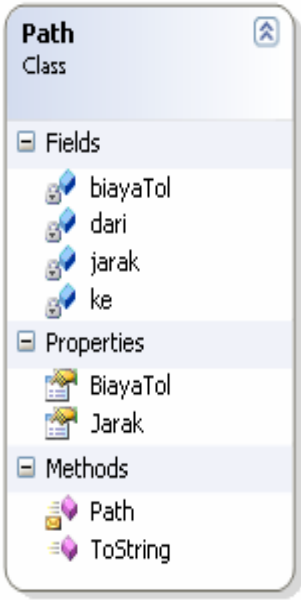
Gambar 3.7 Diagram kelas *garage*



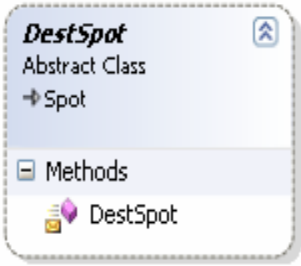
Gambar 3.8 Diagram kelas *spot*

7. Kelas *Path*
   Kelas ini merupakan kelas yang menggambarkan obyek jarak antar *spot* yang ada. Diagram kelas *path* dapat dilihat pada Gambar 3.9.

8. Kelas *DestSpot*
   Kelas ini merupakan kelas yang menggambarkan obyek *spot* tujuan kendaraan. Diagram kelas *dest spot* dapat dilihat pada Gambar 3.10.



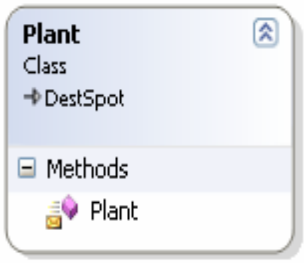
Gambar 3.9 Diagram kelas *path*



Gambar 3.10 Diagram kelas *destspot*

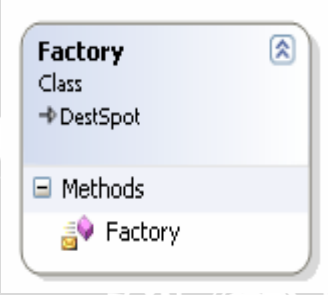
9. Kelas *Plant*

Kelas ini merupakan kelas yang menggambarkan obyek *plant.* Diagram kelas *plant* dapat dilihat pada Gambar 3.11.



Gambar 3.11 Diagram kelas *plant*

10. Kelas *Factory*
    Kelas ini merupakan kelas yang menggambarkan obyek *factory*. Diagram kelas *factory* dapat dilihat pada Gambar 3.12.



Gambar 3.12 Diagram kelas *factory*

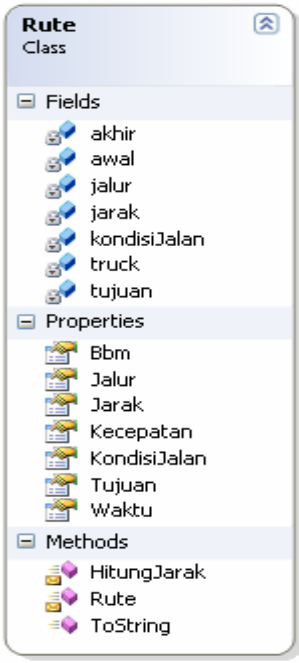
11. Kelas *Warehouse*
    Kelas ini merupakan kelas yang menggambarkan obyek *warehouse.* Diagram kelas *warehouse* dapat dilihat pada Gambar 3.13.

Gambar 3.13 Diagram kelas *warehouse*

12. Kelas Rute

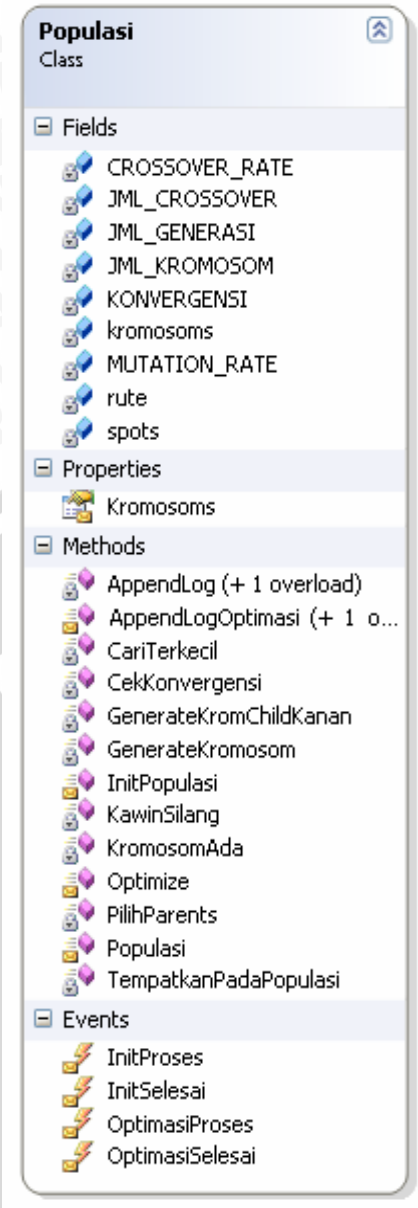
Kelas ini merupakan kelas yang menggambarkan rute yang dilalui oleh kendaraan dari garasi awal sampai ke garasi akhir dengan beberapa tujuan yang harus dilewati. Di kelas ini dilakukan proses penghitungan jarak antara garasi awal sampai garasi akhir, perhitungan waktu tempuh kendaraan dan biaya pemakaian bbm. Diagram kelas rute dapat dilihat pada Gambar 3.14.

13. Kelas Populasi
    Kelas ini merupakan kelas yang melakukan proses genetika. Di kelas ini dilakukan proses inisialisasi populasi awal, seleksi *parents*, *crossover*, dan proses pengecekan konvergensi. Diagram kelas populasi dapat dilihat pada gambar 3.15.

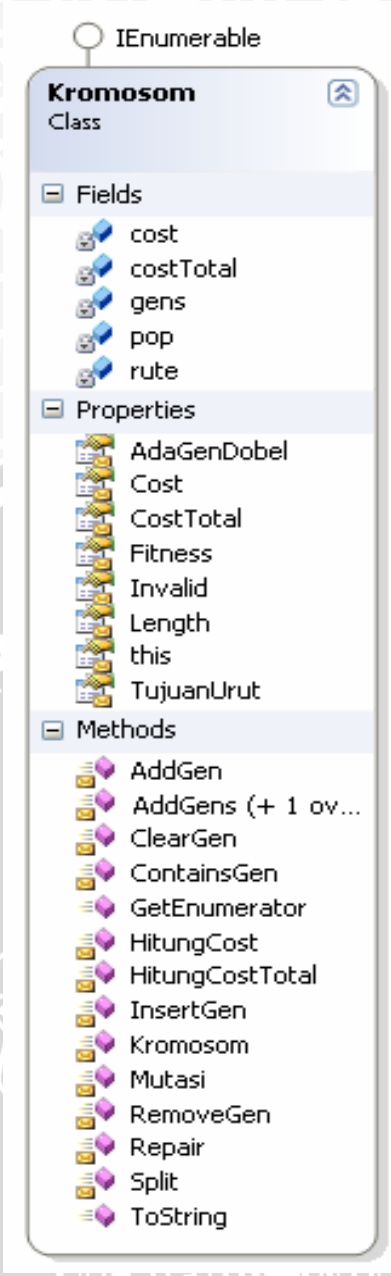


Gambar 3.14 Diagram kelas rute

14. Kelas Kromosom
    Kelas ini merupakan kelas yang menggambarkan obyek kromosom. Di kelas ini dilakukan proses genetika terhadap 1 kromosom, yaitu proses perhitungan *cost,* mutasi, dan repair. Diagram kelas kromosom dapat dilihat pada Gambar 3.16.



Gambar 3.15 Diagram kelas populasi



Gambar 3.16 Diagram kelas kromosom

3.6 Inisialisasi Kromosom

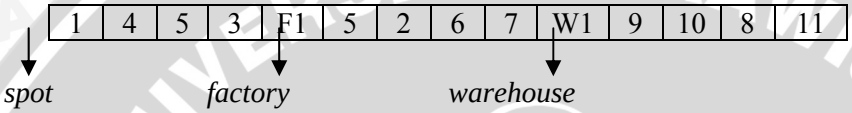
Pencarian rute terpendek pada algoritma genetika dilakukan sekaligus atas sejumlah solusi yang mungkin terjadi yang dikenal sebagai kromosom. Individu-individu yang terdapat dalam suatu populasi disebut kromosom.

Inisialisasi kromosom bergantung pada masalah yang akan diselesaikan. Pengkodean yang digunakan untuk menentukan rute ini yaitu pengkodean permutasi. Masing-masing kromosom berisi *gen-gen* yang merepresentasikan *spot-spot* yang harus dilalui untuk mencapai tujuan.

Dalam distribusi komoditi ini, dalam satu kali perjalanan minimal kendaraan bergerak ke dua titik tujuan. Rute perjalanan kendaraan adalah bergerak dari *garage* menuju titik tujuan pertama (tempat mengambil barang), kemudian menuju ke titik tujuan selanjutnya (tempat bongkar muatan) dan terakhir menuju *garage*.

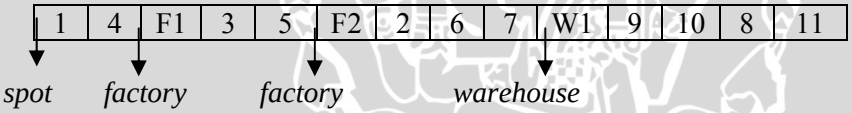
garage→factory→warehouse→garage

Pada urutan perjalanan diatas dapat direpresentasikan dalam bentuk kromosom sehingga menjadi seperti pada Gambar 3.17.



Gambar 3.17 Inisialisai kromosom

Pada Gambar 3.3 dapat dilihat adanya representasi kromosom dengan 2 titik tujuan yaitu *factory1* dan *warehouse1*. Sedangkan untuk perjalanan yang mempunyai titik tujuan lebih dari dua dapat dilihat pada Gambar 3.18.

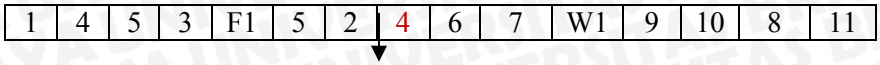


Gambar 3.18 Inisialisai kromosom dengan titik tujuan lebih dari dua

Jumlah *gen* yang ada dalam satu kromosom sama dengan jumlah *spot* yang dilalui dan dapat diketahui urutan perjalanan dan titik mulai (*garage1*) serta titik akhir (*garage2*) dari perjalanan tersebut.

Dalam permasalahan ini, memungkinkan adanya *spot* yang harus dilewati lebih dari satu kali. Namun, pada inisialisasi kromosom hal ini

tidak ditampilkan. Pada inisialisasi kromosom hanya ditampilkan *spot-spot* yang dilewati sebanyak satu kali. Tetapi dalam perhitungan nilai *cost*, *spot* yang dilewati lebih dari satu kali akan dihitung. Pada Gambar 3.19 dapat dilihat *spot-spot* yang sebenarnya dilewati dalam satu perjalanan. Tetapi inisialisasi kromosom yang digunakan tetap seperti pada Gambar 3.17.



Spot yang dilewati lebih dari satu kali

Gambar 3.19 *Spot-spot* yang dilalui dalam perjalanan

Dalam permasalahan ini, dalam satu populasi terdapat banyak individu. Dalam satu individu hanya terdapat satu kromosom, maka dapat dikatakan individu adalah kromosom. Sehingga dalam populasi terdapat banyak kromosom. Dengan bentuk kromosom seperti Gambar 3.16, maka dapat direpresentasikan dalam variabel `'kromosoms'`(banyak kromosom), yang bertipe data `generic List`, yaitu tipe pengembangan tipe data `array` dan `array List`. Tipe data `generic List` bisa menyimpan koleksi data dengan tipe data sama, misal tipe data `'kromosom'`, dan dapat menambah, mengurangi, mencari anggota, atau mengurutkan. Sehingga `'kromosoms'` adalah `generic List` dengan tipe`'kromosom'.` Kapasitas `generic List` bisa dikatakan tidak terbatas, tergantung dari kemampuan komputer. Contoh penggunaannya adalah sebagai berikut:

```
List<Kromosom> kromosoms = new List<Kromosom>();
kromosoms.Add(new Kromosom());
```

Sedangkan dalam sebuah kromosom terdiri dari banyak *gen*. *Gen* sendiri dalam kasus ini bertipe data `spot` dan turunnya `DestSpot, factory, warehouse, plant.` *Gen-gen* dalam kromosom disimpan dalam sebuah variabel `gens`, yaitu `generic List` dengan tipe data `spot`. Sehingga `factory, warehouse, plant` bisa masuk sebagai anggota `gens`. Contoh penggunaannya adalah sebagai berikut:

```
List<Spot> gens = new List<Spot>();
gens.Add(new Spot());
gens.Add(new Factory());
```

3.7 Fungsi Evaluasi

Setelah individu-individu dalam populasi telah terbentuk, maka langkah selanjutnya adalah menghitung nilai *fitness* setiap individu. Fungsi *fitness* sendiri bertujuan untuk mengetahui baik tidaknya solusi yang ada pada suatu individu, setiap individu pada populasi harus memiliki nilai pembandingnya. Selanjutnya dari nilai *fitness* didapatkan solusi terbaik dengan cara pengurutan nilai *fitness* dari individu-individu.

Setelah kromosom terbentuk, maka selanjutnya dilakukan proses perhitungan *fitness*. Aturan perhitungan fungsi *fitness* yang digunakan dapat dilihat pada Persamaan 3.1.

$$fitness = \frac{1}{(cost+1)} \quad (3.1)$$

Dimana *cost* adalah total jarak yang ditempuh dari setiap *spot* yang dilewati dari titik awal sampai titik akhir.

Kromosom dengan nilai *fitness* terbesar akan terpilih sebagai kromosom terbaik. Metode seleksi yang digunakan adalah seleksi roda *roulette*, dimana nilai *fitness* terbesar mempunyai kemungkinan lebih besar untuk menjadi induk pada generasi selanjutnya. Sedangkan untuk nilai *fitness* terburuk akan digantikan oleh individu baru dengan nilai *fitness* yang lebih besar.

3.8 Seleksi

Seleksi mempunyai peranan penting dalam algoritma genetika, karena pada proses ini dipilih induk yang akan digunakan untuk menghasilkan individu baru. Pada kasus ini, metode seleksi yang digunakan adalah *roulette wheel,* semakin besar nilai *fitness* maka semakin besar kemungkinan untuk terpilih sebagai induk.

Setelah inisialisasi awal, dilakukan proses pencarian *fitness* dengan nilai terkecil, hanya individu yang memiliki nilai *fitness* terbaik yang dapat bertahan dan lolos seleksi untuk proses lebih lanjut. Setelah dilakukan proses rekombinasi dan mutasi, sistem akan kembali melakukan pencarian nilai *fitness* dan individu yang mempunyai nilai *fitness* terbaik adalah individu yang lolos seleksi, sampai tercapai nilai yang terbaik atau telah mencapai generasi tertentu.

3.9 Perkawinan Silang

Apabila proses seleksi telah dilaksanakan dan sudah terpilih induk baru, maka operator berikutnya adalah perkawinan silang. Perkawinan silang adalah cara mengkombinasikan *gen-gen* induk untuk menghasilkan keturunan baru. Perkawinan silang yang digunakan adalah *one cut point crossover* untuk tiap satu pasang kromosom *parent*. Pada perkawinan silang ini dilakukan dengan cara menukar nilai *gen* pada posisi *gen* yang sama dari kedua induk.

Dalam perkawinan silang, ada kemungkinan panjang kromosom *parent* tidak sama, maka untuk menentukan panjang kromosom *child* akan dibangkitkan angka random 1 sampai 3. Dengan arti sebagai berikut:

- Apabila muncul angka 1 maka panjang kromosom *child* akan mengikuti panjang kromosom *parent* yang terpendek.
- Apabila muncul angka 2 maka panjang kromosom *child* akan mengikuti panjang kromosom *parent* yang terpanjang.
- Apabila muncul angka 3 maka panjang kromosom *child* akan adalah rata-rata dari panjang kromosom kedua *parent.*

Hasil kromosom perkawinan silang adalah *gen* sebelah kiri *cut point* adalah menyalin langsung dari *parent.* Sedangkan sebelah kanan *cut point*, dipilih dengan cara membangkitkan angka biner random sebanyak *gen child* yang harus terisi. Angka biner disini diartikan dengan angka 1 berarti terpilih ke dalam kromosom *child*, sedangkan angka 0 berarti tidak terpilih ke dalam kromosom *child*.

Langkah-langkah pada perkawinan silang ini adalah sebagai berikut:

Induk 1

1	4	F1	3	5	F2	2	6	7	W1	9	10	8	11

Induk 2

2	6	F2	1	4	7	9	F1	8	5	W1	3	10	12

Langkah 1 :

Menyalin *gen* dari induk 1 ke induk anak.

1	4	F1	3	5	F2	2	..	..	..	..	..	..	..

Langkah 2 :

Kemudian menyalin *gen* dari induk 2 ke anak.

F1	8	5	W1	3	10	12

Langkah 3 :
Menggabungkan *gen-gen* dari induk-induk tersebut, untuk melengkapi kromosom anak.

1	4	F1	3	5	F2	2	F1	8	5	W1	3	10	12

Langkah 4 :
Jika setelah proses perkawinan silang menghasilkan kromosom ilegal, maka akan dilakukan *repair, gen* kromosom yang *invalid* diganti dengan angka yang belum muncul, tetapi urutan angka disesuaikan dengan induknya.

1	4	F1	3	5	F2	2	6	8	9	W1	7	10	12

Setelah proses perkawinan silang maka dilakukan perhitungan *cost* untuk memperoleh nilai *fitness.*

Hasil dari perkawinan silang memungkinkan munculnya *child* yang *invalid*, sehingga harus dilakukan *repair*. Dikatakan *invalid* dengan ketentuan sebagai berikut:

- *Gen* yang muncul lebih dari satu kali dalam satu kromosom.
- Urutan tujuan tidak sesuai dengan urutan tujuan awal.
- *Spot* tujuan tidak lengkap (tidak seperti yang dimasukkan oleh *user* pertama kali).

Pada Gambar 3.20 dapat dilihat contoh kromosom yang *invalid* dan dilakukan *repair*.

Pada perkawinan silang ada satu parameter yang sangat penting, yaitu probabilitas *crossover* (pc). Probabilitas *crossover* menunjukkan rasio dari *child* yang dihasilkan dalam setiap generasi dengan ukuran populasi. Misalkan ukuran populasi (popsize=100), sedangkan probabilitas *crossover* (pc=0,25), berarti bahwa ada 25 kromosom dari 100 kromosom yang ada pada populasi tersebut akan mengalami perkawinan silang. Kromosom-kromosom yang akan mengalami perkawinan silang dipilih secara acak.

Sebelum *repair*:

					↓				↓				
1	4	F1	3	5	W1	2	6	7	F2	9	10	8	11

Sesudah *repair*:

1	4	F1	3	5	F2	2	6	7	W1	9	10	8	11

Gambar 3.20 Ilustrasi *repair* pada kromosom

Algoritma untuk perkawinan silang dapat dilihat pada Gambar 3.21.

```
// pilih dua parent
get parent1, parent2

// pilih jenis kawin
jenisKawin <- random(3)

if jenisKawin = 1 then
// panjang kromosom child adalah panjang kromosom parent
   terpendek
      if pjgKromosomParent1 > pjgKromosomParent2
            pjgKromosomChild <- pjgKromosomParent2
      else
            pjgKromosomChild <- pjgKromosomParent1

else if jenisKawin = 2 then
// panjang kromosom child adalah panjang kromosom parent
   terpanjang
      if pjgKromosomParent1 < pjgKromosomParent2
            pjgKromosomChild <- pjgKromosomParent2
      else
            pjgKromosomChild <- pjgKromosomParent1

else if jenisKawin = 3 then
// panjang kromosom child adalah rata2 panjang kromosom parent
   pjgKromosomChild = (pjgKromosomParent1 + pjgKromosomParent2)
   / 2

end if

// cari cut point nya yg tidak terlalu ke tepi
cutPointMin = 0.1 * pjgKromosomChild
cutPointMax = 0.9 * pjgKromosomChild
cutPoint = random(cutPointMin, cutPointMax)

// pecah kromosom parent
kromosomParent1Kiri, kromosomParent1Kanan <- split(kromosomParent1,
cutPoint)
kromosomParent2Kiri, kromosomParent2Kanan <- split(kromosomParent2,
cutPoint)

// kromosom kiri parent menjadi kromosom kiri child
insertGen(child1, kromosomParent1Kiri)
insertGen(child2, kromosomParent2Kiri)

pjgKromosomChildKanan <- pjgKromosomChild – cutPoint

// bikin kromosom child bagian kanan
kromosomChild1Kanan <-
generateKromosomKananChild(pjgKromosomChildKanan,
kromosomParent2Kanan, kromosomParent1Kanan)
```

```
kromosomChild2Kanan <-
generateKromosomKananChild(pjgKromosomChildKanan,
kromosomParent1Kanan, kromosomParent2Kanan)


procedure generateKromosomKananChild(pjgKromosomChildKanan,
kromosomKananParent, kromosomKananParentLain)

// implementasi dari di-mask 0 atau 1, seperti di bawah ini,
   intinya sama

// jika panjang kromosom child kanan lebih kecil dari parent
   kanan
     if pjgKromosomChild < length of kromosomKananParent

          // ambil posisi2 sejml pjg krom child
          for I = 0 to pjgKromosomChild

               // cari posisi gen2, dan tidak boleh dobel
               repeat
                    posGen = random(pjgKromosomChild)
               until (posGen not in daftarPosisiGen)

               // masukkan ke daftar posisi gen
               insert(daftarPosisiGen, posGen)
          end for;

          // urutkan daftar posisi gen
          sort(daftarPosisiGen)

          // masukkan semua kromosom parent kanan ke kromosm
             kanan anak, sesuai daftar posisi
          foreach (posGen in daftarPosisiGen)
               // masukkan ke kromosom anak
               insert(kromosomChildKanan,
         kromosomParentKanan[posGen]);
          end foreach;

     // jika pjg krom child lebih besar dr parent
     else if pjgKromosomChild > length of kromosomKananParent

          // ambil posisi2 sejml pjg kromosom kanan parent
          for I = 0 to length of kromosomKananParent

               // cari posisi gen2, dan tidak boleh dobel
               repeat
               posGen = random(length of kromosomKananParent)
               until (posGen not in daftarPosisiGen)

               // masukkan ke daftar posisi gen
               insert(daftarPosisiGen, posGen)
          end for

          // urutkan daftar posisi gen
```

```
            sort(daftarPosisiGen)

            // masukkan semua kromosom parent kanan ke kromosm
               kanan anak, sesuai daftar posisi
            foreach (posGen in daftarPosisiGen)
                  // masukkan ke kromosom anak
      insert(kromosomChildKanan,kromosomParentKanan[posGen]);
            end foreach

            // kosongkan daftar posisi gen
            empty(daftarPosisiGen)

            // mengisi kromosom child yg masih kosong (krn
               lebih pjg dari parent)
            sisa <- length of kromosomChildKanan – length o
            kromosomParentKanan

            // isi yg kosong dg parent kanan yg lain
            for I = 1 to sisa

                  // cari posisi gen2, dan tidak boleh dobel
                  repeat
                        posGen = random(length of
                          kromosomKananParentLain)
                  until (posGen not in daftarPosisiGen)

                  // masukkan ke daftar posisi gen
                  insert(daftarPosisiGen, posGen)

            end for
            // masukkan semua kromosom parent kanan yg lain ke
               kromosm kanan anak, sesuai daftar posisi
            foreach (posGen in daftarPosisiGen)
                  // masukkan ke kromosom anak

   insert(kromosomChildKanan, kromosomParentKananLain[posGen]);
            end foreach
      end if;
end
```

Gambar 3.21 *Pseudocode* perkawinan silang

3.10 Mutasi

Mutasi dilakukan untuk mencegah terjadinya konvergensi dini. Pada mutasi ada satu parameter yang sangat penting yaitu probabilitas *mutation* (pm). Probabilitas mutasi menunjukkan persentase kromosom pada populasi yang akan mengalami mutasi.

Mutasi dilakuan dengan cara *swap* atau penukaran. Mutasi dilakukan dengan cara menukar nilai dari gen dari 2 titik. Pemilihan *gen* yang akan ditukar dilakukan secara random. Ilustrasi mutasi secara *swap* dapat dilihat pada Gambar 3.22.

Sebelum mutasi:

1	4	F1	3	5	F2	2	6	7	W1	9	10	8	11

Sesudah mutasi:

1	4	F2	3	5	F1	2	6	7	W1	9	10	8	11

Gambar 3.22 Ilustrasi mutasi cara *swap*

Algoritma untuk proses mutasi seperti tampak pada Gambar 3.23.

```
// inputkan mutation rate
input mutationRate

// hitung jumlah gen yang akan dimutasi
jmlGenMutant <- mutationRate * jmlGenKromosom

// cari posisi2 gen yg akan dimutasi
for i = 1 to jmlGenMutant

      repeat
            // cari posisi yg akan dimutasi secara random
            pos = random(panjangKromosom)
      until (pos not in daftarPosMut) //tidak ada di daftar
            posisi mutasi
      //tambahkan ke daftar posisi mutasi
      insert to daftarPosMut
end for

// mutasikan gen2 berdasar daftar posisi mutasi
for i = 1 to length of daftarPosMut

      // tentukan posisi dan posisi pasangannya
      pos1 <- daftarPosMut[i]
      pos2 <- pos1

      // cari posisi pasangan yg berbeda dg posisi 1
      repeat

            // cari posisi pasangan dg acak
            pos2 = random(panjangKromosom);

      until (pos1 <> pos2)
      // tukarkan gen
      swap(kromosom[pos1], kromosom[pos2])
end
```

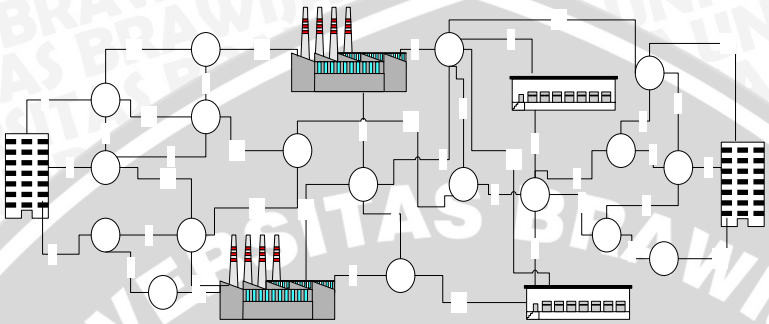
Gambar 3.23 *Pseudocode* mutasi

3.11 Contoh Perhitungan Manual

Misal terdapat sebuah *truck* dengan jenis *truck* tronton, yang melakukan perjalanan sebagai berikut:

Base town awal	Kendaraan	Titik muat barang	Titik bongkar muatan	*Base town* akhir
Garage 1	*Truck* Tronton2	*Factory* 1 *Factory 2*	*Warehouse* 1	*Garage* 2

Dan penggambaran peta pembobotan jarak untuk contoh perhitungan manual dapat dilihat pada Gambar 3.24.



Gambar 3.24 Penggambaran peta pembobotan jarak

3.11.1 Inisialisasi Kromosom

Kromosom 1 :

1	4	F1	10	9	F2	11	W1	13	14	17

Kromosom 2 :

2	1	4	F1	9	F2	11	W1	13	15	18

Kromosom 3 :

2	5	4	F1	10	9	F2	11	W1	13	15	18

Kromosom 4 :

3	6	F2	9	F1	10	W1	13	14	18

Pada contoh inisialisai kromosom, kromosom ini diambil secara acak dan dengan panjang yang sudah ditentukan.

9 1

3

2 2

Garage 3

3.11.2 Evaluasi

Setelah melakukan proses inisialisasi kromosom, selanjutnya adalah melakukan pengecekan, untuk mengetahui apakah kromosom-kromosom tersebut telah memenuhi persyaratan atau tidak. Persyaratan yang dimaksud adalah sampainya *truck* ke titik-titik tujuan.
Proses evaluasi juga menghitung besarnya nilai *fitness* untuk masing-masing kromosom atau individu. Perhitungan nilai *fitness* adalah sebagai berikut:

Kromosom 1 :

1	4	F1	10	9	F2	11	W1	13	14	17

$$cost = 9 + 15 + 14 + 3 + 8 + 10 + 4 + 18 + 6 + 9 + 3 + 12 = 111$$

$$fitness = \frac{1}{cost + 1} = \frac{1}{111 + 1} = 0{,}008928$$

Kromosom 2 :

2	1	4	F1	9	F2	11	W1	13	15	18

$$cost = 2 + 3 + 15 + 14 + 9 + 10 + 4 + 18 + 6 + 7 + 5 + 6 = 99$$

$$fitness = \frac{1}{cost + 1} = \frac{1}{99 + 1} = 0{,}010000$$

Kromosom 3 :

2	5	4	F1	10	9	F2	11	W1	13	15	18

$$cost = 2 + 8 + 4 + 14 + 3 + 8 + 10 + 4 + 18 + 6 + 7 + 5 + 6 = 95$$

$$fitness = \frac{1}{cost + 1} = \frac{1}{95 + 1} = 0{,}010417$$

Kromosom 4 :

3	6	F2	9	F1	10	W1	13	14	18

$$cost = 5 + 7 + 9 + 10 + 9 + 3 + 20 + 6 + 9 + 9 + 6 = 93$$

$$fitness = \frac{1}{cost + 1} = \frac{1}{93 + 1} = 0{,}010638$$

3.11.3 Seleksi

Seleksi yang digunakan adalah seleksi roda *roulette*. Pada seleksi ini, hanya individu yang memiliki *fitness* terbaik yang akan bertahan dan lolos

seleksi untuk diproses lebih lanjut, sedangkan nilai *fitness* terburuk akan digantikan oleh nilai *fitness* yang lebih baik. Hasil dari perhitungan nilai *fitness* dapat dilihat pada Tabel 3.1.

Tabel 3.1 Nilai *fitness* kromosom

Index	***Cost***	***Fitness***
Kromosom 1	111	0.008928
Kromosom 2	99	0.010000
Kromosom 3	95	0.010417
Kromosom 4	93	0.010638

Setelah mendapat nilai *fitness*, maka hasilnya diurutkan dari nilai *fitness* terbaik sampai terburuk. Hasil pengurutan nilai *fitness* dapat dilihat pada Tabel 3.2.

Tabel 3.2 Nilai *fitness* kromosom setelah terurut

Index	***Cost***	***Fitness***
Kromosom 4	93	0.010638
Kromosom 3	95	0.010417
Kromosom 2	99	0.010000
Kromosom 1	111	0.008928

3.11.4Perkawinan Silang

Apabila proses seleksi telah dilakukan dan sudah terpilih induk baru, maka operator berikutnya adalah perkawinan silang. Dalam contoh ini diambil dua kromosom untuk dijadikan *parent* dan kemudian di *crossover*, sehingga menghasilkan 2 *child*.

Perkawinan silang yang digunakan adalah *one-cut-point crossover* yaitu kromosom yang mempunyai 13 *gen* dibagi menjadi 2 bagian.

Parent 1 :

2	5	4	F1	10	9	F2	11	W1	13	15	18

Parent 2 :

3	6	F2	9	F1	10	W1	13	14	18

Setelah ditentukan garis potongnya maka langsung dilakukan penukaran gen antara *parent* 1 dan *parent* 2 untuk menghasilkan *child* 1 dan *child* 2 sesuai dengan ketentuan yang ada. Dalam contoh ini panjang kromosom *child* adalah rata-rata dari panjang kromosom *parent*.

Child 1

2	5	4	F1	10	10	W1	13	F2	14	18

Chld 2

3	6	F2	9	F1	F2	11	W1	13	15	18

Setelah proses penukaran *gen* tersebut, dilakukan pengecekan apakah individu baru yang terbentuk ilegal. Kromosom dikatakan ilegal apabila terdapat 1 atau lebih *gen* yang sama pada satu kromosom dan apabila terdapat kromosom yang ilegal maka dilakukan perbaikkan

Child 1

2	5	4	F1	10	9	F2	13	W1	14	18

Chld 2

3	6	F2	9	F1	10	11	W1	13	15	18

Setelah dilakukan perbaikan pada kromosom maka dilakukan lagi proses penghitungan nilai *fitness* untuk masing-masing kromosom *child.*

Child 1

2	5	4	F1	10	9	F2	13	W1	14	18

$$cost = 2+8+4+14+3+8+10+\sim+6+\sim9+6 = \sim$$

$$fitness = \frac{1}{cost+1} = \frac{1}{\sim+1} = 0$$

Chld 2

3	6	F2	9	F1	10	11	W1	13	15	18

$$cost = 5+7+9+10+9+3+8+7+18+6+7+5+6 = 100$$

$$fitness = \frac{1}{cost+1} = \frac{1}{100+1} = 0{,}009901$$

Setelah dilakukan perhitungan pada *child* 1 dan *child* 2 maka dilakukan penghitungan *fitness* masing-masing kromosom sehingga didapatkan data seperti pada Tabel 3.3.

Tabel 3.3 Nilai *fitness* kromosom *parent* dan *child*

Index	***Cost***	***Fitness***
Kromosom 1	95	0.010417
Kromosom 2	93	0.010638
Child 1	~	0
Child 2	100	0,009901

3.11.5 Mutasi

Misalkan probabilitas mutasi yang digunakan adalah 0,01. Diasumsikan *child1* mengalami mutasi maka proses mutasi terjadi seperti ilustrasi di bawah ini.

Misal *gen* yang akan ditukar pada kromosom 1 adalah *gen* pada posisi 8 dan 9.

Mutasi 1

2	5	4	F1	10	9	F2	13	W1	14	18

↓

2	5	4	F1	10	9	F2	W1	13	14	18

$$cost = 2 + 8 + 4 + 14 + 3 + 8 + 10 + 11 + 18 + 6 + 9 + 9 + 6 = 108$$

$$fitness = \frac{1}{cost + 1} = \frac{1}{108 + 1} = 0{,}009174$$

Setelah dilakukan mutasi pada *child1* didapat hasil nilai *cost* dan *fitness* yang lebih baik. Sebelum dilakukan mutasi nilai *fitness* dari *child1* adalah 0,0093. Setelah dilakukan mutasi, nilai *fitness* dari *child1* menjadi 0,01.

3.11.6 Pembentukan Generasi Baru

Setelah proses seleksi, perkawinan silang, dan mutasi terhadap kromosom dilakukan, proses pembentukan generasi baru dilakukan dengan cara membandingkan *fitness* kromosom anak dengan *fitness* kromosom

induk. Kromosom anak akan menggantikan kromosom induk jika kromosom anak mempunyai nilai *fitness* yang lebih baik. Hasil nilai *fitness* kromosom *parents, child* dan mutasi dapat dilihat pada Tabel 3.4.

Tabel 3.4 Hasil nilai *fitness* kromosom *parent, child* dan mutasi

Index	***Cost***	***Fitness***
Kromosom 1	95	0.010417
Kromosom 2	93	0.010638
Child 1	~	0
Child 2	100	0,009901
Mutasi	108	0.009174

Selanjutnya 2 kromosom dengan nilai *fitness* terbaik diambil untuk dijadikan *parent* pada generasi selanjutnya. Dari kasus ini kromosom yang diambil sebagai *parent* adalah kromosom 1 dan kromosom 2 karena nilai *fitness child* yang dihasilkan tidak lebih baik dari nilai *fitness parent*.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini akan dijelaskan seluruh proses yang sudah dirancang pada bab sebelumnya, tampilan antarmuka dan bagian-bagian *source code* yang dibuat, serta analisa terhadap data yang dihasilkan sistem.

4.1 Lingkungan Implementasi

Implementasi merupakan proses transformasi representasi rancangan ke dalam bahasa pemrograman yang dapat dimengerti oleh komputer. Pada bab ini, lingkungan implementasi yang akan dijelaskan meliputi lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem optimasi pencarian rute pendistribusian komoditi berdasarkan jarak dan kondisi jalan untuk meminimalkan biaya bahan bakar dengan algoritma dengan studi kasus pada PT.XYZ ini adalah sebagai berikut :

1. *Processor* Intel® Core™2 Duo T7100 @ 1.80 GHz (2CPUs)

2. 512 MB RAM
3. 120 GB HDD
4. Monitor 12.1”
5. Keyboard
6. Mouse

Perangkat keras ini akan difungsikan sebagai tempat perangkat lunak untuk penelitian dijalankan.

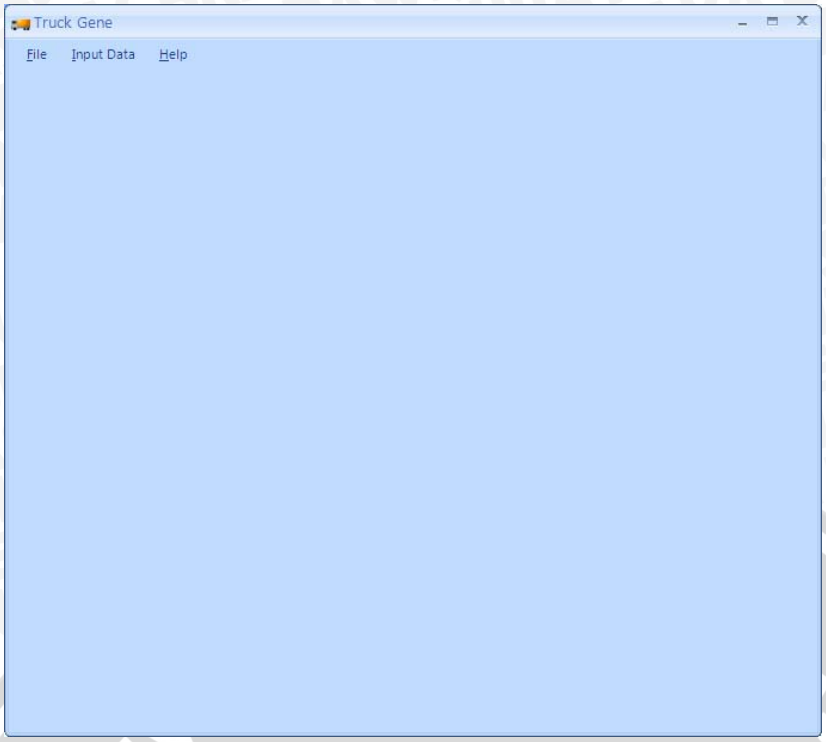
4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem penghitungan jumlah kendaraan ini adalah :

1. Sistem operasi *Microsoft Windows XP Professional Edition Service Pack 2* sebagai tempat aplikasi dijalankan.
2. *Microsoft Visual Studio 2005* sebagai *software development* dalam mengembangkan aplikasi optimasi pencarian rute pendistribusian komoditi.
3. Bahasa pemrograman menggunakan C# 2.0

4.2 Implementasi Program

Tampilan utama dari aplikasi optimasi pencarian rute pendistribusian komoditi dengan algoritma genetika dapat dilihat pada Gambar 4.1. Dalam tampilan aplikasi optimasi rute pendistribusian komoditi ini ada tiga menu utama yaitu *file*, *input* data dan *help*. Dalam menu file terdapat dua submenu yaitu optimasi dan *exit*. Dalam menu *input* data terdapat empat submenu, yaitu *truck, spot,* jarak *spot* dan rute. Sedangkan dalam menu *help* hanya ada satu submenu yaitu *about.*

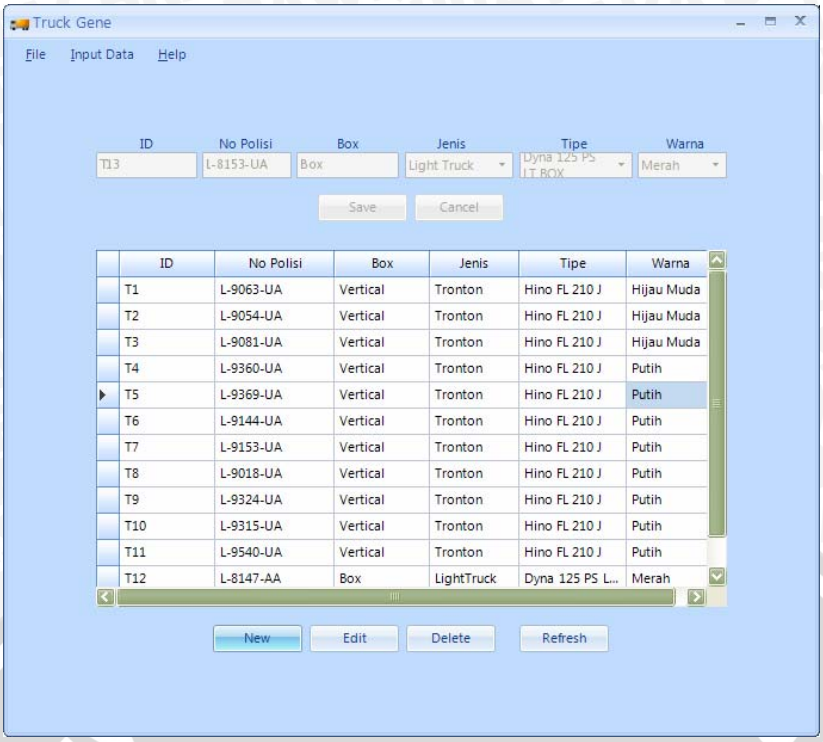


Gambar 4.1 Tampilan utama aplikasi

4.2.1 *Input* Data

Input data terdiri dari komponen utama optimasi rute pendistribusian komoditi, yang meliputi *truck, spot,* jarak spot, dan rute. Berikut penjelasan masing-masing komponen beserta tampilan:

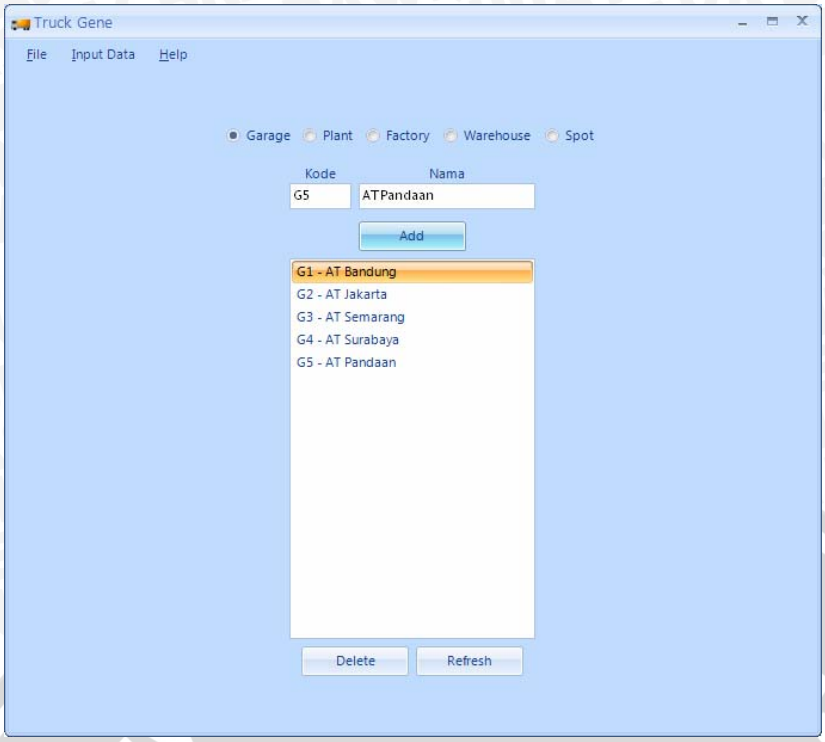
1. *Truck*
   Form truck digunakan untuk menyimpan data *truck*. Data yang disimpan adalah id *truck*, no polisi, jenis box, jenis *truck*, tipe *truck*, warna. Tampilan *form* data *truck* dapat dilihat pada Gambar 4.2.



Gambar 4.2 *Form* data *truck*

2. *Spot*

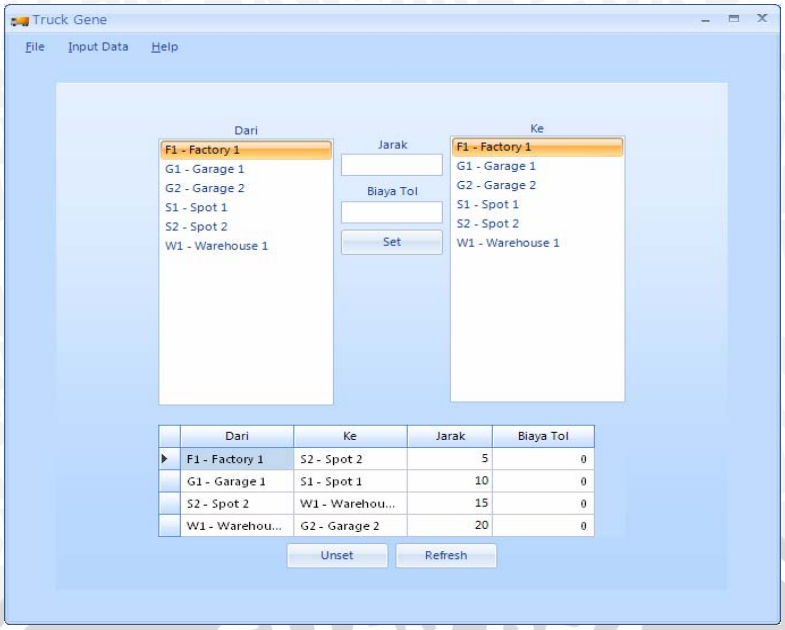
   Form spot digunakan untuk menyimpan data *spot*. Data yang disimpan adalah jenis *spot* (*garage*, *plant, factory, warehouse, spot*), id *spot*,dan nama *spot*. Tampilan *form* data *spot* dapat dilihat pada Gambar 4.3.



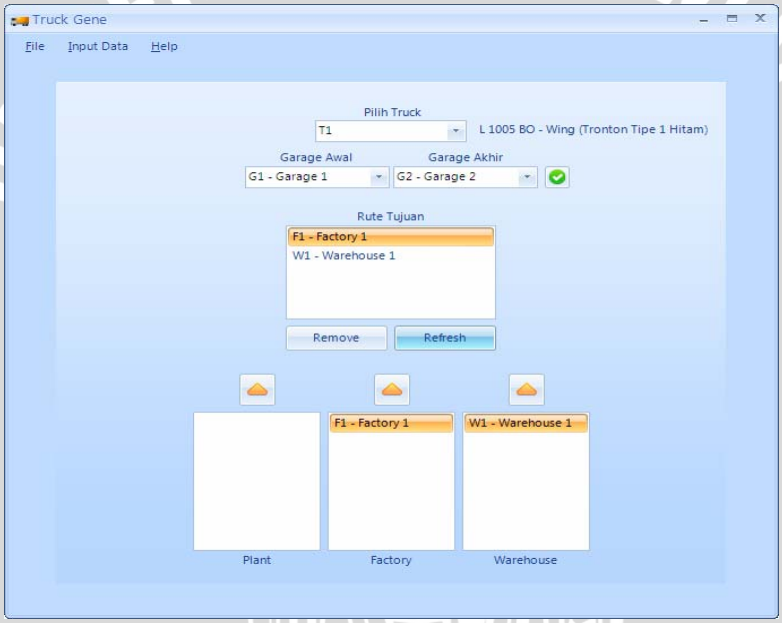
Gambar 4.3 *Form* data *spot*

3. Jarak *Spot*
   Form jarak *spot* digunakan untuk menyimpan data jarak antar *spot*. Data yang disimpan adalah *spot* awal, *spot* tujuan, jarak dan biaya tol antara *spot* awal dan *spot* tujuan. Tampilan *form* data jarak *spot* dapat dilihat pada Gambar 4.4.

4. Rute
   Form rute digunakan untuk menyimpan data rute. Data yang disimpan adalah id *truck* yang digunakan, *garage* awal, *garage* akhir, rute tujuan yang harus dicapai. Tampilan *form* data rute dapat dilihat pada Gambar 4.5.



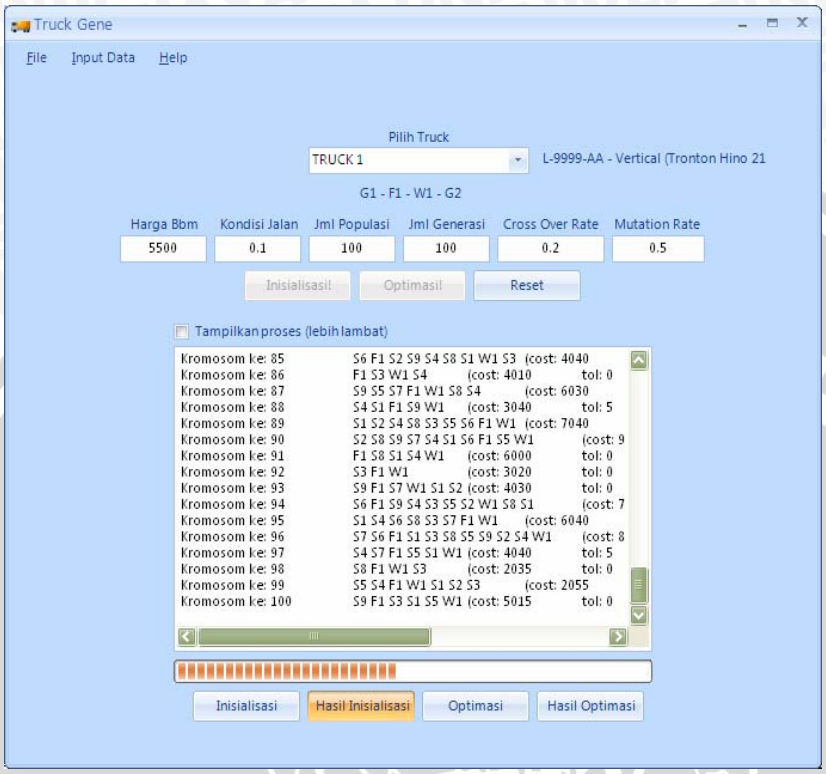
Gambar 4.4 *Form* data jarak *spot*



Gambar 4.5 *Form* data rute

4.2.2 Optimasi

Optimasi merupakan proses algoritma genetika pada aplikasi optimasi rute pendistribusian komoditi. Proses yang ada adalah inisialisasi kromosom, hasil inisialisasi, optimasi dan hasil optimasi. Proses ini dapat berjalan dengan memasukkan rute perjalanan *truck* yang dipilh, dan parameter genetika (jumlah populasi, jumlah generasi, *crossover rate*, *mutation rate*). Tampilan dari *form* optimasi dapat dilihat pada Gambar 4.6.



Gambar 4.6 *Form* Optimasi

4.3 Deskripsi Program

Berdasarkan analisa dan perancangan proses yang terdapat pada Bab III, maka pada sub bahasan ini akan dijelaskan implementasi proses-proses tersebut.

4.3.1 Inisialisasi Kromosom

Tahap awal dalam algoritma genetika adalah pembentukan individu yang dapat disebut dengan inisialisasi kromosom. Prosedur untuk inisialisasi kromosom diawali dengan membuat kromosom sejumlah individu. Prosedur untuk membuat kromosom sejumlah individu dapat dilihat pada *Sourcecode* 4.1.

```
AppendLog(InitProses, "Inisialisasi populasi. ({0}
kromosom)\r\n", jmlKromosom);

kromosoms = new List<Kromosom>(jmlKromosom);
```

***Sourcecode* 4.1** Prosedur membuat kromosom

Setelah membuat kromosom sejumlah individu dalam populasi. Di dalam membuat kromosom, akan dirandom kromosom baru sampai tidak ada kromosom yang berbeda. Inisialisasi akan berhenti setelah tidak menemukan kromosom baru yang berbeda dengan kromosom yang ada. Setiap kromosom yang *valid* akan dimasukan ke dalam populasi. Kromosom tersebut akan dihitung nilai *cost* dan biaya tol. Prosedur membuat kromosom dapat dilihat pada *Sourcecode* 4.2.

```
for (int kromKe = 0; kromKe < jmlKromosom; kromKe++)
{
  Kromosom krom = null;
  int jmlKandidat = 0;
  double persen = (double)(kromKe + 1) / jmlKromosom * 100;

  LagiProses(persen);
  do
  {
    if (jmlKandidat >= MAX_KANDIDAT)
    {
     LagiProses(100);

     kromosomsAwal = new List<Kromosom>();
     kromosomsAwal.AddRange(kromosoms);

     AppendLog(InitProses, "Tidak ditemukan kandidat
     lainnya. Pencarian kandidat dihentikan");
     AppendLog(InitSelesai, "Inisialisasi Selesai.");

      return;
      }
     else
```

```
        {
        krom = GenerateKromosom();

        AppendLog(InitProses, "Kandidat kromosom {0} ke
        {1}: {2}", kromKe + 1, jmlKandidat + 1,krom.GensInfo);

         jmlKandidat++;
         }
        }
        while (KromosomAda(krom));

        kromosoms.Add(krom);
        krom.HitungCost();
        krom.HitungBiayaTol();

        AppendLog(InitProses, "Kromosom ke: {0}\r\n{1}\r\n",
        kromKe + 1, krom);
       }

      kromosomsAwal = new List<Kromosom>();
      kromosomsAwal.AddRange(kromosoms);

      LagiProses(100);

      AppendLog(InitSelesai, "Inisialisasi Selesai.");
```

***Sourcecode* 4.2** Prosedur inisialisasi kromosom

Di dalam inisialiasi kromosom terdapat prosedur *generate* kromosom yang berfungsi untuk membuat kromosom baru. Proses *generate* kromosom dimulai dengan merandom gen dan panjang kromosom. Panjang kromosom dirandom dengan panjang maksimal sejumlah *spot* yang ada, dan minimal sejumlah satu. Prosedur membuat kromosom baru dan merandom panjang kromosom dapat dilihat pada *Sourcecode* 4.3.

```
    Kromosom krom = new Kromosom(this);

    Random r = new Random();

    int maxPjg = spots.Count;
    int minPjg = 1;
    int pjgKrom = r.Next(minPjg, maxPjg + 1);
```

***Sourcecode* 4.3** Prosedur random panjang kromosom

Setelah panjang kromosom didapatkan, kemudian merandom *gen*. *Gen* yang dirandom adalah *spot* yang bukan merupakan *spot* tujuan. *Gen* yang

diambil sejumlah panjang kromosom, dan berbeda dengan *gen* yang telah diambil sebelumnya. Prosedur merandom *gen* dapat dilihat pada *Sourcecode* 4.4.

```
List<int> listPos = new List<int>();

for (int i = 0; i < pjgKrom; i++)
{
  int pos;
  Spot gen;

  bool tujuan = false;

  do
  {
     pos = r.Next(0, maxPjg);

     gen = spots[pos];

     if (gen is DestSpot)
     {
        DestSpot dest = (DestSpot)gen;
        tujuan = rute.Tujuan.Contains(dest);
     }
     else
     {
        tujuan = false;
     }

  } while (krom.Contains(gen) || tujuan);

  krom.Add(gen);
 }

 int jmlTujuan = rute.Tujuan.Count;

 int posGen = -1;
```

***Sourcecode* 4.4** Prosedur merandom *gen*

Setelah *gen* didapatkan seceara random, maka *spot* tujuan akan disisipkan ke dalam kromosom. Sehingga membentuk suatu kromosom yang utuh. Prosedur menyisipkan *spot* tujuan dapat dilihat pada *Sourcecode* 4.5.

```
for (int tujKe = 0; tujKe < jmlTujuan; tujKe++)
{
```

```
        int posMin = posGen + 1;
        int posMax = pjgKrom + tujKe;

        posGen = r.Next(posMin, posMax + 1);

        krom.InsertGen(posGen, rute.Tujuan[tujKe]);
      }

      return krom;
```

***Sourcecode* 4.5** Prosedur menyisispkan *spot* tujuan

Di dalam inisialisasi kromosom dan *generate* kromosom terdapat prosedur `KromosomAda`. Prosedur ini berfungsi untuk melihat apakah *gen* dalam kromosom berbeda dan melihat apakah kromosom yang satu berbeda dengan kromosom yang lainnya dalam satu populasi. Prosedur `KromosomAda` dapat dilihat pada *Sourcecode* 4.6.

```
    bool ada = false;
    foreach (Kromosom cek in kromosoms)
    {
      if (cek.Length == krom.Length)
      {
         for (int pos = 0; pos < krom.Length; pos++)
         {
         if (cek[pos] != krom[pos])
           {
             ada = false;
           break;                          }
        else
        {
           ada = true;
        }
      }

      if (ada)
      {
         break;
      }
    }

    return ada;
  }
```

***Sourcecode* 4.6** Prosedur `kromosom_ada`

4.3.2 Seleksi

Seleksi dimulai dengan menghitung nilai semua *fitness* kromosom. Kemudian memilih kromosom *parent* pada populasi secara random

berdasarkan persentase dari *fitness* yang dimiliki. Setelah mendapatkan *parent1* maka mencari *parent2* secara random sesuai dengan persentase *fitness*. Cari kromosom yang berbeda dengan kromosom yang telah terpilih sebagai *parent1*. Prosedur pilih *parent* dapat dilihat pada *Sourcecode* 4.7.

```
Kromosom[] parents = new Kromosom[2];

double totFitness = 0;

foreach (Kromosom krom in kromosoms)
{
  totFitness += krom.Fitness;
}

Random r = new Random();

int pros1 = r.Next(0, 100);
int pros2 = r.Next(0, 100);

AppendLog(OptimasiProses, "Total fitness: {0}\tRandom %
parents: {1} & {2}", totFitness, pros1, pros2);

Dictionary<Kromosom, double> kromsProsent = new
Dictionary<Kromosom, double>(kromosoms.Count);

double totPros = 0;

foreach (Kromosom krom in kromosoms)
{
  totPros += (krom.Fitness / totFitness) * 100;
   kromsProsent.Add(krom, totPros);
 }

 foreach (KeyValuePair<Kromosom, double> kromPros in
 kromsProsent)
 {
   if (pros1 < kromPros.Value)
   {
     parents[0] = kromPros.Key;
     break;
   }
  }

  do
  {
    foreach (KeyValuePair<Kromosom, double> kromPros in
    kromsProsent)
    {
      if (pros2 < kromPros.Value && parents[0] !=
      kromPros.Key)
      {
```

```
                parents[1] = kromPros.Key;
                break;
              }
            }

            pros2 = r.Next(0, 100);
          }
          while (parents[1] == null);

          return parents;
```

***Sourcecode* 4.7** Prosedur pilih *parent*

4.3.3 Perkawinan Silang

Proses perkawinan silang dimulai dengan merandom jenis perkawinan yang akan dilakukan. Jenis perkawinan disini adalah panjang kromosom anak yang akan dihasilkan. Ada 3 jenis perkawinan yang ada, yaitu:

- Panjang kromosom anak sama dengan panjang kromosom *parent* terpendek
- Panjang kromosom anak sama dengan panjang kromosom *parent* terpanjang
- Panjang kromosom anak rata-rata dari panjang kromosom *parent*

Prosedur pemilihan jenis perkawinan ini dapat dilihat pada *Sourcecode* 4.8.

```
Random r = new Random();
int jenisKawin = r.Next(1, 4);
int pjgKromChild = 0;

AppendLog(OptimasiProses, "Jenis perkawinan: {0}",
jenisKawin);

switch (jenisKawin)
{
  case 1:
    pjgKromChild = parent1.Length > parent2.Length ?
    parent2.Length : parent1.Length;
  break;
  case 2:
    pjgKromChild = parent1.Length < parent2.Length ?
    parent2.Length : parent1.Length;
  break;

  case 3:
    pjgKromChild = (int)((parent1.Length + parent2.Length)
    / 2);
  break;
```

```
        default:
        break;
    }
```

***Sourcecode* 4.8** Prosedur pilih jenis perkawinan

Setelah menentukan jenis perkawinan, proses selanjutnya adalah mencari *cut point* untuk proses perkawinan. Dalam aplikasi ini menggunakan *one-cut-point*. Posisi *cut point* dipilih secara random, namun tidak boleh terlalu ke tepi dari kromosom yang ada. Prosedur memilih posisi *cut point* dapat dilihat pada *Sourcecode* 4.9.

```
    int cutPointMin = (int)(0.2 * pjgKromChild);
    int cutPointMax = (int)(0.8 * pjgKromChild);

    if (cutPointMin == 0)
     {
        cutPointMin = 1;
     }

    int cutPoint = r.Next(cutPointMin, cutPointMax + 1);
```

***Sourcecode* 4.9** Prosedur memilih posisi *cut point*

Setelah *cut point* didapatkan, maka dilakukan proses perkawinan silang, dengan memecah kromosom *parent* menjadi dua. Dari hasil perkawinan silang akan menghasilkan dua *child*. Kromosom *parent* sebelah kiri *cut point* akan menjadi awal dari kromosom *child*. Dan kromosom *parent* sebelah kanan *cut point* akan menjadi kromosom akhir dari *child*. Procedur perkawinan silang dapat dilihat pada *Sourcecode* 4.10.

```
    Kromosom[] kromParent1 = parent1.Split(cutPoint);
    Kromosom[] kromParent2 = parent2.Split(cutPoint);


    Kromosom kromParent1Kiri = kromParent1[0];
    Kromosom kromParent1Kanan = kromParent1[1];
    Kromosom kromParent2Kiri = kromParent2[0];
    Kromosom kromParent2Kanan = kromParent2[1];

    Kromosom[] childs = new Kromosom[2];
    hilds[0] = new Kromosom(this);
    childs[1] = new Kromosom(this);

    Kromosom kromChild1 = childs[0];
```

```
    Kromosom kromChild2 = childs[1];

    kromChild1.AddGens(kromParent1Kiri);
    kromChild2.AddGens(kromParent2Kiri);

    int pjgKromChildKanan = pjgKromChild - cutPoint;

    Kromosom kromChild1Kanan =
    GenerateKromChildKanan(pjgKromChildKanan, kromParent2Kanan,
    kromParent1Kanan);
    Kromosom kromChild2Kanan =
    GenerateKromChildKanan(pjgKromChildKanan, kromParent1Kanan,
    kromParent2Kanan);
    kromChild1.AddGens(kromChild1Kanan);
    kromChild2.AddGens(kromChild2Kanan);

     return childs;
```

***Sourcecode* 4.10** Prosedur perkawinan silang

Pada aplikasi ini, panjang kromosom *child* dapat berbeda dengan panjang kromosom *parent*. Kromosom sebelah kanan dari *child* dapat diisi dengan cara membangkitkan angka biner random sebanyak *gen child* yang harus terisi. Angka 1 berart terpilih ke dalam kromosom *child*, sedangkan angka 0 berarti tidak terpilih ke dalam kromosom *child*. Prosedur memilih *gen* yang dimasukkan ke dalam kromosom *child* dapat dilihat pada *Sourcecode* 4.11.

```
      Kromosom kromChild = new Kromosom(this);

      List<int> listPosisi = new List<int>(pjgKromChildKanan);

      Random r = new Random();
```

***Sourcecode* 4.11** Prosedur memilih *gen* untuk kromosom *child*

Apabila panjang kromosom *child* lebih pendek dari panjang kromosom *parent*, maka diambil *gen* dari kromosom *parent* sejumlah panjang kromosom *child*. *Gen* yang telah didapatkan diletakkan secara terurut sesuai dengan urutan *gen* pada kromosom *parent*, dan dimasukkan ke dalam kromosom *child*. Prosedur memilih *gen* untuk *child* yang lebih pendek dapat dilihat pada *Sourcecode* 4.12.

```
    if (pjgKromChildKanan < parentKanan.Length)
```

```
    {
      for (int i = 0; i < pjgKromChildKanan; i++)
      {
          int genPos = 0;

          do
          {
            genPos = r.Next(0, pjgKromChildKanan);
          }
          while (listPosisi.Contains(genPos));

          listPosisi.Add(genPos);
       }
        listPosisi.Sort();

        foreach (int genPos in listPosisi)
        {
            kromChild.AddGen(parentKanan[genPos]);
        }
     }
```

***Sourcecode* 4.12** Prosedur memilih *gen* untuk *child* yang memiliki kromosom lebih pendek dari kromosom *parent*

Sedangkan untuk *child* yang memiliki panjang kromosom lebih panjang dari *parent*, jumlah *gen* yang belum terisi diambil dari kromosom kanan *parent* yang lainnya. Prosedur memilih *gen* untuk *child* yang lebih panjang dari *parent* dapat dilihat pada *Sourcecode* 4.13.

```
   else            {
      for (int i = 0; i < parentKanan.Length; i++)
      {
        int genPos = 0;

        do
        {
        genPos = r.Next(0, parentKanan.Length);
        }
       while (listPosisi.Contains(genPos));

       listPosisi.Add(genPos);
      }

      listPosisi.Sort();

      foreach (int genPos in listPosisi)
      {
        kromChild.Add(parentKanan[genPos]);
       }
      listPosisi.Clear();
```

```
int sisa = pjgKromChildKanan - parentKanan.Length;

for (int i = 0; i < sisa; i++)
{
   int genPos = 0;

   do
   {
     genPos = r.Next(0, parentKananLain.Length);
    }
   while (listPosisi.Contains(genPos));

   listPosisi.Add(genPos);
}

listPosisi.Sort();

foreach (int genPos in listPosisi)
{
  if (genPos < kromChild.Length)
  {
    kromChild.Insert(genPos, parentKananLain[genPos]);
  }
  else
  {
    kromChild.Add(parentKananLain[genPos]);
   }
  }
 }
 return kromChild;
```

***Sourcecode* 4.13** Prosedur memilih *gen* untuk *child* yang memiliki kromosom lebih panjang dari kromosom *parent*

4.3.4 Mutasi

Mutasi dimulai dengan membangkitkan angka random untuk menentukan kromosom mana yang akan terkena mutasi. Kemudian membangkitkan angka random untuk menentukan posisi *gen* yang akan dimutasi. Pasangan *gen* yang akan dimutasi tidak boleh sama. Setelah menemukan posisi *gen* yang akan dimutasi, maka *gen* tersebut ditukar dengan pasangan *gen* mutasinya. Prosedur mutasi dapat dilihat pada *Sourcecode* 4.14.

```
Kromosom mutant = new Kromosom(pop);
mutant.AddRange(gens);

int jmlMutGen = (int)(mutant.Length * mutRate);
```

```
        int[] mutPos = new int[jmlMutGen];

        pop.AppendLogOptimasi("Jml gen yg dimutasi: {0}",
        jmlMutGen);
        Random r = new Random();
        for (int i = 0; i < jmlMutGen; i++)
        {
          int pos;

          do
          {
            pos = r.Next(0, mutant.Length);
          }
         while (Array.IndexOf(mutPos, pos) > -1);   // -1: ga ada

         mutPos[i] = pos;
        }
        for (int i = 0; i < mutPos.Length; i++)
        {
          pop.AppendLogOptimasi("Mutasi ke: {0}", i + 1);

          int pos1 = mutPos[i];
          int pos2 = pos1;
          do
          {
            pos2 = r.Next(0, mutant.Length);
          }
          while (pos1 == pos2);

          pop.AppendLogOptimasi("Mutasikan gen pd posisi: {0}
          ({1}) dg {2} ({3})", pos1, mutant[pos1].Id, pos2,
          mutant[pos2].Id);

          Spot mutGen = mutant[pos1];
          mutant[pos1] = mutant[pos2];
          mutant[pos2] = mutGen;
         }
         return mutant;
```

***Sourcecode* 4.14** Prosedur mutasi

4.3.5 Perbaikan Kromosom

Perbaikan kromosom dilakukan untuk memperbaiki kromosom yang *invalid*. Prosedur *repair* dapat dilihat pada *Sourcecode* 4.15.

```
        if (AdaGenDobel)
        {
           RepairDobel(parentAcuan, parentLain);
         }
```

```
        if (!TujuanLengkap)
        {
            RepairTujuanTidakLengkap(parentAcuan, parentLain);
         }

        if (!TujuanUrut)
        {
            RepairTujuanTidakUrut(parentAcuan);
         }
```

***Sourcecode* 4.15** Prosedur *repair*

Di dalam prosedur *repair* ada prosedur `repair dobel`, yang berfungsi untuk proses *repair* kromosom dengan kesalahan ada *gen* yang sama dalam satu kromosom. Proses pertama adalah mencari *gen* yang muncul dua kali dengan cara melihat setiap *gen* dan membandingkan dengan *gen* setelahnya. Prosedur mencari *gen* dan membandingkan *gen* setelahnya dapat dilihat pada *Sourcecode* 4.16.

```
        List<int> listPosGenDobel = new List<int>();

        for (int i = 0; i < gens.Count; i++)
        {
         Spot cek = gens[i];

         for (int j = i + 1; j < gens.Count; j++)
         {
            Spot target = gens[j];

            if (cek == target)
            {
               listPosGenDobel.Add(j);
            }
          }
         }
        pop.AppendLogOptimasi("Repair: {0} gen dobel...",
        listPosGenDobel.Count);

        Queue<Spot> listGenPengganti = new Queue<Spot>();
```

***Sourcecode* 4.16** Prosedur mencari *gen* dan muncul dua kali

Setelah mencari *gen* yang muncul dua kali, proses selanjutnya adalah mencari *gen* pada *parent* yang belum ada di dalam kromosom *child*. Dalam kasus ini dimungkinkan *gen* pada *parent* acuan telah terpakai, maka

mencari *gen* yang lain pada *parent2*. Prosedur mencari *gen parent* dapat dilihat pada *soucecode* 4.17.

```
foreach (Spot genParent in parentAcuan)
{
  if (!gens.Contains(genParent))
  {
    listGenPengganti.Enqueue(genParent);
  }
}
foreach (Spot genParentLain in parentLain)
{
 if (!gens.Contains(genParentLain))
 {
   listGenPengganti.Enqueue(genParentLain);
 }
}
```

***Sourcecode* 4.17** Prosedur mencari *gen* pada *parent*

Setelah mencari *gen* yang belum ada di dalam kromosom *child*, proses selanjutnya adalah mengganti *gen* yang muncul dua kali dengan *gen* yang telah ditemukan. Prosedur mengganti *gen* dapat dilihat pada *soucecode* 4.18

```
foreach (int pos in listPosGenDobel)
{
 if (listGenPengganti.Count > 0)
 {
   pop.AppendLogOptimasi("Gantikan gen ke: {0} ({1}) dg
  ({2})", pos, gens[pos].Id, listGenPengganti.Peek().Id);

   gens[pos] = listGenPengganti.Dequeue();
 }
 else
 {
   pop.AppendLogOptimasi("Tidak bisa menggantikan gen ke:
     {0} ({1}), tdk ditemukan pengganti di parent", pos,
 gens[pos].Id);
 }
}
```

***Sourcecode* 4.18** Prosedur mengganti *gen*

Di dalam prosedur *repair* ada prosedur *repair* tujuan tidak lengkap, yang berfungsi untuk proses *repair* kromosom dengan kesalahan ada *spot* tujuan yang tidak lengkap. Proses pertama adalah mendaftar semua *spot* tujuan yang ada di dalam kromosom *child*. Kemudian mencari posisi *spot* tujuan yang tidak ada di kromsosom *parent* dengan *parent* acuan yang diutamakan. Kemudian mengambil nilai posisi *spot* tujuan yang dicari. Pada kasus ini dimungkinkan nilai posisi *spot* yang dicari lebih besar dari jumlah *gen* pada kromosom *child*, maka pencarian posisi *gen* menggunakan persentase posisi *gen* pada *parent*. Prosedur mencari *repair* tujuan tidak lengkap dapat dilihat pada *Sourcecode* 4.19.

```
List<DestSpot> listSpotTujuan = new List<DestSpot>();

foreach (DestSpot dest in rute.Tujuan)
{
  if (!gens.Contains(dest))
  {
    listSpotTujuan.Add(dest);
  }
}

pop.AppendLogOptimasi("Repair: {0} tujuan tidak ada",
listSpotTujuan.Count);

foreach (DestSpot dest in listSpotTujuan)
{
  Kromosom parent = parentAcuan.Contains(dest) ?
  parentAcuan : parentLain;

  int pos = parent.gens.IndexOf(dest);

  if (pos >= gens.Count)
  {
    double pros = (pos / parent.Length);
    pos = (int)Math.Round(pros);
  }

  pop.AppendLogOptimasi("Memasukkan spot tujuan ({0}) di
  posisi ke: {1}\r\n", dest.Id, pos);

  gens.Insert(pos, dest);
 }
```

***Sourcecode* 4.19** Prosedur *repair* tujuan tidak lengkap

Di dalam prosedur *repair* ada proses *repair* tujuan tidak urut, yang berfungsi untuk memperbaiki kromosom yang memiliki *spot* tujuan

yang tidak terurut sesuai dengan urutan tujuan yang dikehendaki oleh pengguna. Proses *repair* dimulai dengan melacak posisi dari *spot* tujuan yang ada di kromosom *child*. Kemudian mengurutan posisi tujuan, dan mengganti *spot* tujuan sesuai dengan urutannya. Prosedur `repair tujuan tidak urut` dapat dilihat pada *Sourcecode* 4.20.

```
pop.AppendLogOptimasi("Repair: tujuan tidak urut.");

int[] listPosTujuan = new int[rute.Tujuan.Count];


for (int i = 0; i < rute.Tujuan.Count; i++)
{
  DestSpot tujuan = rute.Tujuan[i];
  int pos = gens.IndexOf(tujuan);
  listPosTujuan[i] = pos;
}

Array.Sort(listPosTujuan);

for (int i = 0; i < listPosTujuan.Length; i++)
{
  int pos = listPosTujuan[i];

  pop.AppendLogOptimasi("Gantikan Spot tujuan posisi: {0}
 ({1}) dg ({2})", pos, gens[pos].Id, rute.Tujuan[i].Id);

  gens[pos] = rute.Tujuan[i];
 }
```

***Sourcecode* 4.20** Prosedur `repair tujuan tidak urut`

4.3.6 Proses Perhitungan *Cost*

Proses perhitungan *cost* merupakan penjumlahan dari jarak antar *spot* dalam rute, yang mana merupakan *gen* dalam satu kromosom. Perhitungan *cost* dimulai dengan membuat *network* antar *spot* untuk mengetahui jarak antar *spot* tersebut. Perhitungan *cost* untuk *spot* yang merupakan pencilan, harus menambahkan *spot* yang disembunyikan atau yang tidak Nampak pada kromosom. Nilai *cost* untuk antar *spot* yang tidak memiliki jalur, diberi nilai maksimal atau 1000. Prosedur `perhitungan cost` dapat dilihat pada *Sourcecode* 4.21.

```
Network net = Network.GetNetwork();
```

```
        cost = net.GetJarak(rute.Awal, gens[0]);

        for (int pos = 0; pos < gens.Count - 1; pos++)
        {
          Spot gen = gens[pos];
          Spot next = gens[pos + 1];

          double jarak = net.GetJarak(gen, next);

          if (jarak == Network.JARAK_TANPA_JALUR)
          {
             bool pencilan = false;
             double totJarakPenc = 0;

             for (int prevPos = pos - 1; prevPos > 0; prevPos--)
             {
               Spot prev = gens[prevPos];
               double jarakPencilan = net.GetJarak(prev, next);

               Spot now = gens[prevPos + 1];
               totJarakPenc += net.GetJarak(prev, now);

               if (jarakPencilan != Network.JARAK_TANPA_JALUR)
               {
                  pencilan = true;
                  cost += jarakPencilan + totJarakPenc;
                  break;
               }
             }

             if (!pencilan)
             {
               cost += jarak;
             }
          }
          else
          {
            cost += jarak;
          }
        }
        cost += net.GetJarak(gens[gens.Count - 1], rute.Akhir);
```

***Sourcecode* 4.21** Prosedur `perhitungan cost`

4.3.7 Proses Perhitungan Biaya Tol

Perhitungan biaya tol dimulai dengan membuat *network* antar *spot* untuk mencari jarak antar *spot* tersebut. Kemudian melihat apakah antar *spot* tersebut memiliki biaya tol. Apabila ada, maka biaya tol dijumlahkan dari awal sampai akhir rute tersebut. Prosedur `perhitungan biaya tol` dapat dilihat pada *Sourcecode* 4.22.

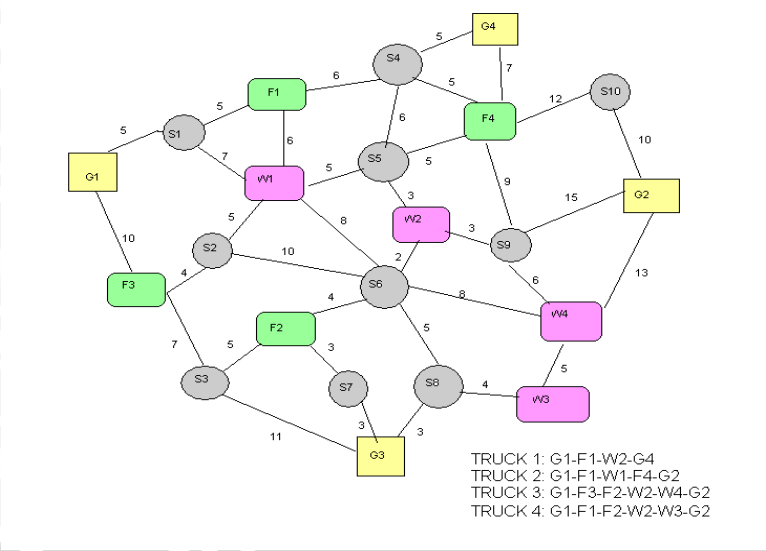
```
    Network net = Network.GetNetwork();

    biayaTol = net.GetBiayaTol(rute.Awal, gens[0]);
    for (int pos = 0; pos < gens.Count - 1; pos++)
    {
      Spot gen = gens[pos];
      Spot next = gens[pos + 1];

      double jarak = net.GetJarak(gen, next);

      if (jarak == Network.JARAK_TANPA_JALUR)
      {
        bool pencilan = false;
        double totBiayaPenc = 0;

        for (int prevPos = pos - 1; prevPos > 0; prevPos--)
        {
          Spot prev = gens[prevPos];
          double jarakPencilan = net.GetJarak(prev, next);

          Spot now = gens[prevPos + 1];
          totBiayaPenc += net.GetBiayaTol(prev, now);

          if (jarakPencilan != Network.JARAK_TANPA_JALUR)
          {
            pencilan = true;
            biayaTol += net.GetBiayaTol(prev, next) +
            totBiayaPenc;
            break;
          }
        }

        if (!pencilan)
        {
          biayaTol = net.GetBiayaTol(gen, next);
        }
      }
      else
      {
        biayaTol += net.GetBiayaTol(gen, next);
      }
    }
```

```
    biayaTol += net.GetBiayaTol(gens[gens.Count - 1],
    rute.Akhir);
```

***Sourcecode* 4.22** Prosedur `perhitungan biaya tol`

4.4 Penerapan Aplikasi

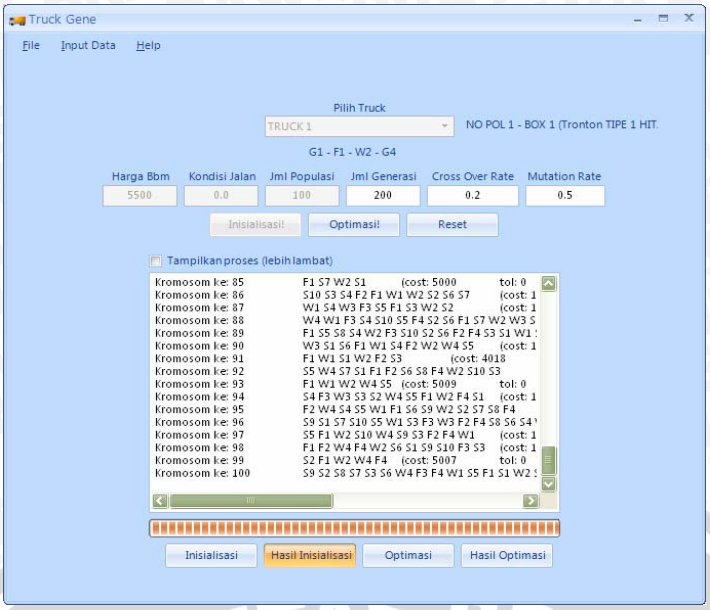
Aplikasi diterapkan dengan memasukkan data sesuai dengan keinginan *user*. Dalam kasus ini data yang ingin dicari adalah rute *truck1* dengan rute perjalanan dari G1-F1-W2-G4. Penggambaran jarak antar *spot* dapat dilihat pada Gambar 4.7.



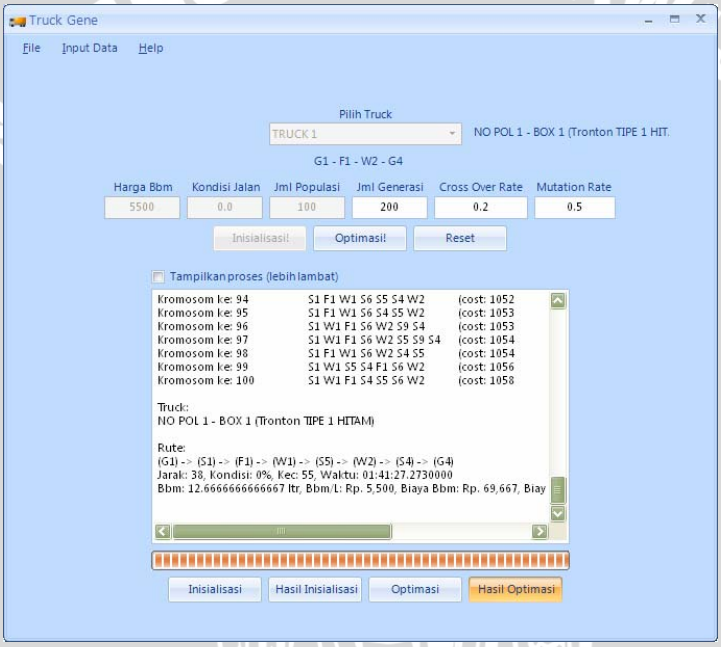
Gambar 4.7 Penggambaran rute perjalanan

Dalam kasus ini parameter genetika yang digunakan adalah jumlah populasi 100, iterasi/generasi yang dilakukan sebanyak 200 kali, dengan seleksi *roulette whell* serta *crossover rate* sebesar 0,2 dan *mutation rate* sebesar 0,5. Untuk lebih jelasnya seperti pada Gambar 4.8.

Apabila proses inisialisasi kromosom telah selesai, maka proses selanjutnya adalah proses optimasi. Proses optimasi disini adalah proses genetika yang dimulai dengan *crossover*. Setelah melalui proses genetika maka akan diperoleh urutan perjalanan, lama perjalanan, biaya BBM dan biaya tol, seperti pada Gambar 4.9



Gambar 4.8 Tampilan aplikasi setelah inisialisasi kromosom



Gambar 4.9 Tampilan aplikasi setelah proses optimasi

4.5 Analisa Hasil

Perubahan nilai *fitness* dari inisialisasi awal sampai menjadi *fitness* terbaik dapat dikarenakan model kromosom, *crossover rate, mutation rate*, banyaknya iterasi atau generasi yang digunakan, serta besarnya ukuran individu dalam satu populasi.

Pada penelitian ini mencoba melakukan analisa terhadap nilai *fitness* yang dihasilkan dengan mengganti nilai parameter *crossover rate*. Uji coba dilakukan untuk mengetahui adanya perubahan nilai *cost* dan nilai *fitness.* Nilai *crossover rate* yang diujikan adalah 0.2, 0.3, 0.4, dan 0.5. Uji coba dilakukan masing-masing 5 kali percobaan untuk setiap nilai probabilitas perkawinan silang.

Berikut hasil dari uji coba perubahan nilai parameter perkawinan silang:

Tabel 4.1 Hasil uji *crossover rate* 0.2

	Min cost	Average Cost	Maks Fit	Average fit
1	38	808.22	0.025641026	0.005546573
2	53	1757.2	0.018518519	0.000923636
3	38	190.97	0.025641026	0.015987724
4	40	1298.64	0.024390244	0.001484555
5	38	521.19	0.025641026	0.010665828
Rata-rata	41.4	915.244	0.0239664	0.0069217

Tabel 4.2 Hasil uji *crossover rate* 0.3

	Min cost	Average Cost	Maks Fit	Average fit
1	38	443.11	0.025641026	0.011995725
2	38	668.3	0.025641026	0.008098619
3	38	649.18	0.025641026	0.008416872
4	38	228.02	0.025641026	0.015772316
5	40	1035.01	0.024390244	0.001610169
Rata-rata	38.4	604.724	0.0253909	0.0091787

Tabel 4.3 Hasil uji *crossover rate* 0.4

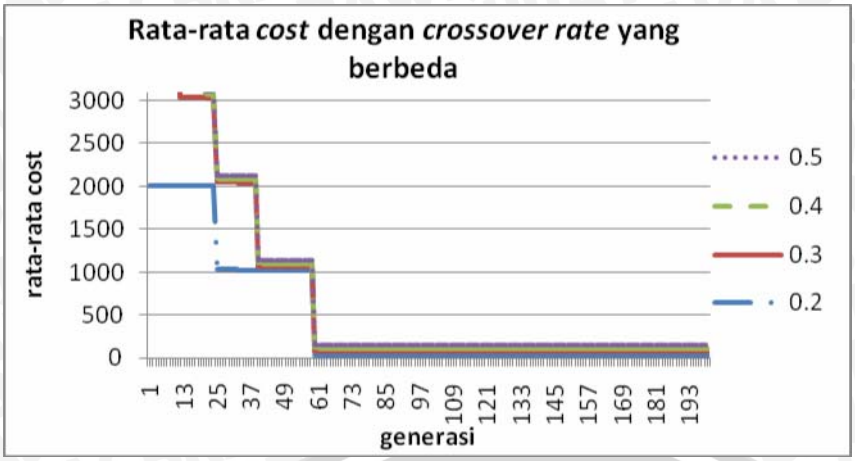
	Min cost	Average Cost	Maks Fit	Average fit
1	38	639.69	0.025641026	0.008546855
2	38	927.58	0.025641026	0.003148382
3	38	169.7	0.025641026	0.0167575
4	38	247.67	0.025641026	0.015390989
5	38	295.85	0.025641026	0.014710902
Rata-rata	38	456.098	0.025641	0.0117109

Tabel 4.4 Hasil uji *crossover rate* 0.5

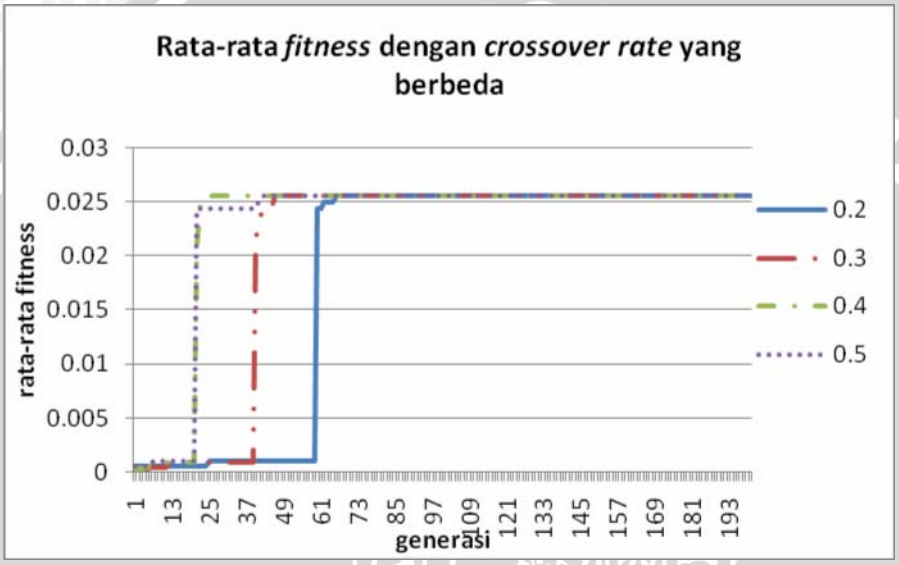
	Min cost	Average Cost	Maks Fit	Average fit
1	38	246,94	0.025641026	0.01558354
2	38	917.33	0.025641026	0.003303121
3	38	111,91	0.025641026	0.004503797
4	38	247.18	0.025641026	0.015511722
5	38	620.37	0.025641026	0.008911548
Rata-rata	38	594.96	0.025641	0.0095627

Pada tabel hasil uji dapat diketahui nilai minimal *cost*, rata-rata *cost,* maksimal *fitness* dan rata-rata *fitness* pada setiap kali dilakukan percobaan. Dari hasil uji coba dapat dilihat bahwa nilai minimum *cost* dari masing-masing perubahan *crossover rate* memiliki nilai yang sama yaitu 38. Tetapi yang membedakan dari masing-masing perubahan *crossover rate* pada waktu 5 kali *runtime* adalah rata-rata dari nilai *cost* dimana pada *crossover rate* 0.2 rata-rata nilai minimum *cost* adalah 41.4, ketika *crossover rate* dinaikkan menjadi 0.3 rata-rata nilai minimum *cost* mengalami penurunan menjadi 38.4. Pada saat *crossover rate* dinaikkan menjadi 0.4 dan 0.5 rata-rata nilai minimum *cost* mengalami penurunan menjadi 38. Perubahan rata-rata nilai *cost* dapat dilihat pada Gambar 4.10.

Pada Gambar 4.10 dapat dilihat rata-rata *cost* dari 5 kali *runtime* dengan jumlah generasi 200 dan probabilitas perkawinan silang yang berbeda. Saat *crossover rate* 0.2, 0.3, 0.4 dan 0.5 *cost* dan *fitness* paling optimal pada satu kali *runtime*. Pada saat terjadi perubahan *crossover rate* nilai *fitness* paling besar mengalami perubahan. Perubahan rata-rata *fitness* dapat dilihat pada Gambar 4.11.

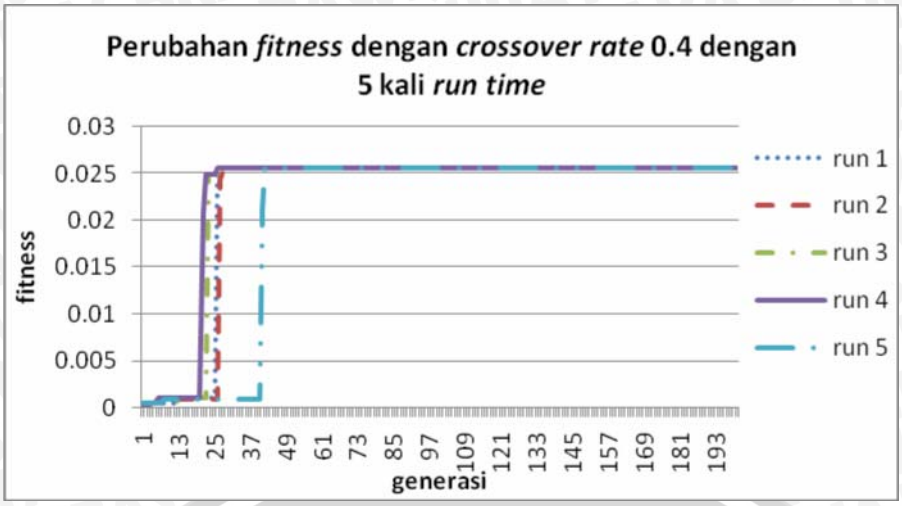


Gambar 4.10 Grafik perbandingan rata-rata *cost* dengan *crossover rate* yang berbeda



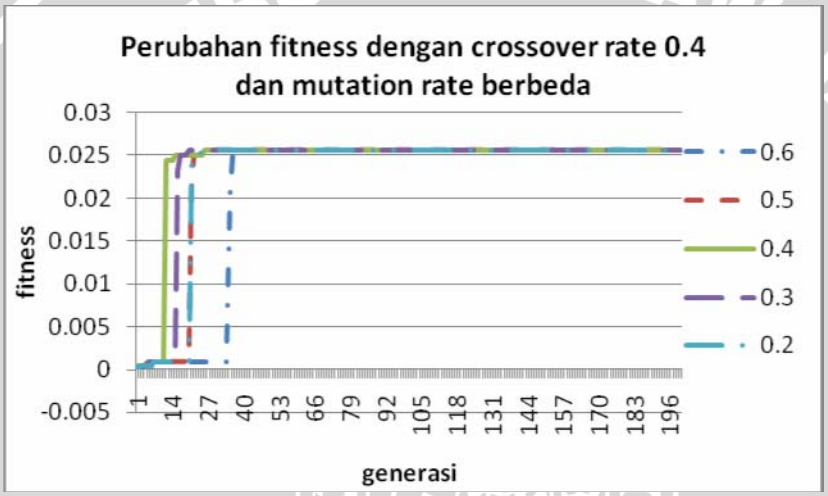
Gambar 4.11 Grafik perbandingan rata-rata *fitness* dengan *crossover rate* yang berbeda

Pada Gambar 4.11 dapat dilihat rata-rata *fitness* lima kali *runtime* dengan jumlah generasi 200 dan *crossover rate* yang berbeda.



Gambar 4.12 Grafik Perubahan *fitness* dengan *crossover rate* 0.4

Dari Gambar 4.12 dapat diketahui bahwa kondisi optimal *crossover rate* 0.4 karena kestabilan hasil yang ada setiap *runtime*.



Gambar 4.13 Grafik Perubahan *fitness* dengan *crossover rate* 0.4 dengan *mutation rate* yang berbeda

Dari Gambar 4.13 dapat dilihat rata-rata *fitness* lima kali *runtime* dengan jumlah generasi 200 dan *crossover rate* 0.4 dengan *mutation rate* yang berbeda.

Tabel 4.5 Hasil uji berdasarkan waktu komputasi

c.rate	0.5		0.4		0.3		0.2	
No	gen ke-	wkt komp.	gen ke-	wkt komp.	gen ke-	wkt komp.	gen ke-	wkt komp.
1	117	435.9063	117	248.4375	140	203.75	111	94.09375
2	98	343.0156	180	400.4219	125	175.188	131	107.4063
3	199	756.7656	152	284.9688	138	190.016	177	143.6719
4	89	335.1094	103	206.9219	200	329.531	196	174.0156
5	68	341.65	140	216.5	172	240.875	200	185.76
Rata-rata	114.2	442.4894	138.4	271.45	155	227.872	163	140.9895

Pada Tabel 4.5 dapat dilihat hasil uji berdasarkan waktu komputasi percobaan sampai mendapatkan hasil yang konvergen dengan 5 kali *runtime.*

Berdasarkan uji coba yang dilakukan, dapat diambil kesimpulan bahwa nilai *cost* akan semakin rendah dan nilai *fitness* akan semakin tinggi dengan semakin besar *crossover rate*. *Crossover rate* yang lebih stabil adalah 0.4, karena pada probabilitas 0.4 didapatkan jumlah nilai *fitness* tertingggi paling banyak pada waktu 5 kali *runtime.* *Mutation rate* yang lebih stabil adalah 0.2. Semakin besar *crossover rate* maka generasi akan semakin cepat konvergen, namun waktu komputasi yang diperlukan akan semakin besar.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari Tugas Akhir ini adalah:

1. Algoritma genetika dapat digunakan sebagai alternatif solusi untuk masalah optimasi rute pendistribusian komoditi.
2. Pada penelitian ini, telah dibuat model genetika untuk masalah optimasi rute pendistribusian komoditi dengan menggunakan inisialisasi kromosom secara random, metode seleksi yang digunakan adalah roda *roulette* dengan metode kawin silang *one cut point crossover* dan mutasi kromosom dilakukan dengan menukarkan 2 titik.
3. Penggunaan *crossover rate* yang berbeda pada optimasi rute pendistribusian komoditi ini, tidak menimbulkan perbedaan yang signifikan terhadap nilai *fitness* yang dihasilkan. Namun penggunaan *mutation rate* yang berbeda menyebabkan perubahan waktu komputasi yang diperlukan. Sehingga dapat dikatakan bahwa algoritma genetika dapat memecahkan dan menyelesaikan permasalahan optimasi rute pendistribusian komoditi dengan sebaik mungkin dengan menggunakan parameter genetika yang tepat.

5.2 Saran

Aplikasi yang dibangun masih belum sempurna. Hal yang dapat bermanfaat untuk mengembangkan aplikasi ini adalah:

1. Aplikasi rute pendistribusian komoditi ini dapat dikembangkan menjadi sebuah sistem informasi yang memiliki kemampuan menentukan rute sesuai kapasitas *truck* dan alur perjalanan distribusi.
2. Pencarian rute pada pendistribusian komoditi ini dilakukan pada kondisi jalan dengan adanya hambatan yang mengakibatkan perubahan waktu tempuh, sehingga dapat dikembangkan dengan adanya kondisi jalan yang searah, batas waktu yang diperbolehkan untuk dilewati oleh *truck*.

(halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

Ballou, Ronald H. 1992. *Business Logistic Management*. Third Edition. Prentice-Hall Int.

Gen, Mitsuo dan Cheng, Runwei. 1997. *Genetic Algorithms and Engineering Design*. John Wiley and Sons.

Ivan Leonardo, Vincentius. 2003. *Pengunaan Algoritma Genetik Untuk Rute Perencanaan Rute Perjalanan Di Jawa Timur*.Universitas Kristen Petra. Surabaya .http://dewey.petra.ac.id/spektra/module/catalog/docs/library/search_simple.php?keyword=sampah&source=title&mode=display&result_per_page=20&npage=1. Tanggal akses: 12 Maret 2008

Kurnia Mawaddah, Nia. 2006. *Penjadwalan Mata Kuliah Menggunakan Algoritma Genetika.* Universitas Brawijaya, Malang.

Kusumadewi, Sri. 2003. *Artificial Intelligence (Teknik dan Aplikasinya).* Graha Ilmu. Yogyakarta.

Kusumadewi, Sri dan Puromo, Hari 2005. *Penyelesaian Masalah Optimasi dengan Teknik-teknik Heuristik.* Graha Ilmu. Yogyakarta.

Michalewicz, Zbigniew. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

Novitasari, Artha. 2007. *Pemecahan Permasalahan Pencarian Rute Terpendek Untuk Jasa Antar Jemput Penumpang Menggunakan Algoritma Genetik.* Universitas Brawijaya. Malang.

Sulistiyaningtias, Hesty. 2003. *Sistem Pengangkutan Sampah di Kecamatan Sukun, Kota Malang dengan Gauled Container System.*Universitas Brawijaya, Malang.

Suyanto. 2005. *Algoritma Genetika dalam MATLAB.* ANDI. Yogyakarta.