TUGAS AKHIR – SM 141501

# ALGORITMA GENETIKA GANDA UNTUK *CAPACITATED VEHICLE ROUTING PROBLEM*

MUHAMMAD LUTHFI SHAHAB
NRP. 1211 100 047

Dosen Pembimbing
Prof. Dr. Mohammad Isa Irawan, M.T.

JURUSAN MATEMATIKA
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Teknologi Sepuluh Nopember
Surabaya 2015

**TUGAS AKHIR – SM 141501**

# ALGORITMA GENETIKA GANDA UNTUK *CAPACITATED VEHICLE ROUTING PROBLEM*

MUHAMMAD LUTHFI SHAHAB
NRP. 1211 100 047

Dosen Pembimbing
Prof. Dr. Mohammad Isa Irawan, M.T.

JURUSAN MATEMATIKA
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Teknologi Sepuluh Nopember
Surabaya 2015

**FINAL PROJECT – SM 141501**

# DOUBLE GENETIC ALGORITHM FOR CAPACITATED VEHICLE ROUTING PROBLEM

MUHAMMAD LUTHFI SHAHAB
NRP. 1211 100 047

Supervisor
Prof. Dr. Mohammad Isa Irawan, M.T.

DEPARTMENT OF MATHEMATICS
Faculty of Mathematics and Natural Sciences
Institut Teknologi Sepuluh Nopember
Surabaya 2015

# LEMBAR PENGESAHAN

## ALGORITMA GENETIKA GANDA UNTUK *CAPACITATED VEHICLE ROUTING PROBLEM*

## *DOUBLE GENETIC ALGORITHM FOR CAPACITATED VEHICLE ROUTING PROBLEM*

**TUGAS AKHIR**
Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Sains
pada
Program Studi S-1 Jurusan Matematika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Teknologi Sepuluh Nopember Surabaya

Oleh :
**MUHAMMAD LUTHFI SHAHAB**
NRP. 1211 100 047

Menyetujui,
Dosen Pembimbing

Prof. Dr. Mohammad Isa Irawan, M.T.
NIP. 19631225 198903 1 001

Mengetahui,
Ketua Jurusan Matematika
FMIPA ITS

Prof. Dr. Erna Apriliani, M.Si.
NIP. 19660414 199102 2 001
**Surabaya, Juli 2015**

**ALGORITMA GENETIKA GANDA UNTUK**
***CAPACITATED VEHICLE ROUTING PROBLEM***

**Nama Mahasiswa : Muhammad Luthfi Shahab**
**NRP : 1211 100 047**
**Jurusan : Matematika**
**Dosen Pembimbing : Prof. Dr. Mohammad Isa Irawan, M.T.**

**Abstrak**

*Capacitated vehicle routing problem (CVRP) adalah salah satu variasi dari vehicle routing problem (VRP) yang menggunakan batasan kapasitas pada kendaraan yang dipakai. Ada banyak metode yang telah diteliti untuk bisa menyelesaikan CVRP, namun penggunaan algoritma genetika masih belum memberikan hasil yang memuaskan. Untuk mempermudah menyelesaikan CVRP, dapat dilakukan dekomposisi pada CVRP agar terbagi menjadi beberapa daerah yang dapat diselesaikan secara independen. Berdasarkan hal tersebut, dirumuskan algoritma genetika ganda yang terlebih dahulu berusaha untuk mendekomposisi CVRP dan kemudian mencari rute terpendek pada setiap daerah menggunakan dua algoritma genetika sederhana yang berbeda. Algoritma genetika ganda kemudian dibandingkan dengan algoritma genetika. Untuk membandingkan dua algoritma tersebut, dibuat empat permasalahan yaitu P50, P75, P100, dan P125 dengan pengujian pada setiap permasalahan menggunakan empat belas variasi kapasitas kendaraan yang berbeda. Didapatkan hasil bahwa algoritma genetika ganda lebih baik dari algoritma genetika dari segi waktu komputasi dan generasi. Dari segi jarak, algoritma genetika ganda juga lebih baik dari algoritma genetika kecuali untuk beberapa kapasitas kendaraan yang kecil pada permasalahan P50 dan P75.*

*Kata Kunci* : CVRP, algoritma genetika, algoritma genetika ganda

**DOUBLE GENETIC ALGORITHM FOR CAPACITATED VEHICLE ROUTING PROBLEM**

**Nama of Student : Muhammad Luthfi Shahab**
**NRP : 1211 100 047**
**Department : Matematika**
**Supervisor : Prof. Dr. Mohammad Isa Irawan, M.T.**

**Abstract**

*Capacitated vehicle routing problem (CVRP) is one of the variations of the vehicle routing problem (VRP), that use the capacity restriction on the vehicle used. There are many methods have been studied to solve the CVRP, but the use of genetic algorithm still not give a satisfactory result. To ease completing CVRP, a decomposition can be done on CVRP so that separate into several areas that can be solved independently. Based on that, a double genetic algorithm formulated in advance trying to decompose CVRP and then find the shortest route for each region using two different simple genetic algorithm. Double genetic algorithm is then compared with genetic algorithm. To compare these two algorithms, four problems then formed, those are P50, P75, P100, and P125. For each problems, testing is done using fourteen different variations vehicle capacity. The result show that double genetic algorithm is better than genetic algorithm in terms of computational time and generation. In terms of distance, the double genetic algorithm is also better than genetic algorithm except for some small capacity vehicles at P50 and P75.*

*Keywords* : CVRP, genetic algorithm, double genetic algorithm

# KATA PENGANTAR

Bismillahirrahmanirrahim.

Puji syukur penulis panjatkan kehadirat Allah SWT, Maha Kuasa atas segala sesuatu, yang telah mengizinkan penulis untuk dapat menyelesaikan Tugas Akhir yang berjudul **"Algoritma Genetika Ganda untuk *Capacitated Vehicle Routing Problem*"**. Tidak lupa, sholawat serta salam penulis haturkan kepada Nabi Muhammad SAW, atas cahaya lurus yang senantiasa Beliau sebarkan.

Ucapan terima kasih penulis sampaikan kepada pihak-pihak yang membantu dalam menyelesaikan proses Tugas Akhir ini, khususnya kepada:

1. Bapak Prof. Dr. Mohammad Isa Irawan, M.T. selaku dosen pembimbing atas segala bimbingan, saran, dukungan, kesabaran dan waktu yang diberikan kepada penulis hingga Tugas Akhir ini selesai.
2. Bapak Dr. Imam Mukhlash, S.Si., M.T., Bapak Drs. Daryono Budi Utomo, M.Si., dan Ibu Dra. Wahyu Fistia Doctorina, M.Si. selaku dosen penguji atas kritik dan saran demi perbaikan Tugas Akhir ini.
3. Bapak Dr. Darmaji, S.Si., M.T. selaku dosen wali yang telah membimbing penulis selama kuliah di Matematika ITS.
4. Ibu Prof. Dr. Erna Apriliani, M.Si. selaku Ketua Jurusan Matematika ITS dan Bapak Dr. Chairul Imron, MI. Komp. Selaku Ketua Program Studi S1 Matematika ITS.
5. Seluruh dosen Jurusan Matematika ITS, atas ilmu yang telah diberikan selama penulis berada di bangku kuliah.
6. Ayahanda tercinta Ayah Zaid Hasan Shahab dan *the best mom ever* Mama Sri Rahayu atas segenap cinta kasih sayang, doa, serta perhatian moril maupun materil yang tidak pernah henti diberikan untuk penulis.
7. Keluarga tercinta, Mbak Evi, Mas Dafid, Mas Risqi, Nadiyya yang telah tanpa henti memberikan dukungan dan doa kepada penulis.

8. Tim Steering Commitee Padamu Himatika 2013 atas segala cerita hidup yang pernah dilakukan bersama.
9. Cordova Ulin Nuha Kamila atas segala perhatian dan waktu yang telah diberikan selama mendampingi penulis.
10. Teman-teman pejuang wisuda 112 yang telah berbagi suka duka selama mengerjakan Tugas Akhir ini.
11. Semua pihak yang tidak dapat disebutkan satu-persatu yang telah membantu hingga pelaksanaan Tugas Akhir ini dapat terselesaikan dengan baik.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan baik dari segi teknik penulisan maupun materi, oleh karena itu kritik dan saran yang membangun sangat penulis harapkan demi kesempurnaan Tugas Akhir ini. Semoga Tugas Akhir ini dapat meberikan banyak manfaat bagi semua pihak.

Surabaya, Juli 2015

Penulis

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

# DAFTAR LAMPIRAN

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

*Vehicle routing problem* (VRP) adalah salah satu permasalahan optimasi kombinatorial yang memiliki banyak aplikasi pada bidang industri (Nazif dan Lee, 2012). Aplikasi dari VRP tersebut banyak digunakan pada permasalahan transportasi dan distribusi. VRP memiliki banyak variasi disesuaikan pada batasan-batasan yang digunakan. Salah satu dari variasi tersebut adalah *capacitated vehicle routing problem* (CVRP) yang menggunakan batasan kapasitas pada kendaraan yang dipakai. Permasalahan penentuan rute distribusi bahan bakar minyak (Hermawan, 2012), permasalahan pengangkutan sampah oleh truk sampah, dan permasalahan pengambilan barang dari pemasok ke gudang pusat adalah contoh-contoh CVRP di dunia nyata.

Banyaknya aplikasi dari CVRP yang sesuai dengan permasalahan di dunia nyata mengakibatkan CVRP menjadi salah satu bidang ilmu yang banyak diteliti (Yucenur dan Demirel, 2011). Penelitian-penelitian untuk menyelesaikan CVRP tersebut dilakukan dengan berbagai metode yang berbeda. Metode eksak dapat menyelesaikan CVRP yang kecil dengan tepat, namun tidak dapat menyelesaikan CVRP yang besar. Metode-metode *metaheuristic* lebih sering digunakan karena dapat menyelesaikan CVRP dengan hasil yang cukup baik dan waktu komputasi yang lebih singkat. Beberapa metode *metaheuristic* yang dapat digunakan antara lain adalah *variable neighborhood search*, *greedy randomized adaptive search procedure*, *stochastic local search*, *iterated local search*, *particle swarm optimization*, *scatter search*, *differential evolution*, *simulated annealing*, *tabu search* dan algoritma genetika (Karakatic dan Podgorelec, 2015). Metode *metaheuristic* yang lain seperti *ant colony system* juga dapat digunakan untuk menyelesaikan CVRP (Hermawan, 2012).

Toth dan Vigo (2002) menyatakan bahwa penggunaan algoritma genetika untuk menyelesaikan CVRP masih belum memberikan hasil yang memuaskan. Namun, keberhasilan algoritma genetika untuk menyelesaikan permasalahan-permasalahan lain seperti *travelling salesman problem* (TSP) dan *vehicle routing problem with time windows* (VRPTW) menunjukkan bahwa penggunaan algoritma genetika akan memberikan hasil yang semakin baik jika terus diteliti. Algoritma genetika juga telah berhasil digunakan untuk menyelesaikan permasalahan penempatan *base transceiver station* (Pramsistya, 2010) dan permasalahan transportasi nonlinier (Soelistyowati, 2010). Dalam penelitian lain, pengembangan dari algoritma genetika yaitu NSGA-II juga telah berhasil digunakan untuk menyelesaikan permasalahan optimasi dalam distribusi kapal perang di wilayah perairan Indonesia (Hozairi, Buda, Masroeri, dan Irawan, 2014).

Untuk mempermudah menyelesaikan CVRP, Taillard (1993) melakukan dekomposisi pada CVRP agar terbagi menjadi beberapa daerah yang dapat diselesaikan secara independen. Dalam penelitiannya, penggabungan *tabu search* dengan dekomposisi CVRP yang dilakukan telah memberikan hasil yang sangat baik pada empat belas permasalahan klasik dan banyak dari hasil tersebut masih tetap menjadi yang terbaik hingga saat ini (Nazif dan Lee, 2012).

Berdasarkan hal tersebut, akan dirumuskan algoritma genetika ganda yang bekerja dengan terlebih dahulu berusaha untuk mendekomposisi CVRP menjadi beberapa daerah yang independen dan kemudian mencari rute terpendek pada setiap daerah menggunakan dua algoritma genetika sederhana yang berbeda. Algoritma genetika ganda tersebut diharapkan dapat digunakan untuk menyelesaikan CVRP dengan hasil yang baik. Dalam Tugas Akhir ini, algoritma genetika ganda akan dirumuskan secara runtut dan akan dibandingkan dengan algoritma genetika untuk mengetahui seberapa baik algoritma genetika ganda dapat digunakan untuk menyelesaikan CVRP.

## 1.2 Rumusan Masalah

Rumusan masalah dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana perumusan algoritma genetika ganda agar dapat digunakan untuk menyelesaikan CVRP?
2. Bagaimana perbandingan antara algoritma genetika ganda dengan algoritma genetika dalam menyelesaikan CVRP?

## 1.3 Batasan Masalah

Batasan masalah yang digunakan dalam Tugas Akhir ini adalah sebagai berikut:

1. Jenis CVRP yang digunakan adalah CVRP simetri yang berarti bahwa jarak perjalanan pergi dan pulang antara dua titik adalah sama.
2. Titik-titik tujuan dalam permasalahan yang dibuat, diatur agar menyebar di sekitar depot.
3. Program dibuat dengan menggunakan bahasa pemrograman Java dalam NetBeans IDE 8.0.2.

## 1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Merumuskan algoritma genetika ganda agar dapat digunakan untuk menyelesaikan CVRP.
2. Membandingkan algoritma genetika ganda dengan algoritma genetika dalam menyelesaikan CVRP.

## 1.5 Manfaat

Manfaat yang bisa diperoleh dari pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Memperkenalkan algoritma genetika ganda yang dapat digunakan untuk menyelesaikan CVRP. Algoritma genetika ganda tersebut juga dapat dikembangkan agar dapat digunakan untuk menyelesaikan permasalahan yang lain.

2. Mengetahui seberapa baik algoritma genetika ganda dapat digunakan untuk menyelesaikan CVRP jika dibandingkan dengan algoritma genetika.

# BAB II
# TINJAUAN PUSTAKA

## 2.1 *Capacitated Vehicle Routing Problem*

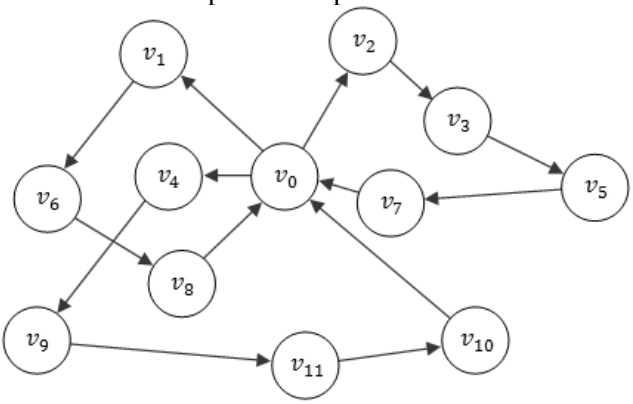
*Vehicle routing problem* (VRP) adalah salah satu permasalahan optimasi kombinatorial yang memiliki banyak aplikasi pada bidang industri (Nazif dan Lee, 2012). Aplikasi dari VRP tersebut banyak digunakan pada permasalahan transportasi dan distribusi. Secara teori, VRP diperoleh dari permasalahan optimasi dasar yaitu *travelling salesman problem* (TSP). VRP mengembangkan TSP dengan adanya depot, yaitu lokasi dari mana kendaraan harus berangkat dan harus pulang. Situasi-situasi dari permasalahan dunia nyata telah menciptakan banyak variasi dari konsep sederhana tersebut (Karakatic dan Podgorelec, 2015). Secara umum, VRP dapat digambarkan dengan permasalahan penyaluran barang dari satu atau beberapa depot ke titik-titik tujuan tertentu yang bertujuan untuk mencari suatu rute yang meminimalkan total jarak yang ditempuh (Yucenur dan Demirel, 2011).

VRP memiliki banyak variasi disesuaikan pada batasan-batasan yang digunakan, diantaranya adalah *heterogeneus vehicle routing problem* (HVRP), *capacitated vehicle routing problem* (CVRP), *multi depot vehicle routing problem* (MDVRP), *vehicle routing problem with time windows* (VRPTW), *vehicle routing problem with backhauls* (VRPB), *vehicle routing problem with pickup and delivery* (VRPPD), dan lain-lain (Yucenur dan Demirel, 2011). Batasan-batasan tersebut digunakan agar VRP dapat diaplikasikan pada permasalahan-permasalahan yang lebih khusus. CVRP menggunakan batasan kapasitas pada kendaraan yang dipakai dalam pengiriman barang (Karakatic dan Podgorelec, 2015). Permasalahan penentuan rute distribusi bahan bakar minyak (Hermawan, 2012), permasalahan pengangkutan sampah oleh truk sampah, dan permasalahan pengambilan barang dari pemasok ke gudang pusat adalah contoh-contoh CVRP di dunia nyata.

CVRP secara formal didefinisikan sebagai graf tak berarah $G = (V, E)$ dimana $V = \{v_0, v_1, \dots, v_n\}$ adalah himpunan *vertex* dan $E = \{(v_i, v_j) | v_i, v_j \in V, i < j\}$ adalah himpunan *edge*. Depot direpresentasikan dengan $v_0$, sedangkan $n$ titik tujuan direpresentasikan dengan $n$ *vertex* yaitu $v_1, \dots, v_n$. Setiap titik tujuan memiliki permintaan sebesar $q_i$ yang harus dilayani. Untuk melakukan pelayanan, digunakan $m$ kendaraan pengangkut barang yang independen. Setiap kendaraan memiliki kapasitas angkutan yang sama yaitu $Q$. Matriks $C = (c_{(i)(j)})$ yang menyediakan jarak perjalanan antara titik tujuan $v_i$ dan $v_j$ didefinisikan pada $E$. Solusi untuk CVRP adalah partisi-partisi $R_1, R_2, \dots, R_m$ dari $V$ yang merepresentasikan rute-rute perjalanan. Setiap rute $R_i$ yaitu$v_{i,0} \rightarrow v_{i,1} \rightarrow \cdots \rightarrow v_{i,k_i+1}$, dimana $v_{i,j} \in V$ dan $v_{i,0} = v_{i,k+1} = v_0$, haruslah memenuhi $\sum_{v_j \in R_i} q_j \leq Q$ yang berarti bahwa total permintaan pada setiap rute tidak boleh melebihi kapasitas kendaraan. Total jarak dari solusi permasalahan adalah jumlah dari jarak-jarak rute $R_i$ yang didefinisikan sebagai berikut:

$$Jarak = \sum_{i=1}^{m} Jarak\ (R_i) = \sum_{i=1}^{m} \sum_{j=0}^{k_i} c_{(i,j)(i,j+1)}. \qquad (2.1)$$

Contoh solusi CVRP dapat dilihat pada Gambar 2.1.




**Gambar 2.1** Contoh Solusi CVRP

CVRP bertujuan untuk menentukan himpunan dari $m$ rute yang total jaraknya minimum, sedemikian sehingga setiap rute dimulai dan berakhir di depot, setiap titik tujuan dikunjungi tepat satu kali dengan tepat satu kendaraan, dan total permintaan dari setiap rute tidak melebihi $Q$ (Nazif dan Lee, 2012).

## 2.2 Algoritma Genetika

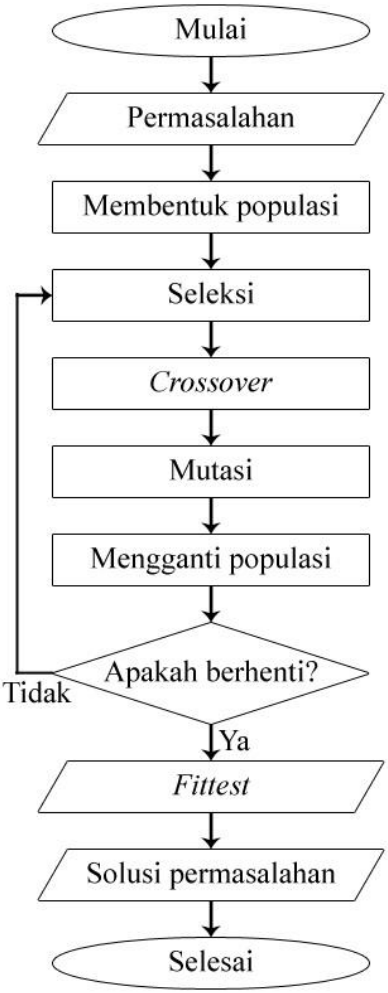
Gagasan dibalik algoritma genetika adalah memodelkan evolusi alami dengan menggunakan pewarisan genetik (Nazif dan Lee, 2012). Ada dua hal pokok yang harus dipikirkan sebelum menggunakan algoritma genetika, yang pertama adalah representasi kromosom dan yang kedua adalah pemilihan fungsi *fitness*. Kromosom tersebut akan menjadi solusi untuk CVRP dan fungsi *fitness* akan menjadi alat untuk mengukur kebaikan dari suatu kromosom.

Algoritma genetika dimulai dengan dugaan populasi yang merepresentasikan kromosom-kromosom sebagai solusi permasalahan. Biasanya dugaan tersebut dipilih secara acak yang tersebar di seluruh *search space*. Algoritma genetika kemudian menggunakan tiga operator penting yaitu seleksi, *crossover*, dan mutasi untuk mengolah populasi hingga konvergen atau menemukan hasil yang terbaik (Coley, 1999).

Seleksi berusaha untuk memberikan tekanan pada populasi seperti pada seleksi alami yang ditemukan pada sistem biologis. Kromosom yang buruk akan terbuang dan kromosom yang baik akan memiliki kesempatan yang lebih besar untuk berkembang pada generasi berikutnya. *Crossover* memungkinkan pertukaran informasi pada kromosom untuk membentuk kromosom-kromosom baru. Mutasi digunakan agar informasi-informasi yang ada dalam kromosom tidak mudah hilang (Coley, 1999).

Setelah seleksi, *crossover*, dan mutasi selesai dilakukan pada populasi awal, maka populasi baru akan terbentuk dan generasi dari populasi baru tersebut telah naik satu tingkat. Proses seleksi, *crossover*, dan mutasi terus dilanjutkan hingga jumlah generasi tertentu telah dilampaui atau kriteria pemberhentian dipenuhi

(Coley, 1999). Langkah-langkah algoritma genetika adalah seperti dalam Gambar 2.2.



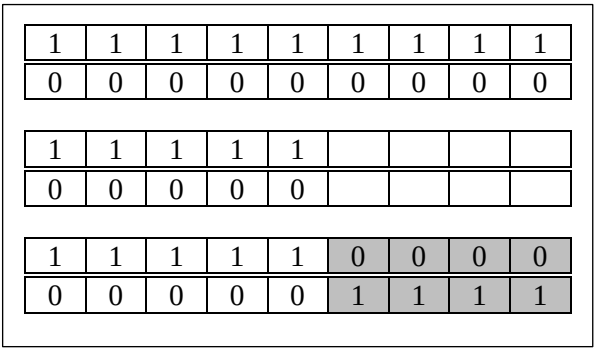

**Gambar 2.2** Langkah-Langkah Algoritma Genetika

### 2.2.1 *Tournament Selection*

*Tournament selection* adalah salah satu operator seleksi yang bekerja dengan terlebih dahulu memilih $n$ kromosom dalam populasi secara acak dan kemudian mengambil yang terbaik dari $n$ kromosom tersebut. Dengan *tournament selection* tersebut, kromosom yang baik akan memiliki kesempatan yang lebih besar untuk terpilih.

### 2.2.2 1-*Point Crossover*

1-*point crossover* adalah salah satu operator *crossover* yang biasa digunakan pada representasi kromosom biner. 1-*point crossover* bekerja dengan memilih titik secara acak pada kromosom dan kemudian menukar subkromosom yang berada di belakang titik bersebut. Contoh 1-*point crossover* dapat dilihat pada Gambar 2.3.




**Gambar 2.3** Contoh 1-*Point Crossover*

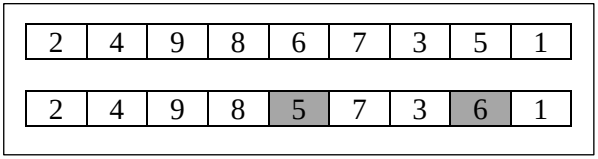
### 2.2.3 *Sequential Constructive Crossover*

*Sequential constructive crossover* (SCX) diperkenalkan oleh Ahmed (2005). SCX digunakan untuk menghasilkan kromosom baru dengan sisi baik yang ada pada orang tuanya juga dengan sisi baik yang tidak ada pada orang tuanya. SCX adalah operator *crossover* yang baik untuk menyelesaikan TSP baik dari segi jarak maupun dari segi waktu komputasi. Langkah-langkah dari SCX adalah sebagai berikut:

1. Mulai dari titik pertama.
2. Dicari titik yang belum dikunjungi dari kromosom orang tua yang muncul setelah titik $i$ dari setiap orang tua. Jika titik tersebut tidak ditemukan, maka dicari titik yang belum dikunjungi dari $(2,3,\dots,n)$.
3. Misalkan bahwa titik $p$ dan titik $q$ terpilih dari orang tua 1 dan orang tua 2. Apabila $c_{ip} < c_{iq}$ maka dipilih titik $p$ sebagai titik berikutnya dalam kromosom anak, jika $c_{ip} > c_{iq}$ maka dipilih titik $q$.
4. Jika kromosom belum lengkap maka kembali ke langkah 2. Jika kromosom telah lengkap maka berhenti.

### 2.2.4 *Exchange*

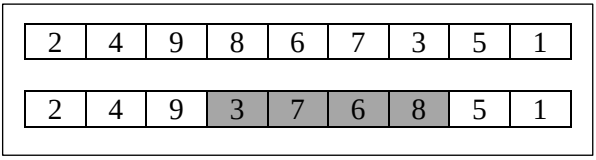
*Exchange* adalah salah satu operator mutasi yang dapat digunakan pada representasi kromosom berupa permutasi. *Exchange* dilakukan dengan memilih dua titik pada kromosom secara acak dan kemudian menukar posisinya (Karakatic dan Podgorelec, 2015). Contoh mutasi dengan *exchange* dapat dilihat pada Gambar 2.4.




**Gambar 2.4** Contoh *Exchange*

### 2.2.5 *Inversion*

*Inversion* adalah salah satu operator mutasi yang dapat digunakan pada representasi kromosom berupa permutasi. *Inversion* dilakukan dengan memilih subkromosom secara acak pada kromosom dan kemudian membalik urutan dari subkromosom tersebut (Karakatic dan Podgorelec, 2015). Contoh mutasi dengan *inversion* dapat dilihat pada Gambar 2.5.




**Gambar 2.5** Contoh *Inversion*

### 2.2.6 *Elitism Replacement with Filtration*

Skema *elitism replacement* digunakan agar populasi dalam algoritma genetika terus berkembang menjadi lebih baik. Skema ini akan menyaring kromosom-kromosom yang baik. *Elitism replacement* berjalanan dengan menggabungkan populasi awal dengan populasi baru menjadi satu populasi dan kemudian mengurutkan kromosom-kromosom berdasarkan *fitness*. Setengah pertama dari populasi tersebut akan diteruskan untuk menjadi populasi baru yang sesungguhnya. *Filtration* digunakan untuk mengidentifikasi kromosom yang identik dari populasi. Kromosom yang identik akan dibuang dan digantikan dengan kromosom dengan *fitness* terbaik dari sisa populasi yang tidak terpakai.

# BAB III
# METODOLOGI PENELITIAN

## 3.1 Studi Literatur

Dilakukan studi literatur untuk mendukung pengerjaan Tugas Akhir dan pemahaman yang lebih mendalam mengenai CVRP dan algoritma genetika. Literatur yang dipelajari dapat bersumber dari jurnal, buku, internet, maupun bimbingan dengan dosen pembimbing.

## 3.2 Pembuatan CVRP

Pembuatan CVRP dilakukan agar algoritma genetika dan algoritma genetika ganda dapat diimplementasikan untuk menyelesaikan CVRP tersebut. Dalam Tugas Akhir ini akan dibuat empat CVRP dengan jumlah titik tujuan yang berbeda.

## 3.3 Perumusan Algoritma Genetika untuk CVRP

Sebelum merumuskan algoritma genetika ganda, terlebih dahulu akan dirumuskan algoritma genetika yang dapat digunakan untuk menyelesaikan CVRP. Yang akan dirumuskan dalam algoritma genetika adalah representasi kromosom, besar populasi, fungsi *fitness*, operator seleksi, operator *crossover*, operator mutasi, skema penggantian populasi, dan kondisi pemberhentian.

## 3.4 Perumusan Algoritma Genetika Ganda untuk CVRP

Algoritma genetika ganda dirumuskan dengan merumuskan dua algoritma genetika sederhana yang dapat digunakan untuk menyelesaikan CVRP dengan cara yang berbeda. Yang akan dirumuskan untuk masing-masing algoritma genetika adalah representasi kromosom, besar populasi, fungsi *fitness*, operator seleksi, operator *crossover*, operator mutasi, skema penggantian populasi, dan kondisi pemberhentian.

## 3.5 Perancangan Program

Perancangan program dilakukan agar dapat digunakan untuk mengetahui perbandingan antara algoritma genetika dan algoritma genetika ganda dalam menyelesaikan CVRP. Program yang akan dibuat adalah program yang mengimplementasikan algoritma genetika dan algoritma genetika ganda untuk menyelesaikan CVRP.

## 3.6 Pembandingan Algoritma Genetika dengan Algoritma Genetika Ganda

Pembandingan algoritma genetika dengan algoritma genetika ganda dilakukan dengan memanfaatkan CVRP dan program yang telah dibuat. Perbandingan antara algoritma genetika dan algoritma genetika ganda akan dilihat dari tiga segi yang berbeda yaitu dari segi jarak, waktu komputasi, dan total generasi yang dibutuhkan untuk menyelesaikan CVRP.

## 3.7 Penarikan Kesimpulan

Penarikan kesimpulan dilakukan dengan memperhatikan hasil dan pembahasan yang telah diselesaikan pada tahap-tahap sebelumnya. Kesimpulan yang ditarik adalah mengenai apakah algoritma genetika ganda dapat digunakan untuk menyelesaikan CVRP dan apakah algoritma genetika ganda dapat menyelesaikan CVRP dengan hasil yang baik.

# BAB IV
# HASIL DAN PEMBAHASAN

## 4.1 Pembuatan CVRP

Pembuatan CVRP dilakukan agar algoritma genetika dan algoritma genetika ganda dapat diimplementasikan untuk menyelesaikan CVRP tersebut. Dengan diimplementasikannya algoritma genetika dan algoritma genetika ganda untuk menyelesaikan CVRP, dapat dilakukan pembandingan antara algoritma genetika dan algoritma genetika ganda. Agar dapat melakukan penarikan kesimpulan dengan cukup baik, dibuat empat CVRP yaitu P50, P75, P100, dan P125. Absis, ordinat, dan permintaan dari setiap titik tujuan pada CVRP yang dibuat, dipilih secara acak dari suatu rentang tertentu.

Permasalahan P50 terdiri dari depot dan 50 titik tujuan. Dalam permasalahan tersebut, absis dan ordinat depot adalah 50 sedangkan absis dan ordinat titik tujuan bernilai antara 0 sampai 100. Setiap titik tujuan memiliki permintaan yang bernilai antara 10 sampai 30. Total permintaan dari semua titik tujuan adalah 1040. Absis, ordinat, dan permintaan dari setiap titik tujuan pada permasalahan P50 dapat dilihat pada Lampiran 1.

Permasalahan P75 terdiri dari depot dan 75 titik tujuan. Dalam permasalahan tersebut, absis dan ordinat depot adalah 75 sedangkan absis dan ordinat titik tujuan bernilai antara 0 sampai 150. Setiap titik tujuan memiliki permintaan yang bernilai antara 10 sampai 30. Total permintaan dari semua titik tujuan adalah 1413. Absis, ordinat, dan permintaan dari setiap titik tujuan pada permasalahan P75 dapat dilihat pada Lampiran 2.

Permasalahan P100 terdiri dari depot dan 100 titik tujuan. Dalam permasalahan tersebut, absis dan ordinat depot adalah 100 sedangkan absis dan ordinat titik tujuan bernilai antara 0 sampai 200. Setiap titik tujuan memiliki permintaan yang bernilai antara 10 sampai 30. Total permintaan dari semua titik tujuan adalah 2044. Absis, ordinat, dan permintaan dari setiap titik tujuan pada permasalahan P100 dapat dilihat pada Lampiran 3.

Permasalahan P125 terdiri dari depot dan 125 titik tujuan. Dalam permasalahan tersebut, absis dan ordinat depot adalah 125 sedangkan absis dan ordinat titik tujuan bernilai antara 0 sampai 250. Setiap titik tujuan memiliki permintaan yang bernilai antara 10 sampai 30. Total permintaan dari semua titik tujuan adalah 2472. Absis, ordinat, dan permintaan dari setiap titik tujuan pada permasalahan P125 dapat dilihat pada Lampiran 4.

## 4.2 Perumusan Algoritma Genetika untuk CVRP

Sebelum merumuskan algoritma genetika ganda, terlebih dahulu akan dirumuskan algoritma genetika yang dapat digunakan untuk menyelesaikan CVRP. Algoritma genetika tersebut dirumuskan dengan karakteristik sebagai berikut:

a) Representasi kromosom yang digunakan dalam algoritma genetika adalah permutasi dari titik-titik tujuan. Setiap kromosom yang terbentuk adalah unik dan setiap kromosom hanya bisa merepresentasikan satu solusi CVRP. Sebagai contoh, apabila permasalahan CVRP yang digunakan terdiri dari 9 titik tujuan, salah satu kromosom yang dapat digunakan adalah $v_1 v_6 v_8 v_5 v_3 v_2 v_7 v_4 v_9$. Untuk merubah kromosom tersebut menjadi solusi yang diinginkan, digunakan informasi mengenai kapasitas kendaraan dan permintaan dari setiap titik tujuan. Misalkan kapasitas kendaraan dalam permasalahan adalah 17 dan permintaan dari setiap titik tujuan $v_i$ adalah $i$, $i = 1, \dots, 9$, maka rute pertama adalah $R_1$ yaitu $v_0 \rightarrow v_1 \rightarrow v_6 \rightarrow v_8 \rightarrow v_0$, rute kedua adalah $R_2$ yaitu $v_0 \rightarrow v_5 \rightarrow v_3 \rightarrow v_2 \rightarrow v_7 \rightarrow v_0$, dan rute ketiga adalah $R_3$ yaitu $v_0 \rightarrow v_4 \rightarrow v_9 \rightarrow v_2$.

b) Besar populasi yang digunakan dalam algoritma genetika adalah 100. Nilai tersebut dipilih agar algoritma genetika dapat memberikan hasil yang cukup baik dengan waktu komputasi yang tidak terlalu lama.

c) Misalkan dari suatu kromosom didapatkan rute-rute $R_i$, $i = 1, \dots, n$, dimana $R_i$ adalah $v_{i,0} \rightarrow v_{i,1} \rightarrow \cdots \rightarrow v_{i,k_i+1}$ dengan

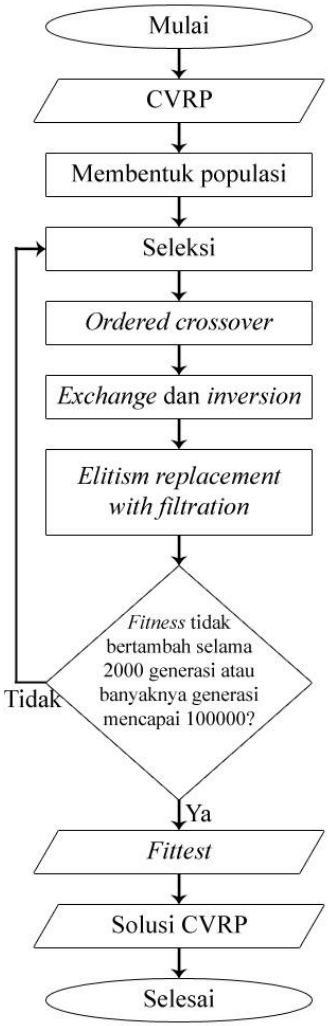
$v_{i,0} = v_{i,k_i+1} = v_0$, maka fungsi *fitness* yang dapat digunakan untuk algoritma genetika adalah

$$f(X,Y) = \sum_{i=1}^{n} \sum_{j=1}^{k_i+1} \sqrt{\left(x_{i,j} - x_{i,j-1}\right)^2 + \left(y_{i,j} - y_{i,j-1}\right)^2} \qquad (4.1)$$

dimana $x_{i,j}$ dan $y_{i,j}$ adalah absis dan ordinat dari $v_{i,j}$, $X$ adalah matriks dengan elemen pada baris ke-$i$ dan kolom ke-$j$ adalah $x_{i,j-1}$, dan $Y$ adalah matriks dengan elemen pada baris ke-$i$ dan kolom ke-$j$ adalah $y_{i,j-1}$. Semakin kecil nilai *fitness* dari suatu kromosom, maka semakin baik kromosom tersebut.

d) Seleksi yang digunakan dalam algoritma genetika adalah pemilihan dua kromosom dalam populasi secara acak. Hal ini dilakukan karena pemilihan dua kromosom yang baik tidak menjamin akan didapatkannya keturunan yang baik pula.
e) Operator *crossover* yang digunakan dalam algoritma genetika adalah *ordered crossover* (OX) dengan kemungkinan terjadinya *crossover* adalah 1,0. OX tersebut dipilih karena dapat memberikan keturunan yang *feasible* untuk representasi kromosom berupa permutasi.
f) Operator mutasi yang digunakan adalah *exchange* dan *inversion* dengan kemungkinan terjadinya masing-masing mutasi adalah 0,1. Operator tersebut dipilih agar informasi-informasi yang ada dalam kromosom tidak mudah hilang.
g) Skema penggantian populasi yang digunakan dalam algoritma genetika adalah *elitism replacement with filtration*. Dengan skema tersebut, *fittest* yang ada pada populasi baru tidak akan lebih buruk dari *fittest* yang ada pada populasi sebelumnya.
h) Kondisi pemberhentian yang digunakan dalam algoritma genetika adalah tidak bertambahnya *fitness* selama 2000 generasi atau banyaknya generasi telah mencapai 100000. Nilai-nilai tersebut dipilih agar algoritma genetika dapat berhenti dengan hasil yang cukup baik.

Berdasarkan perumusan algoritma genetika tersebut, langkah-langkah algoritma genetika untuk menyelesaikan CVRP adalah seperti yang ditunjukkan dalam Gambar 4.1.



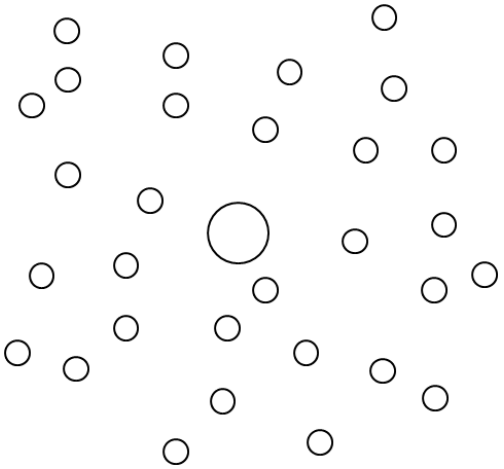

**Gambar 4.1** Langkah-Langkah Algoritma Genetika untuk Menyelesaikan CVRP

## 4.3 Perumusan Algoritma Genetika Ganda untuk CVRP

Algoritma genetika ganda bekerja dengan menggabungkan dua algoritma genetika sederhana agar dapat digunakan untuk menyelesaikan CVRP dengan cara yang berbeda dari algoritma genetika yang biasa. Algoritma genetika berusaha untuk menyelesaikan CVRP secara langsung, sedangkan algoritma genetika ganda akan terlebih dahulu berusaha untuk mendekomposisi CVRP menjadi beberapa daerah yang independen dengan AG1 (algoritma genetika pertama dalam algoritma genetika ganda) dan kemudian mencari rute terpendek pada setiap daerah yang terbentuk oleh AG1 dengan AG2 (algoritma genetika kedua dalam algoritma genetika ganda). Daerah-daerah yang terbentuk dari hasil dekomposisi yang dilakukan oleh AG1 haruslah memenuhi dua karakteristik sebagai berikut:

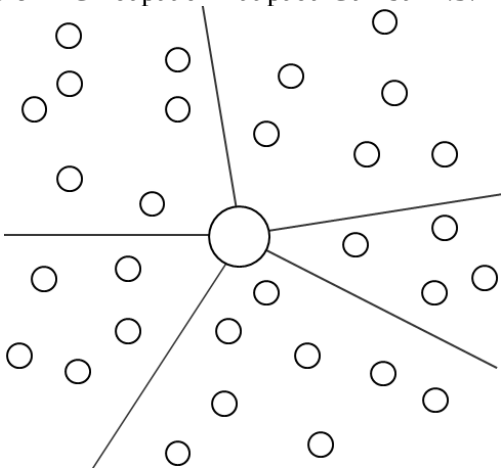
a) Setiap daerah hanya membutuhkan satu kendaraan untuk melayani setiap titik tujuan yang ada dalam daerah tersebut. Dengan kata lain, total permintaan semua titik tujuan yang ada dalam setiap daerah tidak boleh melebihi kapasitas kendaraan.
b) Titik-titik tujuan yang terletak dalam satu daerah harus terletak saling berdekatan.

Karakteristik yang pertama diambil dari sifat dasar CVRP yang menyatakan bahwa setiap rute yang terbentuk dalam solusi CVRP harus dilayani oleh satu kendaraan. AG1 akan berusaha memenuhi karakteristik pertama tersebut dengan mempertimbangkan kapasitas kendaraan dan permintaan dari setiap titik tujuan. Karakteristik yang kedua dibuat agar nantinya solusi CVRP yang terbentuk akan menjadi cukup baik. AG1 akan berusaha memenuhi karakteristik kedua tersebut dengan mempertimbangkan kemiringan garis yang menghubungkan titik tujuan dengan depot. Dalam hal ini, digunakannya kemiringan garis didasarkan pada kenyataan bahwa apabila kemiringan antara dua garis saling berdekatan, maka titik-titik yang ada pada garis tersebut juga akan cukup berdekatan.



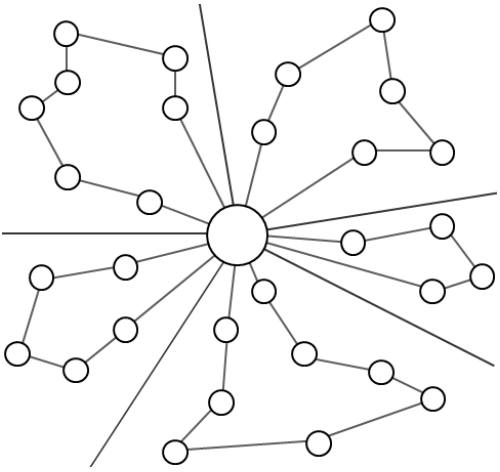
**Gambar 4.2** Contoh CVRP

Perhatikan contoh CVRP sederhana dalam Gambar 4.2 dimana lingkaran besar merepresentasikan depot dan lingkaran-lingkaran kecil merepresentasikan titik-titik tujuan. Dari permasalahan tersebut, salah satu dekomposisi yang bisa dihasilkan oleh AG1 dapat dilihat pada Gambar 4.3.



**Gambar 4.3** Contoh Dekomposisi oleh AG1

Dengan dilakukannya dekomposisi oleh AG1, solusi untuk setiap daerah yang terbentuk akan menjadi solusi untuk CVRP. Solusi untuk setiap daerah yang terbentuk adalah rute terpendek yang berangkat dari depot, kemudian menghubungkan setiap titik yang ada dalam daerah, dan kemudian kembali lagi ke depot. Pemilihan rute tersebut akan dilakukan oleh AG2. Perhatikan bahwa karena setiap daerah yang terbentuk dari hasil dekomposisi hanya membutuhkan satu kendaraan untuk melayani setiap titik tujuan yang ada dalam daerah tersebut, maka informasi mengenai permintaan dari setiap titik tujuan dapat dihilangkan sehingga permasalahan pemilihan rute terpendek pada setiap daerah dapat disebut sebagai *travelling salesman problem* (TSP). Oleh karena itu, AG2 akan mencari rute terpendek pada setiap daerah dengan hanya mempertimbangkan lokasi depot dan lokasi titik tujuan. Apabila rute terpendek untuk setiap daerah telah didapatkan dengan AG2, maka setiap rute tersebut akan digabung menjadi satu sehingga menjadi solusi dari CVRP. Untuk contoh CVRP pada Gambar 4.2 yang telah didekomposisi seperti pada Gambar 4.3, penggabungan rute yang telah diperoleh dengan AG2 dapat dilihat pada Gambar 4.4.



**Gambar 4.4** Contoh Solusi CVRP dengan Algoritma Genetika Ganda

### 4.3.1 Perumusan AG1

AG1 digunakan untuk mendekomposisi CVRP menjadi beberapa daerah yang independen. Sebelum AG1 dapat digunakanan, harus dihitung terlebih dahulu setiap kemiringan garis yang menghubungkan titik tujuan dengan depot. Setelah setiap kemiringan didapatkan, kemiringan tersebut diurutkan mulai dari yang terkecil hingga yang terbesar. Setelah setiap kemiringan selesai diurutkan, setiap titik tujuan yang bersesuaian dengan kemiringan tersebut dilabeli dengan $v_1, v_2, \dots, v_n$, dimana $n$ adalah jumlah titik tujuan dalam CVRP yang digunakan, dengan urutan yang sesuai dengan urutan kemiringan. AG1 dirumuskan dengan karakteristik sebagai berikut:

a) Representasi kromosom yang digunakan dalam AG1 adalah representasi kromosom biner. Sebagai contoh, apabila CVRP yang digunakan terdiri dari 20 titik tujuan, salah satu kromosom yang dapat digunakan adalah 00010000010010000000. Kromosom tersebut menunjukkan bahwa CVRP didekomposisi menjadi tiga daerah. Banyaknya daerah yang terbentuk adalah sama dengan banyaknya digit 1 yang muncul dalam kromosom. Daerah pertama ditandai dengan subkromosom 100000. Karena subkromosom tersebut mengisi kromosom pada posisi ke-4 sampai ke-9, maka yang menjadi titik-titik tujuan pada daerah pertama adalah $v_4, v_5, \dots, v_9$. Daerah kedua ditandai dengan subkromosom 100. Karena subkromosom tersebut mengisi kromosom pada posisi ke-10 sampai ke-12, maka yang menjadi titik-titik tujuan pada daerah kedua adalah $v_{10}, v_{11}, v_{12}$. Daerah ketiga ditandai dengan subkromosom 000 dan 10000000. Karena subkromosom tersebut mengisi kromosom pada posisi ke-1 sampai ke-3 dan posisi ke-13 sampai ke-20, maka yang menjadi titik-titik tujuan pada daerah ketiga adalah $v_1, v_2, v_3, v_{13}, v_{14}, \dots, v_{20}$.
b) Besar populasi yang digunakan dalam AG1 adalah 100. Nilai tersebut dipilih agar AG1 dapat memberikan hasil yang cukup baik dengan waktu komputasi yang tidak terlalu lama.

c) Misalkan dari suatu kromosom didapatkan daerah-daerah $D_i$, $i = 1, \dots, n$, dimana $D_i$ adalah $\{v_{i,1}, \dots, v_{i,k_i}\}$. Didefinisikan beberapa fungsi sebagai berikut:

$$f_1(a, m) = \mid a - m \mid \tag{4.2}$$

$$f_2(Q, Q_i) = \max \left\{ 0 \, , Q - \sum_{j=1}^{k_i} q_{i,j} \right\} \tag{4.3}$$

$$f_3(X_i) = \frac{1}{k_i} \sum_{j=1}^{k_i} x_{i,j} \tag{4.4}$$

$$f_4(Y_i) = \frac{1}{k_i} \sum_{j=1}^{k_i} y_{i,j} \tag{4.5}$$

$$f_5(X_i, Y_i) = \sum_{j=1}^{k_i} \sqrt{(x_{i,j} - f_3(X_i))^2 + (y_{i,j} - f_4(Y_i))^2} \tag{4.6}$$

$$f(a, m, Q, Q_i, X_i, Y_i) = 10^6 f_1(a, m) + 10^3 \sum_{i=1}^{m} f_2(Q, Q_i) + \sum_{i=1}^{m} f_5(X_i, Y_i) \tag{4.7}$$

dimana $x_{i,j}$, $y_{i,j}$, dan $q_{i,j}$ adalah absis, ordinat, dan permintaan dari $v_{i,j}$, $a$ adalah banyaknya digit 1 dalam kromosom, $m$ adalah banyak kendaraan yang digunakan, $Q$ adalah kapasitas kendaraan, $Q_i$ adalah matriks baris dengan elemen pada kolom ke-$j$ adalah $q_{i,j}$, $X_i$ adalah matriks baris dengan elemen pada kolom ke-$j$ adalah $x_{i,j}$, dan $Y_i$ adalah matriks baris dengan elemen pada kolom ke-$j$ adalah $y_{i,j}$. Fungsi $f(a, m, Q, Q_i, X_i, Y_i)$ adalah fungsi *fitness* yang digunakan dalam AG1. Semakin kecil nilai *fitness* dari suatu kromosom, maka semakin baik kromosom tersebut. Nilai $10^6$ dan $10^3$ yang digunakan dalam *fitness* tersebut akan menyebabkan tujuan utama dari AG1 adalah memperkecil $f_1(a, m)$, kemudian memperkecil $\sum_{i=1}^{m} f_2(Q, Q_i)$, dan yang terakhir memperkecil $\sum_{i=1}^{m} f_5(X_i, Y_i)$.

d) Seleksi yang digunakan dalam AG1 adalah pemilihan dua kromosom dalam populasi secara acak. Hal ini dilakukan karena pemilihan dua kromosom yang baik tidak menjamin akan didapatkannya keturunan yang baik pula.
e) Operator *crossover* yang digunakan dalam AG1 adalah 1-*point crossover* dengan kemungkinan terjadinya *crossover* adalah 1,0. *Crossover* tersebut dipilih karena dapat memberikan keturunan yang *feasible* untuk representasi kromosom biner.
f) Mutasi dalam AG1 dilakukan dengan memilih secara acak suatu digit dalam kromosom dan kemudian merubah nilainya. Jika yang terpilih adalah digit 1, maka dirubah menjadi 0. Jika yang terpilih adalah digit 0, maka dirubah menjadi 1. Kemungkinan terjadinya mutasi adalah 0,5. Mutasi tersebut dilakukan agar informasi-informasi yang ada dalam kromosom tidak mudah hilang.
g) Skema penggantian populasi yang digunakan dalam AG1 adalah *elitism replacement with filtration*. Dengan skema tersebut, *fittest* yang ada pada populasi baru tidak akan lebih buruk dari *fittest* yang ada pada populasi sebelumnya.
h) Kondisi pemberhentian yang digunakan dalam AG1 adalah tidak bertambahnya *fitness* selama 2000 generasi atau banyaknya generasi telah mencapai 100000. Nilai-nilai tersebut dipilih agar AG1 dapat berhenti dengan hasil yang cukup baik.

### 4.3.2 Perumusan AG2

AG2 digunakan untuk mencari rute terpendek pada setiap daerah yang terbentuk dari hasil dekomposisi oleh AG1. Rute-rute tersebut kemudian digabungkan untuk menjadi solusi CVRP. AG2 dirumuskan dengan karakteristik sebagai berikut:

a) Representasi kromosom yang digunakan dalam AG2 adalah permutasi dari titik-titik tujuan yang ada dalam daerahnya. Sebagai contoh, apabila titik-titik tujuan pada suatu daerah adalah $v_4, v_5, \dots, v_9$, salah satu kromosom yang dapat

digunakan adalah $v_6v_7v_4v_9v_5v_8$. Kromosom tersebut menunjukan bahwa rute perjalanan yang terbentuk adalah $v_0 \rightarrow v_6 \rightarrow v_7 \rightarrow v_4 \rightarrow v_9 \rightarrow v_5 \rightarrow v_8 \rightarrow v_0$.

b) Besar populasi yang digunakan dalam AG2 adalah 50. Nilai tersebut dipilih agar AG2 dapat memberikan hasil yang cukup baik dengan waktu komputasi yang tidak terlalu lama.

c) Misalkan sebuah kromosom $v_1v_2 \ldots v_n$, fungsi *fitness* yang dapat digunakan untuk mengolah kromosom tersebut adalah

$$f(X,Y) = \sum_{i=1}^{n+1} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \qquad (4.2)$$

dimana $v_{n+1} = v_0$, $x_i$ dan $y_i$ adalah absis dan ordinat dari $v_i$, $X$ adalah matriks baris dengan elemen pada kolom ke-$j$ adalah $x_j$, dan $Y$ adalah matriks baris dengan elemen pada kolom ke-$j$ adalah $y_j$.

d) Operator seleksi yang digunakan dalam AG2 adalah *tournament selection* dengan besar 5. Operator tersebut digunakan karena *crossover* dapat memberikan keturunan yang baik dari dua kromosom yang baik.

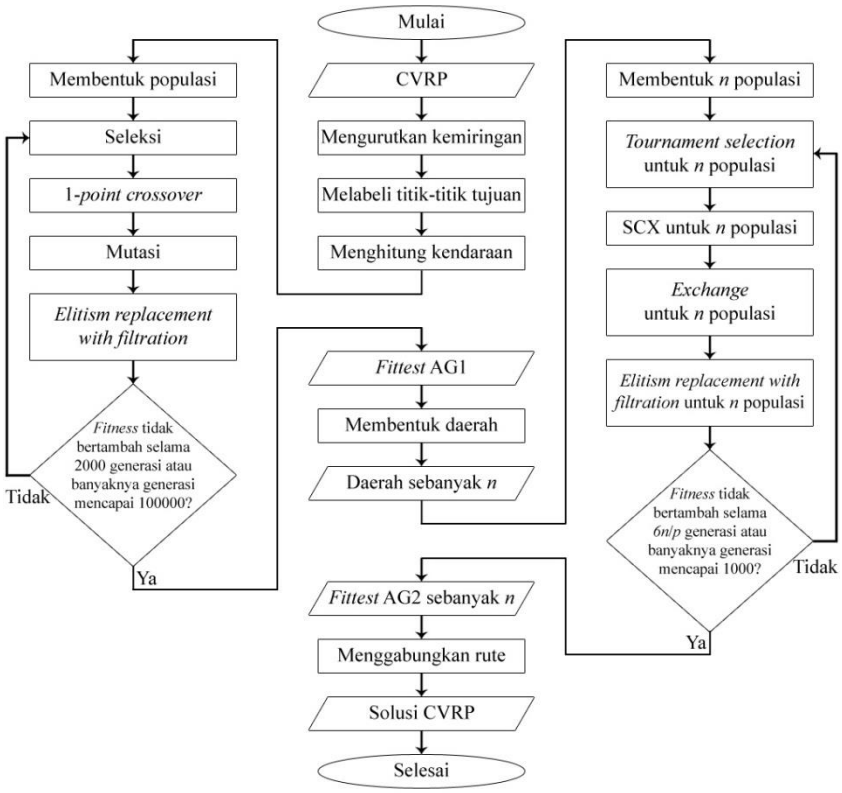
e) Operator *crossover* yang digunakan dalam AG2 adalah *sequential constructive crossover* (SCX) dengan kemungkinan terjadinya *crossover* adalah 1,0. SCX tersebut adalah operator *crossover* yang sangat baik untuk menyelesaikan TSP. SCX dipilih karena permasalahan pemilihan rute untuk daerah-daerah yang terbentuk dari hasil dekomposisi oleh AG1 merupakan TSP.

f) Operator mutasi yang digunakan dalam AG2 adalah *exchange* dengan kemungkinan terjadinya mutasi adalah 0,2. Operator tersebut dipilih agar informasi-informasi yang ada dalam kromosom tidak mudah hilang.

g) Skema penggantian populasi yang digunakan dalam AG2 adalah *elitism replacement with filtration*. Dengan skema tersebut, *fittest* yang ada pada populasi baru tidak akan lebih buruk dari *fittest* yang ada pada populasi sebelumnya.

h) Kondisi pemberhentian yang digunakan dalam AG2 adalah tidak bertambahnya *fitness* selama $\frac{6n}{p}$ generasi ($n$ adalah banyak titik tujuan pada suatu daerah dan $p$ adalah banyak daerah yang terbentuk dari hasil dekomposisi oleh AG1) atau banyaknya generasi telah mencapai 1000. Nilai-nilai tersebut dipilih agar AG2 dapat berhenti dengan hasil yang cukup baik.

Langkah-langkah algoritma genetika ganda untuk menyelesaikan CVRP dapat dilihat pada Gambar 4.5.




**Gambar 4.5** Langkah-Langkah Algoritma Genetika Ganda untuk Menyelesaikan CVRP

## 4.4 Implementasi pada Program

Agar dapat mengetahui perbandingan antara algoritma genetika dengan algoritma genetika ganda dalam menyelesaikan CVRP, dilakukan perancangan suatu program yang mengimplementasikan algoritma genetika dan algoritma genetika ganda. Perancangan program tersebut dilakukan dengan memperhatikan perumusan algoritma genetika dan perumusan algoritma genetika ganda yang telah dibahas sebelumnya.

Untuk mengimplementasikan algoritma genetika dan algoritma genetika ganda dalam menyelesaikan CVRP, digunakan beberapa *class* sebagai berikut:

a) CVRP.java

*Class* ini adalah *class* utama yang digunakan untuk menjalankan program. Isi dari *class* ini adalah mendeklarasikan dan memanggil Simulation.java. Keseluruhan *source code* dari CVRP.java dapat dilihat pada Lampiran 5.

b) City.java

*Class* ini digunakan untuk mendefinisikan suatu depot atau titik tujuan pada permasalahan CVRP yang akan diselesaikan dengan algoritma genetika. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah absis, ordinat, dan permintaan suatu depot atau titik tujuan tertentu. Dalam *class* ini juga terdapat sebuah fungsi yang dapat digunakan untuk menghitung jarak antara suatu titik dengan titik lainnya. Fungsi tersebut dijalankan dengan *source code* sebagai berikut:

```
public double distanceTo(City city){
   int xx = Math.abs(getX() - city.getX());
   int yy = Math.abs(getY() - city.getY());
   double distance = Math.sqrt((xx*xx)+(yy*yy));
   return distance;
}
```

Keseluruhan *source code* dari City.java dapat dilihat pada Lampiran 6.

c) TourManager.java

*Class* ini digunakan untuk menyimpan depot dan titik-titik tujuan. Depot dan titik-titik tujuan tersebut adalah representasi dari permasalahan CVRP yang akan diselesaikan dengan algoritma genetika. Keseluruhan *source code* dari TourManager.java dapat dilihat pada Lampiran 7.

d) Tour.java

*Class* ini digunakan untuk merepresentasikan kromosom dalam algoritma genetika. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah kromosom dan *fitness* dari kromosom. Suatu kromosom dibuat secara acak dengan menjalankan *source code* sebagai berikut:

```
public void generateIndividual() {
    int n = TourManager.numberOfCities();
    for (int i = 0; i < n; i++) {
      setCity(i, TourManager.getCity(i));
    }
    Collections.shuffle(tour);
}
```

Keseluruhan *source code* dari Tour.java dapat dilihat pada Lampiran 8.

e) Population.java

*Class* ini digunakan untuk mendefinisikan populasi dalam algoritma genetika. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah populasi dan untuk mendapatkan kromosom terbaik yang ada dalam populasi. Suatu populasi dibuat dengan menjalankan *source code* sebagai berikut:

```
public Population(int populationSize, boolean bol) {
    tours = new Tour[populationSize];
    if (bol) {
        for (int i=0; i < populationSize(); i++) {
            Tour newTour = new Tour();
            newTour.generateIndividual();
            saveTour(i, newTour);
        }
    }
}
```

Untuk mendapatkan kromosom terbaik yang ada dalam populasi, digunakan *source code* sebagai berikut:

```
public Tour getFittest() {
    Tour fittest = tours[0];
    for (int i = 1; i < populationSize(); i++) {
        if (fittest.getFitness() <= getTour(i).getFitness()) {
            fittest = getTour(i);
        }
    }
    return fittest;
}
```

Keseluruhan *source code* dari Population.java dapat dilihat pada Lampiran 9.

f) AG.java

*Class* ini digunakan untuk mengimplementasikan algoritma genetika. Dalam *class* ini terdapat beberapa fungsi yang digunakan untuk melakukan *ordered crossover*, *exchange* dan *inversion*, dan *elitism replacement with filtration*. *Ordered crossover* dilakukan dengan menjalankan *source code* sebagai berikut:

```
public static Tour crossover(Tour p1, Tour p2) {
    Tour child = new Tour();

    int a = (int) (Math.random() * p1.tourSize());
    int b = (int) (Math.random() * p1.tourSize());

    for (int i = 0; i < child.tourSize(); i++) {
        if (a < b && i > a && i < b) {
            child.setCity(i, p1.getCity(i));
        }
        else if (a > b) {
            if (!(i < a && i > b)) {
                child.setCity(i, p1.getCity(i));
            }
        }
    }

    for (int i = 0; i < child.tourSize(); i++) {
        if (!child.containsCity(p2.getCity(i))) {
            for (int ii = 0; ii < child.tourSize(); ii++) {
                if (child.getCity(ii) == null) {
```

```
                child.setCity(ii, p2.getCity(i));
                break;
            }
        }
    }
  }
  return child;
}
```

*Exchange* dan *inversion* dilakukan dengan menjalankan *source code* sebagai berikut:

```
private static void mutate(Tour tour) {
  double a =  Math.random();

  if(a < (mutationRate/2)){
    int r1 = (int) (tour.tourSize() * Math.random());
    int r2 = (int) (tour.tourSize() * Math.random());
    City city1 = tour.getCity(r1);
    City city2 = tour.getCity(r2);
    tour.setCity(r2, city1);
    tour.setCity(r1, city2);
  }
  else if (a < mutationRate){
    int r1 = (int) (tour.tourSize() * Math.random());
    int r2 = (int) (tour.tourSize() * Math.random());
    if (r2 > r1){
      int x = r2;
      r2 = r1;
      r1 = x;
    }
    int b = 0;
    City city = new City();
    for (int i = r1 ; i < r2 ; i++){
      city = tour.getCity((r1+b));
      tour.setCity((r1+b), tour.getCity((r2-b)));
      tour.setCity((r2-b), city);
      b++;
    }
  }
}
```

*Elitism replacement with filtration* dilakukan dengan menjalankan *source code* sebagai berikut:

```
int m = 0;
```

```
newPopulation3.saveTour(0, newPopulation2.getTour(0));
for (int i = 1; i < b; i++){
   if (newPopulation2.getTour(i).getDistance() !=
                                    newPopulation3.getTour(m).getDistance()){
      m++;
      newPopulation3.saveTour(m, newPopulation2.getTour(i));
   }
}
if ((m+1) != b){
   for (int i = m+1; i < b; i++){
      Tour acak = new Tour();
      acak.generateIndividual();
      newPopulation3.saveTour(i, acak);
   }
}
```

Keseluruhan *source code* dari AG.java dapat dilihat pada Lampiran 10.

g) Location.java

*Class* ini digunakan untuk mendefinisikan suatu depot atau titik tujuan pada permasalahan CVRP yang akan diselesaikan dengan algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah absis, ordinat, permintaan, dan kemiringan garis suatu depot atau titik tujuan tertentu. Dalam *class* ini juga terdapat sebuah fungsi yang dapat digunakan untuk menghitung jarak antara suatu titik dengan titik lainnya. Keseluruhan *source code* dari Location.java dapat dilihat pada Lampiran 11.

h) DataAG1.java

*Class* ini digunakan untuk menyimpan depot, titik-titik tujuan, kapasitas kendaraan, dan jumlah kendaraan. Depot dan titik-titik tujuan tersebut adalah representasi dari permasalahan CVRP yang akan didekomposisi oleh AG1 dari algoritma genetika ganda. Keseluruhan *source code* dari DataAG1.java dapat dilihat pada Lampiran 12.

i) ChromosomeAG1.java

*Class* ini digunakan untuk merepresentasikan kromosom dalam AG1 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk

mengolah kromosom dan *fitness* dari kromosom. Suatu kromosom dibuat secara acak dengan menjalankan *source code* sebagai berikut:

```
public ChromosomeAG1(){
   int a;
   a = number_of_customers/number_of_vehicles;
   int[] position_of_ones = new int[number_of_vehicles];
   for (int i=0 ; i < number_of_vehicles ; i++){
      position_of_ones[i] = (int) (Math.random() * a);
      chromosome[position_of_ones[i]+(a*i)] = 1;
   }
   for (int i=0; i < number_of_customers ; i++) {
      if (chromosome[i] != 1)
         chromosome[i] = 0;
   }
}
```

Keseluruhan *source code* dari ChromosomeAG1.java dapat dilihat pada Lampiran 13.

j) PopulationAG1.java

*Class* ini digunakan untuk mendefinisikan populasi dalam AG1 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah populasi dan untuk mendapatkan kromosom terbaik yang ada dalam populasi. Keseluruhan *source code* dari PopulationAG1.java dapat dilihat pada Lampiran 14.

k) AG1.java

*Class* ini digunakan untuk mengimplementasikan AG1 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang digunakan untuk melakukan 1-*point crossover*, mutasi, dan *elitism replacement with filtration*. 1-*point crossover* dilakukan dengan menjalankan *source code* sebagai berikut:

```
public static ChromosomeAG1 crossover(ChromosomeAG1 p1,
                                       ChromosomeAG1 p2) {
   ChromosomeAG1 child = new ChromosomeAG1();

   int position = (int) (Math.random() * p1.chromosomeSize());

   for (int i = 0 ; i < position ; i++)
```

```
        child.setValue(i, p1.getValue(i));

    for (int i = position ; i < p1.chromosomeSize() ; i++)
        child.setValue(i, p2.getValue(i));

    return child;
}
```

Mutasi dilakukan dengan menjalankan *source code* sebagai berikut:

```
private static void mutate(ChromosomeAG1 a) {
    double dummy = Math.random();
    if (dummy < 0.5){
        int position = (int) (Math.random() * a.chromosomeSize());
        a.setValue(position, Math.abs((a.getValue(position)-1)));
    }
}
```

Keseluruhan *source code* dari AG1.java dapat dilihat pada Lampiran 15.

l) DataAG2.java

*Class* ini digunakan untuk menyimpan depot, titik-titik tujuan yang akan diolah oleh AG2 dari algoritma genetika ganda. Depot dan titik-titik tujuan tersebut didapatkan dari hasil dekomposisi yang dilakukan oleh AG1 dari algoritma genetika ganda. Keseluruhan *source code* dari DataAG2.java dapat dilihat pada Lampiran 16.

m) ChromosomeAG2.java

*Class* ini digunakan untuk merepresentasikan kromosom dalam AG2 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah kromosom dan *fitness* dari kromosom. Keseluruhan *source code* dari ChromosomeAG2.java dapat dilihat pada Lampiran 17.

n) PopulationAG2.java

*Class* ini digunakan untuk mendefinisikan populasi dalam AG2 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang dapat digunakan untuk mengolah populasi dan untuk mendapatkan kromosom terbaik yang ada

dalam populasi. Keseluruhan *source code* dari PopulationAG2.java dapat dilihat pada Lampiran 18.

o) AG2.java

*Class* ini digunakan untuk mengimplementasikan AG2 dari algoritma genetika ganda. Dalam *class* ini terdapat beberapa fungsi yang digunakan untuk melakukan *tournament selection*, *sequential constructive crossover* (SCX), *exchange*, dan *elitism replacement with filtration*. *Tournament selection* dilakukan dengan menjalankan *source code* sebagai berikut:

```
private static ChromosomeAG2 tournamentSelection
                                                    (PopulationAG2 pop) {

   PopulationAG2 tournament = new PopulationAG2(tournamentSize,
                                                  false, pop.getDataAG2());

   for (int i = 0; i < tournamentSize; i++) {
      int randomId = (int) (Math.random() * pop.populationSize());
      tournament.saveChromosome(i, pop.getChromosome(randomId));
   }

   ChromosomeAG2 fittest=tournament.getFittest();
   return fittest;
}
```

*Exchange* dilakukan dengan menjalankan *source code* sebagai berikut:

```
private static void mutate(ChromosomeAG2 tour) {
   for(int i=0; i < tour.tourSize(); i++){
      if(Math.random() < mutationRate){
         int r = (int) (tour.tourSize() * Math.random());

         Location city1 = tour.getLocation(i);
         Location city2 = tour.getLocation(r);

         tour.setLocation(r, city1);
         tour.setLocation(i, city2);
      }
   }
}
```

Keseluruhan *source code* dari AG2.java dapat dilihat pada Lampiran 19.

p) Simulation.java

*Class* ini digunakan sebagai *graphical user inrterface* (GUI) yang merupakan penghubung antara pengguna dengan program. Pengguna dapat memilih permasalahan P50, P75, P100, P125, atau membuat sendiri CVRP yang akan diselesaikan dengan algoritma genetika dan algoritma genetika ganda. Dalam pembuatan CVRP, titik-titik tujuan dijamin tidak ada yang sama dengan menggunakan *source code* sebagai berikut:

```
public void cekBeda(int[] a, int[] b, int j){
   for (int i = 0 ; i < j ; i++){
      if (a[j] == a[i] && b[j] == b[i]){
         a[j] = (int)(Math.random()*X);
         cekBeda(a, b, j);
      }
   }
}
```

Setelah permasalahan selesai dipilih, pengguna dapat memasukkan kapasitas kendaraan yang akan digunakan dalam algoritma genetika dan algoritma genetika ganda. Keseluruhan *source code* dari Simulation.java dapat dilihat pada Lampiran 20.

q) Pelanggan.java

*Class* ini digunakan untuk menampilkan titik-titik tujuan yang merupakan representasi dari CVRP yang telah dipilih. Keseluruhan *source code* dari Pelanggan.java dapat dilihat pada Lampiran 21.

r) PelangganAG.java

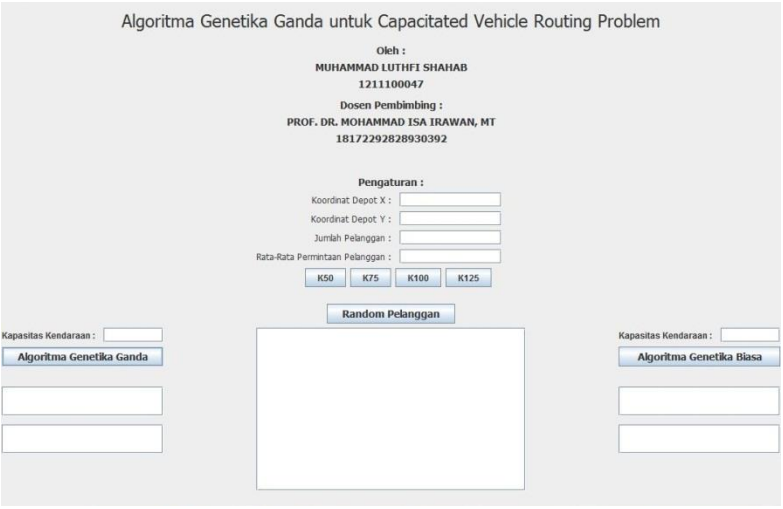
*Class* ini digunakan untuk menampilkan solusi rute dari CVRP yang telah diselesaikan dengan algoritma genetika. Keseluruhan *source code* dari PelangganAG.java dapat dilihat pada Lampiran 22.

s) PelangganAGG.java

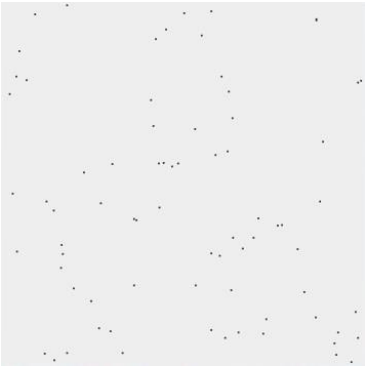
*Class* ini digunakan untuk menampilkan solusi rute dari CVRP yang telah diselesaikan dengan algoritma genetika

ganda. Keseluruhan *source code* dari PelangganAGG.java dapat dilihat pada Lampiran 23.

Apabila program dijalankan, maka akan muncul tampilan seperti dalam Gambar 4.6.



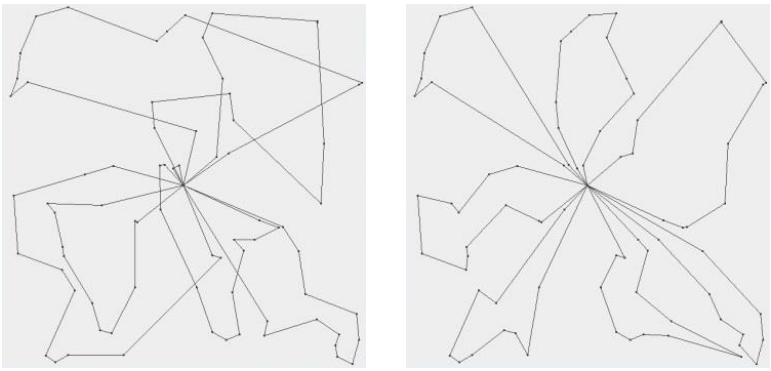

**Gambar 4.6** Tampilan Awal Program saat Dijalankan



**Gambar 4.7** Lokasi dari Titik-Titik Tujuan

Sebagai contoh, misalkan dibuat CVRP dengan absis dan ordinat depot adalah 300, jumlah titik tujuan adalah 75 dan rata-rata permintaan adalah 20. Maka akan muncul tampilan seperti

pada Gambar 4.7. Apabila kapasitas kendaraan yang digunakan adalah 250 dan kemudian permasalahan tersebut diselesaikan dengan algoritma genetika dan algoritma genetika ganda, maka akan muncul solusi rute seperti pada Gambar 4.8.



(a) (b)

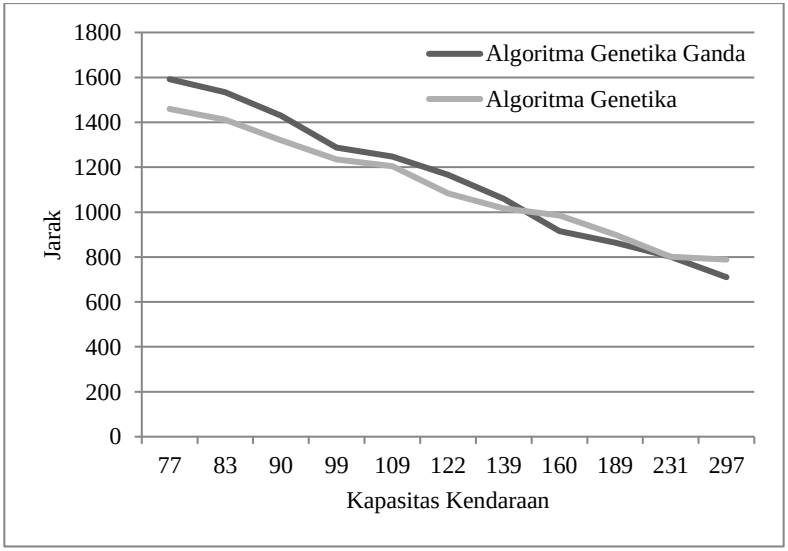
**Gambar 4.8** Solusi Rute untuk CVRP dengan Menggunakan (a) Algoritma Genetika dan (b) Algoritma Genetika Ganda

## 4.5 Pembandingan Algoritma Genetika dengan Algoritma Genetika Ganda

Pembandingan algoritma genetika dengan algoritma genetika ganda dilakukan dengan memanfaatkan permasalahan P50, P75, P100, P125 dan program yang telah dibuat. Perbandingan antara algoritma genetika dan algoritma genetika ganda akan dilihat dari tiga segi yang berbeda yaitu dari segi jarak, waktu komputasi, dan total generasi yang dibutuhkan untuk menyelesaikan CVRP. Dalam segi jarak, digunakan $P = \frac{jarak1 - jarak2}{jarak1} \times 100$ dimana $jarak1$ adalah jarak yang diperoleh algoritma genetika dan $jarak2$ adalah jarak yang diperoleh algoritma genetika ganda. Nilai $P$ tersebut mempunyai arti berapa persen algoritma genetika ganda lebih baik atau lebih buruk dari algoritma genetika. Generasi dari algoritma genetika ganda adalah total generasi yang digunakan oleh AG1 dan semua AG2.

### 4.5.1 Pembandingan Algoritma pada Permasalahan P50

Untuk pembandingan algoritma genetika dengan algoritma genetika ganda pada permasalahan P50, digunakan empat belas variasi kapasitas kendaraan yang berbeda. Kapasitas kendaraan yang digunakan antara lain adalah 77, 83, 90, 109, 122, 139, 160, 189, 231, dan 297. Kapasitas-kapasitas tersebut dipilih agar jumlah kendaraan minimal yang dapat digunakan untuk melayani semua titik tujuan bervariasi mulai dari 4 hingga 14 kendaraan. Sebagai contoh, apabila kapasitas kendaraan yang digunakan adalah 77 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{1040}{77} \right\rceil = \lceil 13{,}507 \rceil = 14$, dan apabila kapasitas kendaraan yang digunakan adalah 297 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{1040}{297} \right\rceil = \lceil 3{,}502 \rceil = 4$. Untuk setiap kapasitas kendaraan yang berbeda, dilakukan tiga kali pengujian dan dipilih satu yang terbaik. Hasil yang didapat dari algoritma genetika dan algoritma ganda disajikan dalam Gambar 4.9 dan Tabel 4.1.




**Gambar 4.9** Perbandingan Jarak untuk Permasalahan P50

**Tabel 4.1** Perbandingan untuk Permasalahan P50

| KK | Algoritma Genetika Ganda | | | Algoritma Genetika | | | *P* |
|---|---|---|---|---|---|---|---|
| | Jarak | Waktu | Gen | Jarak | Waktu | Gen | |
| 77 | 1591.732 | **10** | 7956 | **1459.428** | 25 | **4420** | -9.06 |
| 83 | 1533.234 | **10** | 7938 | **1411.733** | 31 | **5504** | -8.60 |
| 90 | 1429.179 | **10** | 7522 | **1319.67** | 38 | **6633** | -8.29 |
| 99 | 1287.732 | **7** | 6299 | **1234.57** | 24 | **4314** | -4.30 |
| 109 | 1247.899 | **5** | 4546 | **1205.129** | 13 | **3034** | -3.54 |
| 122 | 1166.375 | **6** | **4612** | **1083.415** | 29 | 5834 | -7.65 |
| 139 | 1059.293 | **5** | **2366** | **1015.836** | 64 | 11541 | -4.27 |
| 160 | **915.0866** | **4** | **3715** | 985.2263 | 28 | 4662 | 7.11 |
| 189 | **863.5713** | **3** | **2345** | 899.1628 | 18 | 4338 | 3.95 |
| 231 | 801.5071 | **3** | **2428** | **801.136** | 22 | 5038 | -0.04 |
| 297 | **710.1832** | **3** | **2514** | 788.5825 | 44 | 8489 | 9.94 |

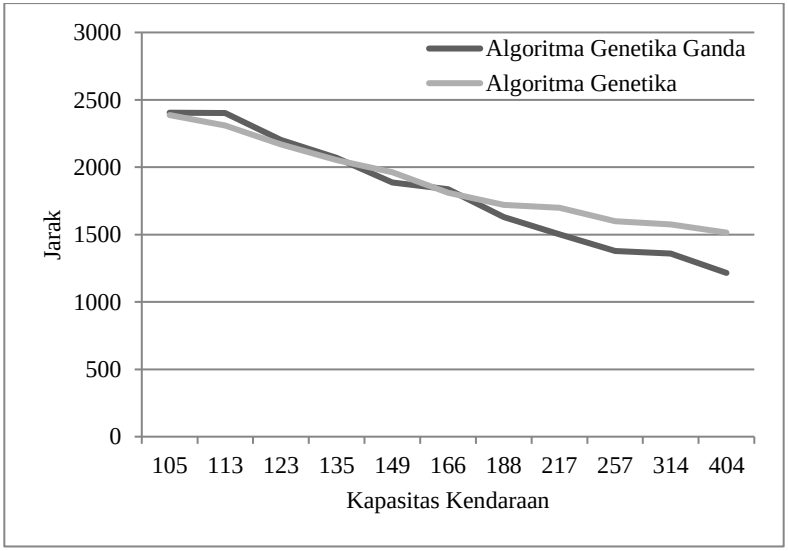
KK – Kapasitas kendaraan, Gen – Generasi

Dari segi jarak, dapat dilihat bahwa algoritma genetika ganda masih kalah dari algoritma genetika untuk kapasitas kendaraan 77, 83, 90, 99, 109, 122, 139, dan 231. Namun untuk kapasitas kendaraan 160, 189, dan 297, algoritma genetika ganda memberikan jarak yang lebih baik. Dapat dilihat dari Gambar 4.9 bahwa seiring dengan bertambahnya kapasitas kendaraan, jarak yang diperoleh algoritma genetika ganda turun lebih cepat dari jarak yang diperoleh algoritma genetika. Hal ini sesuai dengan yang ada pada Tabel 4.1 bahwa seiring dengan bertambahnya kapasitas kendaraan, nilai *P* juga ikut bertambah kecuali pada beberapa kasus tertentu. Hal ini berarti bahwa seiring dengan bertambahnya kapasitas kendaraan, algoritma genetika ganda menjadi semakin baik dari algoritma genetika.

Dari segi waktu komputasi, rata-rata waktu komputasi algoritma genetika ganda adalah 6,7 detik sedangkan untuk algoritma genetika adalah 30,0 detik. Dari segi generasi, rata-rata total generasi algoritma genetika ganda adalah 5426 sedangkan untuk algoritma genetika adalah 5764. Jadi, waktu komputasi dan generasi algoritma genetika ganda lebih baik dari algoritma genetika.

### 4.5.2 Pembandingan Algoritma pada Permasalahan P75

Untuk pembandingan algoritma genetika dengan algoritma genetika ganda pada permasalahan P75, digunakan empat belas variasi kapasitas kendaraan yang berbeda. Kapasitas kendaraan yang digunakan adalah 105, 113, 123, 135, 149, 166, 188, 217, 257, 314, dan 404. Kapasitas-kapasitas tersebut dipilih agar jumlah kendaraan minimal yang dapat digunakan untuk melayani semua titik tujuan bervariasi mulai dari 4 hingga 14 kendaraan. Sebagai contoh, apabila kapasitas kendaraan yang digunakan adalah 105 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{1413}{105} \right\rceil = \lceil 13{,}457 \rceil = 14$, dan apabila kapasitas kendaraan yang digunakan adalah 404 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{1413}{404} \right\rceil = \lceil 3{,}498 \rceil = 4$. Untuk setiap kapasitas kendaraan yang berbeda, dilakukan tiga kali pengujian dan dipilih satu yang terbaik. Hasil yang didapat dari algoritma genetika dan algoritma genetika ganda disajikan dalam Gambar 4.10 dan Tabel 4.2.




**Gambar 4.10** Perbandingan Jarak untuk Permasalahan P75

**Tabel 4.2** Perbandingan untuk Permasalahan P75

| KK | Algoritma Genetika Ganda | | | Algoritma Genetika | | | *P* |
|---|---|---|---|---|---|---|---|
| | Jarak | Waktu | Gen | Jarak | Waktu | Gen | |
| 105 | 2404.516 | **9** | **6183** | **2386.744** | 85 | 10739 | -0.74 |
| 113 | 2401.982 | **6** | **4767** | **2309.077** | 104 | 13173 | -4.02 |
| 123 | 2203.066 | **9** | **5955** | **2170.193** | 65 | 8807 | -1.51 |
| 135 | 2068.197 | **7** | **4757** | **2053.659** | 115 | 14277 | -0.70 |
| 149 | **1885.894** | **5** | **3336** | 1961.569 | 55 | 7242 | 3.85 |
| 166 | 1835.419 | **6** | **3982** | **1811.217** | 136 | 18191 | -1.33 |
| 188 | **1630.429** | **4** | **2604** | 1719.833 | 85 | 10677 | 5.19 |
| 217 | **1501.133** | **5** | **2829** | 1698.279 | 81 | 9711 | 11.6 |
| 257 | **1377.45** | **5** | **3171** | 1598.278 | 98 | 12585 | 13.8 |
| 314 | **1359.159** | **5** | **2815** | 1574.126 | 95 | 12278 | 13.6 |
| 404 | **1215.419** | **6** | **2884** | 1514.616 | 116 | 15220 | 19.7 |

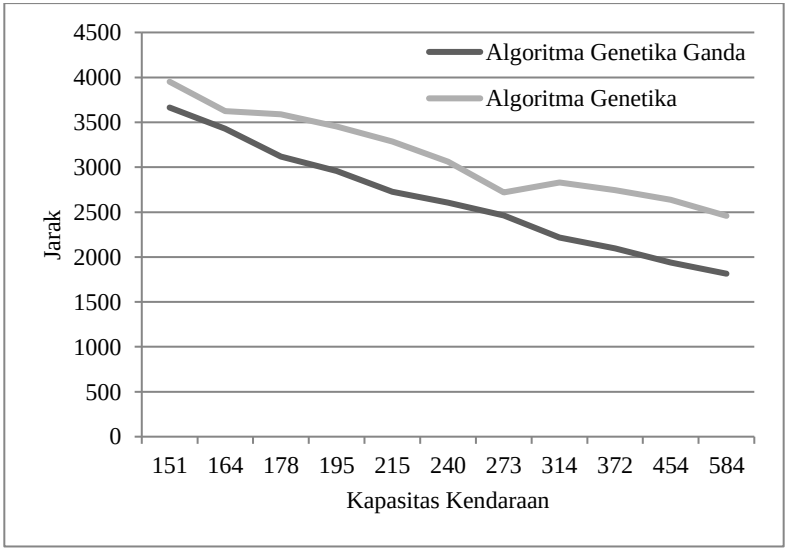
KK – Kapasitas kendaraan, Gen – Generasi

Dari segi jarak yang diperoleh, dapat dilihat bahwa algoritma genetika ganda masih kalah dari algoritma genetika untuk kapasitas kendaraan 105, 113, 123, 135, dan 166. Namun untuk kapasitas kendaraan 149, 188, 217, 257, 314, dan 404, algoritma genetika ganda memberikan jarak yang lebih baik. Dapat dilihat dari Gambar 4.10 bahwa seiring dengan bertambahnya kapasitas kendaraan, jarak yang diperoleh algoritma genetika ganda turun lebih cepat dari jarak yang diperoleh algoritma genetika. Hal ini sesuai dengan yang ada pada Tabel 4.2 bahwa seiring dengan bertambahnya kapasitas kendaraan, nilai *P* juga ikut bertambah kecuali pada beberapa kasus tertentu. Hal ini berarti bahwa seiring dengan bertambahnya kapasitas kendaraan, algoritma genetika ganda menjadi semakin baik dari algoritma genetika.

Dari segi waktu komputasi, rata-rata waktu komputasi algoritma genetika ganda adalah 7,3 detik sedangkan untuk algoritma genetika adalah 91,1 detik. Dari segi generasi, rata-rata total generasi algoritma genetika ganda adalah 4452 sedangkan untuk algoritma genetika adalah 11749. Jadi, waktu komputasi dan generasi algoritma genetika ganda lebih baik dari algoritma genetika.

### 4.5.3 Pembandingan Algoritma pada Permasalahan P100

Untuk pembandingan algoritma genetika dengan algoritma genetika ganda pada permasalahan P100, digunakan empat belas variasi kapasitas kendaraan yang berbeda. Kapasitas kendaraan yang digunakan adalah 151, 164, 178, 195, 215, 240, 273, 314, 372, 454, dan 584. Kapasitas-kapasitas tersebut dipilih agar jumlah kendaraan minimal yang dapat digunakan untuk melayani semua titik tujuan bervariasi mulai dari 4 hingga 14 kendaraan. Sebagai contoh, apabila kapasitas kendaraan yang digunakan adalah 151 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{2044}{151} \right\rceil = \lceil 13{,}536 \rceil = 14$, dan apabila kapasitas kendaraan yang digunakan adalah 584 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{2044}{584} \right\rceil = \lceil 3{,}5 \rceil = 4$. Untuk setiap kapasitas kendaraan yang berbeda, dilakukan tiga kali pengujian dan dipilih satu yang terbaik. Hasil yang didapat dari algoritma genetika dan algoritma genetika ganda disajikan dalam Gambar 4.11 dan Tabel 4.3.




**Gambar 4.11** Perbandingan Jarak untuk Permasalahan P100

**Tabel 4.3** Perbandingan untuk Permasalahan P100

| KK | Algoritma Genetika Ganda | | | Algoritma Genetika | | | *P* |
|---|---|---|---|---|---|---|---|
| | Jarak | Waktu | Gen | Jarak | Waktu | Gen | |
| 151 | **3664.148** | **10** | **6176** | 3951.525 | 123 | 10961 | 7,27 |
| 164 | **3428.708** | **11** | **6267** | 3622.753 | 153 | 13819 | 5,35 |
| 178 | **3119.468** | **9** | **6137** | 3588.194 | 125 | 10521 | 13,0 |
| 195 | **2958.256** | **5** | **2883** | 3454.294 | 166 | 13524 | 14,3 |
| 215 | **2726.513** | **6** | **3908** | 3287.016 | 293 | 26427 | 17,0 |
| 240 | **2604.126** | **5** | **2775** | 3060.496 | 231 | 18616 | 14,9 |
| 273 | **2463.602** | **6** | **2853** | 2717.931 | 171 | 13382 | 9,35 |
| 314 | **2216.962** | **7** | **3742** | 2829.297 | 256 | 22065 | 21,6 |
| 372 | **2094.614** | **8** | **3995** | 2742.865 | 164 | 14749 | 23,6 |
| 454 | **1939.12** | **13** | **4589** | 2638.619 | 148 | 10825 | 26,5 |
| 584 | **1814.5** | **17** | **3398** | 2458.27 | 232 | 20722 | 26,1 |

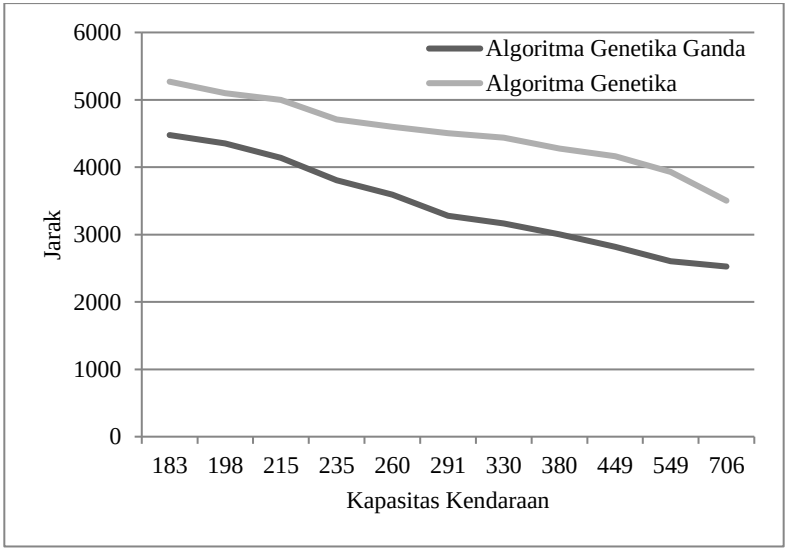
KK – Kapasitas kendaraan, Gen – Generasi

Dari segi jarak yang diperoleh, dapat dilihat bahwa algoritma genetika ganda memberikan hasil yang lebih baik dari algoritma genetika untuk semua variasi kapasitas kendaraan yang digunakan. Dapat dilihat dari Gambar 4.11 bahwa seiring dengan bertambahnya kapasitas kendaraan, jarak yang diperoleh algoritma genetika ganda turun lebih cepat dari jarak yang diperoleh algoritma genetika. Hal ini sesuai dengan yang ada pada Tabel 4.3 bahwa seiring dengan bertambahnya kapasitas kendaraan, nilai *P* juga ikut bertambah kecuali pada beberapa kasus tertentu. Hal ini berarti bahwa seiring dengan bertambahnya kapasitas kendaraan, algoritma genetika ganda menjadi semakin baik dari algoritma genetika.

Dari segi waktu komputasi, rata-rata waktu komputasi algoritma genetika ganda adalah 10,4 detik sedangkan untuk algoritma genetika adalah 171,8 detik. Dari segi generasi, rata-rata total generasi algoritma genetika ganda adalah 5143 sedangkan untuk algoritma genetika adalah 14589.. Jadi, waktu komputasi dan generasi algoritma genetika ganda lebih baik dari algoritma genetika.

### 4.5.4 Pembandingan Algoritma pada Permasalahan P125

Untuk pembandingan algoritma genetika dengan algoritma genetika ganda pada permasalahan P125, digunakan empat belas variasi kapasitas kendaraan yang berbeda. Kapasitas kendaraan yang digunakan adalah 183, 198, 215, 235, 260, 291, 330, 380, 449, 549, dan 706. Kapasitas-kapasitas tersebut dipilih agar jumlah kendaraan minimal yang dapat digunakan untuk melayani semua titik tujuan bervariasi mulai dari 4 hingga 14 kendaraan. Sebagai contoh, apabila kapasitas kendaraan yang digunakan adalah 183 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{2044}{183} \right\rceil = \lceil 13{,}536 \rceil = 14$, dan apabila kapasitas kendaraan yang digunakan adalah 706 maka jumlah kendaraan minimal yang dapat digunakan adalah $\left\lceil \frac{2044}{706} \right\rceil = \lceil 3{,}5 \rceil = 4$. Untuk setiap kapasitas kendaraan yang berbeda, dilakukan tiga kali pengujian dan dipilih satu yang terbaik. Hasil yang didapat dari algoritma genetika dan algoritma ganda disajikan dalam Gambar 4.12 dan Tabel 4.4.




**Gambar 4.12** Perbandingan Jarak untuk Permasalahan P125

**Tabel 4.4** Perbandingan untuk Permasalahan P125

| KK | Algoritma Genetika Ganda | | | Algoritma Genetika | | | *P* |
|---|---|---|---|---|---|---|---|
| | Jarak | Waktu | Gen | Jarak | Waktu | Gen | |
| 183 | **4476.805** | **24** | **11107** | 5269.657 | 364 | 22374 | 15,0 |
| 198 | **4352.449** | **10** | **6188** | 5100.448 | 435 | 24877 | 14,6 |
| 215 | **4141.08** | **14** | **7145** | 4996.808 | 298 | 16720 | 17,1 |
| 235 | **3806.193** | **7** | **3790** | 4706.922 | 444 | 26820 | 19,1 |
| 260 | **3594.609** | **7** | **3166** | 4597.851 | 507 | 30608 | 21,8 |
| 291 | **3280.79** | **8** | **3955** | 4505.065 | 372 | 16397 | 27,1 |
| 330 | **3165.287** | **12** | **5258** | 4440.228 | 643 | 30827 | 28,7 |
| 380 | **3003.622** | **12** | **5045** | 4275.217 | 499 | 28333 | 29,7 |
| 449 | **2817.172** | **17** | **3627** | 4164.521 | 407 | 23457 | 32,3 |
| 549 | **2603.215** | **20** | **3505** | 3928.582 | 514 | 28909 | 33,7 |
| 706 | **2525.264** | **47** | **4063** | 3502.791 | 522 | 30022 | 27,9 |

KK – Kapasitas kendaraan, Gen – Generasi

Dari segi jarak yang diperoleh, dapat dilihat bahwa algoritma genetika ganda memberikan hasil yang lebih baik dari algoritma genetika untuk semua variasi kapasitas kendaraan yang digunakan. Dapat dilihat dari Gambar 4.12 bahwa seiring dengan bertambahnya kapasitas kendaraan, jarak yang diperoleh algoritma genetika ganda turun lebih cepat dari jarak yang diperoleh algoritma genetika. Hal ini sesuai dengan yang ada pada Tabel 4.4 bahwa seiring dengan bertambahnya kapasitas kendaraan, nilai *P* juga ikut bertambah kecuali pada beberapa kasus tertentu. Hal ini berarti bahwa seiring dengan bertambahnya kapasitas kendaraan, algoritma genetika ganda menjadi semakin baik dari algoritma genetika.

Dari segi waktu komputasi, rata-rata waktu komputasi algoritma genetika ganda adalah 15,6 detik sedangkan untuk algoritma genetika adalah 403,7 detik. Dari segi generasi, rata-rata total generasi algoritma genetika ganda adalah 5314 sedangkan untuk algoritma genetika adalah 22254. Jadi, waktu komputasi dan generasi algoritma genetika ganda lebih baik dari algoritma genetika.

# BAB V
# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Berdasarkan pada hasil dan pembahasan yang telah dipaparkan, dapat diambil kesimpulan:

1. Algoritma genetika ganda dapat digunakan untuk menyelesaikan CVRP dengan cara yang berbeda dari algoritma genetika.
2. Dari segi jarak, algoritma genetika ganda lebih baik dari algoritma genetika kecuali untuk beberapa kapasitas kendaraan yang kecil pada permasalahan P50 dan P75. Untuk kapasitas kendaraan yang semakin besar, nilai *P* juga semakin besar yang berarti bahwa penambahan kapasitas kendaraan mengakibatkan algoritma genetika ganda menjadi jauh lebih baik dibandingkan dengan algoritma genetika.
3. Rata-rata waktu komputasi algoritma genetika ganda untuk permasalahan P50, P75, P100, dan P125 tidak melebihi 20 detik sedangkan untuk algoritma genetika bervariasi mulai dari kisaran 30 detik hingga 400 detik. Sehingga dari segi waktu komputasi, algoritma genetika ganda jauh lebih baik dari algoritma genetika.
4. Rata-rata generasi algoritma genetika ganda untuk permasalahan P50, P75, P100, dan P125 berada di kisaran 5000 generasi sedangkan untuk algoritma genetika bervariasi mulai dari kisaran 5000 generasi hingga 20000 generasi. Sehingga dari segi generasi, algoritma genetika ganda lebih baik dari algoritma genetika.

## 5.2 Saran

Saran yang dapat diberikan oleh penulis adalah sebagai berikut:

1. Algoritma genetika ganda dapat digunakan untuk menyelesaikan CVRP di dunia nyata.

2. Algoritma genetika ganda dapat dikembangkan oleh peneliti lain agar dapat digunakan untuk menyelesaikan permasalahan yang lain.

# DAFTAR PUSTAKA


Ahmed, Z.H. 2005. "*Genetic Algorithm for Travelling Salesman Problem using Sequential Constructive Crossover Operator*". **International Journal of Biometrics & Bioinformatics** 3, 96-105.

Coley, D.A. 1999. **An Indtroduction to Genetic Algorithms for Scientists and Engineers**. New Jersey: World Scientific.

Eiben, A.E. dan Smith, J.E. 2003. **Introduction to Evolutionary Computing**. Berlin: Springer.

Hermawan, A. 2012. "Penentuan Rute yang Optimal pada Kegiatan Distribusi BBM Menggunakan *Ant Colony System*". **Surabaya: Tugas Akhir, Jurusan Matematika, Institut Teknologi Sepuluh Nopember**.

Hozairi, Buda, K., Masroeri, dan Irawan, M.I. 2014. "*Implementation of Nondominated Sorting Genetic Algorithm* – II (NSGA-II) *for Multiobjective Optimization Problems on Distribution of Indonesian Navy Warship*". **Journal of Theoretical and Applied Information Technology** 64, 274-281.

Karakatic, S. dan Podgorelec, V. 2015. "*A Suvey of Genetic Algorithms for Solving Multi Depot Vehicle Routing Problem*". **Applied Soft Computing** 27, 519-532.

Nazif, H. dan Lee, L.S. 2012. "*Optimised Crossover Genetic Algorithm for Capacitated Vehicle Routing Problem*". **Applied Mathematical Modelling** 36, 2110-2117.

Pramsistya, Y. 2010. "Optimasi Penempatan BTS dengan Menggunakan Algoritma Genetika". **Surabaya: Tugas Akhir, Jurusan Matematika, Institut Teknologi Sepuluh Nopember**.

Soelistyowati, R. 2010. "Pendekatan Algoritma Genetika untuk Menyelesaikan Masalah Transportasi Nonlinier". **Surabaya: Tugas Akhir, Jurusan Matematika, Institut Teknologi Sepuluh Nopember.**

Taillard, E. 1993. "*Parallel Iterative Search Methods for Vehicle Routing Problem*". **Network** 23, 661-673.

Toth, P. dan Vigo, D. 2002. **The Vehicle Routing Problem**. Philadelpia: University City Science Center.

Yucenur, G.N. dan Demirel, N.C. 2011. "*A New Geometric Shape-Based Genetic Clustering Algorithm for The Multi-Depot Vehicle Routing Problem*". **Expert System with Applications** 38, 11859-11865.

# LAMPIRAN

## Lampiran 1

Absis, Ordinat, dan Permintaan dari Titik-Titik Tujuan untuk Permasalahan P50

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 1 | 50 | 19 | 54 | 36 | 12 |
| 3 | 81 | 11 | 54 | 45 | 24 |
| 3 | 4 | 16 | 57 | 99 | 27 |
| 9 | 3 | 25 | 59 | 72 | 30 |
| 9 | 32 | 29 | 60 | 21 | 18 |
| 13 | 99 | 13 | 60 | 14 | 18 |
| 14 | 21 | 26 | 62 | 95 | 16 |
| 24 | 9 | 14 | 62 | 31 | 28 |
| 25 | 82 | 16 | 63 | 20 | 29 |
| 29 | 10 | 12 | 65 | 11 | 17 |
| 30 | 33 | 29 | 70 | 2 | 28 |
| 30 | 50 | 26 | 70 | 30 | 26 |
| 32 | 52 | 18 | 73 | 86 | 10 |
| 32 | 5 | 27 | 74 | 17 | 27 |
| 34 | 27 | 17 | 75 | 88 | 25 |
| 35 | 91 | 15 | 76 | 27 | 27 |
| 35 | 67 | 28 | 77 | 51 | 17 |
| 37 | 80 | 29 | 80 | 44 | 19 |
| 37 | 87 | 13 | 89 | 87 | 13 |
| 38 | 49 | 13 | 94 | 86 | 17 |
| 39 | 42 | 22 | 97 | 81 | 26 |
| 40 | 11 | 27 | 97 | 59 | 25 |
| 45 | 47 | 19 | 100 | 52 | 24 |
| 47 | 36 | 20 | | | |
| 50 | 83 | 10 | | | |
| 51 | 94 | 24 | | | |
| 53 | 82 | 19 | | | |

**Lampiran 2**

Absis, Ordinat, dan Permintaan dari Titik-Titik Tujuan untuk Permasalahan P75

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 1 | 19 | 24 | 50 | 64 | 10 |
| 2 | 2 | 11 | 51 | 79 | 17 |
| 4 | 120 | 29 | 52 | 17 | 18 |
| 4 | 141 | 29 | 56 | 91 | 17 |
| 7 | 117 | 13 | 56 | 18 | 25 |
| 8 | 138 | 16 | 58 | 40 | 24 |
| 10 | 133 | 14 | 63 | 91 | 27 |
| 11 | 65 | 17 | 65 | 112 | 27 |
| 12 | 92 | 17 | 65 | 55 | 16 |
| 22 | 87 | 25 | 67 | 35 | 14 |
| 24 | 17 | 25 | 68 | 122 | 30 |
| 24 | 149 | 28 | 72 | 82 | 13 |
| 25 | 122 | 11 | 74 | 30 | 24 |
| 25 | 101 | 11 | 74 | 64 | 16 |
| 29 | 7 | 11 | 74 | 113 | 10 |
| 30 | 140 | 10 | 77 | 103 | 21 |
| 33 | 87 | 15 | 78 | 85 | 14 |
| 35 | 108 | 18 | 79 | 28 | 27 |
| 37 | 138 | 14 | 79 | 65 | 26 |
| 38 | 57 | 22 | 82 | 59 | 14 |
| 38 | 145 | 12 | 83 | 102 | 10 |
| 39 | 82 | 29 | 86 | 77 | 18 |
| 39 | 49 | 24 | 88 | 63 | 29 |
| 39 | 144 | 20 | 90 | 18 | 14 |
| 42 | 128 | 27 | 93 | 20 | 24 |
| 44 | 73 | 29 | 94 | 101 | 12 |
| 48 | 21 | 24 | 94 | 77 | 14 |
| 50 | 76 | 27 | 99 | 41 | 19 |
| 50 | 108 | 12 | 103 | 33 | 13 |
| 50 | 129 | 24 | 110 | 73 | 13 |

| Absis | Ordinat | Permintaan |
|---|---|---|
| 114 | 33 | 21 |
| 115 | 85 | 18 |
| 117 | 85 | 11 |
| 118 | 37 | 15 |
| 120 | 61 | 30 |
| 126 | 1 | 23 |
| 127 | 63 | 11 |
| 128 | 31 | 13 |
| 134 | 2 | 16 |
| 134 | 84 | 10 |
| 138 | 53 | 14 |
| 142 | 115 | 24 |
| 149 | 11 | 26 |
| 149 | 143 | 18 |
| 149 | 28 | 19 |

**Lampiran 3**

Absis, Ordinat, dan Permintaan dari Titik-Titik Tujuan untuk Permasalahan P100

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 0 | 30 | 14 | 62 | 60 | 11 |
| 2 | 31 | 19 | 65 | 173 | 30 |
| 2 | 165 | 23 | 68 | 26 | 30 |
| 3 | 46 | 11 | 68 | 48 | 15 |
| 5 | 155 | 22 | 70 | 32 | 25 |
| 6 | 102 | 22 | 70 | 182 | 13 |
| 8 | 163 | 24 | 72 | 130 | 24 |
| 8 | 101 | 27 | 75 | 130 | 17 |
| 15 | 119 | 29 | 76 | 130 | 24 |
| 21 | 121 | 28 | 79 | 7 | 10 |
| 22 | 103 | 18 | 79 | 11 | 12 |
| 24 | 187 | 27 | 83 | 11 | 16 |
| 28 | 172 | 22 | 85 | 122 | 25 |
| 29 | 7 | 28 | 86 | 35 | 18 |
| 30 | 194 | 15 | 87 | 161 | 25 |
| 32 | 2 | 27 | 90 | 103 | 11 |
| 32 | 61 | 16 | 92 | 31 | 11 |
| 34 | 57 | 13 | 92 | 13 | 24 |
| 36 | 73 | 18 | 93 | 96 | 29 |
| 38 | 179 | 23 | 101 | 150 | 25 |
| 42 | 141 | 17 | 101 | 15 | 15 |
| 44 | 127 | 30 | 103 | 83 | 25 |
| 49 | 24 | 15 | 103 | 74 | 25 |
| 50 | 124 | 11 | 105 | 74 | 13 |
| 51 | 106 | 16 | 105 | 200 | 23 |
| 52 | 14 | 17 | 106 | 181 | 12 |
| 53 | 147 | 21 | 108 | 97 | 12 |
| 53 | 111 | 20 | 109 | 2 | 20 |
| 58 | 139 | 25 | 117 | 6 | 11 |
| 62 | 36 | 13 | 117 | 58 | 25 |

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 117 | 73 | 28 | 187 | 11 | 17 |
| 119 | 120 | 19 | 193 | 123 | 20 |
| 125 | 167 | 10 | 194 | 171 | 30 |
| 125 | 37 | 24 | 195 | 16 | 25 |
| 128 | 128 | 25 | 195 | 60 | 17 |
| 131 | 190 | 15 | 197 | 51 | 28 |
| 135 | 178 | 11 | 199 | 122 | 18 |
| 147 | 107 | 26 | | | |
| 148 | 167 | 21 | | | |
| 150 | 162 | 24 | | | |
| 150 | 77 | 30 | | | |
| 151 | 0 | 12 | | | |
| 151 | 151 | 23 | | | |
| 152 | 21 | 25 | | | |
| 153 | 100 | 12 | | | |
| 158 | 0 | 10 | | | |
| 159 | 199 | 30 | | | |
| 162 | 93 | 29 | | | |
| 166 | 148 | 15 | | | |
| 167 | 193 | 26 | | | |
| 168 | 27 | 23 | | | |
| 168 | 37 | 17 | | | |
| 171 | 62 | 26 | | | |
| 171 | 33 | 27 | | | |
| 171 | 85 | 27 | | | |
| 173 | 129 | 16 | | | |
| 173 | 55 | 16 | | | |
| 175 | 45 | 19 | | | |
| 176 | 23 | 25 | | | |
| 177 | 9 | 26 | | | |
| 179 | 123 | 25 | | | |
| 184 | 46 | 26 | | | |
| 186 | 66 | 14 | | | |

**Lampiran 4**

Absis, Ordinat, dan Permintaan dari Titik-Titik Tujuan untuk Permasalahan P125

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 4 | 152 | 15 | 49 | 116 | 10 |
| 4 | 12 | 20 | 51 | 201 | 23 |
| 5 | 179 | 24 | 52 | 81 | 16 |
| 8 | 225 | 18 | 52 | 60 | 21 |
| 9 | 6 | 10 | 52 | 149 | 17 |
| 10 | 246 | 15 | 56 | 102 | 29 |
| 10 | 135 | 28 | 61 | 71 | 29 |
| 14 | 78 | 22 | 63 | 77 | 15 |
| 14 | 119 | 11 | 65 | 32 | 11 |
| 15 | 103 | 22 | 67 | 246 | 15 |
| 16 | 34 | 13 | 67 | 238 | 28 |
| 17 | 119 | 11 | 72 | 140 | 14 |
| 19 | 139 | 25 | 73 | 177 | 11 |
| 20 | 192 | 11 | 76 | 74 | 23 |
| 22 | 85 | 29 | 77 | 19 | 10 |
| 27 | 7 | 30 | 83 | 184 | 20 |
| 30 | 142 | 17 | 86 | 124 | 16 |
| 31 | 240 | 10 | 86 | 96 | 29 |
| 32 | 240 | 15 | 87 | 233 | 25 |
| 32 | 246 | 14 | 87 | 97 | 25 |
| 33 | 217 | 10 | 87 | 127 | 27 |
| 38 | 226 | 16 | 88 | 153 | 13 |
| 41 | 165 | 16 | 89 | 1 | 15 |
| 43 | 108 | 30 | 93 | 174 | 15 |
| 43 | 106 | 22 | 93 | 182 | 16 |
| 43 | 165 | 18 | 94 | 233 | 13 |
| 45 | 194 | 19 | 94 | 145 | 29 |
| 47 | 112 | 26 | 95 | 30 | 17 |
| 48 | 74 | 29 | 96 | 14 | 28 |
| 49 | 230 | 15 | 99 | 147 | 15 |

| Absis | Ordinat | Permintaan | Absis | Ordinat | Permintaan |
|---|---|---|---|---|---|
| 102 | 108 | 26 | 194 | 38 | 21 |
| 103 | 15 | 21 | 195 | 123 | 25 |
| 103 | 43 | 13 | 197 | 33 | 24 |
| 107 | 132 | 28 | 197 | 187 | 28 |
| 112 | 136 | 12 | 199 | 47 | 30 |
| 112 | 19 | 23 | 200 | 148 | 22 |
| 115 | 236 | 17 | 204 | 179 | 13 |
| 117 | 35 | 25 | 205 | 6 | 19 |
| 124 | 195 | 19 | 211 | 124 | 21 |
| 129 | 7 | 19 | 215 | 138 | 11 |
| 132 | 48 | 19 | 216 | 77 | 24 |
| 135 | 83 | 18 | 217 | 66 | 29 |
| 136 | 153 | 23 | 217 | 28 | 20 |
| 143 | 139 | 13 | 218 | 237 | 13 |
| 144 | 126 | 16 | 219 | 140 | 29 |
| 147 | 112 | 11 | 221 | 116 | 22 |
| 152 | 44 | 30 | 221 | 212 | 28 |
| 154 | 237 | 14 | 221 | 120 | 28 |
| 155 | 212 | 18 | 228 | 209 | 10 |
| 159 | 19 | 30 | 228 | 181 | 19 |
| 159 | 111 | 26 | 229 | 25 | 21 |
| 161 | 220 | 22 | 231 | 52 | 20 |
| 162 | 24 | 16 | 234 | 142 | 12 |
| 164 | 108 | 22 | 235 | 197 | 16 |
| 167 | 180 | 24 | 235 | 132 | 14 |
| 170 | 150 | 29 | 241 | 237 | 25 |
| 174 | 0 | 16 | 241 | 177 | 30 |
| 175 | 198 | 13 | 243 | 166 | 24 |
| 178 | 18 | 24 | 244 | 77 | 22 |
| 178 | 114 | 14 | 244 | 83 | 11 |
| 183 | 4 | 16 | 247 | 155 | 23 |
| 190 | 132 | 13 | 247 | 242 | 30 |
| 190 | 167 | 22 | | | |

## Lampiran 5

*Source Code* dari CVRP.java

```
package cvrp;

public class CVRP {

    public static void main(String[] args) {
        Simulation simulation = new Simulation();
        simulation.setVisible(true);
    }
}
```

## Lampiran 6

*Source Code* dari City.java

```
package cvrp;

public class City {
    int x;
    int y;
    int d;

    public City(){
        this.x = (int)(Math.random()*200);
        this.y = (int)(Math.random()*200);
        this.d = (int)(Math.random()*20);
    }

    public City(int x, int y, int d){
        this.x = x;
        this.y = y;
        this.d = d;
    }

    public void setCity(int x, int y, int d){
        this.x = x;
        this.y = y;
        this.d = d;
    }

    public int getX(){
        return this.x;
    }

    public int getY(){
        return this.y;
    }

    public int getD(){
        return this.d;
    }

    public double distanceTo(City city){
        int xx = Math.abs(getX() - city.getX());
        int yy = Math.abs(getY() - city.getY());
```

```
        double distance = Math.sqrt((xx*xx)+(yy*yy));
        return distance;
    }

    @Override
    public String toString(){
        return getX()+","+getY()+","+getD();
    }
}
```

## Lampiran 7

*Source Code* dari TourManager.java

```
package cvrp;

import java.util.ArrayList;

public class TourManager {
   private static ArrayList destinationCities = new ArrayList<City>();
   private static City depot;

   public static void setDepot(City city) {
      depot = new City(city.getX(), city.getY(), city.getD());
   }

   public static City getDepot() {
      return depot;
   }

   public static void addCity(City city) {
      destinationCities.add(city);
   }

   public static City getCity(int index){
      return (City)destinationCities.get(index);
   }

   public static int numberOfCities(){
      return destinationCities.size();
   }
}
```

## Lampiran 8

*Source Code* dari Tour.java

```
package cvrp;

import java.util.ArrayList;
import java.util.Collections;

public class Tour {
   private ArrayList tour = new ArrayList<City>();
   private double fitness = 0;
   private double distance = 0;

   public Tour(){
      for (int i = 0; i < TourManager.numberOfCities(); i++) {
         tour.add(null);
      }
   }

   public Tour(ArrayList tour){
      this.tour = tour;
   }

   public void generateIndividual() {
      int n = TourManager.numberOfCities();
      for (int i = 0; i < n; i++) {
         setCity(i, TourManager.getCity(i));
      }
      Collections.shuffle(tour);
   }

   public City getCity(int tourPosition) {
      return (City)tour.get(tourPosition);
   }

   public void setCity(int tourPosition, City city) {
      tour.set(tourPosition, city);
      fitness = 0;
      distance = 0;
   }

   public double getFitness() {
      if (fitness == 0) {
```

```
        fitness = 1/(double)getDistance();
    }
    return fitness;
}

public double getDistance(){
    if (distance == 0) {
        double tourDistance = 0;
        City depot = TourManager.getDepot();
        int kapasitas = depot.getD();
        tourDistance += depot.distanceTo(getCity(0));
        kapasitas -= getCity(0).getD();

        for (int cityIndex=0; cityIndex < tourSize(); cityIndex++) {
            City fromCity = getCity(cityIndex);
            City destinationCity;
            if(cityIndex+1 < tourSize()){
                destinationCity = getCity(cityIndex+1);
                if((kapasitas-destinationCity.getD()) < 0){
                    kapasitas = depot.getD();
                    tourDistance += fromCity.distanceTo(depot);
                    tourDistance += depot.distanceTo(destinationCity);
                    kapasitas -= destinationCity.getD();
                }
                else{
                    tourDistance += fromCity.distanceTo(destinationCity);
                    kapasitas -= destinationCity.getD();
                }
            }
            else{
                tourDistance += fromCity.distanceTo(depot);
            }
        }

        distance = tourDistance;
    }
    return distance;
}

public int tourSize() {
    return tour.size();
}
```

```
    public boolean containsCity(City city){
        return tour.contains(city);
    }

    @Override
    public String toString() {
        String geneString = "|";
        for (int i = 0; i < tourSize(); i++) {
            geneString += getCity(i)+"|";
        }
        return geneString;
    }
}
```

## Lampiran 9

*Source Code* dari Population.java

```
package cvrp;

public class Population {
   Tour[] tours;

   public Population(int populationSize, boolean bol) {
      tours = new Tour[populationSize];
      if (bol) {
         for (int i = 0; i < populationSize(); i++) {
            Tour newTour = new Tour();
            newTour.generateIndividual();
            saveTour(i, newTour);
         }
      }
   }

   public void saveTour(int index, Tour tour) {
      tours[index] = tour;
   }

   public Tour getTour(int index) {
      return tours[index];
   }

   public Tour getFittest() {
      Tour fittest = tours[0];
      for (int i = 1; i < populationSize(); i++) {
         if (fittest.getFitness() <= getTour(i).getFitness()) {
            fittest = getTour(i);
         }
      }
      return fittest;
   }

   public int populationSize() {
      return tours.length;
   }
}
```

## Lampiran 10

*Source Code* dari AG.java

```
package cvrp;

public class AG {
   private static final double mutationRate = 0.2;

   public static Population evolvePopulation(Population pop) {
      int a = 2*pop.populationSize();
      int b = pop.populationSize();

      Population newPopulation = new Population(pop.populationSize(), false);

      for (int i = 0; i < newPopulation.populationSize(); i++) {
         int random1 = (int) (Math.random() * pop.populationSize());
         int random2 = (int) (Math.random() * pop.populationSize());
         Tour parent1 = pop.getTour(random1);
         Tour parent2 = pop.getTour(random2);
         Tour child = crossover(parent1, parent2);
         newPopulation.saveTour(i, child);
      }

      Tour aa = new Tour ();
      aa = newPopulation.getFittest();

      for (int i = 0; i < newPopulation.populationSize(); i++) {
         if (newPopulation.getTour(i) != aa){
            mutate(newPopulation.getTour(i));
         }
      }

      Population newPopulation2 = new Population(a, false);

      for (int i = 0; i < pop.populationSize(); i++) {
         newPopulation2.saveTour(i, pop.getTour(i));
      }

      for (int i = 0; i < pop.populationSize(); i++) {
         newPopulation2.saveTour((b+i), newPopulation.getTour(i));
      }
```

```
    for (int i = 0; i < a; i++){
        int k = i;
        Tour fittest = newPopulation2.getTour(i);
        for (int j = i+1; j < a; j++) {
            if (fittest.getFitness() <= newPopulation2.getTour(j).getFitness()) {
                fittest = newPopulation2.getTour(j);
                k = j;
            }
        }
        newPopulation2.saveTour(k, newPopulation2.getTour(i));
        newPopulation2.saveTour(i, fittest);
    }

    Population newPopulation3 = new Population(b, false);
    int m = 0;
    newPopulation3.saveTour(0, newPopulation2.getTour(0));

    for (int i = 1; i < b; i++){
        if (newPopulation2.getTour(i).getDistance() !=
                                        newPopulation3.getTour(m).getDistance()){
            m++;
            newPopulation3.saveTour(m, newPopulation2.getTour(i));
        }
    }

    if ((m+1) != b){
        for (int i = m+1; i < b; i++){
            Tour acak = new Tour();
            acak.generateIndividual();
            newPopulation3.saveTour(i, acak);
        }
    }

    return newPopulation3;
}

public static Tour crossover(Tour p1, Tour p2) {
    Tour child = new Tour();
    int a = (int) (Math.random() * p1.tourSize());
    int b = (int) (Math.random() * p1.tourSize());

    for (int i = 0; i < child.tourSize(); i++) {
        if (a < b && i > a && i < b) {
```

```
            child.setCity(i, p1.getCity(i));
        }
        else if (a > b) {
            if (!(i < a && i > b)) {
                child.setCity(i, p1.getCity(i));
            }
        }
    }

    for (int i = 0; i < child.tourSize(); i++) {
        if (!child.containsCity(p2.getCity(i))) {
            for (int ii = 0; ii < child.tourSize(); ii++) {
                if (child.getCity(ii) == null) {
                    child.setCity(ii, p2.getCity(i));
                    break;
                }
            }
        }
    }

    return child;
}

private static void mutate(Tour tour) {
    double a =  Math.random();

    if(a < (mutationRate/2)){
        int r1 = (int) (tour.tourSize() * Math.random());
        int r2 = (int) (tour.tourSize() * Math.random());
        City city1 = tour.getCity(r1);
        City city2 = tour.getCity(r2);
        tour.setCity(r2, city1);
        tour.setCity(r1, city2);
    }

    else if (a < mutationRate){
        int r1 = (int) (tour.tourSize() * Math.random());
        int r2 = (int) (tour.tourSize() * Math.random());
        if (r2 > r1){
            int x = r2;
            r2 = r1;
            r1 = x;
        }
```

```
            int b = 0;
            City city = new City();
            for (int i = r1 ; i < r2 ; i++){
                city = tour.getCity((r1+b));
                tour.setCity((r1+b), tour.getCity((r2-b)));
                tour.setCity((r2-b), city);
                b++;
            }
        }
    }
}
```

## Lampiran 11

*Source Code* dari Location.java

```
package cvrp;

public class Location {
    private int x;
    private int y;
    private int demand;
    private double tetha;

    public Location(){
    }

    public Location(int x, int y){
        this.x = x;
        this.y = y;
    }

    public Location(int x, int y, int demand){
        this.x = x;
        this.y = y;
        this.demand = demand;
        double dummy1 = y-DataAG1.getDepot().getY();
        double dummy2 = x-DataAG1.getDepot().getX();
        tetha = Math.toDegrees(Math.atan2(dummy1, dummy2));
        if (tetha < 0)
            tetha = tetha + 360;
        tetha = 360 - tetha;
    }

    public void setLocation(int x, int y){
        this.x = x;
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
```

```
    public int getDemand(){
        return demand;
    }

    public double getTetha(){
        return tetha;
    }

    public double distanceTo(Location a){
        double dummy1 = Math.abs(getX() - a.getX());
        double dummy2 = Math.abs(getY() - a.getY());
        return Math.sqrt(dummy1*dummy1 + dummy2*dummy2);
    }

    @Override
    public String toString(){
        return "x : "+x+"\ty : "+y+"\tdemand : "+demand+"\ttetha : "+tetha;
    }
}
```

## Lampiran 12

*Source Code* dari DataAG1.java

```
package cvrp;

import java.util.ArrayList;

public class DataAG1 {
   private static int number_of_vehicles;
   private static int capacity_of_vehicles;
   private static double number;
   private static Location depot;
   private static ArrayList customers = new ArrayList<Location>();

   public static void addCustomer(Location a) {
      customers.add(a);
   }

   public static Location getCustomer(int a){
      return (Location)customers.get(a);
   }

   public static int numberOfCustomers(){
      return customers.size();
   }

   public static void setDepot(Location a) {
      depot = new Location(a.getX(), a.getY());
   }

   public static Location getDepot() {
      return depot;
   }

   public static void setNumber(double a) {
      number = a;
   }

   public static double getNumber() {
      return number;
   }
```

```
    public static void setCapacityOfVehicles(int a){
        capacity_of_vehicles = a;
    }

    public static int getCapacityOfVehicles() {
        return capacity_of_vehicles;
    }

    public static void setNumberOfVehicles(int a) {
        number_of_vehicles = a;
    }

    public static int numberOfVehicles(){
        return number_of_vehicles;
    }
}
```

## Lampiran 13

*Source Code* dari ChromosomeAG1.java

```
package cvrp;

public class ChromosomeAG1 {
   private int number_of_customers = DataAG1.numberOfCustomers();
   private int number_of_vehicles = DataAG1.numberOfVehicles();
   private int capacity = DataAG1.getCapacityOfVehicles();
   private double number = DataAG1.getNumber();
   private int[] chromosome = new int[number_of_customers];
   private double fitness = 0;

   public ChromosomeAG1(){
      int a;
      a = number_of_customers/number_of_vehicles;
      int[] position_of_ones = new int[number_of_vehicles];
      for (int i = 0 ; i < number_of_vehicles ; i++){
         position_of_ones[i] = (int) (Math.random() * a);
         chromosome[position_of_ones[i] + (a*i)] = 1;
      }
      for (int i = 0; i < number_of_customers; i++) {
         if (chromosome[i] != 1)
            chromosome[i] = 0;
      }
   }

   public ChromosomeAG1(int[] a){
      chromosome = a;
   }

   public int getValue(int a) {
      return chromosome[a];
   }

   public int numberOfOnes(){
      int a = 0;
         for (int i = 0 ; i < number_of_customers ; i++){
            a = a + chromosome[i];
         }
      return a;
   }
```

```
public void setValue(int index, int value) {
   chromosome[index] = value;
   fitness = 0;
}

public int chromosomeSize() {
   return number_of_customers;
}

@Override
public String toString(){
   String string = "";
   for (int i = 0 ; i < number_of_customers ; i++){
      string = string + chromosome[i];
   }
   return string;
}

public double getFitness(){
   if (fitness == 0) {
      double dummy1 = 0;
      double dummy2 = 0;
      double dummy3 = 0;
      double jumlah = 0;
      int number_of_ones = numberOfOnes();

      if (number_of_ones == number_of_vehicles){
         int[] demand = new int[number_of_ones];
         double jarak = 0;
         double[] centerX = new double[number_of_vehicles];
         double[] centerY = new double[number_of_vehicles];
         int[] sub = new int[number_of_ones+1];
         sub[number_of_ones] = 0;
         for (int i = 0 ; i < number_of_ones ; i++){
            demand[i] = 0;
            sub[i] = 0;
            centerX[i] = 0;
            centerY[i] = 0;
         }
         int k = 0;

         for (int i = 0 ; i < number_of_customers ; i++){
            if (chromosome[i] == 1){
```

```
        k++;
    }
    if (k == number_of_ones){
        demand[0] = demand[0]+DataAG1.getCustomer(i).getDemand();
    }
    else {
        demand[k] = demand[k]+DataAG1.getCustomer(i).getDemand();
    }
    sub[k]++;
}

for (int i = 0 ; i < number_of_ones ; i++){
    if (demand[i] <= capacity)
        demand[i] = 0;
    else
        demand[i] = demand[i]-capacity;
}

for (int i = 0 ; i < number_of_ones ; i++){
    jumlah = jumlah + demand[i];
}

int m = 0;
Location[] city = new Location[(sub[0]+sub[number_of_vehicles])];

for (int i = 0 ; i < sub[number_of_vehicles] ; i++){
    city[m] = DataAG1.getCustomer(number_of_customers-
                                        sub[number_of_vehicles]+i);
    m++;
}

for (int i = 0 ; i < sub[0] ; i++){
    city[m] = DataAG1.getCustomer(i);
    m++;
}

for (int i = 0 ; i < sub[0]+sub[number_of_vehicles] ; i++){
    centerX[0] = centerX[0] + city[i].getX();
    centerY[0] = centerY[0] + city[i].getY();
}

centerX[0] = centerX[0]/(sub[0]+sub[number_of_vehicles]);
centerY[0] = centerY[0]/(sub[0]+sub[number_of_vehicles]);
```

```
            int n = sub[0];

            for (int p = 1 ; p < number_of_vehicles ; p++){
               for (int i = n ; i < n+sub[p] ; i++){
                  centerX[p] = centerX[p] + DataAG1.getCustomer(i).getX();
                  centerY[p] = centerY[p] + DataAG1.getCustomer(i).getY();
               }
               centerX[p] = centerX[p]/sub[p];
               centerY[p] = centerY[p]/sub[p];
               n = n + sub[p];
            }

            for (int i = 0 ; i < sub[0]+sub[number_of_vehicles] ; i++){
               jarak = jarak +
                     distanceTo(city[i].getX(),centerX[0],city[i].getY(),centerY[0]);
            }

            n = sub[0];

            for (int p = 1 ; p < number_of_vehicles ; p++){
               for (int i = n ; i < n+sub[p] ; i++){
                  jarak = jarak +
                           distanceTo(DataAG1.getCustomer(i).getX(),centerX[p],
                                    DataAG1.getCustomer(i).getY(),centerY[p]);
               }
               n = n + sub[p];
            }

            dummy3 = jarak;
            dummy2 = number*jumlah;
            fitness = dummy2 + dummy3;
         }
         else fitness = number*1000*Math.abs(number_of_ones-
                                                   number_of_vehicles);
      }
      return fitness;
   }

   public double distanceTo(double a, double b, double c, double d){
      double x = Math.abs(a - b);  double y = Math.abs(c - d);
      double distance = Math.sqrt((x*x) + (y*y));  return distance;
   }
}
```

## Lampiran 14

*Source Code* dari PopulationAG1.java

```
package cvrp;

public class PopulationAG1 {
   ChromosomeAG1[] chromosomes;

   public PopulationAG1(int size, boolean b) {
      chromosomes = new ChromosomeAG1[size];
      if (b) {
         for (int i  = 0; i < size; i++) {
            ChromosomeAG1 newChromosome = new ChromosomeAG1();
            saveChromosome(i, newChromosome);
         }
      }
   }

   public void saveChromosome(int a, ChromosomeAG1 b) {
      chromosomes[a] = b;
   }

   public ChromosomeAG1 getChromosome(int a) {
      return chromosomes[a];
   }

   public ChromosomeAG1 getFittest() {
      ChromosomeAG1 fittest = chromosomes[0];
      for (int i = 1; i < sizeOfPopulation(); i++) {
         if (fittest.getFitness() > getChromosome(i).getFitness()) {
            fittest = getChromosome(i);
         }
      }
      return fittest;
   }

   public int sizeOfPopulation() {
      return chromosomes.length;
   }
}
```

## Lampiran 15

*Source Code* dari AG1.java

```
package cvrp;

public class AG1 {
   public static PopulationAG1 evolvePopulation(PopulationAG1 pop) {
      int b = pop.sizeOfPopulation();
      int a = 2*b;
      PopulationAG1 newPopulation = new PopulationAG1(b, false);

      for (int i = 0 ; i < b; i++) {
         int random1 = (int) (Math.random() * b);
         int random2 = (int) (Math.random() * b);
         ChromosomeAG1 parent1 = pop.getChromosome(random1);
         ChromosomeAG1 parent2 = pop.getChromosome(random2);
         ChromosomeAG1 child = crossover(parent1, parent2);
         newPopulation.saveChromosome(i, child);
      }

      ChromosomeAG1 aa = newPopulation.getFittest();

      for (int i = 0; i < b; i++) {
         if (newPopulation.getChromosome(i) != aa){
            mutate(newPopulation.getChromosome(i));
         }
      }

      PopulationAG1 newPopulation2 = new PopulationAG1(a, false);

      for (int i = 0; i < b; i++) {
         newPopulation2.saveChromosome(i, pop.getChromosome(i));
      }

      for (int i = 0; i < b; i++) {
         newPopulation2.saveChromosome((b+i),
                                          newPopulation.getChromosome(i));
      }

      for (int i = 0; i < b; i++) {
         int k = i;
         ChromosomeAG1 fittest = newPopulation2.getChromosome(i);
```

```
        for (int j = i+1; j < a; j++) {
            if (fittest.getFitness() >
                                newPopulation2.getChromosome(j).getFitness()) {
                fittest = newPopulation2.getChromosome(j);
                k = j;
            }
        }
        newPopulation2.saveChromosome(k,
                                            newPopulation2.getChromosome(i));
        newPopulation2.saveChromosome(i, fittest);
    }

    PopulationAG1 newPopulation3 = new PopulationAG1(b, false);
    int m = 0;
    newPopulation3.saveChromosome(0, newPopulation2.getChromosome(0));

    for (int i = 1; i < b; i++){
        if (newPopulation2.getChromosome(i).getFitness() !=
                                newPopulation3.getChromosome(m).getFitness()){
            m++;
            newPopulation3.saveChromosome(m,
                                            newPopulation2.getChromosome(i));
        }
    }

    if ((m+1) != b){
        for (int i = m+1; i < b; i++){
            ChromosomeAG1 acak = new ChromosomeAG1();
            newPopulation3.saveChromosome(i, acak);
        }
    }

    return newPopulation3;
}

public static ChromosomeAG1 crossover(ChromosomeAG1 p1,
                                                    ChromosomeAG1 p2) {
    ChromosomeAG1 child = new ChromosomeAG1();
    int position = (int) (Math.random() * p1.chromosomeSize());

    for (int i = 0 ; i < position ; i++)
        child.setValue(i, p1.getValue(i));
```

```
        for (int i = position ; i < p1.chromosomeSize() ; i++)
            child.setValue(i, p2.getValue(i));

        return child;
    }

    private static void mutate(ChromosomeAG1 a) {
        double dummy = Math.random();
        if (dummy < 0.5){
            int position = (int) (Math.random() * a.chromosomeSize());
            a.setValue(position, Math.abs((a.getValue(position)-1)));
        }
    }
}
```

## Lampiran 16

*Source Code* dari DataAG2.java

```
package cvrp;

import java.util.ArrayList;

public class DataAG2 {
   private ArrayList customers = new ArrayList<Location>();

   public void addCustomer(Location a) {
      customers.add(a);
   }

   public Location getCustomer(int a){
      return (Location)customers.get(a);
   }

   public int totalX(){
      int a = 0;
      for (int i = 0 ; i < numberOfCustomers(); i++){
         a = a + getCustomer(i).getX();
      }
      return a;
   }

   public int totalY(){
      int a = 0;
      for (int i = 0 ; i < numberOfCustomers(); i++){
         a = a + getCustomer(i).getY();
      }
      return a;
   }

   public int numberOfCustomers(){
      return customers.size();
   }

   @Override
   public String toString() {
      String string = "(";
```

```
        for (int i = 0; i < numberOfCustomers(); i++) {
            string += ""+(i+1)+")"+getCustomer(i)+"  \n(";
        }
        return string;
    }

}
```

## Lampiran 17

*Source Code* dari ChromosomeAG2.java

```
package cvrp;

import java.util.ArrayList;
import java.util.Collections;

public class ChromosomeAG2 {
   private ArrayList chromosome = new ArrayList<Location>();
   private double fitness = 0;
   private double distance = 0;

   public ChromosomeAG2(int a){
      for (int i = 0; i < a; i++) {
         chromosome.add(null);
      }
   }

   public ChromosomeAG2(ArrayList a){
      this.chromosome = a;
   }

   public void generateIndividual(DataAG2 a) {
      for (int cityIndex = 0; cityIndex < a.numberOfCustomers(); cityIndex++) {
        setLocation(cityIndex, a.getCustomer(cityIndex));
      }
      Collections.shuffle(chromosome);
   }

   public void setLocation(int index, Location a) {
      chromosome.set(index, a);
      fitness = 0;
      distance = 0;
   }

   public double getFitness() {
      if (fitness == 0) {
         fitness = 1/(double)getDistance();
      }
      return fitness;
   }
```

```
    public Location getLocation(int a) {
        return (Location)chromosome.get(a);
    }

    public double getDistance(){
        if (distance == 0) {
            double chromosomeDistance = 0;

            for (int i=0; i < tourSize(); i++) {
                Location from = getLocation(i);
                Location destination;
                if(i+1 < tourSize()){
                    destination = getLocation(i+1);
                }
                else{
                    destination = getLocation(0);
                }
                chromosomeDistance += from.distanceTo(destination);
            }

            distance = chromosomeDistance;
        }
        return distance;
    }

    public int tourSize() {
        return chromosome.size();
    }

    public boolean containsLocation(Location a){
        return chromosome.contains(a);
    }

    @Override
    public String toString() {
        String geneString = "|";
        for (int i = 0; i < tourSize(); i++) {
            geneString += i+") "+getLocation(i)+"\n|";
        }
        return geneString;
    }
}
```

## Lampiran 18

*Source Code* dari PopulationAG2.java

```
package cvrp;

public class PopulationAG2 {
   ChromosomeAG2[] chromosomes;
   DataAG2 data;

   public PopulationAG2(int size, boolean b, DataAG2 a) {
      this.data = a;
      chromosomes = new ChromosomeAG2[size];
      if (b) {
         for (int i = 0; i < populationSize(); i++) {
            ChromosomeAG2 chromosome = new
                                       ChromosomeAG2(a.numberOfCustomers());
            chromosome.generateIndividual(a);
            saveChromosome(i, chromosome);
         }
      }
   }

   public void saveChromosome(int index, ChromosomeAG2 a) {
      chromosomes[index] = a;
   }

   public ChromosomeAG2 getChromosome(int a) {
      return chromosomes[a];
   }

   public ChromosomeAG2 getFittest() {
      ChromosomeAG2 fittest = chromosomes[0];
      for (int i = 1; i < populationSize(); i++) {
         if (fittest.getFitness() <= getChromosome(i).getFitness()) {
            fittest = getChromosome(i);
         }
      }
      return fittest;
   }

   public int populationSize() {
      return chromosomes.length;
   }
```

```
    public DataAG2 getDataAG2() {
        return data;
    }
}
```

## Lampiran 19

*Source Code* dari AG2.java

```
package cvrp;

public class AG2 {
   private static final double mutationRate = 0.2;
   private static final int tournamentSize = 5;

   public static PopulationAG2 evolvePopulation(PopulationAG2 pop) {
      int a = 2*pop.populationSize();
      int b = pop.populationSize();

      PopulationAG2 newPopulation = new
                                 PopulationAG2(pop.populationSize(), false, null);

      for (int i = 0; i < newPopulation.populationSize(); i++) {
         ChromosomeAG2 parent1 = tournamentSelection(pop);
         ChromosomeAG2 parent2 = tournamentSelection(pop);
         ChromosomeAG2 child = crossover(parent1, parent2);
         newPopulation.saveChromosome(i, child);
      }

      ChromosomeAG2 aa = newPopulation.getFittest();

      for (int i = 0; i < b; i++) {
         if (newPopulation.getChromosome(i) != aa){
            mutate(newPopulation.getChromosome(i));
         }
      }

      PopulationAG2 newPopulation2 = new PopulationAG2(a, false, null);

      for (int i = 0; i < pop.populationSize(); i++) {
         newPopulation2.saveChromosome(i, pop.getChromosome(i));
      }

      for (int i = 0; i < pop.populationSize(); i++) {
         newPopulation2.saveChromosome((b+i),
                                              newPopulation.getChromosome(i));
      }
```

```
    for (int i = 0; i < a; i++){
        int k = i;
        ChromosomeAG2 fittest = newPopulation2.getChromosome(i);

        for (int j = i+1; j < a; j++) {
            if (fittest.getFitness() <=
                                newPopulation2.getChromosome(j).getFitness()) {
                fittest = newPopulation2.getChromosome(j);
                k = j;
            }
        }

        newPopulation2.saveChromosome(k,
                                            newPopulation2.getChromosome(i));
        newPopulation2.saveChromosome(i, fittest);
    }

    PopulationAG2 newPopulation3 = new PopulationAG2(b, false,null);

    int m = 0;
    newPopulation3.saveChromosome(0, newPopulation2.getChromosome(0));

    for (int i = 1; i < b; i++){
        if (newPopulation2.getChromosome(i).getDistance() !=
                                newPopulation3.getChromosome(m).getDistance()){
            m++;
            newPopulation3.saveChromosome(m,
                                            newPopulation2.getChromosome(i));
        }
    }

    if ((m+1) != b){
        for (int i = m+1; i < b; i++){
            int ran = (int) (Math.random() * a);
            newPopulation3.saveChromosome(i,
                                          newPopulation2.getChromosome(ran));
        }
    }

    return newPopulation3;
}
```

```
public static ChromosomeAG2 crossover(ChromosomeAG2 parent1,
                                                    ChromosomeAG2 parent2) {
   int a = parent1.tourSize();
   ChromosomeAG2 child = new ChromosomeAG2(a);
   Location cityFromParent1 = new Location();
   Location cityFromParent2 = new Location();
   child.setLocation(0, parent1.getLocation(0));

   // Inisialisasi kota yang digunakan untuk mencari legitimate node
   Location[] kota = new Location[a];
   for (int i = 0; i < a; i++) {
      kota[i] = new Location();
   }

   // Proses Crossover
   for (int i = 0; i < a-1; i++) {
      kota[i].setLocation(child.getLocation(i).getX(),
                                                    child.getLocation(i).getY());

      // Legitimate node dari Parent 1
      for (int ii = 0; ii < a-1; ii++) {
         if (parent1.getLocation(ii).distanceTo(kota[i]) == 0)  {
            cityFromParent1.setLocation(parent1.getLocation(ii+1).getX(),
                                                parent1.getLocation(ii+1).getY());
         }
      }
      if (parent1.getLocation(a-1).distanceTo(kota[i]) == 0) {
         cityFromParent1.setLocation(parent1.getLocation(0).getX(),
                                                parent1.getLocation(0).getY());
      }

      //Cek apakah kota dari Parent1 ada di Child
      for (int ii = 0; ii < a; ii++)
      {
         if (cityFromParent1.distanceTo(kota[ii]) == 0)
         {
            cityFromParent1.setLocation(0,0);
         }
      }

      // Mencari dari {1,2,3,4,5,6,...,n}
      if (cityFromParent1.getX()==0 && cityFromParent1.getY()==0) {
         for (int ii = 0; ii < a; ii++){
```

```
        int j = 0;
        for (int iii = 0; iii < a; iii++){
            if (parent1.getLocation(ii).distanceTo(kota[iii]) == 0)
                j = 1;
        }
        if (j == 0)
            cityFromParent1.setLocation(parent1.getLocation(ii).getX(),
                                                    parent1.getLocation(ii).getY());
    }
}

// Legitimate node dari Parent 2
for (int ii = 0; ii < a-1; ii++){
    if (parent2.getLocation(ii).distanceTo(kota[i]) == 0) {
        cityFromParent2.setLocation(parent2.getLocation(ii+1).getX(),
                                                parent2.getLocation(ii+1).getY());
    }
}
if (parent2.getLocation(a-1).distanceTo(kota[i]) == 0) {
    cityFromParent2.setLocation(parent2.getLocation(0).getX(),
                                                    parent2.getLocation(0).getY());
}

//Cek apakah kota dari Parent2 ada di Child
for (int ii = 0; ii < a; ii++)
    if (cityFromParent2.distanceTo(kota[ii]) == 0)
        cityFromParent2.setLocation(0,0);
    }
}

// Mencari dari {1,2,3,4,5,6,...,n}
if (cityFromParent2.getX()==0 && cityFromParent2.getY()==0)
    for (int ii = 0; ii < a; ii++) {
        int j = 0;
        for (int iii = 0; iii < a; iii++){
            if (parent2.getLocation(ii).distanceTo(kota[iii]) == 0)
                j = 1;
        }
        if (j == 0)
            cityFromParent2.setLocation(parent2.getLocation(ii).getX(),
                                                    parent2.getLocation(ii).getY());
    }
}
```

```
            // Menentukan kromosom child
            if (kota[i].distanceTo(cityFromParent1) <
                                                kota[i].distanceTo(cityFromParent2)) {
               kota[i+1].setLocation(cityFromParent1.getX(),
                                                            cityFromParent1.getY());
               for (int ii = 1 ; ii < a ; ii++) {
                  child.setLocation(ii, kota[ii]);
               }
            }
            else {
               kota[i+1].setLocation(cityFromParent2.getX(),
                                                            cityFromParent2.getY());
               for (int ii = 1 ; ii < a ; ii++) {
                  child.setLocation(ii, kota[ii]);
               }
            }
         }
         return child;
      }

      private static void mutate(ChromosomeAG2 tour) {
         for(int i=0; i < tour.tourSize(); i++){
            if(Math.random() < mutationRate){
               int r = (int) (tour.tourSize() * Math.random());
               Location city1 = tour.getLocation(i);
               Location city2 = tour.getLocation(r);
               tour.setLocation(r, city1);
               tour.setLocation(i, city2);
            }
         }
      }

      private static ChromosomeAG2 tournamentSelection(PopulationAG2 pop) {
         PopulationAG2 tournament = new PopulationAG2(tournamentSize, false,
                                                            pop.getDataAG2());
         for (int i = 0; i < tournamentSize; i++) {
            int randomId = (int) (Math.random() * pop.populationSize());
            tournament.saveChromosome(i, pop.getChromosome(randomId));
         }
         ChromosomeAG2 fittest = tournament.getFittest();
         return fittest;
      }
   }
```

## Lampiran 20

*Source Code* dari Simulation.java

```
package cvrp;

public class Simulation extends javax.swing.JFrame {

    private int X;
    private int Y;
    private int Jumlah;
    private int Rata;
    private int[] koordinatX;
    private int[] koordinatY;
    private int[] permintaan;
    private int [][] aaa;

    public Simulation() {
        initComponents();
    }

    public void cekBeda(int[] a, int[] b, int j){
        for (int i = 0 ; i < j ; i++){
            if (a[j] == a[i] && b[j] == b[i]){
                a[j] = (int)(Math.random()*X);
                cekBeda(a, b, j);
            }
        }
    }

…

    private void randomActionPerformed(java.awt.event.ActionEvent evt) {
        Jumlah = Integer.parseInt(jumlah.getText());
        Rata = Integer.parseInt(rata.getText());
        X = 2*Integer.parseInt(x.getText()) + 1;
        Y = 2*Integer.parseInt(y.getText()) + 1;
        koordinatX = new int[Jumlah];
        koordinatY = new int[Jumlah];
        permintaan = new int[Jumlah];
        String string = "";
        for(int i = 0 ; i < Jumlah ; i++){
            permintaan[i] = ((int)(Math.random()*21))+Rata-10;
            koordinatX[i] = (int)(Math.random()*X);
```

```
        koordinatY[i] = (int)(Math.random()*Y);
        cekBeda(koordinatX , koordinatY, i);
        string = string + (i+1) + ") x : " + koordinatX[i] + "\t   y : " +
                                        koordinatY[i] + "\tp : " + permintaan[i] + "\n";
    }

    teks.setText(string);
    Pelanggan pelanggan = new Pelanggan (koordinatX, koordinatY, Jumlah);
    pelanggan.setVisible(true);
    aaa = new int [Jumlah][3];

    for (int i = 0 ; i < Jumlah ; i++){
        aaa[i][0] = koordinatX[i];
        aaa[i][1] = koordinatY[i];
        aaa[i][2] = permintaan[i];
    }
}

private void AGGActionPerformed(java.awt.event.ActionEvent evt) {
    double best = 0;
    int a = 0;
    double b = 0;
    int capacity = Integer.parseInt(kapasitasAGG.getText());
    int ganti = Jumlah;
    Location depot = new
            Location(Integer.parseInt(x.getText()),Integer.parseInt(y.getText()));
    double number = 100000;
    int totalGeneration = 0;
    DataAG1.setNumber(number);
    DataAG1.setDepot(depot);
    DataAG1.setCapacityOfVehicles(capacity);
    int numberOfCities = ganti;
    Location[] city = new Location[numberOfCities];

    for (int i = 0 ; i < numberOfCities ; i++){
        city[i] = new Location(aaa[i][0],aaa[i][1],aaa[i][2]);
    }

    int numberOfVehicle;

    for (int i = 0 ; i < numberOfCities ; i++){
        a = a + city[i].getDemand();
    }
```

```
numberOfVehicle = a / DataAG1.getCapacityOfVehicles()+ 1;
a = 0;
DataAG1.setNumberOfVehicles(numberOfVehicle);

int[][] ccc = new int[numberOfCities+numberOfVehicle+1][2];
ccc[numberOfCities+numberOfVehicle][0] = DataAG1.getDepot().getX();
ccc[numberOfCities+numberOfVehicle][1] = DataAG1.getDepot().getY();
double tetha;
Location cityDummy = new Location();

for (int i = 1 ; i < numberOfCities ; i++){
   tetha = city[i-1].getTetha();
   a = i-1;
   for (int k = i ; k < numberOfCities ; k++){
      if (tetha > city[k].getTetha()){
         tetha = city[k].getTetha();
         a = k;
      }
   }
   cityDummy = city[a];
   city[a] = city [i-1];
   city[i-1] = cityDummy;
}

a = 0;

for (int i = 0 ; i < numberOfCities ; i++){
   DataAG1.addCustomer(city[i]);
}

int sizeOfPopulation = 100;
int numberOfGeneration = 100000;
PopulationAG1 pop = new PopulationAG1(sizeOfPopulation, true);
int aa = 0;
double bb = pop.getChromosome(0).getFitness();

for (int i = 0; i < numberOfGeneration; i++) {
   pop = AG1.evolvePopulation(pop);
   System.out.println(aa+") "+pop.getChromosome(0).getFitness());
   if (bb == pop.getChromosome(0).getFitness())
      aa++;
   if (bb != pop.getChromosome(0).getFitness())
      aa = 0;
```

```
    if (aa==2000)
        break;
    bb = pop.getChromosome(0).getFitness();
    totalGeneration++;
}

for (int i = 0 ; i < 20 ; i++){
    if (pop.getChromosome(0).getFitness() >= DataAG1.getNumber()){
        System.out.println("\n\nTambah Kendaraan");
        numberOfVehicle++;
        DataAG1.setNumberOfVehicles(numberOfVehicle);
        PopulationAG1 newPop = new PopulationAG1(sizeOfPopulation,
                                                                    true);
        aa = 0;
        bb = newPop.getChromosome(0).getFitness();

        for (int j = 0; j < numberOfGeneration; j++) {
            newPop = AG1.evolvePopulation(newPop);
            System.out.println(aa+")
                                    "+newPop.getChromosome(0).getFitness());
            if (bb == newPop.getChromosome(0).getFitness())
                aa++;
            if (bb != newPop.getChromosome(0).getFitness())
                aa = 0;
            if (aa==2000)
                break;
            bb = newPop.getChromosome(0).getFitness();
            totalGeneration++;
        }

        pop = newPop;
    }
}

if (pop.getChromosome(0).getFitness() < DataAG1.getNumber()){
    int numberOfVeasibleSolution = 1;

    for (int ii = 0 ; ii < numberOfVeasibleSolution ; ii++){
        a = 0;
        b = 0;
        DataAG2[] tourManagerTSP = new DataAG2[numberOfVehicle];
```

```
for (int i = 0 ; i < numberOfVehicle ; i++){
   tourManagerTSP[i] = new DataAG2();
   tourManagerTSP[i].addCustomer(depot);
}

for (int i = 0 ; i < numberOfCities ; i++){
   if (pop.getChromosome(ii).getValue(i) == 1){
      a++;
   }
   if (a == numberOfVehicle){
      tourManagerTSP[0].addCustomer(city[i]);
   }
   else{
      tourManagerTSP[a].addCustomer(city[i]);
   }
}

PopulationAG2[] popRegion = new
                                    PopulationAG2[numberOfVehicle];

for (int i = 0 ; i < numberOfVehicle ; i++){
   popRegion[i] = new PopulationAG2(50, true, tourManagerTSP[i]);
}

int numberOfGenerations3 = 1000;
int numberOfGenerations2 = numberOfCities*6/numberOfVehicle;

for (int i = 0 ; i < numberOfVehicle ; i++){
   aa = 0;
   bb = popRegion[i].getChromosome(0).getFitness();
   for (int k = 0 ; k < numberOfGenerations3 ; k++){
      popRegion[i] = AG2.evolvePopulation(popRegion[i]);
      if (bb == popRegion[i].getChromosome(0).getFitness())
         aa++;
      if (bb != popRegion[i].getChromosome(0).getFitness())
         aa = 0;
      if (aa==numberOfGenerations2)
         break;
      bb = popRegion[i].getChromosome(0).getFitness();
      totalGeneration++;
   }
}
```

```
for (int i = 0 ; i < numberOfVehicle ; i++){
   b = b + popRegion[i].getFittest().getDistance();
}

System.out.println("\n"+b);

if (ii == 0)
   best = b;

if (b < best)
   best = b;

int[] c = new int [numberOfVehicle];

for (int i = 0 ; i < numberOfVehicle ; i++){
   for (int j = 0 ; j < popRegion[i].getChromosome(0).tourSize() ;
                                                                  j++){
      if (popRegion[i].getChromosome(0).getLocation(j).getX() ==
                                          DataAG1.getDepot().getX() &&
            popRegion[i].getChromosome(0).getLocation(j).getY() ==
                                          DataAG1.getDepot().getY()){
         c[i] = j;
      }
   }
   System.out.println(popRegion[i].getFittest().toString());
   System.out.println(c[i]);
   System.out.println();
}

int m = 0;

for (int i = 0 ; i < numberOfVehicle ; i++){
   for (int j = c[i] ; j < popRegion[i].getChromosome(0).tourSize() ;
                                                                  j++){
      ccc[m][0] =
               popRegion[i].getChromosome(0).getLocation(j).getX();
      ccc[m][1] =
               popRegion[i].getChromosome(0).getLocation(j).getY();
      m++;
   }

   for (int j = 0 ; j < c[i] ; j++){
      ccc[m][0] =
```

```
                    popRegion[i].getChromosome(0).getLocation(j).getX();
                ccc[m][1] =
                    popRegion[i].getChromosome(0).getLocation(j).getY();
                m++;
              }
            }

            for (int i = 0 ; i < numberOfCities+numberOfVehicle ; i++){
               System.out.println(i+") "+ccc[i][0]+"\t"+ccc[i][1]);
            }
          }

          System.out.println("\n\n"+best);
       }

       System.out.println("\nTotal Generasi : "+totalGeneration);

       PelangganAGG pelanggan = new PelangganAGG (koordinatX,
                                                  koordinatY, Jumlah, ccc);
       pelanggan.setVisible(true);
       hasilAGG.setText(""+best);
       generasiAGG.setText(""+totalGeneration);
    }

    private void AGBActionPerformed(java.awt.event.ActionEvent evt) {
       int zz = Integer.parseInt(kapasitasAGG.getText());
       City depot = new
                City(Integer.parseInt(x.getText()),Integer.parseInt(y.getText()),zz);
       TourManager.setDepot(depot);
       int numberOfCities = Jumlah;
       City[] city = new City[numberOfCities];

       for (int i = 0 ; i < numberOfCities ; i++){
          city[i] = new City(aaa[i][0],aaa[i][1],aaa[i][2]);
       }

       for (int i = 0 ; i < numberOfCities ; i++){
          TourManager.addCity(city[i]);
       }

       int sizeOfPopulation = 100;
       int numberOfGeneration = 100000;
       Population pop = new Population(sizeOfPopulation, true);
```

```
int aa = 0;
double bb = pop.getFittest().getDistance();
int totalGeneration=0;

for (int i = 0; i < numberOfGeneration; i++) {
   pop = AG.evolvePopulation(pop);
   if (bb == pop.getFittest().getDistance())
      aa++;
   if (bb != pop.getFittest().getDistance())
      aa = 0;
   if (aa==2000)
      break;
   System.out.println(i+" - "+aa+") "+pop.getFittest().getDistance());
   bb = pop.getFittest().getDistance();
   totalGeneration++;
}

hasilAGB.setText(""+pop.getFittest().getDistance());
generasiAGB.setText(""+totalGeneration);
int kapasitas = depot.getD();
int[][] bbb = new int [1000][2];
int[] c = new int[100];
c[0] = 0;
int m = 1;
int kap = 0;
int n = 0;

for (int i = 0; i < pop.getFittest().tourSize(); i++) {
   kap = kap + pop.getFittest().getCity(i).getD();
   bbb[n][0] = pop.getFittest().getCity(i).getX();
   bbb[n][1] = pop.getFittest().getCity(i).getY();
   n++;
   if (kap > kapasitas){
      kap = pop.getFittest().getCity(i).getD();;
      c[m] = i;
      m++;
      bbb[n][0] = pop.getFittest().getCity(i).getX();
      bbb[n][1] = pop.getFittest().getCity(i).getY();
      bbb[n-1][0] = depot.getX();
      bbb[n-1][1] = depot.getY();
      n++;
   }
}
```

```
        System.out.println(pop.getFittest().toString());

        for (int i = 0; i < m; i++) {
            System.out.println(i+") "+c[i]);
        }

        for (int i = 0; i < 100; i++) {
            System.out.println(i+") "+bbb[i][0]+"\t"+bbb[i][1]);
        }

        int banyakdaerah = m+1;
        int[][] ccc = new int[numberOfCities+banyakdaerah][2];
        ccc[numberOfCities+banyakdaerah-1][0] =
                                                TourManager.getDepot().getX();
        ccc[numberOfCities+banyakdaerah-1][1] =
                                                TourManager.getDepot().getY();
        ccc[0][0] = TourManager.getDepot().getX();
        ccc[0][1] = TourManager.getDepot().getY();

        for (int i = 1; i < numberOfCities+banyakdaerah-1; i++) {
            ccc[i][0] = bbb[i-1][0];
            ccc[i][1] = bbb[i-1][1];
        }

        for (int i = 0; i < numberOfCities+banyakdaerah; i++) {
            System.out.println(i+") "+ccc[i][0]+"\t"+ccc[i][1]);
        }

        PelangganAG pelanggan = new PelangganAG (koordinatX, koordinatY,
                                                            Jumlah, ccc);
        pelanggan.setVisible(true);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        x.setText("50");
        y.setText("50");
        jumlah.setText("50");
        rata.setText("20");
        Jumlah = Integer.parseInt(jumlah.getText());
        Rata = Integer.parseInt(rata.getText());
        X = 2*Integer.parseInt(x.getText()) + 1;
        Y = 2*Integer.parseInt(y.getText()) + 1;
        koordinatX = new int[Jumlah];
```

```
    koordinatY = new int[Jumlah];
    permintaan = new int[Jumlah];
    String string = "";
    aaa = new int [Jumlah][3];

    int[][] bbb = { {1, 50, 19},{3, 81, 11},{3, 4, 16},{9, 3, 25},{9, 32, 29},
                    {13, 99, 13},{14, 21, 26},{24, 9, 14},{25, 82, 16},{29, 10, 12},
                    {30, 33, 29},{30, 50, 26},{32, 52, 18},{32, 5, 27},{34, 27, 17},
                   {35, 91, 15},{35, 67, 28},{37, 80, 29},{37, 87, 13},{38, 49, 13},
                   {39, 42, 22},{40, 11, 27},{45, 47, 19},{47, 36, 20},{50, 83, 10},
                   {51, 94, 24},{53, 82, 19},{54, 36, 12},{54, 45, 24},{57, 99, 27},
                   {59, 72, 30},{60, 21, 18},{60, 14, 18},{62, 95, 16},{62, 31, 28},
                    {63, 20, 29},{65, 11, 17},{70, 2, 28},{70, 30, 26},{73, 86, 10},
                   {74, 17, 27},{75, 88, 25},{76, 27, 27},{77, 51, 17},{80, 44, 19},
                {89, 87, 13},{94, 86, 17},{97, 81, 26},{97, 59, 25},{100, 52, 24}};

    for (int i = 0 ; i < Jumlah ; i++){
       aaa[i][0] = bbb[i][0];
       aaa[i][1] = bbb[i][1];
       aaa[i][2] = bbb[i][2];
       permintaan[i] = bbb[i][2];
       koordinatX[i] = bbb[i][0];
       koordinatY[i] = bbb[i][1];
       string = string + (i+1) + ") x : " + koordinatX[i] + "\t   y : " +
                                        koordinatY[i] + "\tp : " + permintaan[i] + "\n";
    }

    teks.setText(string);
    Pelanggan pelanggan = new Pelanggan (koordinatX, koordinatY, Jumlah);
    pelanggan.setVisible(true);
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    x.setText("75");
    y.setText("75");
    jumlah.setText("75");
    rata.setText("20");
    Jumlah = Integer.parseInt(jumlah.getText());
    Rata = Integer.parseInt(rata.getText());
    X = 2*Integer.parseInt(x.getText()) + 1;
    Y = 2*Integer.parseInt(y.getText()) + 1;
    koordinatX = new int[Jumlah];
    koordinatY = new int[Jumlah];
```

```
    permintaan = new int[Jumlah];
    String string = "";
    aaa = new int [Jumlah][3];

    int[][] bbb = { {1, 19, 24},{2, 2, 11},{4, 120, 29},{4, 141, 29},
                    {7, 117, 13},{8, 138, 16},{10, 133, 14},{11, 65, 17},{12, 92, 17},
                  {22, 87, 25},{24, 17, 25},{24, 149, 28},{25, 122, 11},{25, 101, 11},
                   {29, 7, 11},{30, 140, 10},{33, 87, 15},{35, 108, 18},{37, 138, 14},
                   {38, 57, 22},{38, 145, 12},{39, 82, 29},{39, 49, 24},{39, 144, 20},
                   {42, 128, 27},{44, 73, 29},{48, 21, 24},{50, 76, 27},{50, 108, 12},
                    {50, 129, 24},{50, 64, 10},{51, 79, 17},{52, 17, 18},{56, 91, 17},
                    {56, 18, 25},{58, 40, 24},{63, 91, 27},{65, 112, 27},{65, 55, 16},
                    {67, 35, 14},{68, 122, 30},{72, 82, 13},{74, 30, 24},{74, 64, 16},
                  {74, 113, 10},{77, 103, 21},{78, 85, 14},{79, 28, 27},{79, 65, 26},
                    {82, 59, 14},{83, 102, 10},{86, 77, 18},{88, 63, 29},{90, 18, 14},
                  {93, 20, 24},{94, 101, 12},{94, 77, 14},{99, 41, 19},{103, 33, 13},
               {110, 73, 13},{114, 33, 21},{115, 85, 18},{117, 85, 11},{118, 37, 15},
                  {120, 61, 30},{126, 1, 23},{127, 63, 11},{128, 31, 13},{134, 2, 16},
            {134, 84, 10},{138, 53, 14},{142, 115, 24},{149, 11, 26},{149, 143, 18},
                                                                    {149, 28, 19}};

    for (int i = 0 ; i < Jumlah ; i++){
       aaa[i][0] = bbb[i][0];
       aaa[i][1] = bbb[i][1];
       aaa[i][2] = bbb[i][2];
       permintaan[i] = bbb[i][2];
       koordinatX[i] = bbb[i][0];
       koordinatY[i] = bbb[i][1];
       string = string + (i+1) + ") x : " + koordinatX[i] + "\t   y : " +
                                        koordinatY[i] + "\tp : " + permintaan[i] + "\n";
    }

    teks.setText(string);
    Pelanggan pelanggan = new Pelanggan (koordinatX, koordinatY, Jumlah);
    pelanggan.setVisible(true);
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    x.setText("100");
    y.setText("100");
    jumlah.setText("100");
    rata.setText("20");
    Jumlah = Integer.parseInt(jumlah.getText());
```

```
Rata = Integer.parseInt(rata.getText());
X = 2*Integer.parseInt(x.getText()) + 1;
Y = 2*Integer.parseInt(y.getText()) + 1;
koordinatX = new int[Jumlah];
koordinatY = new int[Jumlah];
permintaan = new int[Jumlah];
String string = "";
aaa = new int [Jumlah][3];

int[][] bbb = { {0, 30, 14},{2, 31, 19},{2, 165, 23},{3, 46, 11},
            {5, 155, 22},{6, 102, 22},{8, 163, 24},{8, 101, 27},{15, 119, 29},
          {21, 121, 28},{22, 103, 18},{24, 187, 27},{28, 172, 22},{29, 7, 28},
            {30, 194, 15},{32, 2, 27},{32, 61, 16},{34, 57, 13},{36, 73, 18},
         {38, 179, 23},{42, 141, 17},{44, 127, 30},{49, 24, 15},{50, 124, 11},
         {51, 106, 16},{52, 14, 17},{53, 147, 21},{53, 111, 20},{58, 139, 25},
            {62, 36, 13},{62, 60, 11},{65, 173, 30},{68, 26, 30},{68, 48, 15},
         {70, 32, 25},{70, 182, 13},{72, 130, 24},{75, 130, 17},{76, 130, 24},
            {79, 7, 10},{79, 11, 12},{83, 11, 16},{85, 122, 25},{86, 35, 18},
          {87, 161, 25},{90, 103, 11},{92, 31, 11},{92, 13, 24},{93, 96, 29},
     {101, 150, 25},{101, 15, 15},{103, 83, 25},{103, 74, 25},{105, 74, 13},
      {105, 200, 23},{106, 181, 12},{108, 97, 12},{109, 2, 20},{117, 6, 11},
    {117, 58, 25},{117, 73, 28},{119, 120, 19},{125, 167, 10},{125, 37, 24},
{128, 128, 25},{131, 190, 15},{135, 178, 11},{147, 107, 26},{148, 167, 21},
     {150, 162, 24},{150, 77, 30},{151, 0, 12},{151, 151, 23},{152, 21, 25},
    {153, 100, 12},{158, 0, 10},{159, 199, 30},{162, 93, 29},{166, 148, 15},
     {167, 193, 26},{168, 27, 23},{168, 37, 17},{171, 62, 26},{171, 33, 27},
     {171, 85, 27},{173, 129, 16},{173, 55, 16},{175, 45, 19},{176, 23, 25},
      {177, 9, 26},{179, 123, 25},{184, 46, 26},{186, 66, 14},{187, 11, 17},
    {193, 123, 20},{194, 171, 30},{195, 16, 25},{195, 60, 17},{197, 51, 28},
                                                        {199, 122, 18}};

for (int i = 0 ; i < Jumlah ; i++){
    aaa[i][0] = bbb[i][0];
    aaa[i][1] = bbb[i][1];
    aaa[i][2] = bbb[i][2];
    permintaan[i] = bbb[i][2];
    koordinatX[i] = bbb[i][0];
    koordinatY[i] = bbb[i][1];
    string = string + (i+1) + ") x : " + koordinatX[i] + "\t   y : " +
                                  koordinatY[i] + "\tp : " + permintaan[i] + "\n";
}

teks.setText(string);
```

```
    Pelanggan pelanggan = new Pelanggan (koordinatX, koordinatY, Jumlah);
    pelanggan.setVisible(true);
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    x.setText("125");
    y.setText("125");
    jumlah.setText("125");
    rata.setText("20");
    Jumlah = Integer.parseInt(jumlah.getText());
    Rata = Integer.parseInt(rata.getText());
    X = 2*Integer.parseInt(x.getText()) + 1;
    Y = 2*Integer.parseInt(y.getText()) + 1;
    koordinatX = new int[Jumlah];
    koordinatY = new int[Jumlah];
    permintaan = new int[Jumlah];
    String string = "";
    aaa = new int [Jumlah][3];

    int[][] bbb = { {4, 152, 15},{4, 12, 20},{5, 179, 24},{8, 225, 18},
                {9, 6, 10},{10, 246, 15},{10, 135, 28},{14, 78, 22},{14, 119, 11},
            {15, 103, 22},{16, 34, 13},{17, 119, 11},{19, 139, 25},{20, 192, 11},
              {22, 85, 29},{27, 7, 30},{30, 142, 17},{31, 240, 10},{32, 240, 15},
          {32, 246, 14},{33, 217, 10},{38, 226, 16},{41, 165, 16},{43, 108, 30},
            {43, 106, 22},{43, 165, 18},{45, 194, 19},{47, 112, 26},{48, 74, 29},
              {49, 230, 15},{49, 116, 10},{51, 201, 23},{52, 81, 16},{52, 60, 21},
              {52, 149, 17},{56, 102, 29},{61, 71, 29},{63, 77, 15},{65, 32, 11},
            {67, 246, 15},{67, 238, 28},{72, 140, 14},{73, 177, 11},{76, 74, 23},
              {77, 19, 10},{83, 184, 20},{86, 124, 16},{86, 96, 29},{87, 233, 25},
              {87, 97, 25},{87, 127, 27},{88, 153, 13},{89, 1, 15},{93, 174, 15},
            {93, 182, 16},{94, 233, 13},{94, 145, 29},{95, 30, 17},{96, 14, 28},
        {99, 147, 15},{102, 108, 26},{103, 15, 21},{103, 43, 13},{107, 132, 28},
      {112, 136, 12},{112, 19, 23},{115, 236, 17},{117, 35, 25},{124, 195, 19},
          {129, 7, 19},{132, 48, 19},{135, 83, 18},{136, 153, 23},{143, 139, 13},
    {144, 126, 16},{147, 112, 11},{152, 44, 30},{154, 237, 14},{155, 212, 18},
      {159, 19, 30},{159, 111, 26},{161, 220, 22},{162, 24, 16},{164, 108, 22},
        {167, 180, 24},{170, 150, 29},{174, 0, 16},{175, 198, 13},{178, 18, 24},
        {178, 114, 14},{183, 4, 16},{190, 132, 13},{190, 167, 22},{194, 38, 21},
      {195, 123, 25},{197, 33, 24},{197, 187, 28},{199, 47, 30},{200, 148, 22},
        {204, 179, 13},{205, 6, 19},{211, 124, 21},{215, 138, 11},{216, 77, 24},
      {217, 66, 29},{217, 28, 20},{218, 237, 13},{219, 140, 29},{221, 116, 22},
    {221, 212, 28},{221, 120, 28},{228, 209, 10},{228, 181, 19},{229, 25, 21},
    {231, 52, 20},{234, 142, 12},{235, 197, 16},{235, 132, 14},{241, 237, 25},
```

```
    {241, 177, 30},{243, 166, 24},{244, 77, 22},{244, 83, 11},{247, 155, 23},
                                                        {247, 242, 30}};

    for (int i = 0 ; i < Jumlah ; i++){
        aaa[i][0] = bbb[i][0];
        aaa[i][1] = bbb[i][1];
        aaa[i][2] = bbb[i][2];
        permintaan[i] = bbb[i][2];
        koordinatX[i] = bbb[i][0];
        koordinatY[i] = bbb[i][1];
        string = string + (i+1) + ") x : " + koordinatX[i] + "\t   y : " +
                                        koordinatY[i] + "\tp : " + permintaan[i] + "\n";
    }

    teks.setText(string);
    Pelanggan pelanggan = new Pelanggan (koordinatX, koordinatY, Jumlah);
    pelanggan.setVisible(true);
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Simulation().setVisible(true);
        }
    });
}

private javax.swing.JButton AGB;
private javax.swing.JButton AGG;
private javax.swing.JTextField generasiAGB;
private javax.swing.JTextField generasiAGG;
private javax.swing.JTextField hasilAGB;
private javax.swing.JTextField hasilAGG;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel31;
private javax.swing.JLabel jLabel35;
private javax.swing.JLabel jLabel36;
```

```
    private javax.swing.JLabel jLabel37;
    private javax.swing.JLabel jLabel38;
    private javax.swing.JLabel jLabel39;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel40;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLayeredPane jLayeredPane1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jumlah;
    private javax.swing.JTextField kapasitasAGB;
    private javax.swing.JTextField kapasitasAGG;
    private javax.swing.JLabel qqqq;
    private javax.swing.JButton random;
    private javax.swing.JTextField rata;
    private javax.swing.JTextArea teks;
    private javax.swing.JTextField x;
    private javax.swing.JTextField y;
}
```

## Lampiran 21

*Source Code* dari Pelanggan.java

```
package cvrp;

import java.awt.*;
import javax.swing.*;

class Surface extends JPanel{
   private int[] X;
   private int[] Y;
   private int Jumlah;

   public Surface (int[] x, int[] y, int jumlah){
      X = new int [jumlah];
      Y = new int [jumlah];
      X = x;
      Y = y;
      Jumlah = jumlah;
   }

   private void doDrawing(Graphics g){
      Graphics g2d = (Graphics2D) g;
      for (int i = 0 ; i < Jumlah ; i++){
         g2d.drawLine(X[i],Y[i],X[i],Y[i]);
         g2d.drawLine(X[i]-1,Y[i],X[i]-1,Y[i]);
         g2d.drawLine(X[i]+1,Y[i],X[i]+1,Y[i]);
         g2d.drawLine(X[i],Y[i]-1,X[i],Y[i]-1);
         g2d.drawLine(X[i],Y[i]+1,X[i],Y[i]+1);
         g2d.drawLine(X[i]-1,Y[i]-1,X[i]-1,Y[i]-1);
         g2d.drawLine(X[i]-1,Y[i]+1,X[i]-1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]+1,X[i]+1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]-1,X[i]+1,Y[i]-1);
      }
   }

   @Override
   public void paintComponent(Graphics g){
      super.paintComponent(g);
      doDrawing(g);
   }
}
```

```
public class Pelanggan extends JFrame{
    public Pelanggan(int[] x, int[] y, int jumlah) {
        initUI(x, y, jumlah);
    }
    private void initUI(int[] x, int[] y, int jumlah){
        add(new Surface(x, y, jumlah));
        setTitle("Posisi Pelanggan");
        setSize(619, 641);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

## Lampiran 22

*Source Code* dari PelangganAG.java

```
package cvrp;

import java.awt.*;
import javax.swing.*;

class SurfaceAGB extends JPanel{
   private int[] X;
   private int[] Y;
   private int[][] XY;
   private int Jumlah;

   public SurfaceAGB (int[] x, int[] y, int jumlah, int[][] xy){
      X = new int [jumlah];
      Y = new int [jumlah];
      XY = xy;
      X = x;
      Y = y;
      Jumlah = jumlah;
   }

   private void doDrawing(Graphics g){
      Graphics g2d = (Graphics2D) g;

      for (int i = 0 ; i < Jumlah ; i++){
         g2d.drawLine(X[i],Y[i],X[i],Y[i]);
         g2d.drawLine(X[i]-1,Y[i],X[i]-1,Y[i]);
         g2d.drawLine(X[i]+1,Y[i],X[i]+1,Y[i]);
         g2d.drawLine(X[i],Y[i]-1,X[i],Y[i]-1);
         g2d.drawLine(X[i],Y[i]+1,X[i],Y[i]+1);
         g2d.drawLine(X[i]-1,Y[i]-1,X[i]-1,Y[i]-1);
         g2d.drawLine(X[i]-1,Y[i]+1,X[i]-1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]+1,X[i]+1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]-1,X[i]+1,Y[i]-1);
      }

      for (int i = 0 ; i < XY.length-1 ; i++){
         g2d.drawLine(XY[i][0],XY[i][1],XY[i+1][0],XY[i+1][1]);
      }
   }
```

```
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        doDrawing(g);
    }
}

public class PelangganAG extends JFrame{
    public PelangganAG(int[] x, int[] y, int jumlah, int[][] xy) {
        initUI(x, y, jumlah, xy);
    }
    private void initUI(int[] x, int[] y, int jumlah, int[][] xy){
        add(new SurfaceAGB(x, y, jumlah, xy));
        setTitle("Hasil Algoritma Genetika Biasa");
        setSize(619, 641);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

## Lampiran 23

*Source Code* dari PelangganAGG.java

```
package cvrp;

import java.awt.*;
import javax.swing.*;

class SurfaceAGG extends JPanel{
   private int[] X;
   private int[] Y;
   private int[][] XY;
   private int Jumlah;

   public SurfaceAGG (int[] x, int[] y, int jumlah, int[][] xy){
      X = new int [jumlah];
      Y = new int [jumlah];
      XY = xy;
      X = x;
      Y = y;
      Jumlah = jumlah;
   }

   private void doDrawing(Graphics g){
      Graphics g2d = (Graphics2D) g;

      for (int i = 0 ; i < Jumlah ; i++){
         g2d.drawLine(X[i],Y[i],X[i],Y[i]);
         g2d.drawLine(X[i]-1,Y[i],X[i]-1,Y[i]);
         g2d.drawLine(X[i]+1,Y[i],X[i]+1,Y[i]);
         g2d.drawLine(X[i],Y[i]-1,X[i],Y[i]-1);
         g2d.drawLine(X[i],Y[i]+1,X[i],Y[i]+1);
         g2d.drawLine(X[i]-1,Y[i]-1,X[i]-1,Y[i]-1);
         g2d.drawLine(X[i]-1,Y[i]+1,X[i]-1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]+1,X[i]+1,Y[i]+1);
         g2d.drawLine(X[i]+1,Y[i]-1,X[i]+1,Y[i]-1);
      }

      for (int i = 0 ; i < XY.length-1 ; i++){
         g2d.drawLine(XY[i][0],XY[i][1],XY[i+1][0],XY[i+1][1]);
      }
   }
   @Override
```

```
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        doDrawing(g);
    }
}

public class PelangganAGG extends JFrame{
    public PelangganAGG(int[] x, int[] y, int jumlah, int[][] xy) {
        initUI(x, y, jumlah, xy);
    }
    private void initUI(int[] x, int[] y, int jumlah, int[][] xy){
        add(new SurfaceAGG(x, y, jumlah, xy));
        setTitle("Hasil Algoritma Genetika Ganda");
        setSize(619, 641);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

# BIODATA PENULIS

Penulis memiliki nama lengkap Muhammad Luthfi Shahab. Penulis lahir di Malang, pada tanggal 31 Maret 1995. Penulis telah menempuh pendidikan di SD Negeri Gondanglegi Wetan 1 (2001-2007), MTs Negeri Malang 3 (2007-2009), dan MA Negeri 3 Malang (2009-2011).

Setelah lulus MA, penulis mendaftar di Jurusan Matematika ITS melalui jalur SNMPTN undangan dan tercatat sebagai mahasiswa Matematika ITS dengan NRP 1211100047. Selama menempuh kuliah di Jurusan Matematika ITS, penulis pernah menjadi asisten dosen serta aktif di organisasi kemahasiswaan. Penulis pernah aktif di organisasi HIMATIKA-ITS sebagai staff DAGRI periode 2012-2013 dan staff SAINSTEK periode 2013-2014. Penulis juga pernah menjadi anggota Steering Committee Padamu Himatika periode 2013-2014.

Segala saran dan kritik yang membangun selalu penulis harapkan untuk kebaikan ke depannya. Penulis dapat dihubungi di luthfi.shahab@gmail.com.