TUGAS AKHIR – SM141501

ADAPTASI ALGORITMA *HARMONY SEARCH* UNTUK MENYELESAIKAN *CAPACITATED VEHICLE ROUTING PROBLEM* (CVRP) DENGAN PERMINTAAN DINAMIS

DINAN FAKHRANA RAMADHANI
NRP 1213 100 111

Dosen Pembimbing
Prof. Dr. Mohammad Isa Irawan, MT

DEPARTEMEN MATEMATIKA
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Teknologi Sepuluh Nopember

FINAL PROJECT – SM141501

ADAPTATION OF THE HARMONY SEARCH ALGORITHM TO SOLVE CAPACITATED VEHICLE ROUTING PROBLEM (CVRP) WITH DYNAMIC DEMANDS

DINAN FAKHRANA RAMADHANI
NRP 1213 100 111

Supervisor
Prof. Dr. Mohammad Isa Irawan, MT

DEPARTMENT OF MATHEMATICS
Faculty of Mathematics and Natural Sciences
Institut Teknologi Sepuluh Nopember
Surabaya 2017

LEMBAR PENGESAHAN

ADAPTASI ALGORITMA *HARMONY SEARCH* UNTUK MENYELESAIKAN *CAPACITATED VEHICLE ROUTING PROBLEM* (CVRP) DENGAN PERMINTAAN DINAMIS

ADAPTATION OF THE HARMONY SEARCH ALGORITHM TO SOLVE CAPACITATED VEHICLE ROUTING PROBLEM (CVRP) WITH DYNAMIC DEMANDS

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
Untuk memperoleh gelar Sarjana Sains
Pada bidang studi Ilmu Komputer
Program Studi S-1 Departemen Matematika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Institut Teknologi Sepuluh Nopember Surabaya

Oleh :
DINAN FAKHRANA RAMADHANI
NRP. 1213100111

Menyetujui,
Dosen Pembimbing,

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

Mengetahui,
Kepala Departemen Matematika
FMIPA ITS

Dr. Imam Mukhlash, S.Si, MT
NIP. 19700831 199403 1 003

Surabaya, Juli 2017

ADAPTASI ALGORITMA *HARMONY SEARCH* UNTUK MENYELESAIKAN *CAPACITATED VEHICLE ROUTING PROBLEM* (CVRP) DENGAN PERMINTAAN DINAMIS

Nama Mahasiswa	: Dinan Fakhrana Ramadhani
NRP	: 1213 100 111
Departemen	: Matematika
Dosen Pembimbing	: Prof. Dr. M. Isa Irawan, MT

Abstrak

Capacitated vehicle routing problem (CVRP) merupakan permasalahan pencarian rute terbaik untuk mendistribusikan barang kepada sejumlah pelanggan menggunakan kendaraan yang memiliki kapasitas terbatas. CVRP merupakan permasalahan *np-hard* yang berarti diperlukan suatu metode untuk dapat menghasilkan solusi penyelesaian yang optimal dengan waktu komputasi lebih cepat. Pertama, pada penelitian ini terdapat dua kasus, kasus pertama adalah CVRP dengan letak depot berada di pinggir dan kasus kedua letak depotnya berada ditengah dengan masing-masing memiliki koordinat depot, koordinat 70 pelanggan, kapasitas kendaraan, dan permintaan setiap pelanggan. Kemudian, dicari rute terpendek untuk melayani pelanggan yang memiliki permintaan. Pada penelitian ini, CVRP dengan permintaan dinamis diselesaikan dengan algoritma *harmony search,* GA-*based clustering algorithm*, dan algoritma *hybrid harmony search.* Hasil yang didapatkan yaitu algortima *hybrid harmony search* dapat digunakan untuk menyelesaikan CVRP dengan permintaan dinamis dan menghasilkan solusi yang baik dibandingkan dengan algoritma *harmony search,* dan GA-*based clustering algorithm.*

Kata Kunci : CVRP, permintaan dinamis, algoritma *harmony search,* algoritma genetika, algoritma *hybrid harmony search.*

ADAPTATION OF THE HARMONY SEARCH ALGORITHM TO SOLVE CAPACITATED VEHICLE ROUTING PROBLEM (CVRP) WITH DYNAMIC DEMANDS

Name of Student	: Dinan Fakhrana Ramadhani
NRP	: 1213 100 111
Department	: Mathematics
Supervisor	: Prof. Dr. M. Isa Irawan, MT

Abstract

Capacitated vehicle routing problem (CVRP) is a matter of finding the best route to distribute goods to a number of customers using vehicles with limited capacity. CVRP is a np-hard problem which means a method is needed that can produce an optimal solution with faster computation time. First, in this research there are two cases, the first case is the CVRP with the location of the depot is on the edge and the second case where the depot is located in the middle. Each case contains coordinate of depot, coordinate of 70 customers, vehicle capacity, and demand of each customers. Then, search for the shortest route to serve customers who have requests. In this study, CVRP with dynamic demands solved by GA-based clustering algorithm, harmony search algorithm and hybrid harmony search algorithm. The results is hybrid harmony search algorithm can be used to solved CVRP with dynamic demands and produce a good solution, while harmony search algorithm and GA-based clustering can not produce a good solution.

Keyword* : *CVRP, dynamic demands, harmony search algorithm, genetic algorithm, hybrid harmony search algorithm.

KATA PENGANTAR

Segala puji syukur penulis panjatkan ke hadirat Allah SWT, karena dengan ridho-Nya penulis dapat menyelesaikan Tugas Akhir dengan judul

"ADAPTASI ALGORITMA *HARMONY SEARCH* UNTUK MENYELESAIKAN *CAPACITATED VEHICLE ROUTING PROBLEM* (CVRP) DENGAN PERMINTAAN DINAMIS"

yang merupakan salah satu persyaratan akademis dalam menyelesaikan Program Sarjana Matematika, Fakultas Matematika dan Ilmu Pengethuan Alam, Institut Teknologi Sepuluh Nopember, Surabaya.

Tugas Akhir ini dapat diselesaikan dengan baik berkat kerja sama, bantuan, dan dukungan dari banyak pihak. Sehubungan dengan hal tersebut, penulis ingin mengucapkan terima kasih kepada :

1. Bpk. Dr. Imam Mukhlash, S.Si, MT selaku Kepala Departemen Matematika ITS.
2. Bpk. Darmaji, S.Si, MT selaku Dosen Wali yang telah memberikan arahan akademik selama penulis menempuh pendidikan di Departemen Matematika ITS.
3. Prof. Dr. Mohammad Isa Irawan, MT selaku Dosen Pembimbing yang telah membimbing penulis dalam mengerjakan Tugas Akhir ini sehingga dapat terselesaikan dengan baik.
4. Dr. Didik Khusnul Arif, S.Si, M.Si selaku Ketua Program Studi S1 Departemen Matematika ITS.
5. Drs. Iis Herisman, M.Si selaku Sekretaris Program Studi S1 Departemen Matematika ITS.
6. Seluruh jajaran dosen dan staf Departemen Matematika ITS.
7. Keluarga tercinta yang senantiasa memberikan dukungan dan do'a yang tak terhingga.
8. Teman-teman angkatan 2013.

9. Semua pihak yang tidak bisa penulis sebutkan satu-persatu, terima kasih telah membantu sampai terselesaikannya Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan. Oleh karena itu, penulis mengharapkan kritik dan saran dari pembaca. Akhir kata, semoga Tugas Akhir ini dapat bermanfaat bagi semua pihak yang berkepentingan.

Surabaya, Juli 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Pada bab ini dibahas mengenai latar belakang yang mendasari penulisan Tugas Akhir ini. Di dalamnya mencakup identifikasi permasalahan pada topik Tugas Akhir kemudian dirumuskan menjadi permasalahan yang diberikan batasan-batasan dalam pembahasan pada Tugas Akhir ini.

1.1 Latar Belakang

Permasalahan distribusi produk atau barang merupakan permasalahan yang sangat krusial di bidang industri. Perusahaan seperti PT. PERTAMINA, PT. Sumber Alfaria Trijaya, PT. Frisian Flag Indonesia (FFI), dan PT. Toyota Astra Motor melibatkan pengelolaan armada kendaraan untuk distribusi barang. Salah satu permasalahan pada perusahaan tersebut adalah menentukan rute optimal kendaraan dalam mengirimkan barang ke pelanggan guna mengendalikan biaya operasi. Permasalahan tersebut disebut juga *Vehicle Routing Problem* (VRP) [1].

VRP merupakan permasalahan penentuan rute kendaraan yang digunakan untuk megirimkan barang dari depot ke pelanggan. Salah satu kategori pada VRP yaitu *capacitated vehicle routing problem* (CVRP). CVRP merupakan permasalahan penentuan rute optimal yang digunakan oleh armada kendaraan untuk melayani sejumlah pelanggan dengan dikenakan kapasitas maksimum pada kendaraan. CVRP termasuk *np-hard problem* [2], sehingga diperlukan suatu metode untuk menghasilkan solusi penyelesaian yang optimal dengan waktu komputasi lebih cepat.

CVRP diperkenalkan pada tahun 1959, sehingga banyak algoritma yang telah dikembangkan untuk menyelesaikan permasalahan ini. Penelitian yang telah dilakukan untuk menyelesaikan *capacitated vehicle routing problem* diantaranya menggunakan *particle swarm optimization algorithm* [3], *bee colony algorithm* [4], *branch and cut algorithm* [5], SR-GCWS *hybrid algorithm* [6], algoritma genetika ganda [7], dan *harmony*

search algorithm [8]. CVRP yang diselesaikan dengan algoritma genetika ganda dan algoritma *harmony search* masing-masing memberikan hasil yang baik.

Pada permasalahan yang nyata dalam perusahaan, adakalanya perusahaan yang memiliki N cabang, namun permintaan hanya berasal dari n cabang, sehingga operator hanya perlu mempertimbangkan posisi dari n cabang saja. Hal tersebut dilakukan untuk mengurangi biaya pengiriman. Permasalahan tersebut menjadi landasan pada penelitian ini untuk menyelesaikan CVRP dengan permintaan yang dinamis. Pada Tugas Akhir yang saya kerjakan, CVRP diselesaikan dengan menggunakan GA-*based clustering algorithm*, algoritma *harmony search,* dan algoritma *hybrid harmony search* untuk mendapatkan hasil yang baik.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, dapat ditarik rumusan masalah yaitu bagaimana menyelesaikan CVRP dengan permintaan dinamis menggunakan GA-*based clustering algorithm*, algoritma *harmony search,* dan algoritma *hybrid harmony search.*

1.3 Batasan Masalah

Pada penelitian ini, penulis membuat batasan masalah sebagai berikut :

1. Data yang digunakan bukan data riil, tetapi berupa titik-titik koordinat dalam bidang 2 dimensi.
2. Titik-titik tujuan CVRP diatur agar menyebar di sekitar depot.
3. Setiap titik tujuan dikunjungi tepat satu kali.
4. Jenis CVRP yang digunakan adalah CVRP simetri yang berarti bahwa jarak perjalanan pergi dan pulang antara dua titik adalah sama.
5. Armada kendaraan yang digunakan memiliki kapasitas maksimum yang sama (*homogenous fleet).*

6. Terdapat satu depot sebagai titik awal dan kembalinya rute semua kendaraan dalam melakukan pengiriman.
7. Tidak ada batasan waktu dan total jarak pada setiap rute.
8. Permintaan pelanggan merupakan bilangan bulat antara 0 sampai 4.
9. Pada algoritma *hybrid harmony search*, tahap pertama adalah GA-*based clustering algorithm* dan tahap kedua adalah algoritma *harmony search.*

1.4 Tujuan

Berdasarkan permasalahan yang telah dirumuskan sebelumnya, tujuan dari Tugas Akhir ini adalah untuk menyelesaikan CVRP dengan permintaan yang dinamis menggunakan GA-*based clustering algorithm*, algoritma *harmony search*, dan algoritma *hybrid harmony search*.

1.5 Manfaat

Manfaat yang dapat diperoleh dari Tugas Akhir ini adalah sebagai referensi bagi penelitian selanjutnya yang bertujuan untuk menyelesaikan CVRP dengan permintaan dinamis.

1.6 Sistematika Penulisan Tugas Akhir

Sistematika dari penulisan Tugas Akhir ini adalah sebagai berikut :

1. BAB I PENDAHULUAN
   Bab ini menjelaskan tentang gambaran umum dari penulisan Tugas Akhir ini yang meliputi latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Bab ini berisi tentang materi-materi yang mendukung Tugas Akhir ini, antara lain penelitian terdahulu, *capacitated vehicle routing problem*, *harmony search algorithm*, algoritma genetika, GA-*based clustering algorithm*, *uniform crossover*, dan *roulette wheel selection*.

3. BAB III METODOLOGI PENELITIAN
   Pada bab ini dibahas tentang langkah – langkah dan metode yang digunakan untuk menyelesaikan Tugas Akhir ini.
4. BAB IV PERANCANGAN IMPLEMENTASI
   Pada bab ini akan menguraikan bagaimana tahapan-tahapan dalam perancangan implementasi.
5. BAB V IMPLEMENTASI
   Bab ini menjelaskan mengenai implementasi dari GA-*based clustering algorithm*, algoritma *harmony* search, dan algortima *hybrid harmony search* untuk menyelesaikan *capacitated vehicle routing problem* dengan permintaan yang dinamis. Setelah itu, dilakukan analisis terhadap hasil implementasi. Analisis ini bertujuan untuk melihat apakah algoritma tersebut dapat diterapkan untuk menyelesaikan CVRP dengan permintaan yang dinamis.
6. BAB VI PENUTUP
   Bab ini berisi kesimpulan yang diperoleh dari pembahasan masalah sebelumnya serta saran yang diberikan untuk pengembangan selanjutnya.

BAB II
TINJAUAN PUSTAKA

Pada bab ini dibahas mengenai dasar teori yang digunakan dalam penyusunan Tugas Akhir ini. Dasar teori yang dijelaskan dibagi menjadi beberapa subbab yaitu penelitian terdahulu, *Capacitated Vehicle Routing Problem*, *Harmony Search Algorithm*, Algoritma Genetika, GA-*Based Clustering Algorithm*, *Uniform Crossover,* dan *Roulette Wheel Selection*.

2.1 Penelitian Terdahulu

Penelitian terdahulu yang digunakan dalam Tugas Akhir ini adalah penelitian yang relevan dengan tema yang diambil yaitu penelitian pada tahun 2013 oleh Pichpibul, T. dan Kawtummachai, R. [8]. HS yang dimodifikasi terdiri dari dua tahapan yaitu tahap pertama yaitu dalam proses menginisialisasi *harmony memory* dimana terdapat solusi dari penelitian sebelumnya menggunakan *improved Clarke-Wright savings algorithm* dan kedua yaitu adanya prosedur *roulette wheel selection* dalam mekanisme improvisasi harmoni baru untuk meningkatkan proses seleksi.

Selain itu, penelitian pada tahun 2000 oleh Maulik dan Bandyopadhyay [9] yang berjudul *Genetic Algorithm-Based Clustering Technique* mengenai *clustering* menggunakan *genetic algorithm.* Pada penelitian tersebut performansi dari GA-*based clustering algorithm* dibandingkan dengan algoritma K-*means* pada 10 *data sets* yang menghasilkan kesimpulan bahwa GA-*based clustering algorithm* menghasilkan performansi yang lebih unggul daripada algoritma K-*means* untuk *data set* yang digunakan.

Penelitian selanjutnya yang berkaitan dengan Tugas Akhir ini adalah penelitian yang dikerjakan oleh Shahab, M. L. dan Irawan M. I. [7] tentang penyelesaian CVRP dengan menggunakan algoritma genetika ganda. Algoritma genetika ganda terlebih dahulu mendekomposisi CVRP menjadi beberapa daerah dan kemudian mencari rute terpendek pada setiap daerah dengan menggunakan dua algoritma genetika sederhana yang berbeda.

Kemudian solusi algoritma genetika ganda dibandingkan dengan algoritma genetika dengan hasil bahwa algortima genetika ganda lebih baik dari algoritma genetika dalam menyelesaikan CVRP dalam segi waktu dan jarak.

Namun, pada penelitian yang dilakukan oleh Pichpibul dan Kawtummachai, serta Shahab, M. L. dan Irawan M. I. jumlah pelanggan yang didefinisikan selalu sama dengan jumlah pelanggan yang akan dikunjungi. Sedangkan, pada permasalahan yang nyata, tidak semua pelanggan melakukan permintaan pada waktu yang sama. Sehingga, tidak perlu mengunjungi pelanggan yang tidak memiliki permintaan. Berdasarkan hal tersebut, dalam pencarian rute, lokasi pelanggan yang tidak memiliki permintaan sebaiknya tidak perlu dipertimbangkan. Oleh karena itu, berdasarkan penelitian sebelumnya, dalam penelitian ini, CVRP dengan permintaan yang dinamis akan diselesaikan dengan menggunakan GA-*based clustering algorithm*, algoritma *harmony search*, dan algoritma *hybrid harmony search*. Dimana algoritma *hybrid harmony search* menggabungkan GA-*based clustering algorithm* yang digunakan untuk mendekomposisi CVRP menjadi beberapa daerah dengan algortima *harmony search* yang digunakan untuk *routing* pelanggan pada masing-masing daerah.

2.2 *Capacitated Vehicle Routing Problem*

Vehicle Routing Problem merupakan salah satu permasalahan yang banyak dipelajari diantara permasalahan optimisasi. VRP berkaitan dengan penentuan rute optimal yang digunakan oleh kendaraan, berasal dari satu atau lebih depot, untuk melayani sejumah pelanggan [10]. VRP dapat diklasifikasikan menjadi 10 kategori [11], yaitu: (1) *Deterministic Vehicle Routing Problem* (DVRP), (2*) Multi-depot* VRP (MDVRP), (3) VRP *with Time Windows* (VRPWTW), (4) *Stochastic* VRP (SVRP), (5*) Capacitated* VRP (CVRP), (6) Fuzzy VRP (FVRP), (7) *Location* VRP (LVRP), (8) *Facility Location* VRP (FLVRP), (9) *Backhauling* VRP (BHVRP), dan (10) *Inventory Control* VRP (ICVRP).

Capacitated vehicle routing problem (CVRP) merupakan bagian dari VRP yang mengundang perhatian dari banyak peneliti [11]. Pada CVRP, jumlah pelanggan sama dengan jumlah pengiriman, permintaannya terdefinisi, diketahui di awal, dan tidak dapat dibagi, kendaraan yang digunakan identik, berangkat dari satu depot yang sama, dan terdapat kapasitas maksimum. Tujuannya yaitu untuk meminimalkan total biaya (terhitung dari jumlah dan jarak rute atau waktu perjalanannya) untuk melayani semua pelanggan. Umumnya, biaya perjalanan dari setiap pasang lokasi pelanggan adalah sama dalam kedua arah, yang menyebabkan matriks biaya simetris, namun dalam beberapa aplikasi, seperti penditribusian barang di daerah perkotaan dengan jalanan *one-way*, matriks biayanya asimetris. [8]

CVRP dapat dideskripsikan sebagai permasalahan teori graf. 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, yang mengunakan m kendaraan dengan kapasitas yang sama Q untuk melayani permintaan q_i dari n pelanggan, $i = 1,2, \dots, n$ yang direpresentasikan dengan n *vertex* yaitu $v_1, \dots, v_n$. 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 $v_{i,j} \in V$, k_i merupakan banyaknya pelanggan dalam rute R_i dan $v_{i,0} = v_{i,k_i+1} = v_0$ (0 menotasikan depot), untuk setiap kluster R_i haruslah memenuhi $\sum_{j=1}^{k_i} q_j \leq Q$ yang berarti bahwa total permintaan pada setiap rute tidak boleh melebihi kapasitas kendaraan. Total jarak dari solusi permasalahan adalah total dari jarak rute R_i yang didefinisikan sebagai berikut:

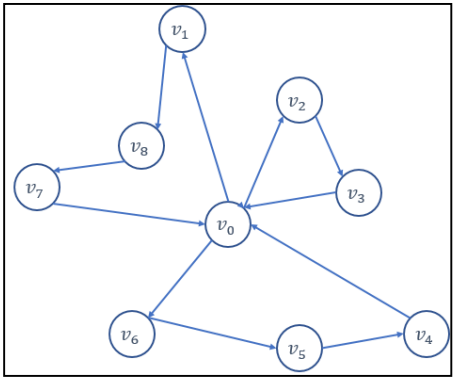
$$Jarak = \sum_{i=1}^{m} Jarak\ (R_i) = \sum_{i=1}^{m} \sum_{j=0}^{k_i} c_{(i,j)(i,j+1)}$$

CVRP bertujuan untuk menentukan himpunan dari m rute yang total jaraknya minimum, 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. [7][12]

Misalkan dicontohkan CVRP sebagai berikut, terdapat 8 pelanggan $(v_1, v_2, \dots, v_8)$ yang secara urut masing-masing permintaannya yaitu 1, 2, 3, 1, 1, 3, 2, dan 2. Terdapat 1 depot (v_0), dengan kapasitas maksimum kendaraan yaitu 5. Kemudian setelah diselesaikan didapatkan 3 kluster dengan rincian sebagai berikut:

Kluster 1	=	v_1	v_8	v_7								
Kluster 2	=	v_2	v_3									
Kluster 3	=	v_4	v_5	v_6								
Rute	=	0	v_1	v_8	v_7	0	v_2	v_3	0	v_4	v_5	v_6	0

Representasi rute CVRP dapat dilihat pada gambar 2.1



Gambar 2.1 Rute dari Contoh CVRP

2.3 *Harmony Search Algorithm*

Algoritma *Harmony Search* pertama kali dikemukakan oleh Zong Woo Geem, Joong Hoon Kim, dan G. V. Loganathan pada tahun 2001 yang terinspirasi dari fenomena buatan yaitu harmoni musik. Pada tabel 2.1 Wang, Gao, & Zenger menjelaskan

bagaimana proses improvisasi harmoni dapat merepresentasikan optimisasi [13].

Tabel 2.2.1 Perbandingan Improvisasi Harmoni dan Optimisasi

Faktor perbandingan	Improvisasi harmoni	Optimisasi
Target	Standar estetika	Fungsi objektif
Kondisi terbaik	Harmoni yang fantastis	Optimum global
Komponen	Nada pada instrumen	Nilai variabel
Unit proses	Setiap latihan	Setiap iterasi

Improvisasi musik adalah sebuah proses pencarian harmoni yang lebih baik dengan mencoba berbagai kombinasi nada yang mengikuti salah satu dari 3 aturan [13], yaitu:

1. Memainkan satu nada dari memorinya;
2. Memainkan nada yang berdekatan dengan salah satu nada dalam memorinya;
3. Memainkan nada acak dari rentang suara yang mungkin.

Proses tesebut ditiru dalam seleksi variabel pada algoritma *harmony search.* Demikian pula, seleksi pada algoritma *harmony search* harus mengikuti salah satu dari 3 aturan berikut:

1. Memilih salah satu nilai dari *harmony memory* (yang didefinisikan sebagai *memory considerations*);
2. Memilih nilai yang berdekatan dengan salah satu nilai dari *harmony search memory* (yang didefinisikan sebagai *pitch adjusment*);
3. Memilih nilai secara acak dari rentang nilai yang mungkin (didefinisikan sebagai *randomization*).

Langkah-langkah dari algoritma *harmony search* adalah sebagai berikut [14]:

1. Inisialisasi parameter permasalahan optimisasi dan parameter algoritma.
2. Inisialisasi *harmony memory* (HM).
3. Improvisasi sebuah harmoni baru dari HM.

4. Perbaharui HM.
5. Ulangi langkah ke 3 dan 4 sampai kriteria pemberhentian terpenuhi.

Langkah 1: Inisialisasi paremeter permasalahan optimisasi dan parameter algoritma. Parameter algoritma HS yang dibutuhkan untuk menyelesaikan permasalahan optimisasi juga di spesifikasikan pada langkah ini: *harmony memory size* (jumlah vektor solusi dalam *harmony memory*, HMS), *harmony memory considering rate* (HMCR), *pitch adjusting rate* (PAR), dan kriteria pemberhentian (jumlah maksimum pencarian), HMCR dan PAR merupakan parameter yang digunakan untuk meningkatkan vektor solusi.

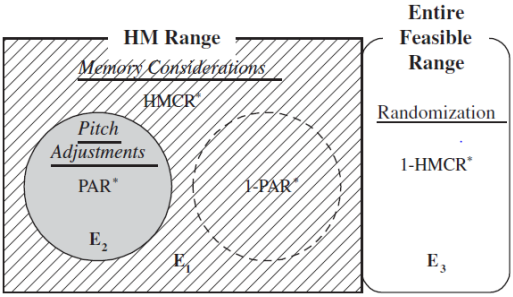
Langkah 2: Inisialisasi *harmony memory* (HM). Pada langkah kedua, matriks HM berisi vektor solusi yang dibentuk secara acak dan diurutkan oleh nilai dari fungsi objektif.

$$HM = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^{HMS} \end{bmatrix}$$

Langkah 3: Improvisasi harmoni baru dari HM. Pada langkah ketiga, sebuah vektor harmoni baru, $x' = (x_1', x_2', \dots, x_N')$ berdasarkan *memory considerations, pitch adjustments,* dan *randomization*. Hal tersebut memungkinkan untuk memilih nilai yang baru menggunakan parameter HMCR, yang berkisar antara 0 sampai 1. Apabila bilangan acak yang dibangkitkan $r_1 <$ HMCR maka lakukan tahap *memory considerations* dengan probabilitas HMCR, jika tidak maka lakukan dengan tahap *randomization* dengan probabilitas $(1 -$ HMCR$)$.

HMCR adalah probabilitas dalam memilih satu nilai yang tersimpan pada HM, dan (1-HMCR) adalah peluang dari pemilihan satu nilai secara acak tidak terbatas untuk nilai yang tersimpan di HM. Setiap komponen dalam vektor harmoni yang baru, $x' = (x_1', x_2', \dots, x_N')$ yang melalui proses *memory consideration* diperiksa untuk menentukan apakah harus melalui proses *pitch adjusment*. Proses ini menggunakan parameter PAR yang menentukan tingkat

penyesuaian untuk *pitch* yang terpilih dari tahap *memory considerations*, yang digambarkan sebagai berikut:



Gambar 2.2 Konsep Improvisasi *Harmony* Baru [14]

Apabila bilangan acak yang dibangkitkan $r_2 <$ PAR maka lakukan *pitch adjusting* dengan probabilitas PAR. Proses *pitch adjusting* hanya dilakukan setelah satu nilai dipilih dari HM. Nilai (1-PAR) artinya tidak melakukan proses *pitch adjusting*. Proses *pitch adjusting* yaitu memilih satu nilai yang berdekatan dengan nilai yang terpilih pada tahap *memory considerations*.

Langkah 4: Memperbaharui HM. Pada langkah ke-empat, apabila nilai fungsi objektif vektor harmoni baru lebih baik daripada harmoni yang paling buruk pada HM, harmoni baru dimasukkan ke dalam HM dan harmoni yang buruk dikeluarkan dari HM.

Langkah 5: Pada langkah kelima, komputasi dihentikan ketika kriteria pemberhentian terpenuhi. Jika belum, ulangi langkah 3 dan 4.

2.4 Algoritma Genetika

Algoritma genetika (AG) ditemukan oleh John Holland pada tahun 1960an dan dikembangkan oleh Holland, muridnya, dan koleganya di University of Michigan pada tahun 1960an dan 1970an. [15]

Unsur – unsur dalam algoritma genetika yaitu populasi kromosom, *selection* berdasarkan *fitness value*, *crossover* untuk

menghasilkan *offspring* baru, dan *random mutation* dari *offspring* baru. Setiap kromosom merupakan kandidat solusi. AG biasanya membutuhkan fungsi *fitness* yang memberikan skor (*fitness*) pada setiap kromosom pada populasi. *Fitness* dari suatu kromosom bergantung pada seberapa baik kromosom menyelesaikan permasalahan. [15][16]

Bentuk algoritima genetika yang paling sederhana melibatkan 3 jenis operator: *selection, crossover* (*single point*) dan *mutation.* [15]

Selection Operator ini memilih kromosom pada populasi untuk reproduksi. Semakin baik kromosomnya, semakin besar kemungkinan dipilih untuk bereproduksi.

Crossover Operator ini memilih lokus secara acak dan menukar urutan sebelum dan sesudah lokus antara 2 kromosom untuk menghasilkan 2 keturunan. Misalnya, 100000100 dan 111111111 dapat disilangkan setelah lokus ketiga pada masing-masing kromosom untuk menghasilkan 2 keturunan 100111111 dan 111000100.

Mutation Operator ini membalik beberapa bit didalam kromosom. Misalnya, 00000100 salah satu kemungkinannya yaitu bermutasi pada posisi kedua untuk menghasilkan 01000100.

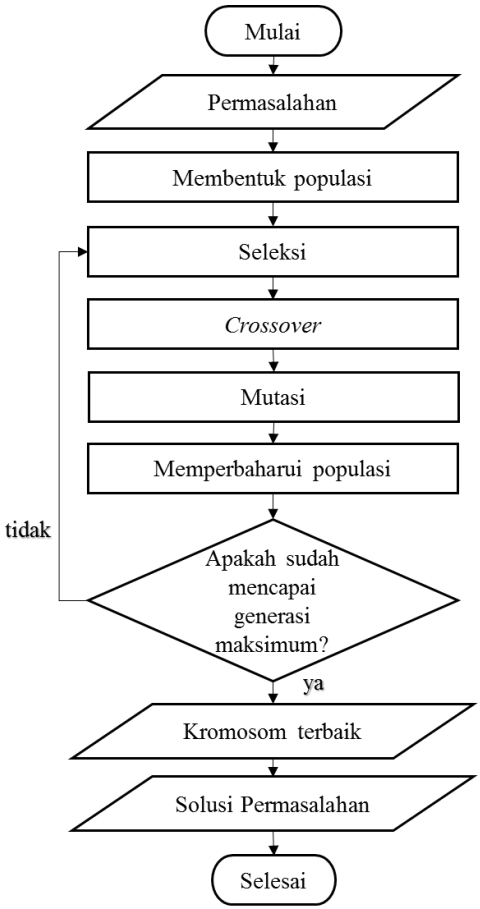
Langkah-langkah dari algoritma genetika sederhana yaitu:

1. Mulai dengan membangkitkan populasi secara acak yang berisi n kromosom.
2. Menghitung nilai *fitness* $f(x)$ dari masing-masing x kromosom pada populasi.
3. Membangkitkan kromosom baru.
   a. Pilih sepasang *parent chromosomes* pada populasi berdasarkan probabilitas seleksi.
   b. Dengan probabilitas p_c (probabilitas *crossover*), silangkan sepasang kromosom pada titik yang terpilih secara acak (dipilih dengan probabilitas yang seragam) untuk membentuk keturunan.

c. Mutasi keturunan pada masing-masing lokus dengan probabilitas p_m (probabiltas mutasi).

4. Memperbaharui populasi.
5. Kembali ke langkah 3 hingga kriteria pemberhentian terpenuhi.

Berikut merupakan diagram alir algoritma genetika.



Gambar 2.3 Langkah – Langkah Algoritma Genetika

2.4.1 GA-*Based Clustering Algorithm*

Langkah-langkah GA- *based clustering algorithm* adalah sebagai berikut [9] :

1. Inisialisasi populasi
   Pusat K kluster diinisialisasi secara acak dengan K titik yang terpilih pada data. Proses ini diulang untuk masing-masing P kromosom pada populasi, dimana P merupakan ukuran populasi.
2. Komputasi *fitness*
   Pada tahap pertama, kluster dibentuk dengan mempertimbangkan pusat kluster. Hal ini dilakukan dengan menetapkan masing-masing titik $x_i, i = 1,2, \dots, n$ ke dalam salah satu kluster C_j dengan pusat z_j, sehingga
   $\|x_i - z_j\| < \|x_i - z_p\|, p = 1,2, \dots K$ dan $p \neq j$.
   Setelah pengelompokkan selesai, kemudian hitung nilai *fitness* kromosom.
3. Tahapan selanjutnya adalah seleksi, *crossover*, dan mutasi seperti tahapan pada algoritma sederhana.

2.4.2 *Uniform Crossover*

Uniform crossover membuat setiap lokus menjadi titik yang dapat di *crossover*. Dibangkitkan secara acak sebuah *crossover mask* yang panjangnya sama dengan struktur individu dan paritas bit pada *mask* menunjukkan induk mana yang akan menurunkan bitnya. [17]

2.4.3 *Roulette Wheel Selection*

Pada *roulette wheel selection* probabilitas seleksi sebanding dengan nilai *fitness.* Analogi *roulette wheel* muncul karena seseorang dapat membayangkan seluruh populasi membentuk sebuah *roulette wheel* dengan ukuran slot individu sebanding dengan nilai *fitness.* Roda kemudian diputar dan bola dilempar masuk. Probabilitas bola yang akan berhenti di slot tertentu sebanding dengan ukuran slot tersebut dan demikian sesuai dengan *fitness value* yang bersangkutan. [16]

BAB III
METODE PENELITIAN

Pada bab ini dijelaskan objek, peralatan, dan langkah-langkah yang digunakan dalam penyusunan Tugas Akhir. Disamping itu, dijelaskan pula prosedur dan proses pelaksanaan setiap langkah yang dilakukan dalam menyelesaikan Tugas Akhir.

3.1 Objek dan Aspek Penelitian

Objek yang digunakan dalam penelitian ini adalah CVRP dengan permintaan dinamis yang meliputi koordinat depot, koordinat 70 pelanggan, permintaan setiap pelanggan, dan kapasitas kendaraan. Sedangkan aspek penelitian yang diteliti adalah menyelesaikan CVRP dengan permintaan dinamis. Algoritma yang digunakan yaitu GA-*based clustering algorithm*, algoritma *harmony search,* dan algoritma *hybrid harmony search.*

3.2 Peralatan

Penelitian ini menggunakan peralatan berupa perangkat keras dan perangkat lunak, antara lain:

1. Perangkat keras

   Perangkat keras yang digunakan laptop dengan spesifikasi:
   - Sistem operasi windows 8.1, 64-bit.
   - Prosesor Intel® Core™ i5-3337U CPU @ 1.80GHz.
   - RAM 4,00 GB.
2. Perangkat Lunak

   Perangkat lunak yang digunakan adalah NetBeans IDE 8.2.

3.3 Tahapan Penelitian

Langkah-langkah yang digunakan dalam menyelesaikan Tugas Akhir ini yang diringkas dalam Gambar 3.1 adalah sebagai berikut :

1. Studi Literatur

Studi Literatur ini dilakukan untuk identifikasi permasalahan dengan mencari referensi yang menunjang

penelitian yang berupa Tugas Akhir, jurnal internasional, buku, maupun artikel yang berhubungan dengan topik Tugas Akhir ini.

2. Implementasi

Implementasi pada penelitian ini dibuat dengan bahasa pemrograman Java dan menggunakan aplikasi NetBeans IDE 8.2. Pada tahap ini dilakukan proses dimana persoalan diterjemahkan melalui suatu rangkaian aktivitas sesuai model proses, metode, dan alat bantu yang digunakan. Berikut adalah tahapan dari proses implementasi :

a. Mendefinisikan CVRP

Pada tahap ini akan didefinisikan koordinat depot, koordinat pelanggan, permintaan pelanggan, dan kapasitas kendaraan.

b. Melakukan pengecekan terhadap permintaan pelanggan.

Setelah CVRP didefinisikan, kemudian dilakukan pengecekan permintaan pelanggan yang selanjutnya akan diputuskan pelanggan yang akan dikunjungi oleh kendaraan.

c. Menyelesaikan CVRP

Pada penelitian ini CVRP dengan permintaan dinamis akan diselesaikan dengan GA-*based clustering algorithm*, algoritma *harmony search,* dan algoritma *hybrid harmony search.*

d. Membuat representasi hasil implementasi

Setelah CVRP berhasil diselesaikan dengan masing-masing algoritma, kemudian akan dibuat graf untuk menggambarkan solusi percobaan dari masing-masing algoritma.

3. Pengujian

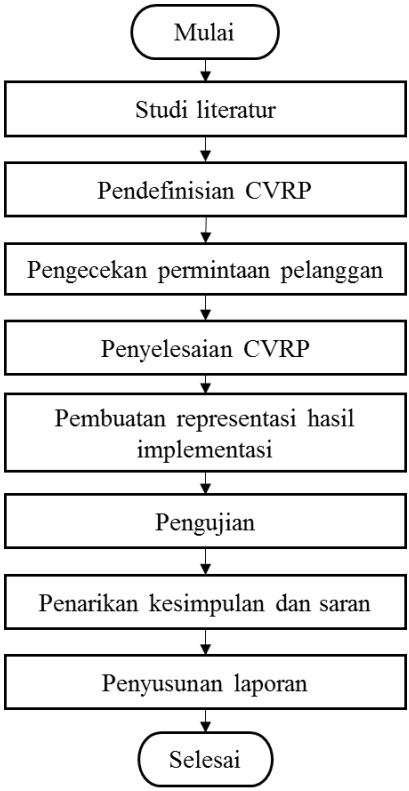
Selanjutnya, dilakukan pengujian untuk memeriksa apakah hasil implementasi sudah sesuai atau terjadi error. Pengujian dilakukan dengan 2 data CVRP yaitu *case 1* dan *case 2.* Dimana pada *case 1* letak depot berada dipinggir, dan pada *case 2* letak depotnya berada di tengah.

4. Penarikan Kesimpulan dan Saran

Setelah dilakukan pengujian, maka dapat ditarik suatu kesimpulan dan saran sebagai masukan untuk pengembangan penelitian lebih lanjut.

5. Penyusunan Laporan Tugas Akhir

Setelah semua proses selesai dilakukan maka tahap terakhir adalah penyusunan laporan Tugas Akhir.



Gambar 3.1 Diagram Alir Metode Penelitian

BAB IV
PERANCANGAN IMPLEMENTASI

Bab ini menjelaskan rancangan implementasi yang menggambarkan proses rancang bangun secara terperinci dan digunakan sebagai acuan untuk implementasi.

4.1 Perancangan Pendefinisian CVRP

Tahap ini bertujuan untuk mendefinisikan CVRP yang terdiri dari N pelanggan, depot, dan kapasitas maksimum kendaraan Q. Pelanggan dan depot didefinisikan sebagai $v_i,\ i = 0, \dots, N$ dimana v_0 merupakan depot. Selain itu, didefinisikan pula koordinat pelanggan $(x_i, y_i),\ i = 1, \dots, N$, koordinat depot (x_0, y_0), dan permintaan pelanggan (d_i). Sebagai contoh didefinisikan CVRP dengan koordinat depot (0,0), kapasitas kendaraan 5, dan 7 pelanggan dengan koordinat dan permintaan masing-masing pelanggan sebagai berikut :

Tabel 4.1 Contoh Data Pelanggan pada CVRP

Pelanggan	x	y	Permintaan
1	-2	3	1
2	0	3	2
3	1	1	2
4	3	0	0
5	0	-4	2
6	-1	-2	3
7	-5	0	0

4.2 Perancangan Pengecekan Permintaan

Pelanggan yang akan dikunjungi pada penelitian ini adalah sebanyak n, dengan n merupakan kumpulan pelanggan yang memiliki permintaan $(d_i > 0)$ dari keseluruhan pelanggan N. Sedangkan pelanggan yang tidak memiliki permintaan $(d_i = 0)$ diabaikan. Dengan contoh CVRP yang didefinisikan pada 4.1, hanya

pelanggan 1, 2, 3, 5, dan 6 saja yang masuk ke perhitungan selanjutnya. Sedangkan pelanggan 4 dan 7 diabaikan karena tidak memiliki permintaan.

4.3 Perancangan Perhitungan Jarak Terdekat Antara Pelanggan dan Depot.

Untuk menghitung jarak terdekat antar pelanggan dan depot digunakan rumus jarak euclidan sebagai berikut:

$$c_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Dengan $c_{i,j}$ merupakan jarak antara titik i ke titik j. Karena pada penelitian ini menggunakan CVRP simetris, maka jarak antara dua lokasi dalam kedua arah adalah sama atau $c_{i,j} = c_{j,i}$. Jarak $c_{i,j}$ inilah yang digunakan pada masing-masing algoritma pada penelitian ini. Berikut merupakan contoh perhitungan jarak antara v_1 dan v_2 dan jarak pelanggan dan depot dapat dilihat pada tabel 4.2.

$$\begin{aligned} c_{1,2} &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ &= \sqrt{((-2) - 0)^2 + (3 - 3)^2} \\ &= 2.0000 \end{aligned}$$

Tabel 4.2 Contoh Hasil Perhitungan Jarak Pelanggan dan Depot

	v_0	v_1	v_2	v_3	v_5	v_6
v_0	0	3.6056	3.0000	1.4142	4.0000	2.2361
v_1	3.6056	0	2.0000	3.6056	7.2801	5.0990
v_2	3.0000	2.0000	0	2.2361	7.0000	5.0990
v_3	1.4142	3.6056	2.2361	0	5.0990	3.6056
v_5	4.0000	7.2801	7.0000	5.0990	0	2.2361
v_6	2.2361	5.0990	5.0990	3.6056	2.2361	0

4.4 Perancangan Penyelesaian CVRP dengan *GA-Based Clustering Algorithm*

Langkah-langkah GA-*based clustering algorithm* untuk menyelesaikan CVRP dengan permintaan dinamis adalah sebagai berikut:

1. Inisialisasi parameter GA-*based clustering algorithm* yaitu *crossover rate*, *mutation rate*, *number of iteration*, dan *population size.* Sebagai contoh, untuk menyelesaikan CVRP yang didefinisikan pada 4.1, diberikan parameter *crossover rate* = 0.8, *mutation rate* = 0.6, *number of iteration* = 1, dan *population size* = 3.

2. Hitung jumlah kluster yang akan terbentuk dengan cara sebagai berikut :

$$K = \frac{\sum_{i=1}^{n} d_i}{Q}$$

dengan :
K = jumlah kluster
n = jumlah pelanggan yang mempunyai permintaan
d_i = permintaan pelanggan ke i
Q = kapasitas kendaraan
Pada contoh 4.1, jumlah kendaraan yang diperlukan adalah 2, dengan perhitungan sebagai berikut :

$$K = \frac{10}{5} = 2$$

3. Inisialisasi *K centroid* $z_{m,k}$, dengan $k = 1,2,\ldots K$ secara *random* sebanyak m kromosom.

Tabel 4.3 Contoh Inisialisasi Centroid

Kromosom ke-	*Center* ke-	Simbol	x	y
1	1	$z_{1,1}$	-2	3
	2	$z_{1,2}$	1	1

2	1	$z_{2,1}$	0	-4
	2	$z_{2,2}$	-1	-2
3	1	$z_{3,1}$	0	3
	2	$z_{3,2}$	1	1

4. Bentuk kluster pada kromosom. Kluster dibentuk berdasarkan *centroid* pada kromosom dengan cara memasukkan pelanggan $v_j,\ j = 1,2, \dots, l$ ke dalam kluster C_k dengan letak *centroid* terdekat. Sebelum membuat kluster, pelanggan diurutkan dari yang paling jauh dengan depot hingga yang paling dekat. Kemudian pelanggan dimasukkan ke dalam kluster dengan *centroid* terdekat. Apabila kluster yang paling dekat sudah penuh, maka pelanggan akan masuk ke kluster dengan *centroid* terdekat kedua dan seterusnya. Urutan pelanggan pada contoh CVRP pada 4.1 dapat dilihat pada tabel 4.4 dan hasil kluster dapat dilihat pada tabel 4.5.

Tabel 4.4 Urutan Pelanggan untuk Proses *Clustering*

Urutan	Pelanggan	Jarak ke Depot
1	5	4.0000
2	1	3.6056
3	2	3.0000
4	6	2.2361
5	3	1.4142

Tabel 4.5 Contoh Hasil Kluster dengan GA-Based Clustering Algorithm

Centroid	Pelanggan	Permintaan	Jarak
$z_{1,1}$	1	1	0
	2	2	2.0000
	3	3	3.6056
$z_{1,2}$	5	2	5.0990
	6	3	3.6056
$z_{2,1}$	5	2	0
	6	3	2.2361
$z_{2,2}$	1	1	2.2361
	2	2	5.0990
	3	2	3.6056
$z_{3,1}$	2	2	0
	6	3	5.0990
$z_{3,2}$	5	2	5.0990
	1	1	1.4142
	3	2	0

5. Kemudian setelah pelanggan masuk ke dalam masing-masing kluster, hitung t_d yaitu total jarak antara setiap

pelanggan ke masing-masing *center*-nya pada setiap kromosom.

Tabel 4.6 Contoh Total Jarak Setiap Kromosom dengan GA-Based Clustering Algorithm

Kromosom	*Center*	Jarak	Total Jarak
1	$z_{1,1}$	5.6056	14.3102
	$z_{1,2}$	8.7046	
2	$z_{2,1}$	2.2361	13.1768
	$z_{2,2}$	10.9407	
3	$z_{3,1}$	5.099	12.4341
	$z_{3,2}$	7.3351	

6. Menghitung nilai *fitness* f setiap kromosom.

$$f = \frac{1}{t_d}$$

Tabel 4.7 Contoh Nilai *Fitness* pada GA-*Based Clustering Algorithm*

Kromosom	Total Jarak	Nilai *Fitness*
1	14.3102	0.069880225
2	13.1768	0.07589096
3	12.4341	0.080423995

7. Kemudian untuk melakukan proses seleksi, pertama-tama hitung probabilitas yang dihasilkan oleh proses *roulette wheel*. Hitung *selection probability* p_m setiap kromosom dan

cumulative probability q_m setiap rute. *Selection probability* didapatkan dengan membagi nilai *fitness* kromosom ke m dengan jumlah total *fitness* keseluruhan kromosom. Kemudian dicari probabilitas kumulatif setiap rute.

$$p_m = \frac{f_m}{\sum_{i=1}^{M} f_i} \quad \text{dimana } m = 1,2,\dots M$$

$$q_m = \sum_{i=1}^{m} p_i \quad \text{dimana } m = 1,2,\dots M$$

M merupakan banyaknya kromosom. Seleksi proses dimulai dengan memutar *roulette wheel* dengan bilangan acak r_1 antara 0 dan 1. Apabila $r_1 \leq q_1$ maka kromosom pertama yang terpilih merupakan pelanggan yang berada pada populasi ke 1. Jika tidak, pilih kromosom ke m, dimana $2 \leq m \leq M$. Kemudian pilih kromosom kedua menggunakan cara yang sama dengan bilangan acak r_2. Sebagai contoh pada tahapan ini, digunakan CVRP yang didefinisikan pada 4.1.

Tabel 4.8 Contoh *Selection* dan *Cumulative Probaility* dengan GA-*Based Clustering Algorithm*

Kromosom	Nilai *Fitness*	p_m	q_m
1	0.069880225	0.301353984	0.308937729
2	0.07589096	0.327274891	0.644448679
3	0.080423995	0.355551321	1
Total *Fitness*	0.231887511		

Misalkan $r_1 = 0.3$ dan $r_2 = 0.7$ maka terpilih kromosom 1 dan 3.

8. Kromosom yang terpilih pada proses seleksi merupakan *parents* pada proses *crossover*. Kemudian lakukan *crossover* dengan probabilitas *crossover*. Dibangkitkan bilangan acak r_i dengan $i = 1, \dots, K$, apabila $r_i <$ *crossover rate* maka gen *offspring* pada lokus ke i berasal dari gen *parent* 1 pada lokus

ke i. Jika tidak, maka ambil gen dari *parent 2* pada lokus ke i.

Tabel 4.9 Contoh Proses *Crossover*

Parent 1	$z_{1,1}$	$z_{1,2}$
Parent 2	$z_{2,1}$	$z_{2,2}$
r_i	$r_1 = 0.3$	$r_2 = 0.85$
Offspring	$z_{1,1}$	$z_{2,2}$

9. Kemudian *offspring* yang dihasilkan dari proses *crossover* dilanjutkan ke proses mutasi. Dibangkitkan bilangan *random,* r_3. Apabila $r_3 <$ *mutation rate* maka dilakukan proses mutasi pada *offspring.* Pada penelitian ini terdapat dua cara proses mutasi dengan probabilitas yang sama. Pertama, dibangkitkan bilangan *random* r_4, apabila $r_4 \leq 0.5$ maka *center* ke i akan diganti dengan *center* yang baru dimana $i = 1, \dots, K$. Namun, jika $r_4 > 0.5$ maka *center* ke i akan ditukar dengan *center* kluster lain secara acak. Proses mutasi ini dilakukan untuk setiap *center*. Misalkan $r_3 = 0.7$ maka akan dilakukan proses mutasi pada *offspring.* Untuk lokus 1 pada *offspring*, dibangkitkan $r_4 = 0.4$ maka *center* akan diganti dengan *center* yang baru. Kemudian untuk lokus 2 $r_5 = 0.6$, maka *center* akan ditukar dengan *center* kluster lain secara acak. Contoh proses mutasi adalah sebagai berikut :

Tabel 4.10 Contoh Proses Mutasi

Sebelum mutasi		
Offspring	$z_{1,1}$	$z_{3,2}$
Setelah mutasi lokus 1 dengan $r_4 = 0.4$		
Offspring	$z_{3,1}$	$z_{3,2}$
Setelah mutasi lokus 2 dengan $r_5 = 0.6$		
Offspring	$z_{3,2}$	$z_{3,1}$

10. Bentuk kluster pada kromosom baru dan hitung nilai *fitness* kromosom baru. Apabila kromosom lebih baik dari kromosom yang paling buruk pada populasi, maka ganti kromosom yang paling buruk dengan kromosom baru.

Tabel 4.11 Contoh Hasil Kluster Kromosom Baru

Centroid	Pelanggan	Permintaan	Jarak
$z_{3,2}$	6	3	0
	3	2	3.6056
$z_{3,1}$	5	2	0
	1	1	4.0000
	2	2	7.0000

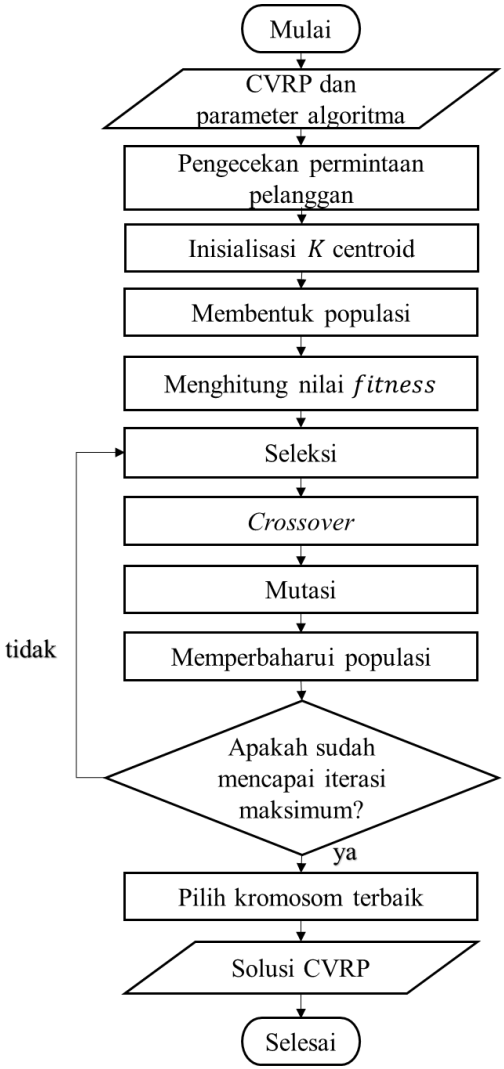
Tabel 4.12 Contoh Perhitungan Nilai *Fitness* Kromosom Baru

Centroid	Jarak	Total Jarak	Nilai *Fitness*
$z_{3,2}$	3.6056	14.6056	0.068466889
$z_{3,1}$	11		

Pada contoh yang diberikan di tabel 4.11 dan 4.12, nilai *fitness* kromosom baru tidak lebih besar dari nilai *fitness* kromosom yang ada pada populasi, maka kromosom baru tidak mengganti kromosom yang ada pada populasi.

11. Ulangi langkah ke-tujuh hingga mencapai iterasi maksimal. Apabila sudah mencapai angka iterasi maksimal maka iterasi berhenti dan solusi dari CVRP merupakan kromosom terbaik pada populasi.

Berikut merupakan diagram alir dari GA-*Based Clustering Algorithm* untuk menyelesaikan CVRP dengan permintaan dinamis.



Gambar 4.1 Diagram Alir GA-*Based Clustering Algorithm* untuk Menyelesaikan CVRP

4.5 Perancangan Penyelesaian CVRP dengan Algoritma *Harmony Search*

Penerapan algoritma *harmony search* untuk menyelesaikan CVRP pada penelitian ini adalah sebagai berikut:

1. Inisialisasi parameter penelitian dan parameter algoritma *harmony search.*
   a. Inisialisasi *harmony memory size* (HMS).
   b. Inisialisasi jumlah iterasi (NI).
   c. Insialisasi HMCR.
   d. Inisialisasi PAR.

   Dengan menggunakan contoh CVRP yang didefinisikan pada 4.1, parameter yang digunakan pada algoritma *harmony search* adalah HMS, NI, HMCR, dan PAR dengan nilai masing-masing parameter secara berurutan adalah 3, 1, 0.95, dan 0.15.

2. Inisialisasi *harmony memory*.
   a. *Harmony memory* berisi kumpulan *harmony vector* sebanyak HMS dan *harmony vector* berisi n pelanggan secara acak.

Tabel 4.13 Contoh *Harmony Memory*

Harmony Vector	Urutan Pelanggan				
	1	2	3	4	5
1	1	2	5	6	3
2	1	3	5	6	2
3	6	3	1	5	2

   b. Kemudian *harmony* vector dibagi menjadi k kluster. Pertama, tetapkan kendaraan berawal dari depot direpresentasikan dengan $i = 0$. Kemudian pelanggan pertama pada *harmony vector* sebagai pelanggan pertama yang dikunjungi oleh kendaraan I. Selanjutnya, pilih pelanggan selanjutnya pada *harmony vector* untuk ditetapkan sebagai pelanggan kedua. Begitu seterusnya hingga sisa kapasitas kendaraan pertama sama dengan 0

atau lebih kecil dari permintaan pelanggan selanjutnya. Apabila sudah mencapai pelanggan terakhir untuk kendaraan I, kendaraan I kembali ke depot dan kemudian kendaraan II mulai dari depot untuk melayani rute selanjutnya dan seterusnya hingga tidak ada pelanggan yang tersisa.

Harmony vector 1	=	1	2	5	6	3			
Demand	=	1	2	2	3	2			
Vehicle	=	1			2				
Rute	=	0	1	2	5	0	6	3	0

Harmony vector 2	=	1	3	5	6	2			
Demand	=	1	2	2	3	2			
Vehicle	=	1			2				
Rute	=	0	1	3	5	0	6	2	0

Harmony vector 3	=	6	3	1	5	2			
Demand	=	3	2	1	2	2			
Vehicle	=	1		2					
Rute	=	0	6	3	0	1	2	2	0

c. Setelah dibentuk rute untuk semua pelanggan, selanjutnya hitung total jarak (t_d) yaitu jumlah perjalanan dari seluruh kluster pada setiap *harmony vector*. Kemudian hitung *fitness* dengan $f = \frac{1}{t_d}$ karena semakin kecil total jaraknya, semakin besar atau semakin bagus nilai *fitness*-nya. Total *distance* dan nilai *fitness* untuk contoh CVRP pada 4.1 dapat dilihat pada tabel 4.14.

Tabel 4.14 Contoh Total *Distance* dan Nilai *Fitness Harmony Vector*

Harmony Vector	t_d	Nilai *Fitness*
1	23.8615	0.041908514
2	26.6453	0.037530071
3	28.1416	0.035534582

3. Improvisasi sebuah *harmony vector* baru.
   a. *Harmony vector* baru $x' = (x_1', x_2', \dots, x_n')$ dihasilkan dengan 2 aturan. Bangkitkan bilangan acak r_1 antara 0 dan 1.
      - Apabila $r_1 <$ HMCR, maka pilih pelanggan baru dari pelanggan yang ada pada HM dengan probabilitas yang dihasilkan oleh *roulette wheel*. Untuk *harmony vector* ke h dengan *fitness* f_h. Probabilitas seleksi p_h dan probabilitas kumulatif q_h dihitung sebagai berikut:

$$p_h = \frac{f_h}{\sum_{i=1}^{H} f_i} \quad \text{dengan } h = 1,2, \dots H$$

$$q_h = \sum_{i=1}^{h} p_i \quad \text{dengan } h = 1,2, \dots H$$

        H merupakan HMS. Seleksi proses dimulai dengan memutar *roulette wheel* dengan bilangan acak r_2 antara 0 dan 1. Apabila $r_2 < q_1$ maka x_i' merupakan pelanggan yang berada pada *harmony vector* ke 1. Jika tidak, pilih x_i' dari *harmony vector* ke h, dimana $2 \leq h \leq H$. Kemudian dibangkitkan bilangan acak r_3. Apabila $r_3 < PAR$, maka ganti nilai x_i' dengan salah satu tetangganya yaitu antara 2 pelanggan yang paling dekat dengan x_i' dengan probabilitas yang sama. Namun, apabila $r_3 \geq PAR$ maka tetap mempertahankan nilai x_i' yang terpilih pada tahap *memory consideration.* Dengan contoh CVRP 4.1, misalkan dibangkitkan $r_1 = 0.3$, maka pelanggan baru dipilih dengan tahap *memory considerations*. *Selection probability dan cumulative probability* berdasarkan nilai *fitness* pada tahap sebelumnya, dapat dilihat pada tabel 4.15.

Tabel 4.15 *Selection* dan *Cumulative Probability* dengan Algoritma *Harmony Search*

Harmony Vector	Nilai *Fitness*	p_h	q_h
1	0.041908514	0.364506909	0.364506909
2	0.037530071	0.326424608	0.690931517
3	0.035534582	0.309068483	1
Total *Fitness*	0.114973167		

Dengan $r_2 = 0.5$ maka pelanggan yang terpilih yaitu pelanggan 1 pada *harmony vector* 2. Kemudian $r_3 = 0.1$, maka dilakukan proses *pitch adjustment*.

Tabel 4.16 Mencari *Neighbouring Customer pada Pitch Adjusment*

Pelanggan	2	3	5	6
Jarak dari Depot	3.0000	1.4142	4.0000	2.2361

Maka *neihbouring customer*nya adalah 3 dan 6. Dengan probabilitas yang sama, didapatkan pelanggan 6 sebagai pelanggan pada *harmony vector* baru. Sehingga *harmony vector* baru sekarang adalah :

Tabel 4.17 Pelanggan Pertama *Harmony Vector baru*

Urutan	1	2	3	4	5
Rute	6				

- Apabila $r_1 \geq$ HMCR maka pilih x_i' dari keseluruhan pelanggan yang belum dikunjungi dengan menggunakan probabilitas yang dihasilkan dari proses seleksi *roulette wheel* sebagai berikut :

$$p_a = \frac{c_{(x'_{i-1},a)}}{\sum_1^A c_{(,a)}} \quad \text{untuk } \forall a \in A$$

$$q_a = \sum_{j=1}^{a} p_j \quad \text{untuk } \forall a \in A$$

dengan :
a = pelanggan yang belum dikunjungi
$A =$ kumpulan pelanggan yang belum dikunjungi
$x_i' =$ pelanggan baru ke i
$p_a =$ probabilitas seleksi pelanggan a
$q_a =$ probabilitas kumulatif pelanggan a
$c_{(x_{i-1}',a)} =$ jarak antara pelanggan baru urutan ke $i-1$ dengan pelanggan a.
Proses seleksi dimulai dengan memutar *roulette wheel* dengan nilai *random* r_4 dimana $0 \leq r_4 \leq 1$. Jika $r_4 \leq q_a$, maka x_i' merupakan pelanggan a.
$r_1 = 0.96$, maka lakukan proses *randomization* untuk mencari pelanggan 2 pada *harmony vector* baru.

Tabel 4.18 Tahap *Randomization*

Pelanggan	Jarak	p_a	q_a
1	2.2361	0.169699775	0.122435737
2	5.0990	0.386968004	0.55666778
3	3.6056	0.273632445	0.830300225
5	2.2361	0.169699775	1

$r_4 = 0.7$ maka pelanggan yang terpilih adalah pelanggan 3. Sehingga pelanggan kedua *harmony vector* baru yaitu 3.

Tabel 4.19 Pelanggan Kedua *Harmony Vector* Baru

Urutan	1	2	3	4	5
Rute	6	3			

b. Lakukan tahap a hingga mendapatkan $x' = (x_1', x_2', \dots, x_n')$. Dengan contoh sebelumnya, *harmony vector* barunya seperti pada tabel 4.20.

Tabel 4.20 *Harmony Vector* Baru

Urutan	1	2	3	4	5
Rute	6	3	1	2	5

4. Memperbaharui *harmony memory*.
   a. Setelah mendapatkan *harmony vector* yang baru kemudian hitung nilai *fitness*-nya.

Harmony vector baru	=	6	3	1	2	5	
Demand	=	3	2	1	2	2	
Vehicle	=	1		2			
Rute	=	0	1	2	0	5	6
		3	0				

   Dengan $t_d = 23.8615$ dan nilai *fitness* = 0.041908514.

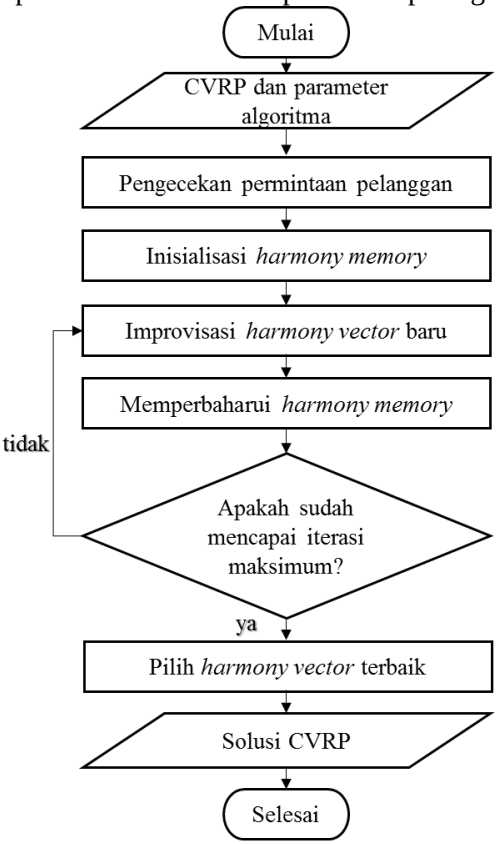
   b. Apabila *fitness value* dari *harmony vector* yang baru lebih baik daripada *harmony vector* yang berada pada *harmony memory*, maka eliminasi *harmony vector* yang paling buruk pada *harmony memory* dan ganti dengan *harmony vector* yang baru. Namun, apabila *fitness value* pada *harmony vector* baru tidak lebih baik daripada *fitness value* yang ada pada *harmony memory*, maka tidak ada pergantian *harmony vector*. Pada contoh yang diberikan sebelumnya, nilai *fitness harmony vector* baru lebih baik dari *harmony vector* yang ada pada HM, maka *harmony vector* baru mengganti *harmony vector* yang memiliki nilai *fitness* terkecil.

Tabel 4.21 *Updated Harmony Memory*

Harmony Vector	Urutan Pelanggan				
	1	2	3	4	5
1	1	2	5	6	3
2	1	3	5	6	2
3	6	3	1	2	5

5. Kondisi pemberhentian.
   Iterasi terus menghasilkan *harmony vector* baru hingga jumlah iterasi (NI) yang sudah didefinisikan pada tahap awal terpenuhi.

Diagram alir algoritma *harmony search* untuk menyelesaikan CVRP dengan permintaan dinamis dapat dilihat pada gambar 4.2.



Gambar 4.2 Diagram Alir Algoritma *Harmony Search* untuk Menyelesaikan CVRP

4.6 Perancangan Penyelesaian CVRP dengan *Hybrid Harmony Search Algorithm*

Langkah-langkah *Hybrid Harmony Search* dalam menyelesaikan CVRP pada penelitian ini adalah sebagai berikut :

1. Inisialisasi paremeter GA-*based clustering* algorithm dan *harmony search.*Parameter yang digunakan pada GA-*based*

clustering algorithm yaitu *crossover rate* = 0.8, *mutation rate* = 0.6, *number of iteration* = 1, dan *population size* = 3. Sedangkan parameter algoritma *harmony search* yang digunakan yaitu HMS, NI, HMCR, dan PAR masing-masing adalah 3, 1, 0.95, dan 0.15.

2. Selesaikan CVRP dengan menggunakan GA-*based clustering algorithm* dan menghasilkan 1 kromosom terbaik dengan *k* kluster. Sebagai contoh, digunakan kromosom terbaik pada 4.4.

Kromosom	:	2	6	5	1	3
Rute kluster 1	:	0	2	6	0	
Rute kluster 2	:	0	5	1	3	0

3. Perbaiki kluster 1 yang dihasilkan oleh GA-*based clustering algorithm* dengan algoritma *harmony search.*

Tabel 4.22 *Harmony Memory* Kluster 1 pada Algoritma HHS

Harmony Vector	Urutan Pelanggan	
	1	2
1	2	6
2	6	2
3	2	6

Tabel 4.23 Nilai *Fitness Harmony Vector* Kluster 1 pada Algoritma HHS

Harmony Vector	Rute				Total Jarak	Nilai *Fitness*
1	0	2	6	0	10.3351	0.096757651
2	0	6	2	0	10.3351	0.096757651
3	0	2	6	0	10.3351	0.096757651

Dengan contoh CVRP 4.1, misalkan dibangkitkan $r_1 = 0.3$, maka pelanggan baru dipilih dengan tahap *memory considerations*. *Selection probability dan cumulative probability* berdasarkan nilai *fitness* pada tahap sebelumnya, dapat dilihat pada tabel 4.24

Tabel 4.24 *Selection* dan *Cumulative Probability* Kluster 1 dengan Algoritma HHS

Harmony Vector	Nilai *Fitness*	p_h	q_h
1	0.096757651	0.333333333	0.333333333
2	0.096757651	0.333333333	0.666666667
3	0.096757651	0.333333333	1
Total *Fitness*	0.290272953		

Dengan $r_2 = 0.4$ maka pelanggan yang terpilih yaitu pelanggan 6 pada *harmony vector* 2. Kemudian $r_3 = 0.5$, maka tidak dilakukan proses *pitch adjustment*.

Rute baru kluster 1 : | 0 | 6 | 2 | 0 |

Nilai Fitness : 0.096757651

4. Perbaharui kromosom. Karena nilai *fitness* rute baru tidak lebih baik dari nilai *fitness* yang ada pada *harmony memory* maka rute pada kromosom tidak berubah.

5. Kemudian ulangi langkah 3 dan 4 hingga semua kluster sudah diperbaiki. Pada contoh, selanjuntya dikerjakan *kluster* 2

Tabel 4.25 *Harmony Memory* Kluster 2 pada Algoritma HHS

Harmony Vector	Urutan Pelanggan		
	1	2	3
1	1	3	5
2	3	5	1
3	5	3	1

Tabel 4.26 Nilai *Fitness Harmony Vector* Kluster 2 pada Algoritma HHS

Harmony Vector	Rute					Total Jarak	Nilai *Fitness*
1	0	1	3	5	0	16.3102	0.061311327

2	0	3	5	1	0	17.3989	0.057474898
3	0	5	3	1	0	16.3102	0.061311327

Dibangkitkan $r_1 = 0.3$ maka pilih pelanggan dengan *memory considerations.*

Tabel 4.27 *Selection* dan *Cumulative Probability* Kluster 2 pada Algoritma HHS

Harmony Vector	Nilai *Fitness*	p_h	q_h
1	0.061311327	0.340433983	0.364506909
2	0.057474898	0.319132034	0.659566017
3	0.061311327	0.340433983	1
Total *Fitness*	0.180097551		

$r_2 = 0.2$ maka pilih pelanggan dari *harmony vector* 1. Sehingga terpilih pelanggan 1. Kemudian, $r_3 = 0.1$ maka *update* pelanggan dengan proses *pitch adjustment*.

Tabel 4.28 *Pitch Adjusment* pada Algoritma HHS

Pelanggan	3	5
Jarak dari Depot	1.4142	4.0000

Dengan probabilitas yang sama, didapatkan pelanggan 3 sebagai pelanggan pada *harmony vector* baru. Sehingga *harmony vector* baru sekarang adalah

Tabel 4.29 Pelanggan Pertama *Harmony Vector baru* pada Algoritma HHS

Urutan	1	2	3
Rute	3		

$r_1 = 0.98$, maka lakukan proses *randomization* untuk mencari pelanggan 2 pada *harmony vector* baru.

Tabel 4.30 Tahap *Randomization* pada HHS

Pelanggan	Jarak	p_a	q_a
1	3.6056	0.41421777	0.41421777
5	5.0990	0.58578223	1

$r_2 = 0.5$ maka pelanggan yang terpilih adalah pelanggan 1.

Maka *harmony vector* baru dari kluster 2 adalah sebagai berikut :

Harmony vector baru	:	3	1	5		
Rute	:	0	3	1	5	0

Total jarak : 16.2999

Nilai *fitness* : 0.06135007

Karena nilai *fitness harmony vector* baru lebih baik daripada salah satu *harmony vector* yang ada pada HM. Maka perbaharui HM menjadi:

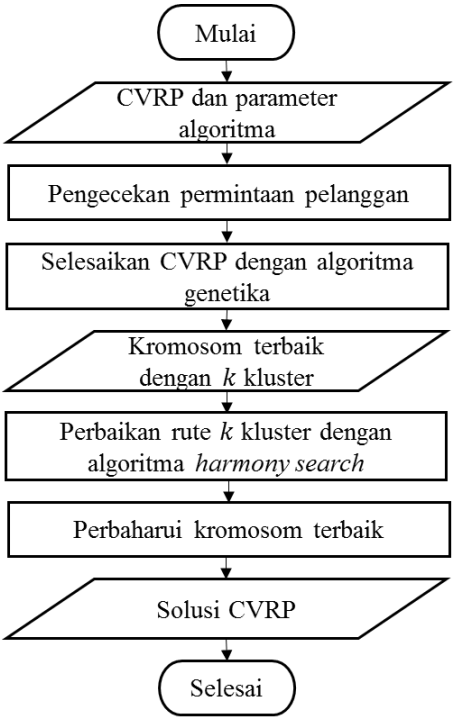
Tabel 4.31 HM yang Sudah Diperbaharui

Harmony Vector	Urutan Pelanggan		
	1	2	3
1	1	3	5
2	3	1	5
3	5	3	1

Sehingga didapatkan *final route* nya yaitu gabungan antara hasil terbaik pada kluster 1 dan kluster 2, yaitu

0	6	2	0	3	1	5	0

Diagram alir langkah – langkah algoritma HHS untuk menyelesaikan CVRP dengan permintaan dinamis dapat dilihat pada gambar 4.3.



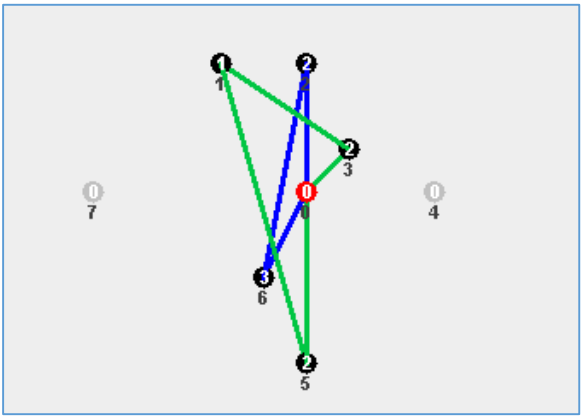
Gambar 4.3 Diagram Alir Algoritma *Hybrid Harmony Search* untuk Menyelesaikan CVRP

4.7 Perancangan Representasi Solusi

Solusi akan direpresentasikan melalui graf.

1. Pelanggan dan depot akan direpresentasikan sebagai node dalam koordinat 2 dimensi.
2. Solusi yang diambil yaitu solusi terbaik dari masing-masing algoritma.
3. Kemudian gambar garis yang menghubungkan antara kedua titik sesuai urutan pelanggan.
4. Depot digambarkan sebagai titik berwarna merah, pelanggan yang memiliki permintaan digambarkan sebagai titik berwarna hitam, dan pelanggan yang tidak mempunyai

permintaan digambarkan sebagai titik berwarana abu-abu. Pelanggan dihubungkan dengan rute yang dihasilkan oleh masing-masing algoritma. Warna rute yang sama berarti pelanggan tersebut berada pada kluster yang sama.



Gambar 4.4 Solusi dari CVRP pada 4.1 dengan Menggunakan Algoritma HHS

BAB V
IMPLEMENTASI DAN PENGUJIAN

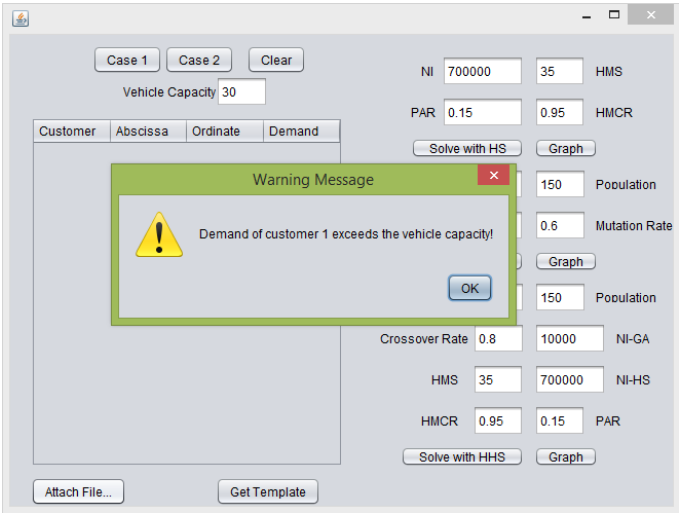
Pada bab ini dijelaskan tentang implementasi dalam bahasa pemrograman Java dan hasil uji coba program dalam menyelesaikan CVRP dengan permintaan yang dinamis.

5.1 Implementasi Perangkat Lunak

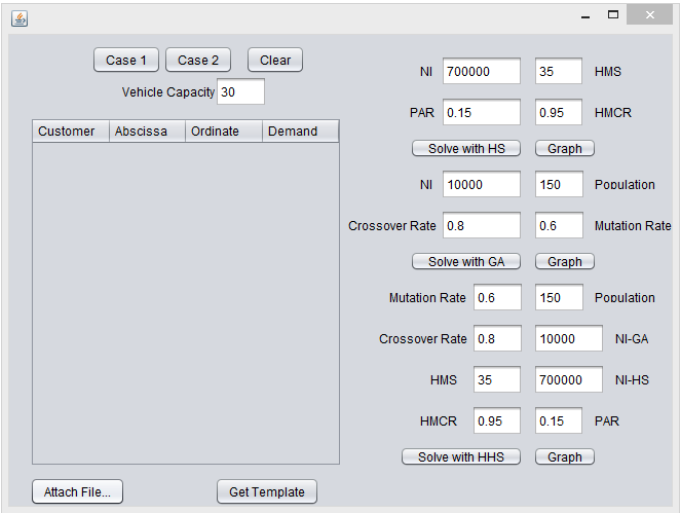
Program untuk menyelesaikan CVRP dengan permintaan dinamis diimplementasikan dalam bahasa pemrograman Java dengan menggunakan *software* Netbeans 8.2 .

5.1.1 Tampilan Aplikasi

Pada program, terdapat beberapa tombol dan isian untuk memudahkan penyelesaian CVRP. Tombol *Case* 1 dan *Case* 2 untuk memilih data CVRP. Kemudian terdapat *input* parameter dari algoritma yang ingin digunakan, dan tombol *Graph* untuk melihat representasi solusi berupa graf dari hasil algoritma yang digunakan. Tombol *Attach File* digunakan untuk mengambil data dari komputer yang berformat .xlsx dengan tujuan untuk memudahkan *user* untuk melihat, dan melakukan perubahan pada data. Tombol *Get Template* digunakan untuk menyimpan data yang telah di-*attach* agar *user* yang awam dapat mengetahui *format* untuk meng-*input* kan data. Kemudian, pada program terdapat fungsi untuk mengetahui apakah permintaan pelanggan melebihi kapasitas kendaraan atau tidak. Peringatan akan muncul untuk memberitahu *user* apabila terdapat permintaan pelanggan yang melebihi kapasitas kendaraan. Untuk mengatasi hal tersebut, *user* dapat membuat kapasitas kendaraan lebih besar atau dapat mengganti permintaan pelanggan tersebut. Contoh *warning* untuk kasus tersebut dapat dilihat pada gambar 5.1 dan contoh tampilan program dapat dilihat pada gambar 5.2.



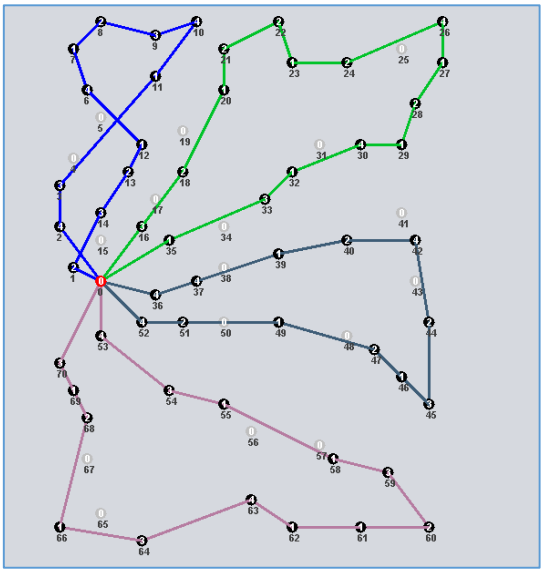
Gambar 5.1 Peringatan Permintaan Apabila Pelanggan Melebihi Kapasitas Kendaraan



Gambar 5.2 Tampilan Awal Program

5.1.2 Pengujian Perangkat Lunak

Berikut pada Gambar 5.3 merupakan contoh representasi solusi CVRP pada *Case* 1 dengan letak depot berada dipinggir. Dimana titik berwarna merah menggambarkan depot, titik berwarna hitam menggambarkan pelanggan yang memiliki permintaan, titik berwarna abu-abu menggambarkan pelanggan yang tidak memiliki permintaan, dan garis penghubung antar pelanggan dan depot dengan warna yang sama menggambarkan pelanggan berada dalam *cluster* yang sama.



Gambar 5.3 Contoh Representasi Solusi CVRP dengan Permintaan Dinamis

5.2 Pengujian Perangkat Lunak

Berikut ini akan dijelaskan pengujian perangkat lunak penyelesaian CVRP dengan permintaan dinamis yang dibagi menjadi 2 case, yaitu: CVRP dengan depot di pinggir (*case* 1) dan CVRP dengan depot berada di tengah (*case* 2).

5.2.1 Pengujian pada *Case* 1

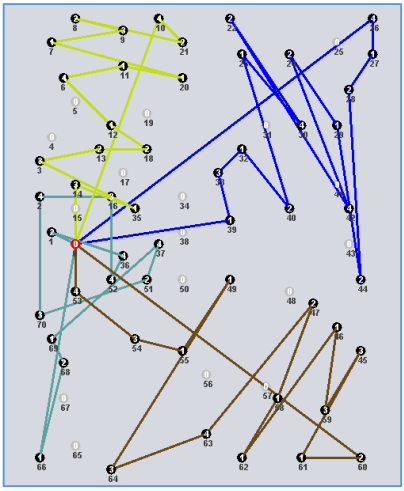
Pada *case* 1 CVRP, depot berada pada koordinat $(-11,0)$. CVRP akan diselesaikan dengan 3 algoritma, yaitu GA-*based clustering algorithm*, algoritma *harmony search*, algoritma *hybrid harmony search.*

5.2.1.1 Pengujian *Case* 1 dengan GA-*based clustering algorithm*

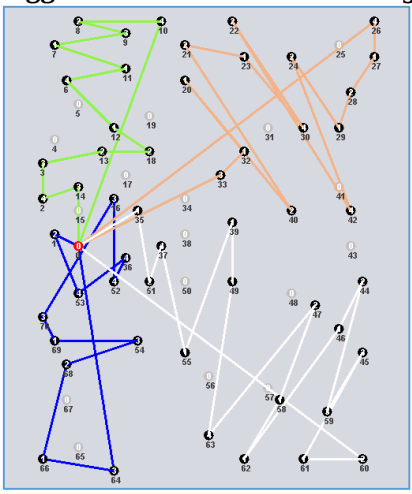
Parameter GA-*based clustering algorithm* yang digunakan untuk menyelesaikan *case* 1 adalah sebagai berikut :

a. *Mutation rate* : 0.6
b. *Crossover rate* : 0.8
c. *Population* : 150
d. *Number of Iteration* : 10000

Nilai *fitness*, total *distance* dan waktu komputasi dari pengujian *case* 1 menggunakan GA-*based clustering algorithm* dengan 30 kali percobaan dapat dilihat pada lampiran B, tabel B.1. Representasi solusi dengan menggunakan graf untuk setiap percobaan dapat dilihat pada lampiran D, tabel D.1. Rata-rata nilai *fitness, total distance* dan waktu komputasi masing-masing adalah 0.002137426, 509.5443764 dan 51.7994 detik. Sedangkan nilai *fitness* terbaik yaitu 0.002179209 dengan total jarak 504.4807444 dan waktu komputasi 49.449 detik. Sedangkan total jarak terkecil yaitu 484.2014538 dengan nilai *fitness* 0.002089231 dan waktu komputasi 55.736 detik. Karena GA-*based clustering algorithm* pada penelitian ini digunakan untuk *clustering*, maka nilai *fitness* yang baik merupakan gambaran dari hasil *clustering* yang baik sedangkan semakin kecil total jarak pada solusi maka semakin baik *routing* yang diperoleh. Hasil pengujian kemudian digambarkan dengan menggunakan graf. Representasi solusi *case* 1 CVRP dengan nilai *fitness* terbaik dapat dilihat pada gambar 5.4. Sedangkan gambar 5.5 merepresentasikan solusi *case* 1 CVRP dengan total jarak terbaik.



Gambar 5.4 Graf Solusi *Case* 1 CVRP dengan Nilai *Fitness* Terbaik Menggunakan GA-*Based Clustering Algorithm*



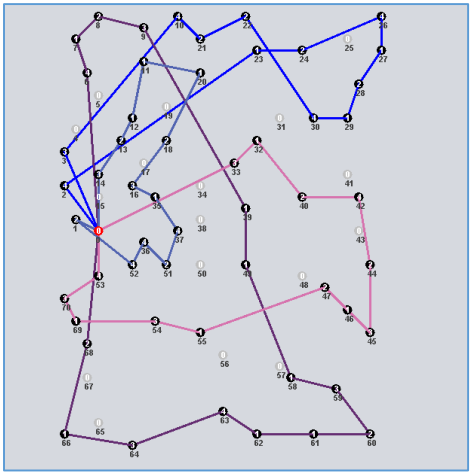
Gambar 5.5 Graf Solusi *Case* 1 CVRP dengan Total Jarak Terbaik Menggunakan GA-*Based Clustering Algorithm*

5.2.1.2 Pengujian *Case* 1 dengan Algoritma *Harmony Search*

Parameter algoritma *harmony search* yang digunakan untuk menyelesaikan *case* 1 adalah sebagai berikut :

a. HMS : 35
b. NI-HS : 700000
c. HMCR : 0.95
d. PAR : 0.15

Total *distance,* nilai *fitness* dan waktu komputasi dari pengujian *case* 1 menggunakan algoritma *harmony search* dengan 30 kali percobaan dapat dilihat pada lampiran B, tabel B.2. Representasi solusi dengan graf untuk setiap percobaan dapat dilihat pada lampiran D, tabel D.2. Rata-rata *total distance,* dan waktu komputasi masing-masing adalah 374.3301029 dan 619.5666667 detik. Sedangkan solusi terbaik pengujian *case* 1 dengan algoritma *harmony search* yaitu dengan *total distance* 338.9870694, dan waktu komputasi 795 detik. Dengan jumlah iterasi yang besar dan waktu komputasi yang lebih lama daripada GA-*based clustering algorithm*, algoritma *harmony search* dapat menghasilkan total jarak yang lebih kecil dikarenakan *routing* yang dihasilkan untuk setiap kluster lebih baik daripada yang dihasilkan dengan menggunakan GA-*based clustering algorithm*. Namun, kluster yang dihasilkan tidak sebaik yang dihasilkan dengan menggunakan GA-*based clustering algorithm.* Representasi solusi terbaik dengan graf dapat dilihat pada gambar 5.6.



Gambar 5.6 Graf Solusi Terbaik *Case* 1 CVRP dengan Algoritma *Harmony Search*

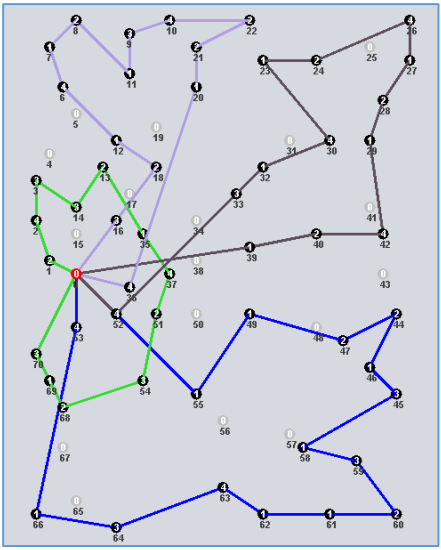
5.2.1.3 Pengujian *Case* 1 dengan Algoritma *Hybrid Harmony Search*

Parameter algoritma *hybrid harmony search* yang digunakan untuk menyelesaikan *case* 1 adalah sebagai berikut :

a. *Mutation rate* : 0.8
b. *Crossover rate* : 0.6
c. *Population* : 150
d. HMS : 35
e. NI-GA : 10000
f. NI-HS : 700000
g. HMCR : 0.95
h. PAR : 0.15

Total *distance* sebelum dan sesudah hibridisasi serta waktu komputasi dari pengujian *case* 1 menggunakan algoritma *hybrid harmony search* dengan 30 kali percobaan dapat dilihat pada lampiran B, tabel B.3. Representasi solusi dengan graf untuk setiap percobaan dapat dilihat pada lampiran D, tabel D.3. Pertama-tama CVRP dibagi menjadi beberapa daerah dengan menggunakan GA-

based clustering algorithm, kemudian hasil *clustering* terbaik yang diperoleh dari algoritma tersebut diambil untuk diperbaiki rute setiap klusternya dengan menggunakan algoritma *harmony search.* Hasil yang diperoleh dengan memanfaatkan keunggulan dari kedua algoritma tersebut cukup baik dengan rata-rata *total distance* sebelum, sesudah hibridisasi dan waktu komputasi masing-masing adalah 510.4367072, 329.2638177 dan 307.574 detik. Dengan hasil tersebut terlihat bahwa adanya perbedaaan total jarak yang cukup jauh berbeda antara sebelum dan setelah dilakukan hibridisasi. Sedangkan solusi terbaik pengujian *case 1* dengan algoritma *hybrid harmony search* yaitu dengan *total distance* sebelum hibridisasi 503.1832153 dan setelah dilakukan hibridisasi adalah 308.7193915 dengan waktu komputasi 276.696 detik. Representasi solusi terbaik dengan graf dapat dilihat pada gambar 5.7.



Gambar 5.7 Graf Solusi Terbaik *Case* 1 CVRP dengan Algoritma *Hybrid Harmony Search*

5.2.2 Pengujian pada *Case* 2

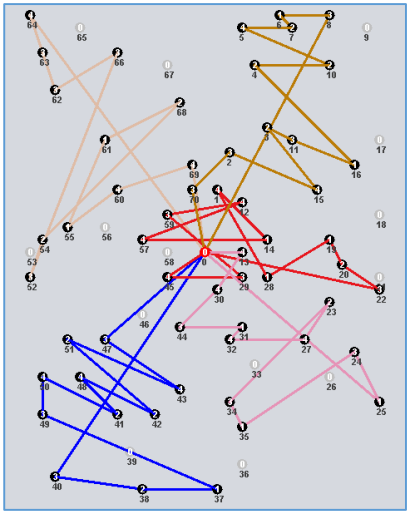
Pada *case* 2 CVRP, depot berada pada koordinat (0,0). CVRP akan diselesaikan dengan 3 algoritma, yaitu GA-*based clustering algorithm*, algoritma *harmony search*, algoritma *hybrid harmony search.*

5.2.2.1 Pengujian Case 2 dengan GA-*based clustering algorithm*

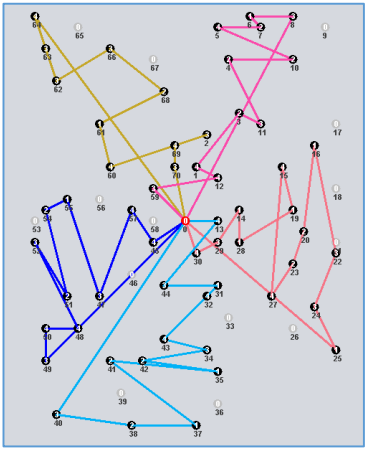
Parameter GA-*based clustering algorithm* yang digunakan untuk menyelesaikan *case* 2 adalah sebagai berikut :

a. *Mutation rate* : 0.6
b. *Crossover rate* : 0.8
c. *Population* : 150
d. *Number of Iteration* : 10000

Nilai *fitness*, total *distance* dan waktu komputasi dari pengujian *case* 2 menggunakan algoritma GA-*based clustering algorithm* dengan 30 kali percobaan dapat dilihat pada lampiran C, tabel C.1. Representasi solusi dengan graf untuk setiap percobaan dapat dilihat pada lampiran E, tabel E.1. Rata-rata nilai *fitness, total distance* dan waktu komputasi masing-masing adalah 0.002446452, 474.9387832 dan 103.8077 detik. Sedangkan nilai *fitness* terbaik yaitu 0.002506969 dengan total jarak 456.7064899 dan waktu komputasi 97.086 detik. Sedangkan total jarak terkecil yaitu 402.2694707 dengan nilai *fitness* 0.002312145 dan waktu komputasi 131.438 detik. Karena GA-*based clustering algorithm* pada penelitian ini digunakan untuk *clustering*, maka nilai *fitness* yang baik merupakan gambaran dari hasil *clustering* yang baik sedangkan semakin kecil total jarak pada solusi maka semakin baik *routing* yang diperoleh. Hasil pengujian kemudian digambarkan dengan menggunakan graf. Representasi solusi CVRP *case* 2 dengan nilai *fitness* terbaik dapat dilihat pada gambar 5.8. Sedangkan gambar 5.9 merepresentasikan solusi CVRP *case* 2 dengan total jarak terbaik.



Gambar 5.8 Graf Solusi *Case 2* CVRP dengan Nilai *Fitness* Terbaik Menggunakan GA-*Based Clustering Algorithm*



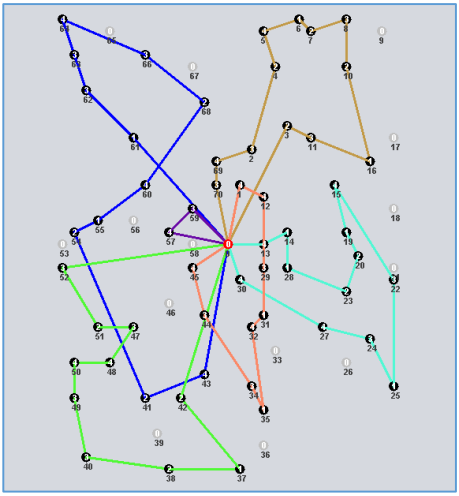
Gambar 5.9 Graf Solusi *Case 2* CVRP dengan Total Jarak Terbaik Menggunakan GA-*Based Clustering Algorithm*

5.2.2.2 Pengujian Case 2 dengan Algoritma *Harmony Search*

Parameter algoritma *harmony search* yang digunakan untuk menyelesaikan *case* 2 adalah sebagai berikut :

a. HMS : 35
b. NI-HS : 700000
c. HMCR : 0.95
d. PAR : 0.15

Total *distance*, nilai *fitness* dan waktu komputasi dari pengujian *case* 2 menggunakan algoritma *harmony search* dengan 30 kali percobaan dapat dilihat pada lampiran C, tabel C.2. Representasi solusi dengan graf untuk setiap percobaan dapat dilihat pada lampiran E, tabel E.2. Rata-rata *total distance* dan waktu komputasi masing-masing adalah 366.926637 dan 516.8 detik. Sedangkan solusi terbaik pengujian *case* 2 dengan algoritma *harmony search* yaitu dengan *total distance* 333.8907553 dan waktu komputasi 420 detik. Dengan jumlah iterasi yang besar dan waktu komputasi yang lebih lama dari GA-*based clustering algorithm*, algoritma *harmony search* dapat menghasilkan total jarak yang lebih kecil dikarenakan *routing* yang dihasilkan untuk setiap kluster lebih baik daripada yang dihasilkan dengan menggunakan GA-*based clustering algorithm*. Namun, kluster yang dihasilkan tidak sebaik yang dihasilkan dengan menggunakan GA-*based clustering algorithm*. Representasi solusi terbaik dengan graf dapat dilihat pada gambar 5.10.



Gambar 5.10 Graf Solusi Terbaik *Case* 2 CVRP dengan Algoritma *Harmony Search*

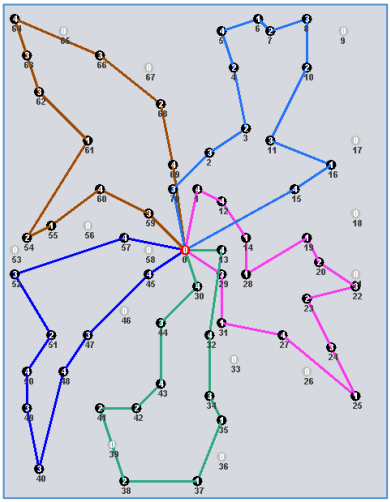
5.2.2.3 Pengujian Case 2 dengan Algoritma *Hybrid Harmony Search*

Parameter algoritma *hybrid harmony search* yang digunakan untuk menyelesaikan *case* 2 adalah sebagai berikut :

a. *Mutation rate* : 0.6
b. *Crossover rate* : 0.8
c. *Population* : 150
d. HMS : 35
e. NI-GA : 10000
f. NI-HS : 700000
g. HMCR : 0.95
h. PAR : 0.15

Total *distance* sebelum dan sesudah hibridisasi serta waktu komputasi dari pengujian *case* 2 menggunakan algoritma *hybrid harmony search* dengan 30 kali percobaan dapat dilihat pada lampiran C, tabel C.3. Representasi solusi dengan graf untuk setiap percobaan dapat dilihat pada lampiran E, tabel E.3. Pertama-tama

CVRP dibagi menjadi beberapa daerah dengan menggunakan GA-*based clustering algorithm*, kemudian hasil klustering terbaik yang diperoleh dari algoritma tersebut diambil untuk diperbaiki rute setiap klusternya dengan menggunakan algoritma *harmony search.* Rata-rata total *distance* sebelum, sesudah hibridisasi dan waktu komputasi masing-masing adalah 466.1549998, 306.1945788 dan 351.7053333 detik. Dengan hasil tersebut terlihat bahwa adanya perbedaaan total jarak yang cukup jauh berbeda antara sebelum dan setelah dilakukan hibridisasi. Sedangkan solusi terbaik pengujian *case* 2 dengan algoritma *hybrid harmony search* yaitu dengan total *distance* sebelum hibridisasi 426.7102333, dan setelah dilakukan hibridisasi adalah 289.592414 dengan waktu komputasi 357.908 detik. Representasi solusi terbaik dengan graf dapat dilihat pada gambar 5.11.



Gambar 5.11 Graf Solusi Terbaik *Case* 2 CVRP dengan Algoritma *Hybrid Harmony Search*

BAB VI
PENUTUP

Pada bab ini berisi tentang beberapa kesimpulan yang dihasilkan berdasarkan penelitian yang telah dilaksanakan dan saran yang dapat digunakan jika penelitian ini dikembangkan.

6.1 Kesimpulan

Berdasarkan analisis terhadap hasil pengujian program, maka dapat diambil kesimpulan sebagai berikut :

1. GA-*based clustering algorithm* dapat menghasilkan kluster yang cukup baik untuk menyelesaikan CVRP dengan permintaan dinamis. Sedangkan algoritma *harmony search* menghasilkan *routing* yang cukup baik untuk masing-masing kluster.
2. Dengan memanfaatkan keunggulan dari algoritma *harmony search* dan GA-*based clustering algorithm*, algoritma *hybrid harmony search* yang merupakan gabungan dari GA-*based clustering algorithm* dan algoritma *harmony search* mendapatkan hasil yang baik dalam menyelesaikan CVRP dengan permintaan dinamis.

6.2 Saran

Hal yang penulis sarankan untuk pengembangan penelitian selanjutnya adalah mengembangkan algoritma *hybrid harmony search* untuk menyelesaikan CVRP dengan permintaan dinamis untuk mendapatkan hasil yang lebih baik.

DAFTAR PUSTAKA

[1] Bramel, J., & Levi, D. S. (1997). **The Logic of Logistics: Theory, Algorithms, and Applications for Logistics Management**. Springer-Verlag New York, Inc., 175 Fifth Avenue New York, NY 10010, USA.

[2] Toth, P., & Vigo, D. (2001). **The Vehice Routing Problem**. Philadephia : SIAM (Society for Industrial and Applied Mathematics).

[3] Ai, T. J., & Kachitvichyanukul V. (2009). "Particle Swarm Optimization and Two Solution Represetations for Solving the Capacitated Vehicle Routing Problem". **Computer & Industrial Engineering** 56, 1:380-387.

[4] Szeto, W. Y., Yongzhong, W., & Ho, S. C. (2011). "An Artificial Bee Colony Algorithm for the Capacitated Vehicle Routing Problem". **European Journal of Operation Research** 215, 126-135.

[5] Augerat, P., Jose, M. B., Benavent, E., Corberan, A., Naddef, D., & Rinaldi, G. (1998). **Computational Results with A Branch and Cut Code for the Capacitated Vehicle Routing Problem**. Grenobe, France: Universite Joseph Fourier.

[6] Juan, A. A., Faulin, J., Ruiz, R., Barrios, B., & Caballe, S. (2010). "The SR-GCWS Hybrid Algorithm for Solving the Capacitated Vehicle Routing Problem". **Applied Soft Computing** 10, 1:215-224.

[7] Shahab, M. L., & Irawan, M. I. (2015). "Algoritma Genetika Ganda untuk Capacitated Vehicle Routing Problem". **Jurnal Sains dan Seni ITS** 4, 2.

[8] Pichibul, T., & Kawtummachai, R. (2013). "Modified Harmony Search Algorithm for the Capacitated Vehicle Routing Problem". **Proceedings of the International MultiConference of Engineers and Computer Scientists 2013** 2. Hong Kong, Maret 13-15.

[9] Maulik, U. & Bandyopadhyay, S. (2000). "Genetic Algorithm-Based Clustering Technique". **Pattern Recognition** 33, 1455-1465.

[10] Toth, P., & Vigo, D. (2002). "Models, Relaxations and Exact Approaches for the Capacitated Vehicle Routing Problem". **Discrete Applied Mathematics** 123, 487-512.

[11] Mehrjedi, Y. Z. (2015). "Multiple-Criteria Decision-Making Combined with VRP: A Categorized Bibliographic Study". **International Journal of Supply and Operations Management** 2, 2:798-820.

[12] Nazif, H., & Lee, L. S. (2012). "Optimised Crossover Genetic Algorithm for Capacitated Vehicle Routing Problem". **Applied Mathematical Modelling** 36, 2110-2117.

[13] Wang, X., Gao, X. Z., & Zenger, K. (2015). **An Introduction to Harmony Search Optimization Method**. Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing.

[14] Lee, K. S., & Geem, Z. W. (2005). "A New Meta-heuristic Algorithm for Continous Engineering Optimization: Harmony Search Theory and Practice". **Computer Methods in Applied Mechanics and Engineering** 194, 3902-3933.

[15] Mitchell, M. (1999). **An Introduction to Genetic Algorithm**. Cambridge, Massachussetts : A Bardford Book The MIT Press.

[16] Coley, D, A. (1999). **An Introduction to Genetic Algorithms for Scienctist and Engineers**. Singapore : World Scienctific Publishing Co. Pte. Ltd.

[17] Picek, S. & Golub, M. (2010). "Comparison of a Crossover Operator in Binary-coded Genetic Algorithms". **WSEAS Transaction on Computers** 9, 9.

LAMPIRAN A
Data Depot dan Pelanggan

a. Data Depot dan Pelanggan pada *Case* 1

Tabel A.1 Data Depot dan Pelanggan pada *Case* 1

Pelanggan	Absis	Ordinat	Permintaan
1	-13	1	2
2	-14	4	4
3	-14	7	3
4	-13	9	0
5	-11	12	0
6	-12	14	4
7	-13	17	1
8	-11	19	2
9	-7	18	3
10	-4	19	4
11	-7	15	1
12	-8	10	1
13	-9	8	2
14	-11	5	3
15	-11	3	0
16	-8	4	3
17	-7	6	0
18	-5	8	2
19	-5	11	0
20	-2	14	1
21	-2	17	2
22	2	19	2
23	3	16	1

24	7	16	2
25	11	17	0
26	14	19	4
27	14	16	1
28	12	13	2
29	11	10	1
30	8	10	4
31	5	10	0
32	3	8	1
33	1	6	3
34	-2	4	0
35	-6	3	1
36	-7	-1	4
37	-4	0	4
38	-2	1	0
39	2	2	1
40	7	3	2
41	11	5	0
42	12	3	4
43	12	0	0
44	13	-3	2
45	13	-9	3
46	11	-7	1
47	9	-5	2
48	7	-4	0
49	2	-3	1
50	-2	-3	0
51	-5	-3	2
52	-8	-3	4

53	-11	-4	4
54	-6	-8	3
55	-2	-9	1
56	0	-11	0
57	5	-12	0
58	6	-13	1
59	10	-14	3
60	13	-18	2
61	8	-18	1
62	3	-18	1
63	0	-16	4
64	-8	-19	3
65	-11	-17	0
66	-14	-18	1
67	-12	-13	0
68	-12	-10	2
69	-13	-8	1
70	-14	-6	3

b. Data Depot dan Pelanggan pada *Case* 2

Tabel A.2 Data Depot dan Pelanggan pada *Case* 2

Pelanggan	Absis	Ordinat	Permintaan
1	1	5	4
2	2	8	3
3	5	10	2
4	4	15	2
5	3	18	4
6	6	19	1

7	7	18	2
8	10	19	3
9	13	18	0
10	10	15	2
11	7	9	3
12	3	4	4
13	3	0	4
14	5	1	1
15	9	5	4
16	12	7	1
17	14	9	0
18	14	3	0
19	10	1	1
20	11	-1	2
21	14	-2	0
22	14	-3	3
23	10	-4	2
24	12	-8	3
25	14	-12	1
26	10	-10	0
27	8	-7	4
28	5	-2	1
29	3	-2	3
30	1	-3	4
31	3	-6	1
32	2	-7	4
33	4	-9	0
34	2	-12	3
35	3	-14	1

36	3	-17	0
37	1	-19	1
38	5	-19	2
39	-6	-16	0
40	-12	-18	3
41	-7	-13	2
42	-4	-13	2
43	-2	-11	4
44	-2	-6	3
45	-3	-2	4
46	-5	-5	0
47	-8	-7	3
48	-10	-10	4
49	-13	-13	3
50	-13	-10	4
51	-11	-7	2
52	-14	-2	3
53	-14	0	0
54	-13	1	2
55	-11	2	1
56	-8	2	0
57	-5	1	4
58	-3	0	0
59	-3	3	3
60	-7	5	4
61	-8	9	1
62	-12	13	3
63	-13	16	3
64	-14	19	4

65	-10	18	0
66	-7	16	3
67	-3	15	0
68	-2	12	2
69	-1	7	4
70	-1	5	3

LAMPIRAN B
Hasil 30 Percobaan *Case* 1 CVRP

a. Hasil 30 Percobaan *Case* 1 CVRP dengan GA-*Based Clustering Algorithm*

Tabel B.1 Nilai *Fitness*, Total Jarak, dan Waktu Komputasi 30 Percobaan *Case* 1 dengan GA-*Based Clustering Algorithm*

No. Percobaan	Nilai *Fitness*	Total Jarak	Waktu Komputasi (detik)
1	0.002139112	514.0584329	47.829
2	0.002121476	514.2176601	47.976
3	0.002103433	498.066608	44.291
4	0.002175218	514.7980411	54.083
5	0.002139112	514.0584329	45.209
6	0.002179209	504.4807444	49.449
7	0.002125485	514.0584329	55.375
8	0.002121476	514.2176601	51.475
9	0.002179209	504.4807444	51.733
10	0.002075763	510.9599258	50.053
11	0.002139112	514.0584329	44.86
12	0.002175218	514.7980411	56.467
13	0.002121476	514.2176601	58.256
14	0.002121476	514.2176601	48.973
15	0.002179209	504.4807444	54.587
16	0.002100244	511.1939957	56.127
17	0.002152794	488.7236656	57.772
18	0.002138791	507.6145498	52.717
19	0.002127118	512.4335322	61.048
20	0.002139112	514.0584329	43.571

21	0.002132462	514.7980411	51.456
22	0.002121476	514.2176601	49.25
23	0.002139112	514.0584329	48.658
24	0.002175218	514.7980411	49.44
25	0.002138507	504.983782	57.834
26	0.002137242	503.4502329	58.429
27	0.002175218	514.7980411	53.616
28	0.002121476	514.2176601	47.022
29	0.002089231	484.2014538	55.736
30	0.002138791	507.6145498	50.69

b. Hasil 30 Percobaan *Case* 1 CVRP dengan Algoritma *Harmony Search*

Tabel B.2 Total Jarak, Nilai *Fitness*, dan Waktu Komputasi 30 Percobaan *Case* 1 dengan Algoritma *Harmony Search*

No. Percobaan	Total Jarak	Nilai *Fitness*	Waktu Komputasi (detik)
1	389.6938	0.002566117	570
2	392.8883	0.002545252	427
3	386.631	0.002586445	420
4	356.1305	0.002807959	429
5	380.3191	0.002629371	404
6	346.7139	0.002884222	774
7	381.4947	0.002621268	412
8	341.8623	0.002925154	750
9	338.9871	0.002949965	795
10	349.8611	0.002858277	524
11	392.0169	0.002550910	745
12	374.8835	0.002667495	484

13	356.2658	0.002806893	692
14	366.0083	0.002732178	855
15	381.3021	0.002622593	923
16	368.5571	0.002713283	932
17	371.3513	0.002692868	918
18	368.4298	0.002714221	406
19	374.6272	0.002669320	565
20	402.1401	0.002486696	728
21	419.0549	0.002386322	401
22	372.8933	0.002681732	398
23	356.5382	0.002804748	457
24	390.9516	0.002557861	461
25	346.6747	0.002884548	787
26	386.4324	0.002587775	686
27	370.3885	0.002699867	415
28	392.1626	0.002549963	1025
29	393.7992	0.002539365	439
30	380.8438	0.002625748	765

c. Hasil 30 Percobaan *Case* 1 CVRP dengan Algoritma *Hybrid Harmony Search*

Tabel B.3 Total Jarak Sebelum dan Sesudah Hibridisasi dan Waktu Komputasi 30 Percobaan *Case* 1 dengan Algoritma *Hybrid Harmony Search*

No. Percobaan	Total Jarak dengan AG	Sesudah di Hybrid	Waktu Komputasi (detik)
1	514.7980411	325.7417403	284.721
2	514.0584329	350.5505218	271.107
3	503.1832153	308.7193915	276.696

4	488.7236656	326.2245298	280.595
5	504.4807444	321.9897102	285.225
6	512.4335322	353.7195602	281.592
7	507.6145498	330.5557959	275.386
8	507.6145498	317.5533059	274.874
9	488.7236656	324.4078063	282.923
10	503.4502329	316.8625504	285.365
11	514.0584329	324.5261341	278.721
12	504.4807444	319.1171908	285.246
13	507.6145498	324.0165038	285.177
14	488.7236656	320.1353139	304.635
15	522.8968775	342.9590178	275.566
16	514.7980411	336.6918905	366.51
17	514.2176601	318.3783092	305.757
18	504.4807444	314.2606022	313
19	514.7980411	329.9760322	342.019
20	514.0584329	356.4246287	321.069
21	503.4502329	331.2754128	380.071
22	512.4335322	347.8646582	374.325
23	504.4807444	323.2282385	349.486
24	514.2176601	337.4731032	342.009
25	514.7980411	327.0369866	349.553
26	488.7236656	328.8419293	337.232
27	593.1592684	345.809117	299.788
28	507.6145498	311.8644578	315.032
29	514.7980411	319.8395208	329.707
30	514.2176601	341.8705718	273.833

LAMPIRAN C
Hasil 30 Percobaan *Case* 2 CVRP

a. Nilai *Fitness,* Total Jarak dan Waktu Komputasi 30 Percobaan *Case* 2 dengan GA-*Based Clustering Algorithm*

Tabel C.1 Nilai *Fitness*, Total Jarak dan Waktu Komputasi 30 Percobaan *Case* 2 dengan GA-*Based Clustering Algorithm*

No. Percobaan	Nilai *Fitness*	Total Jarak	Waktu Komputasi (detik)
1	0.002506969	456.7064899	130.565
2	0.002367446	521.7609052	87.238
3	0.002400062	440.8721394	113.697
4	0.00239767	538.064696	72.801
5	0.002400062	440.8721394	92.888
6	0.002312145	402.2694707	131.438
7	0.002506969	456.7064899	97.086
8	0.002377478	485.1324939	94.286
9	0.002405598	527.8840675	101.214
10	0.002506969	456.7064899	108.549
11	0.002506969	456.7064899	117.455
12	0.002483244	426.7102333	108.852
13	0.002453553	440.8721394	103.638
14	0.002506969	456.7064899	106.682
15	0.002506969	456.7064899	105.081
16	0.00239767	538.064696	83.481
17	0.002506969	456.7064899	103.976
18	0.002390789	459.3979355	110.648
19	0.002506969	456.7064899	102.262
20	0.002506969	456.7064899	107.116

21	0.00242818	521.3843452	106.651
22	0.002376369	459.3979355	122.677
23	0.002506969	456.7064899	112.952
24	0.002506969	456.7064899	107.381
25	0.00241379	493.6679535	102.425
26	0.002403579	538.4985841	104.636
27	0.00239767	538.064696	79.088
28	0.002506969	456.7064899	104.471
29	0.00239767	538.064696	72.08
30	0.002506969	456.7064899	122.917

b. Hasil 30 Percobaan *Case* 2 dengan Algoritma *Harmony Search*

Tabel C.2 Total Jarak Sebelum dan Sesudah Hibridisasi, Nilai *Fitness* dan Waktu Komputasi 30 Percobaan *Case* 2 dengan Algoritma *Harmony Search*

No. Percobaan	Total Jarak	Nilai *Fitness*	Waktu Komputasi (detik)
1	366.7097219	0.002726952	770
2	385.0044341	0.002597372	771
3	368.8472549	0.002711149	430
4	334.4344326	0.002990122	806
5	404.800771	0.002470351	806
6	357.9103709	0.002793996	434
7	350.6672331	0.002851706	454
8	370.6980282	0.002697613	427
9	380.230402	0.002629984	759
10	362.2489963	0.002760532	413
11	368.2505718	0.002715542	459

12	342.8262993	0.002916929	644
13	366.7790434	0.002726437	642
14	383.0752233	0.002610453	437
15	363.5684272	0.002750514	434
16	344.8123465	0.002900128	434
17	357.9146056	0.002793962	432
18	355.3881895	0.002813824	772
19	359.6911618	0.002780163	433
20	353.2207865	0.002831090	436
21	336.7650011	0.002969430	485
22	396.1703094	0.002524167	449
23	395.1390102	0.002530755	419
24	333.8907553	0.002994992	420
25	356.710487	0.002803394	421
26	393.8552303	0.002539004	426
27	375.0982029	0.002665969	419
28	379.5767542	0.002634513	429
29	395.1904851	0.002530425	421
30	368.324575	0.002714997	422

c. Hasil 30 Percobaan *Case* 2 dengan Algoritma *Hybrid Harmony Search*

Tabel C.3 Total Jarak dan Waktu Komputasi 30 Percobaan *Case* 2 dengan Algoritma *Hybrid Harmony Search*

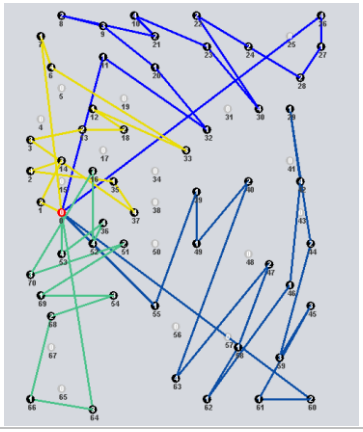
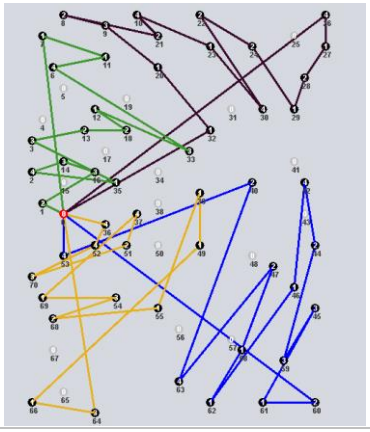
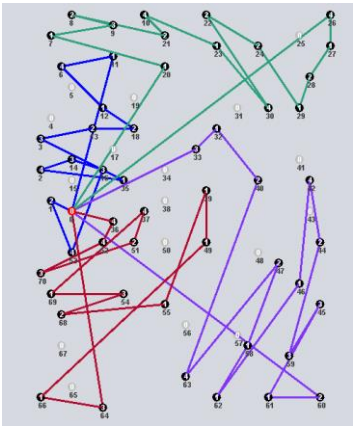
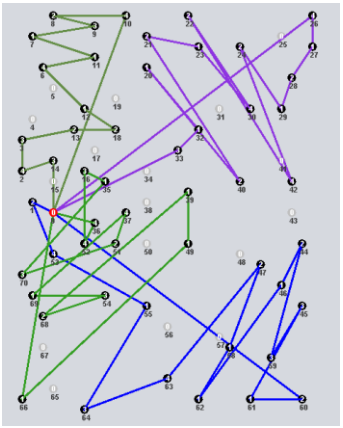
No. Percobaan	Total Jarak dengan AG	Total Jarak Sesudah Hybrid	Waktu Komputasi (detik)
1	456.7064899	307.260758	357.992
2	440.8721394	306.421758	345.989
3	440.8721394	302.271442	337.949

4	538.064696	314.048836	329.956
5	456.706489	296.275989	349.708
6	440.8721394	299.487149	338.329
7	426.7102333	289.592414	357.908
8	456.7064899	306.712793	365.496
9	517.5105063	307.500397	357.422
10	440.8721394	304.258822	386.559
11	456.7064899	313.103533	358.426
12	456.7064899	309.891696	387.036
13	456.7064899	303.264655	358.435
14	538.064696	306.119666	344.914
15	440.8721394	309.875904	372.892
16	518.7265693	319.911749	353.722
17	456.7064899	299.175306	371.477
18	456.7064899	301.461481	374.83
19	440.8721394	306.245727	364.371
20	440.8721394	309.069455	337.767
21	440.8721394	308.004916	336.637
22	440.8721394	303.055197	362.083
23	485.1324939	296.066724	356.926
24	538.064696	303.423446	311.322
25	456.7064899	323.330944	343.832
26	440.8721394	299.289778	336.548
27	527.8840675	320.25741	328.225
28	440.8721394	307.494404	326.224
29	440.8721394	306.767654	350.756
30	493.6679535	306.197365	347.429

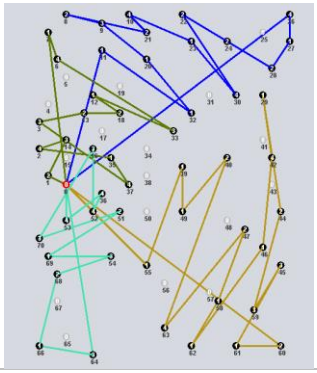
LAMPIRAN D
Graf Solusi *Case* 1 CVRP

a. Graf Solusi *Case* 1 CVRP dengan Menggunakan *GA-Based Clustering Algorithm*

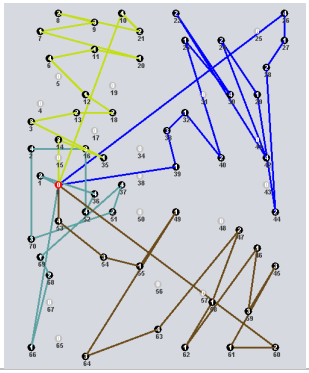
Tabel D.1 Graf Solusi *Case* 1 CVRP dengan Menggunakan GA-*Based Clustering Algorithm* (urutan berdasarkan nomor percobaan)

1	2
	
3	4
	

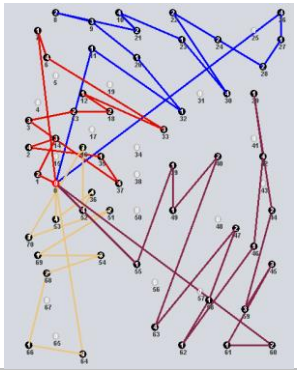
5



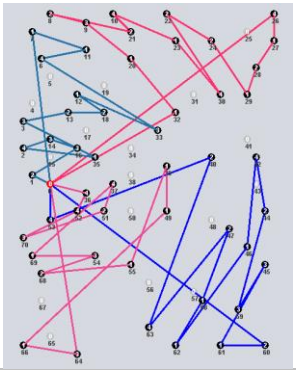
6



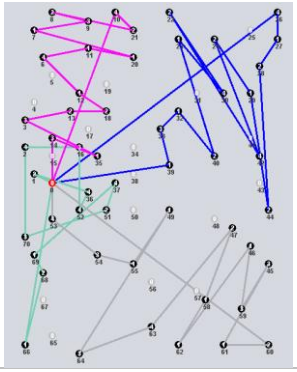
7



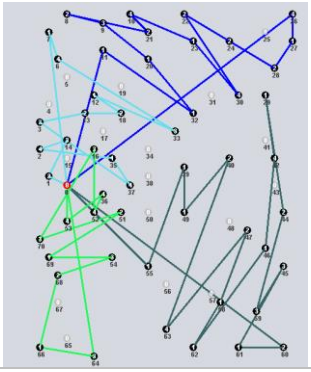
8



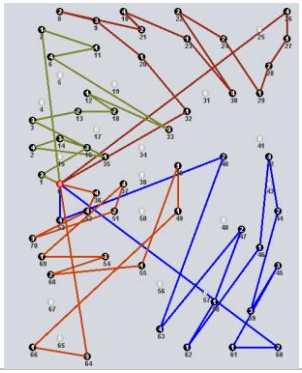
9



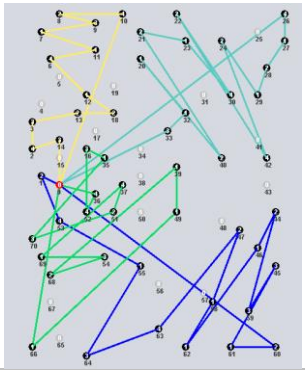
10



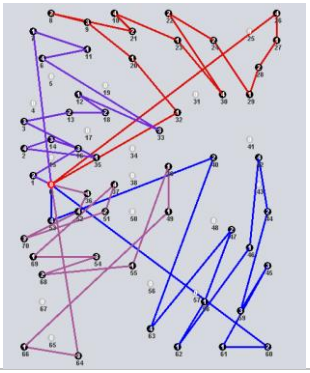
11



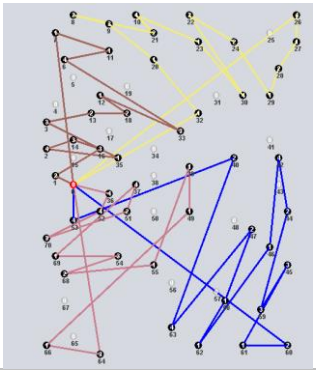
12



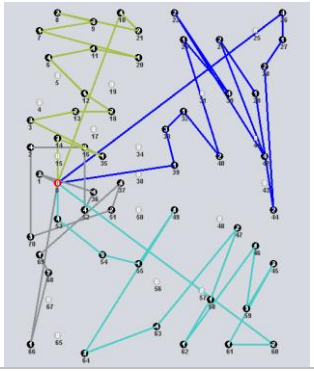
13



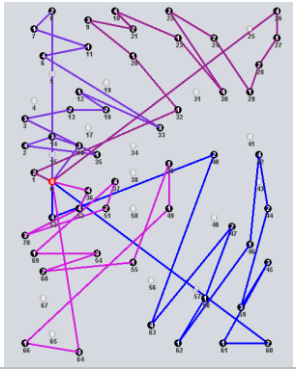
14



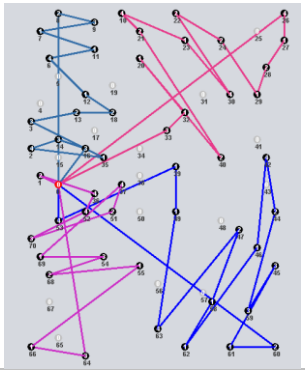
15



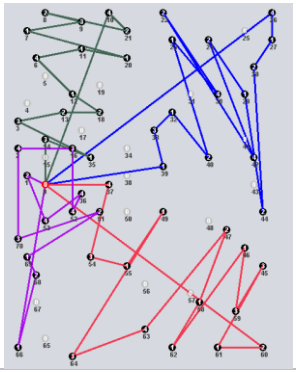
16



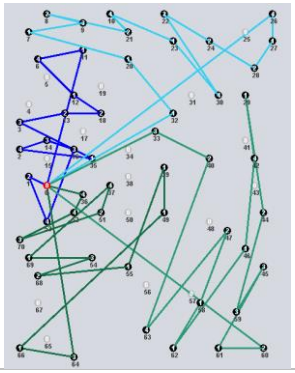
17



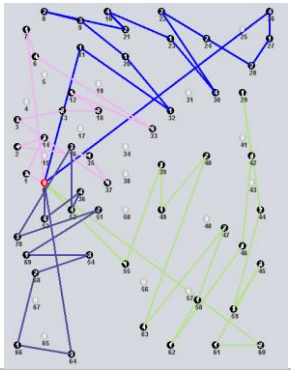
18



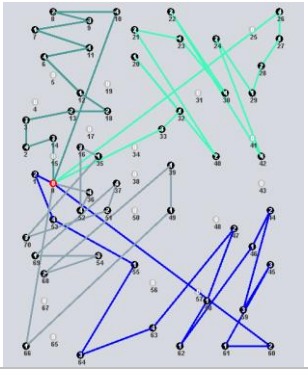
19



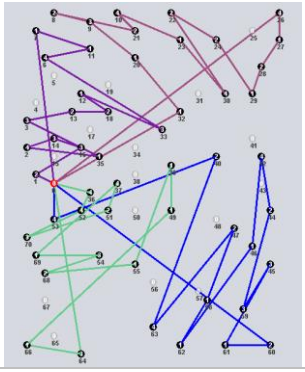
20



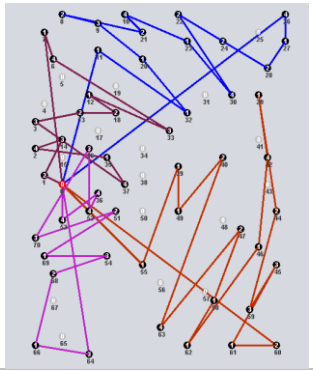
21



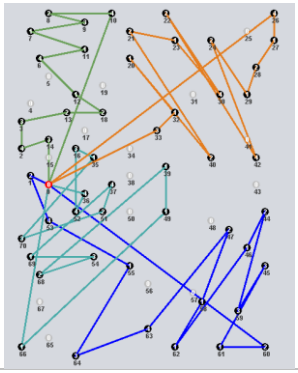
22



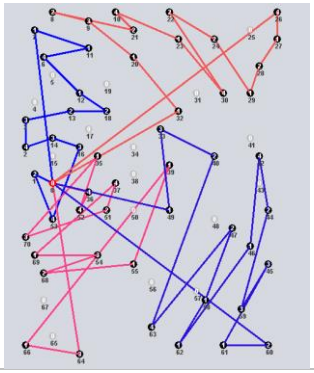
23



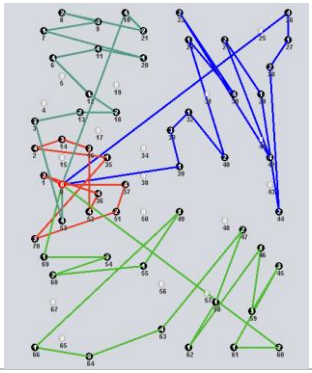
24



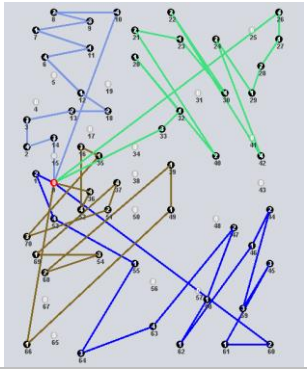
25



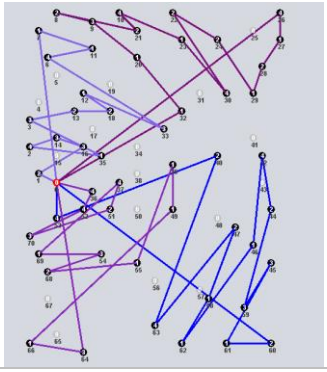
26

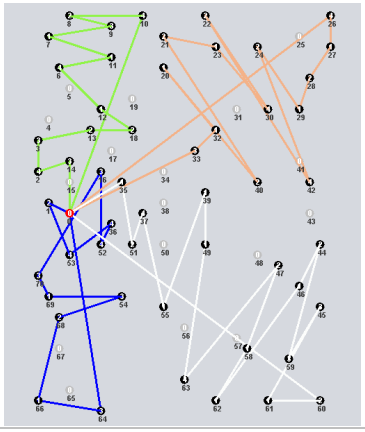
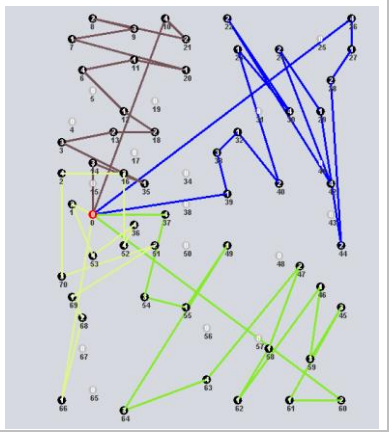


27



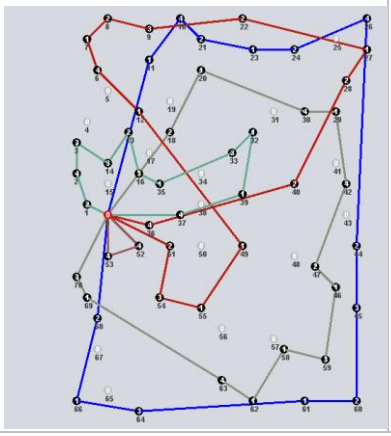
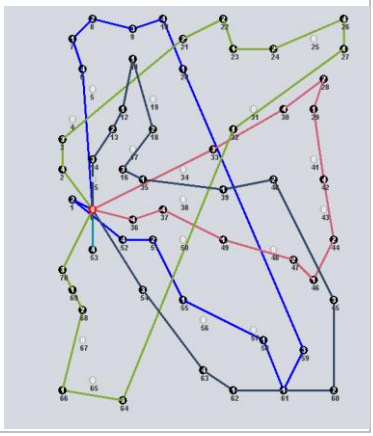
28



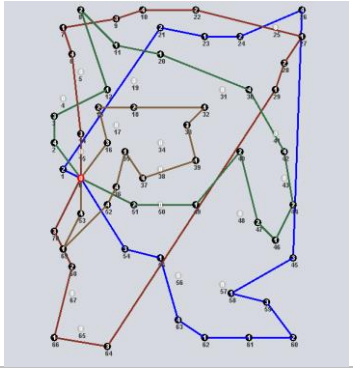
29	30
	

b. Graf Solusi *Case* 1 CVRP dengan Menggunakan Algoritma *Harmony Search*

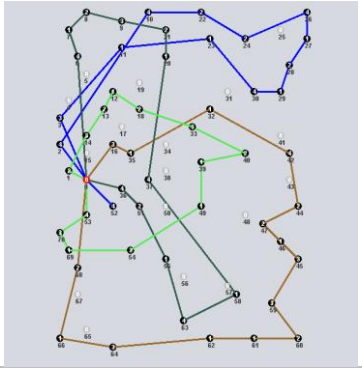
Tabel D.2 Graf Solusi *Case* 1 CVRP dengan Menggunakan Algoritma *Harmony Search* (urutan berdasarkan nomor percobaan)

1	2
	

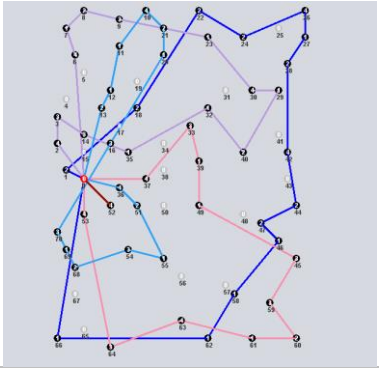
3



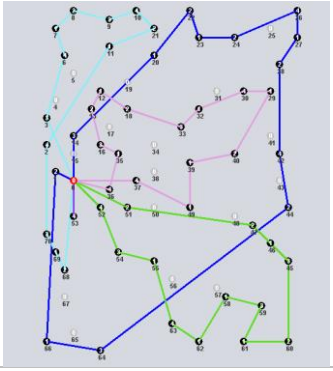
4



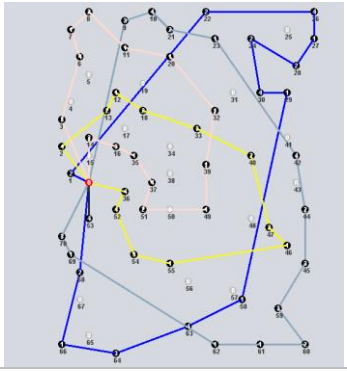
5



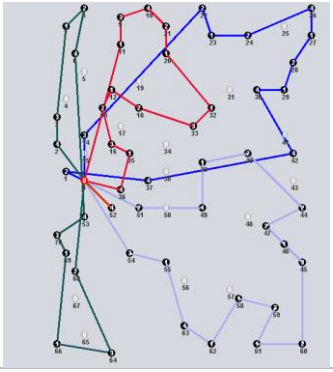
6



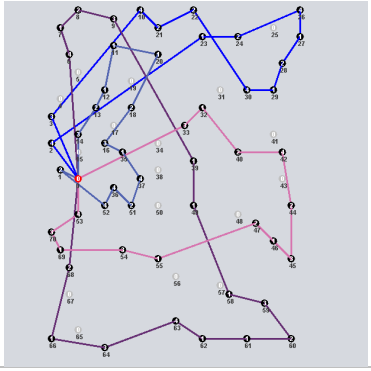
7



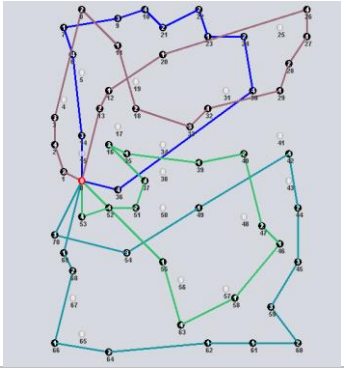
8



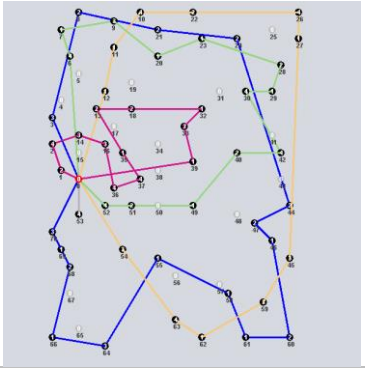
9



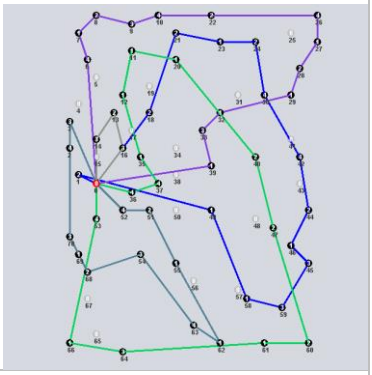
10



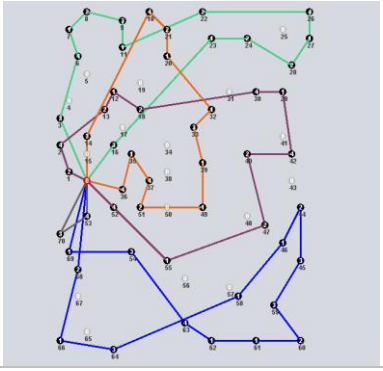
11



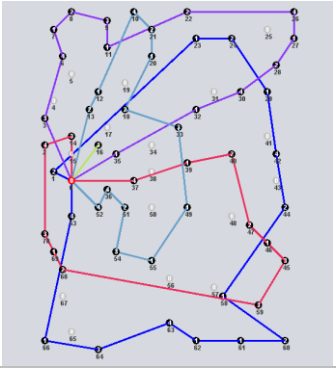
12



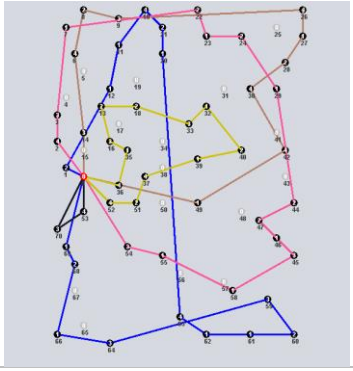
13



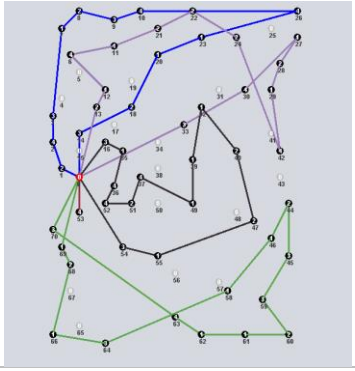
14



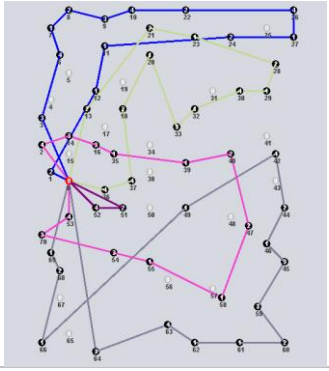
15



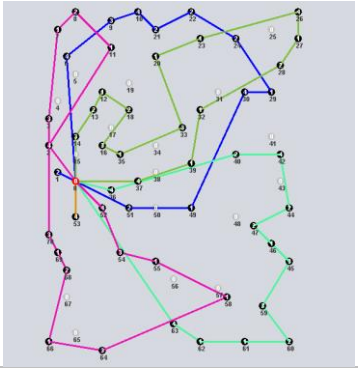
16



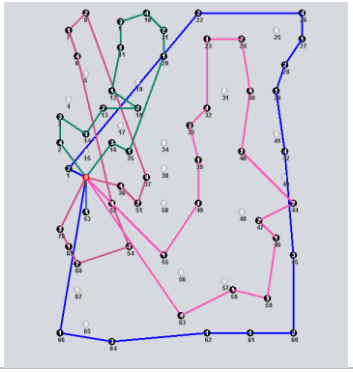
17



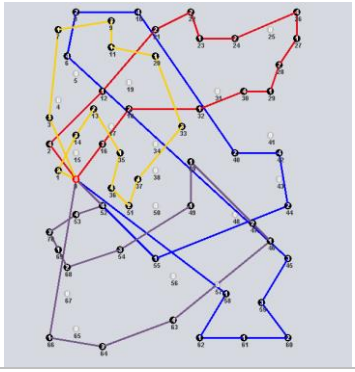
18



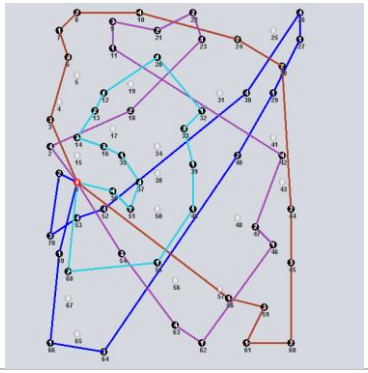
19



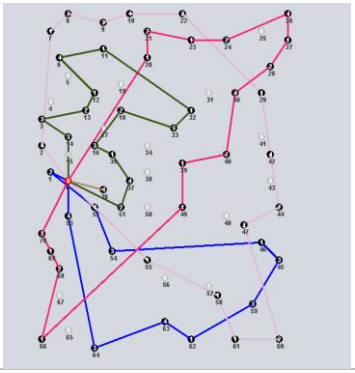
20



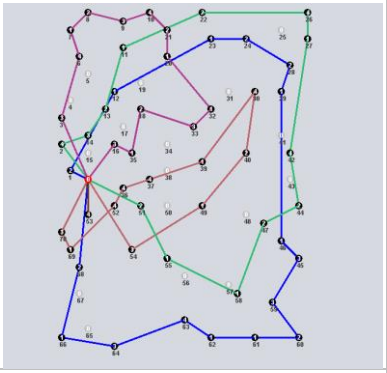
21



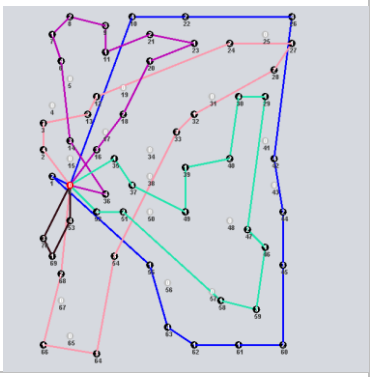
22



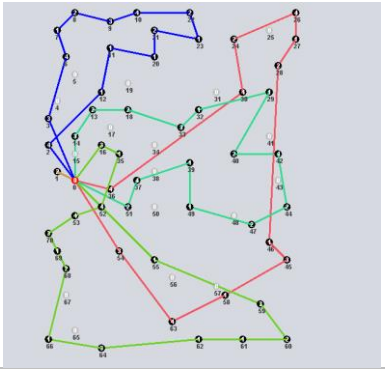
23



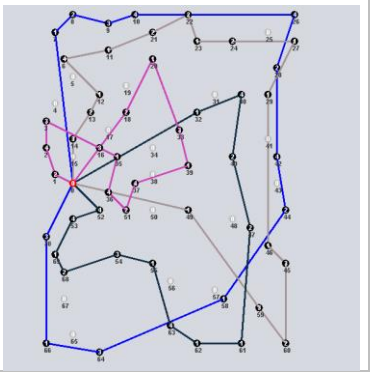
24



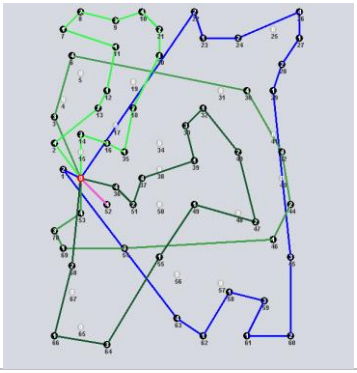
25



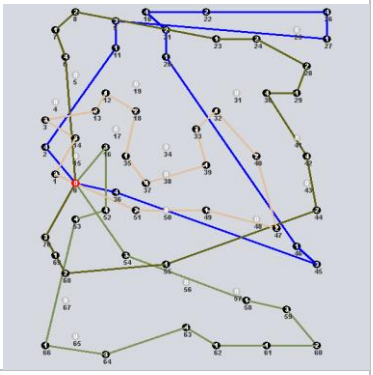
26



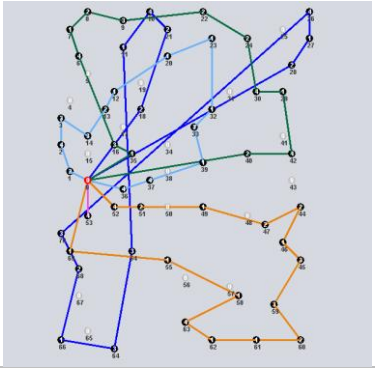
27



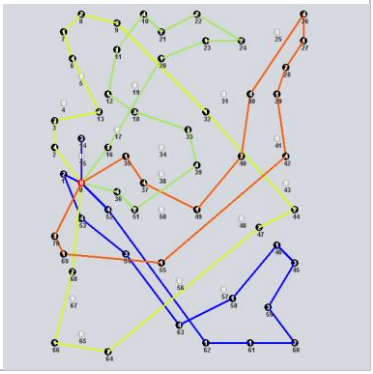
28



29

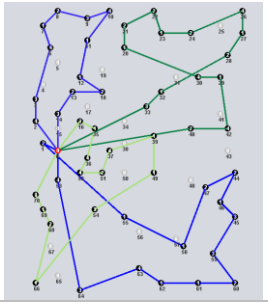
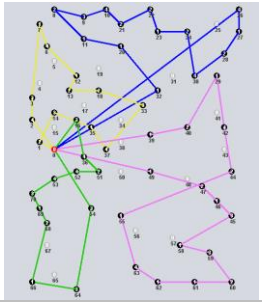
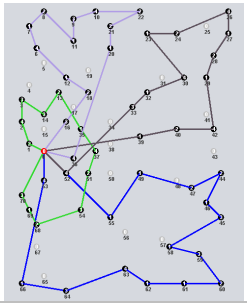
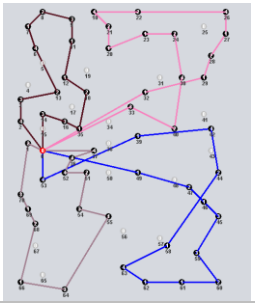
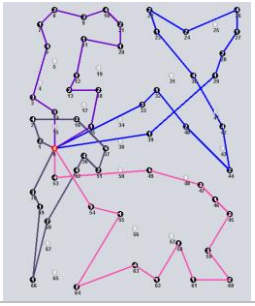
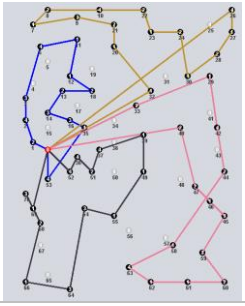


30

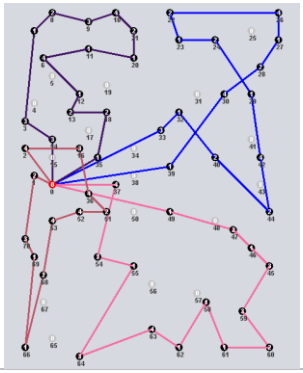


c. Graf Solusi *Case* 1 CVRP dengan Menggunakan Algoritma *Hybrid Harmony Search*

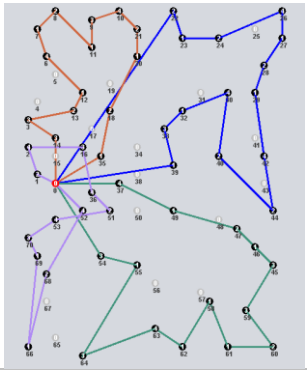
Tabel D.3 Graf Solusi *Case* 1 CVRP dengan Menggunakan Algoritma *Hybrid Harmony Search* (urutan berdasarkan nomor percobaan)

1	2
	
3	4
	
5	6
	

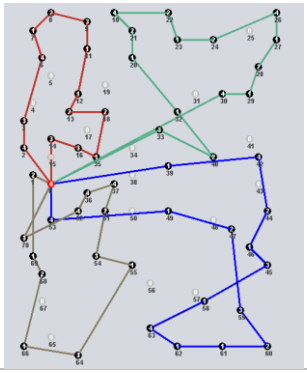
7



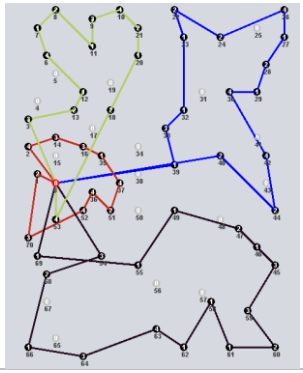
8



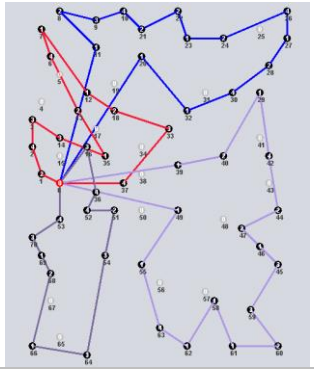
9



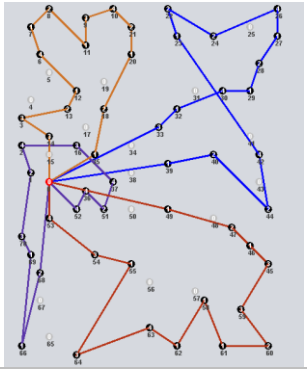
10



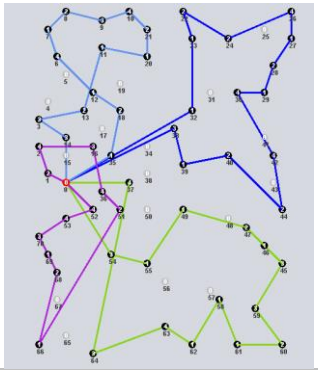
11



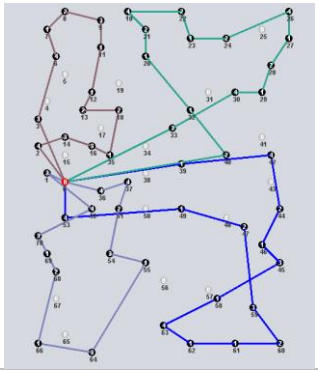
12



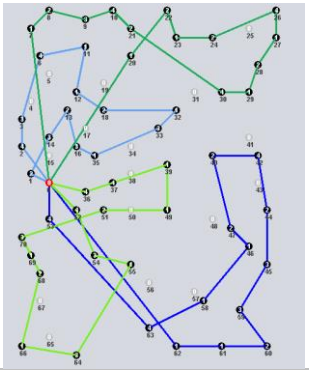
13



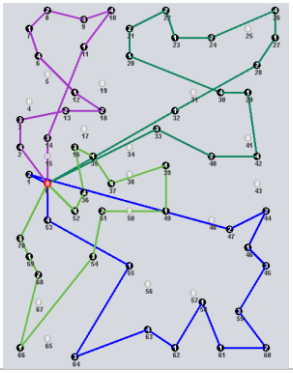
14



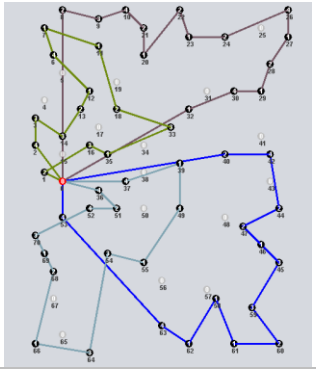
15



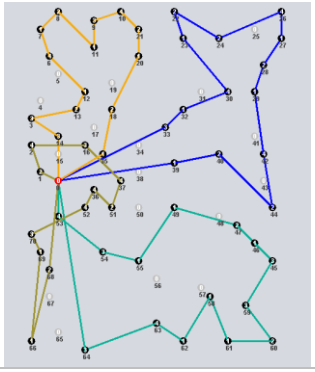
16



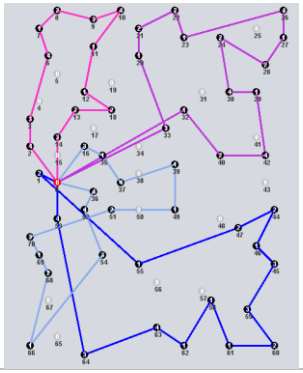
17



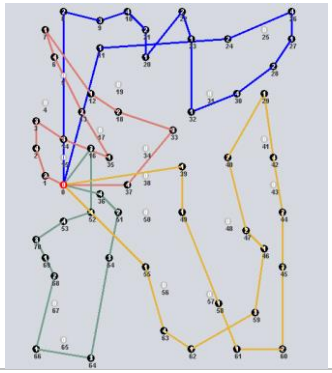
18



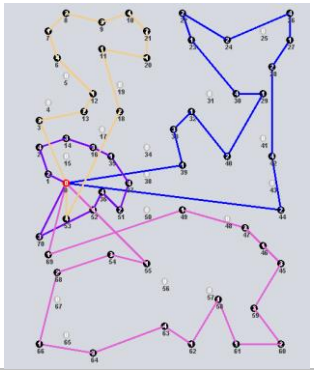
19



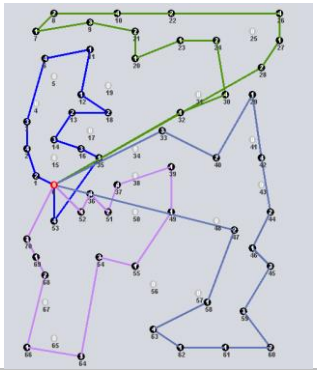
20



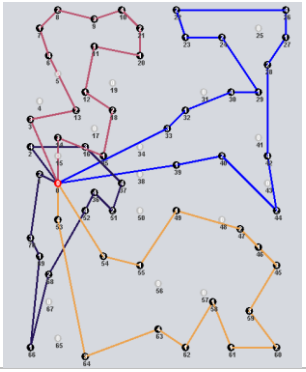
21



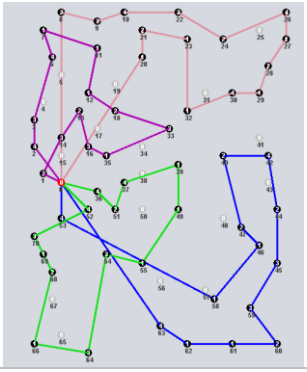
22



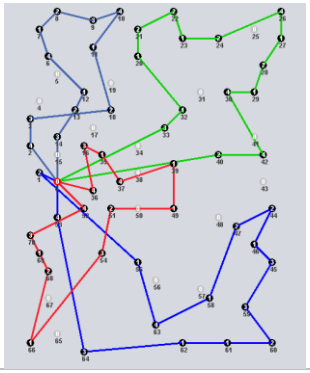
23



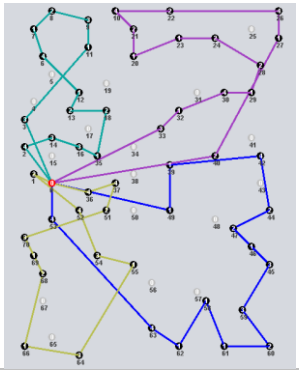
24



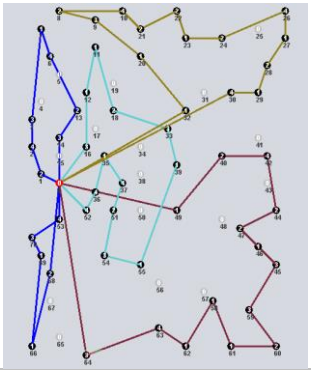
25



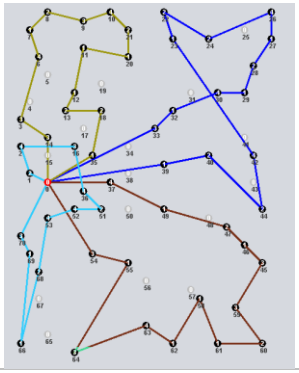
26



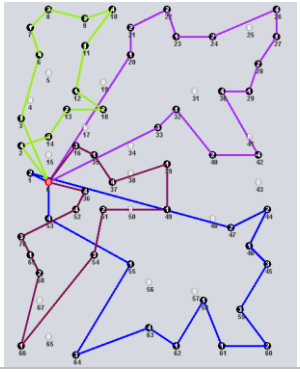
27



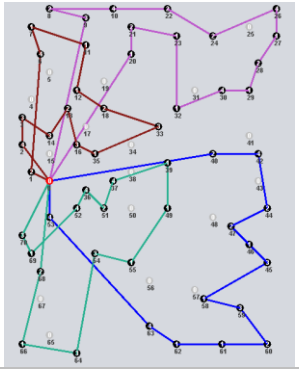
28



29



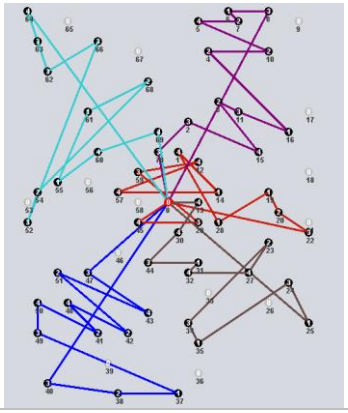
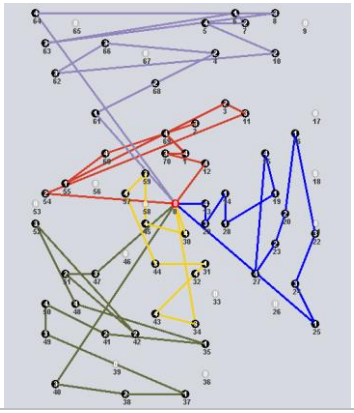
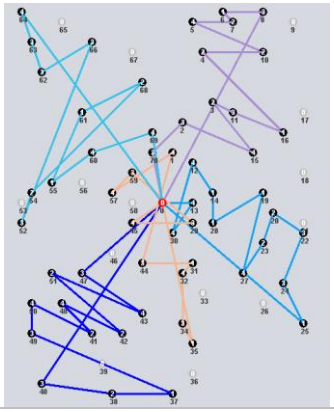
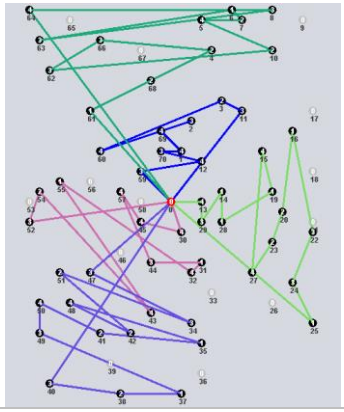
30



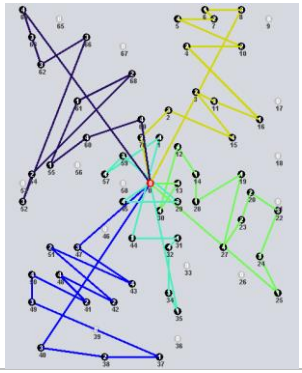
LAMPIRAN E
Graf Solusi *Case* 2 CVRP

a. Graf Solusi *Case* 2 CVRP dengan Menggunakan *GA-Based Clustering Algorithm*

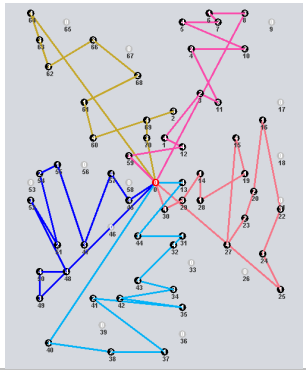
Tabel E.1 Graf Solusi *Case* 2 CVRP dengan Menggunakan GA-*Based Clustering Algorithm* (urutan berdasarkan nomor percobaan)

1	2
	
3	4
	

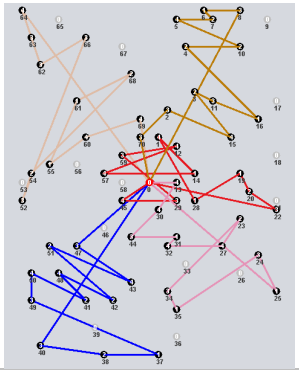
5



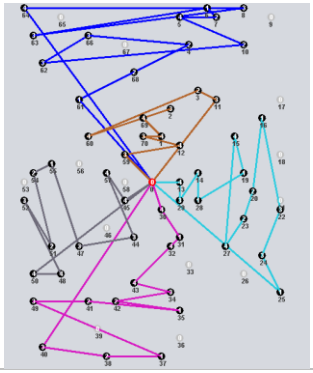
6



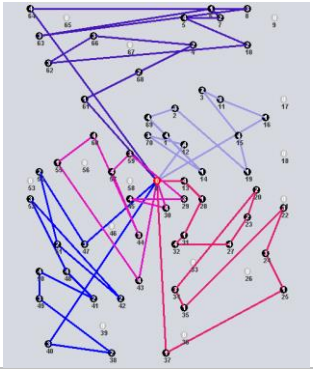
7



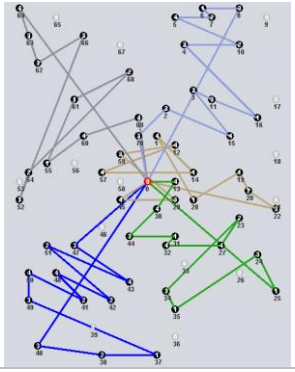
8



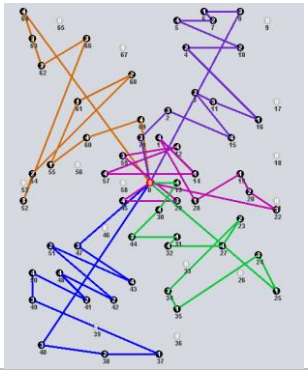
9



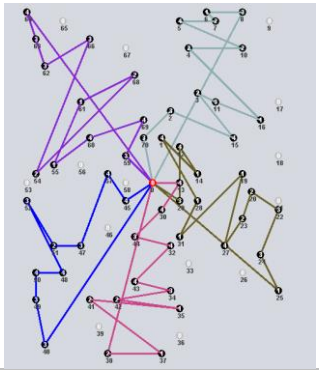
10



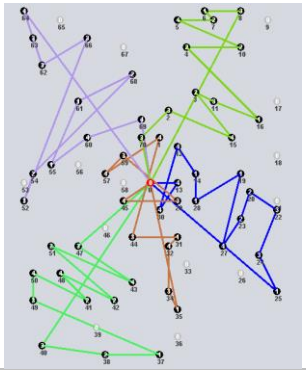
11



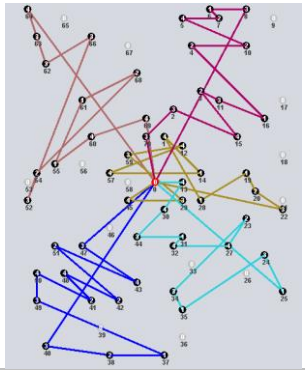
12



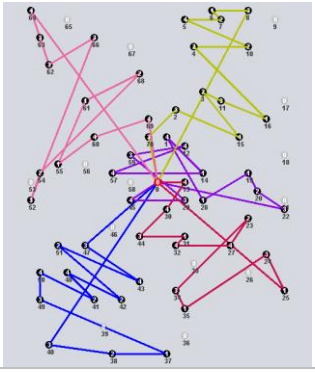
13



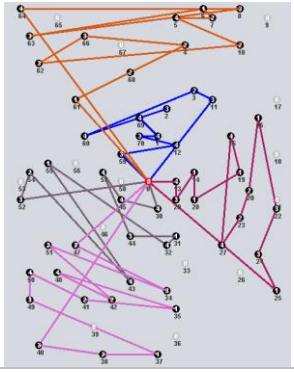
14



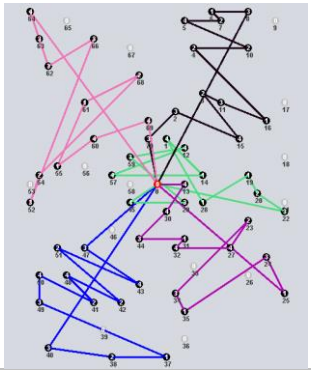
15



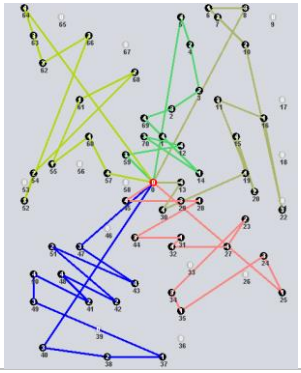
16



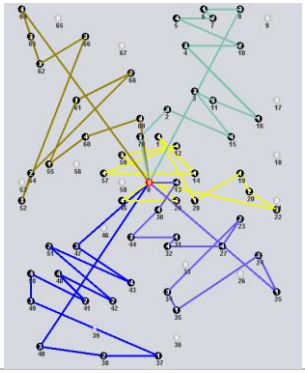
17



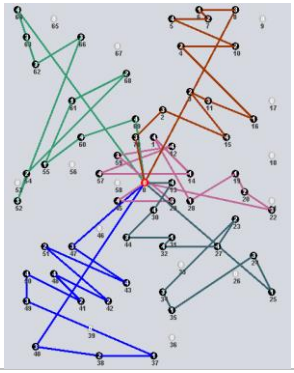
18



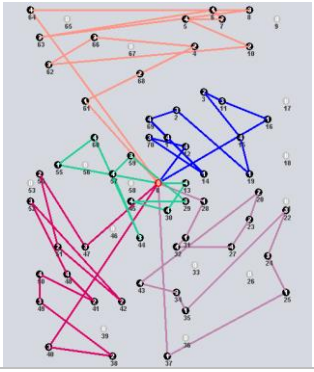
19



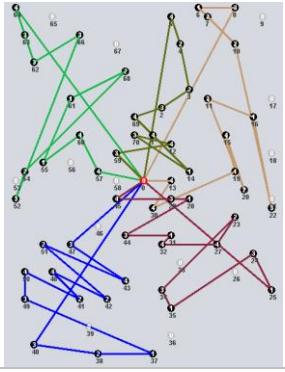
20



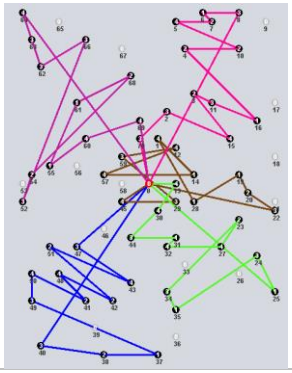
21



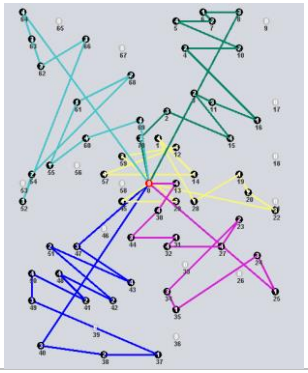
22



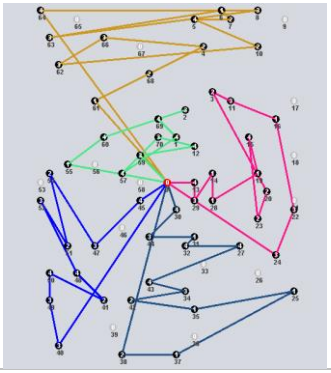
23



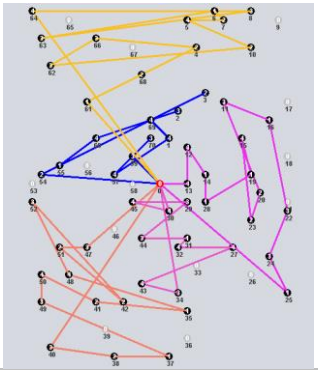
24



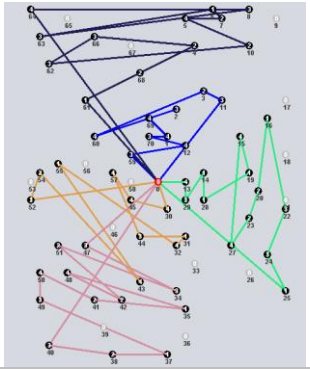
25



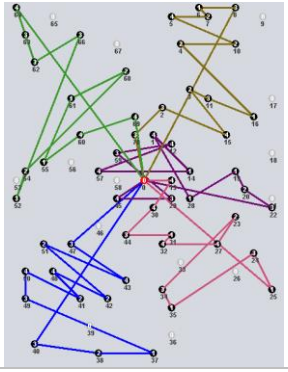
26

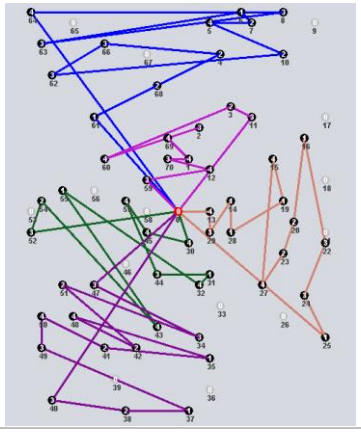
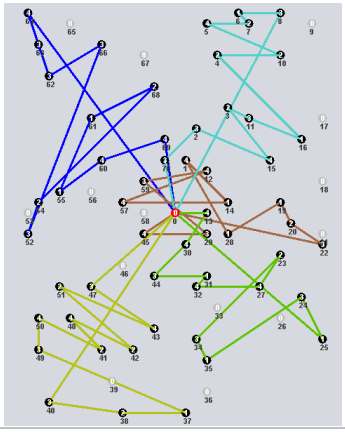


27



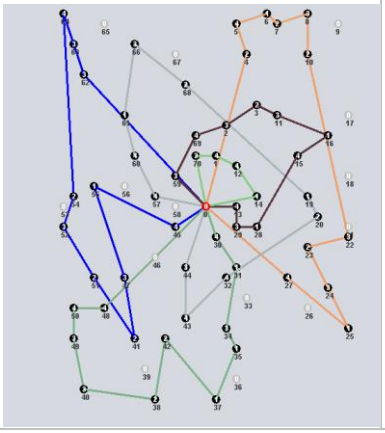
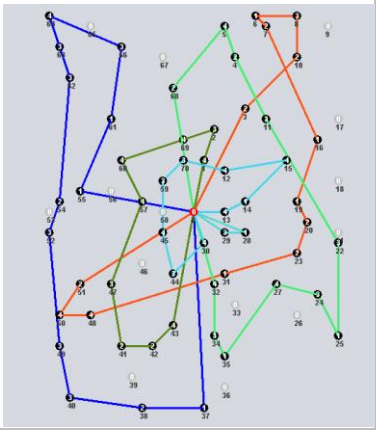
28



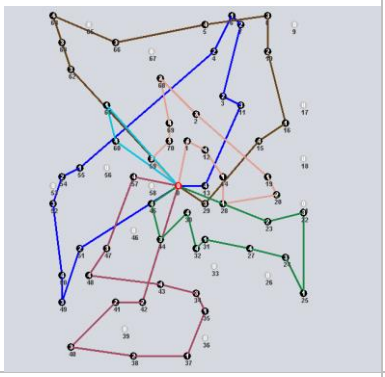
29	30
	

b. Graf Solusi *Case* 2 CVRP dengan Menggunakan Algoritma *Harmony Search*

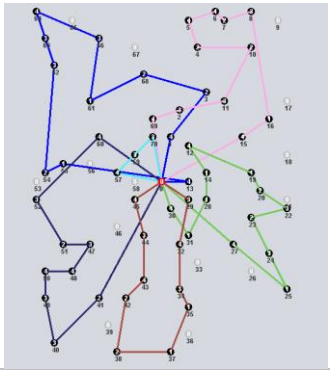
Tabel E.2 Graf Solusi *Case* 2 CVRP dengan Menggunakan Algoritma *Harmony Search* (urutan berdasarkan nomor percobaan)

1	2
	

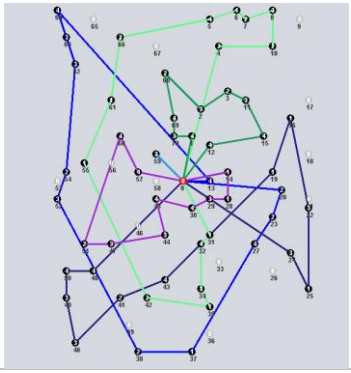
3



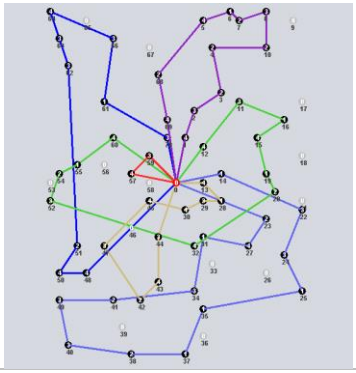
4



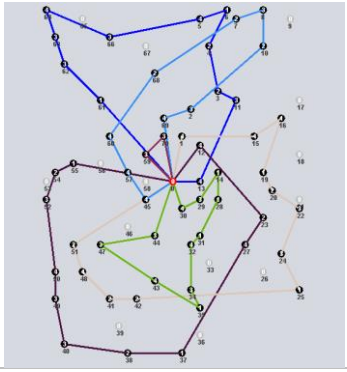
5



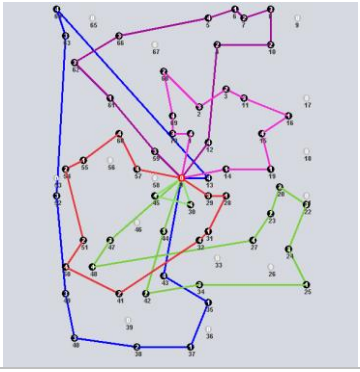
6



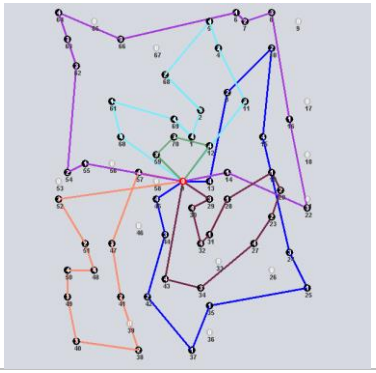
7



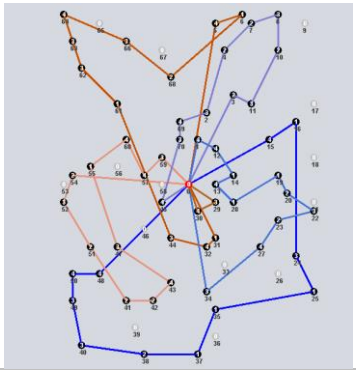
8



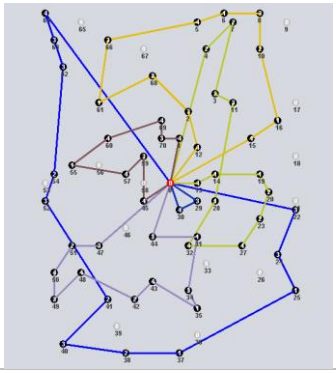
9



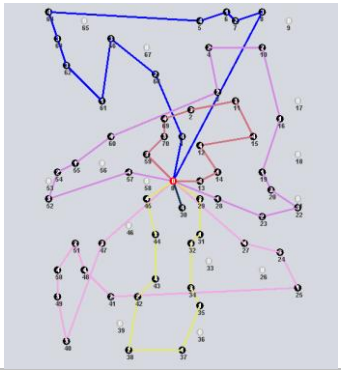
10



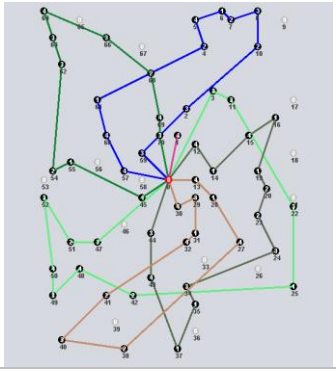
11



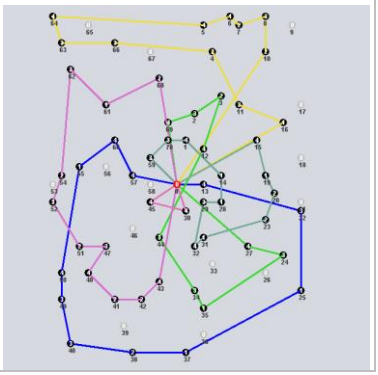
12



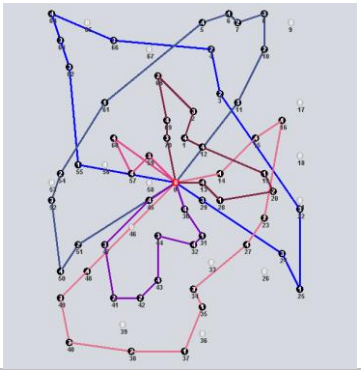
13



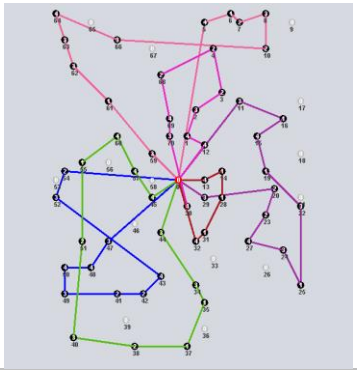
14



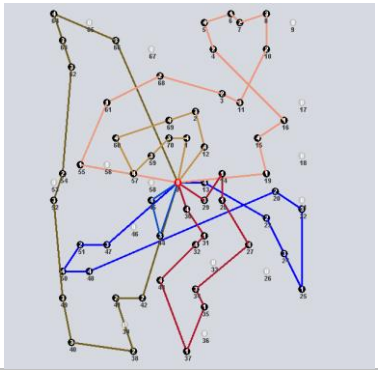
15



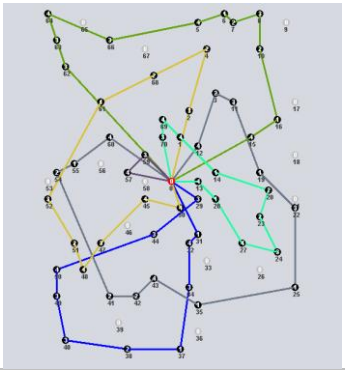
16



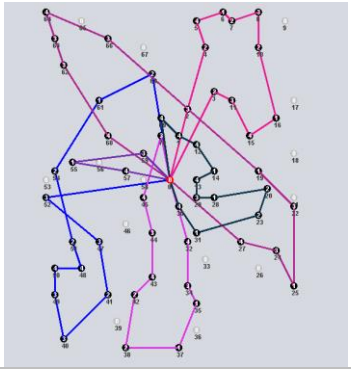
17



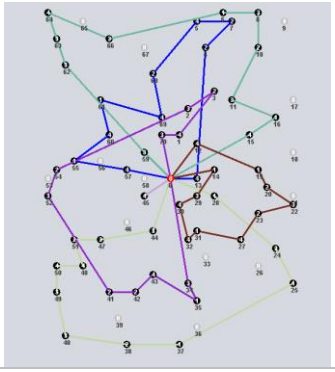
18



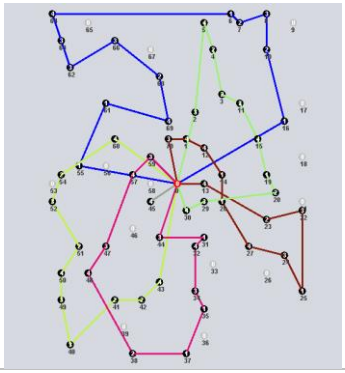
19



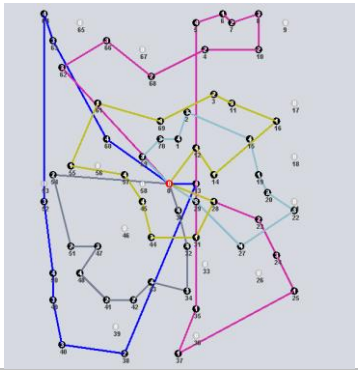
20



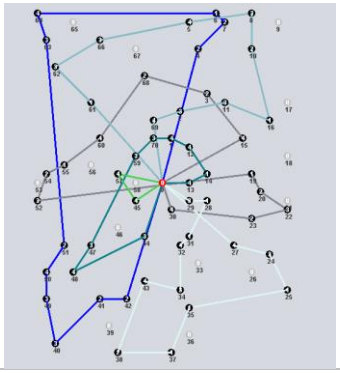
21



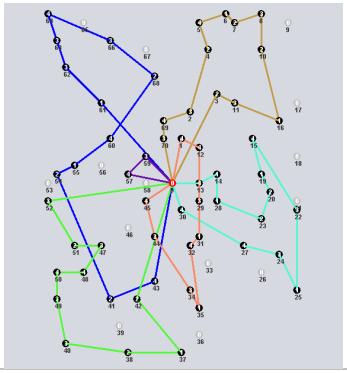
22



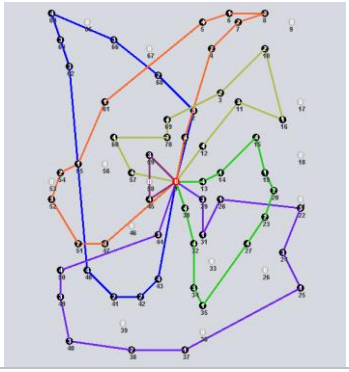
23



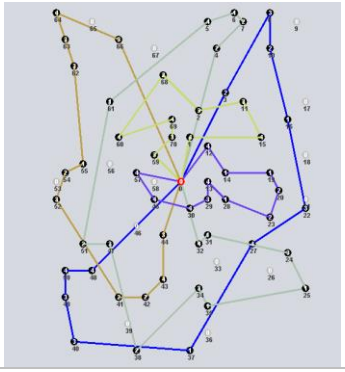
24

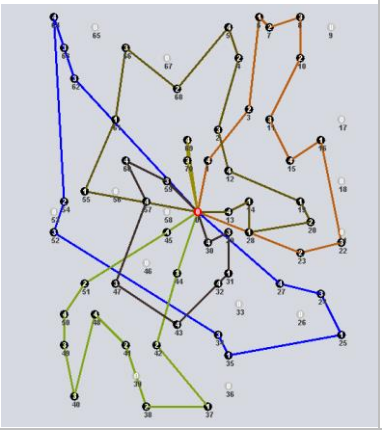
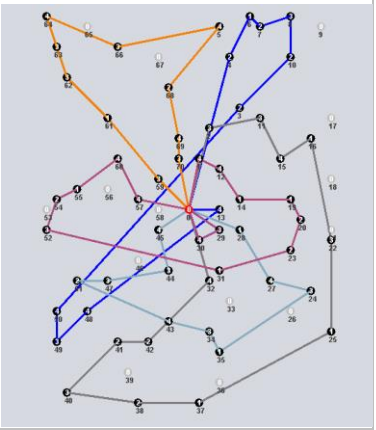
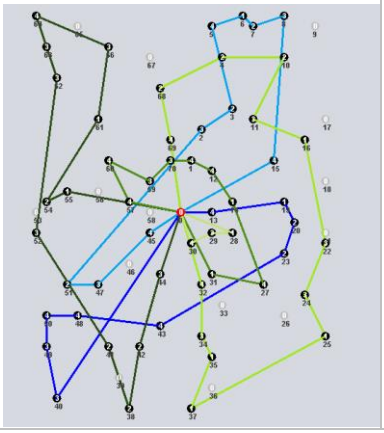
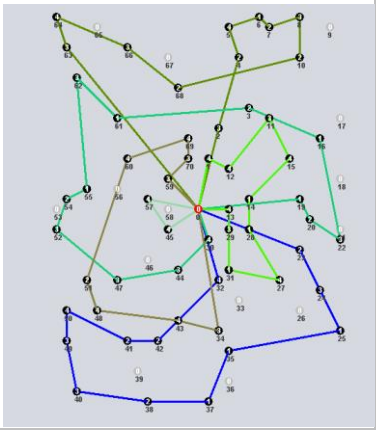


25



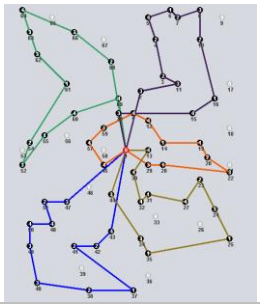
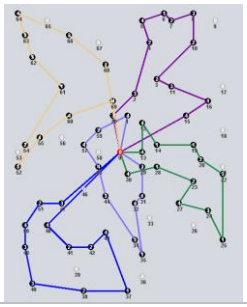
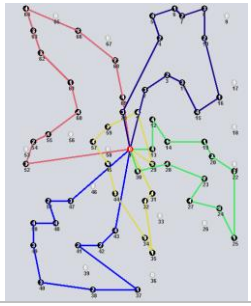
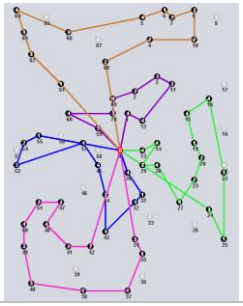
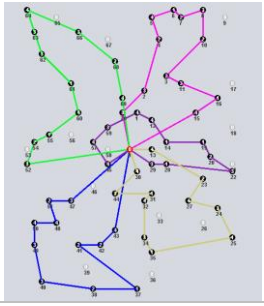
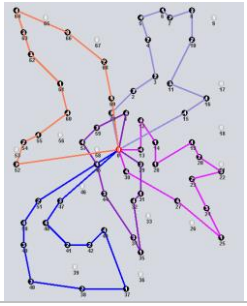
26



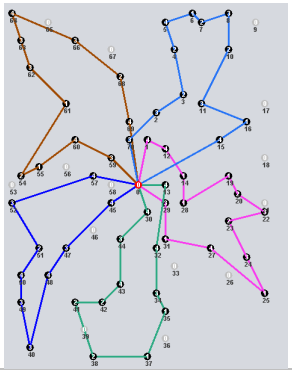
27	28
	
29	30
	

c. Graf Solusi *Case* 2 CVRP dengan Menggunakan Algoritma *Hybrid Harmony Search*

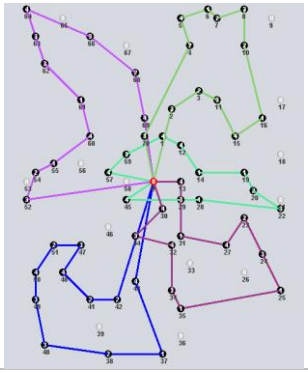
Tabel E.3 Graf Solusi *Case* 2 CVRP dengan Menggunakan Algoritma *Hybrid Harmony Search* (urutan berdasarkan nomor percobaan)

1	2
	
3	4
	
5	6
	

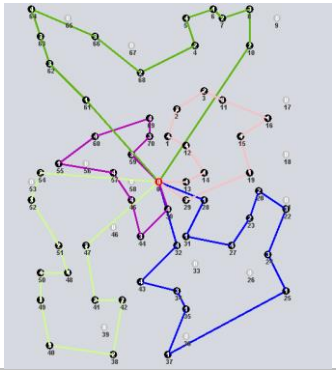
7



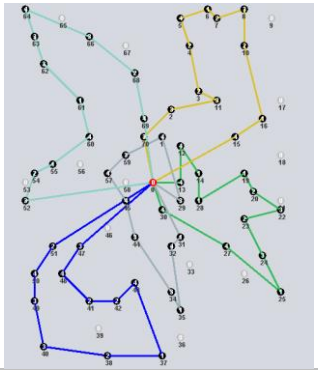
8



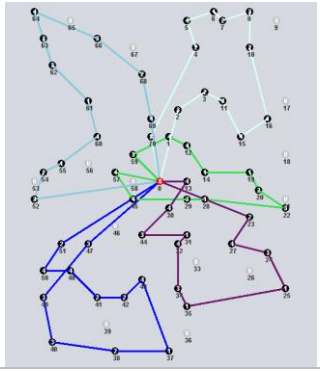
9



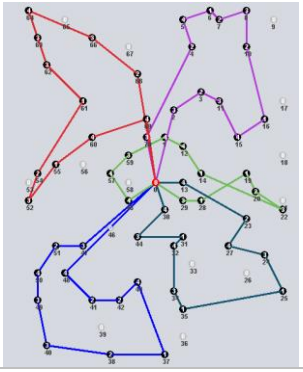
10



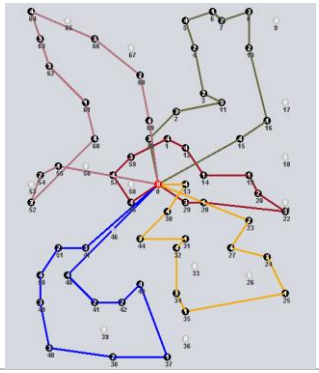
11



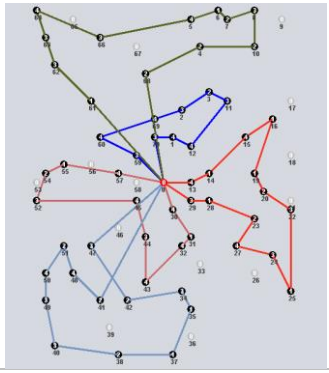
12



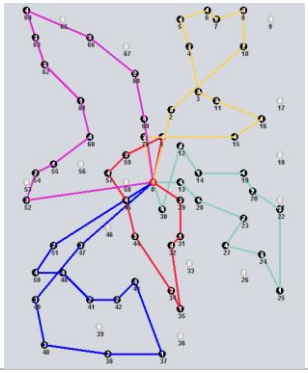
13



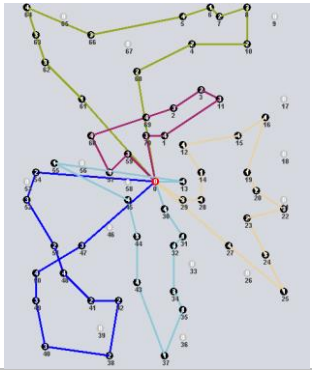
14



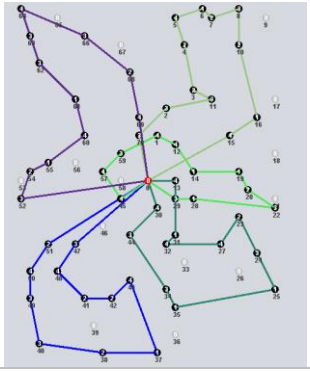
15



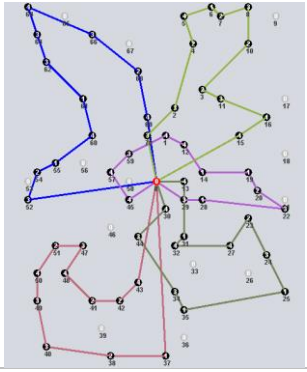
16



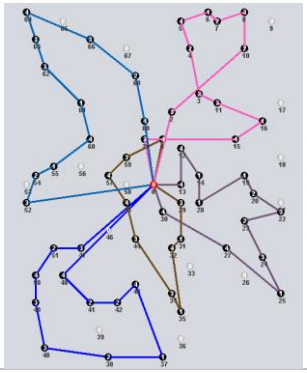
17



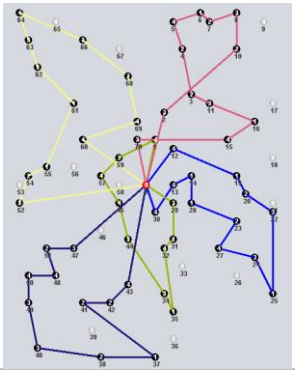
18



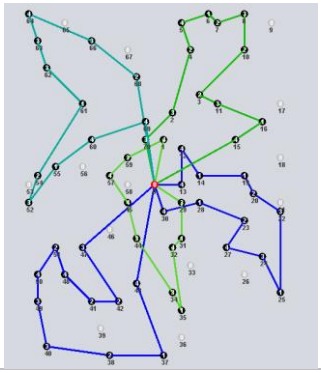
19



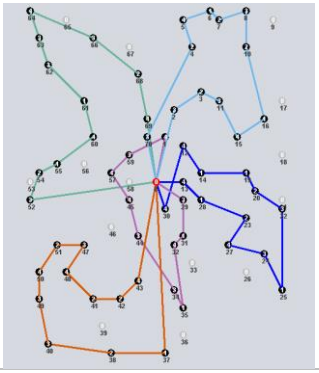
20



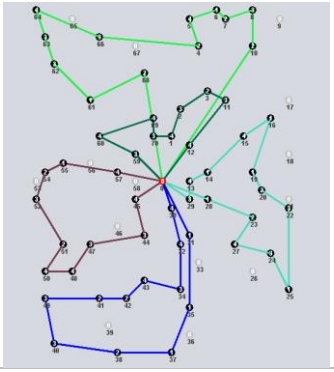
21



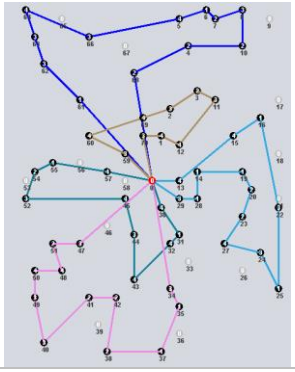
22



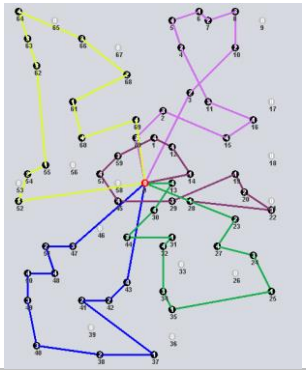
23



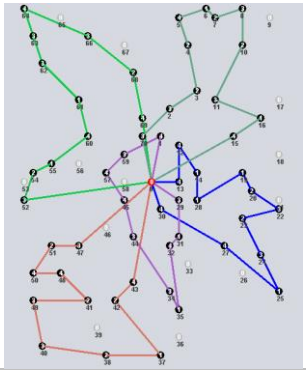
24



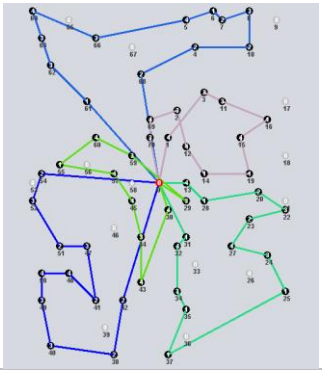
25



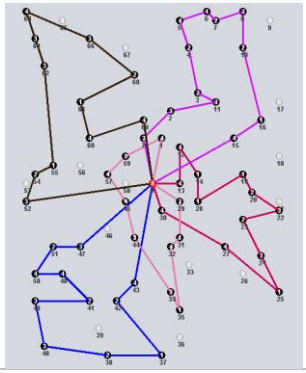
26



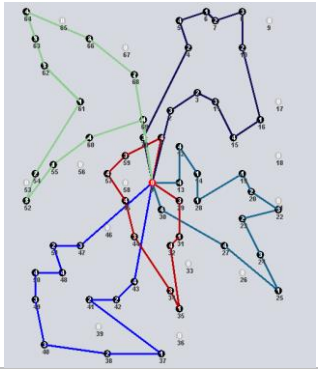
27



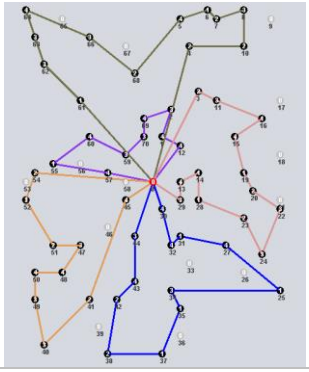
28



29



30



LAMPIRAN F
Source Code

a. *Simulation*

```
package purehs;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileFilter;
import javax.swing.table.DefaultTableModel;

/**
 *
 * @author Dinan
 */
public class simulation extends
javax.swing.JFrame {

     /**
     * Creates new form simulation
     */
    public Data[] chosendata;
    Integer vehiclecapacity;
    Integer hmsValuehs, iterationValuehs;
    Double hmcrValuehs, parValuehs;
    Integer populationValuega,
iterationValuega;
    Double mutationValuega, crossoverValuega;
    Integer hmsValue, iteration1, iteration2,
populationValue;
    Double hmcrValue, parValue, mutationValue,
crossoverValue;
    iFitnessAble fitnessUtils;
    iClusterAble clusterUtils;
```

```
    Integer[] finalRoute;
    Integer[] finalRoutega;
    Integer[] finalRoutehs;
    public Double[][] distances;
    HarmonySearch hSearch;

    public simulation() {
        initComponents();
        fitnessUtils = new iFitnessAble() {
            @Override
            public double
getFitnessValue(Integer[] harmonyVector) {
                double total = 0;
                int tempWeight =
vehiclecapacity;
                int tempatAsal = 0;
                int tempatTujuan;
                int demandcustomer;
                for (int i = 0; i <
harmonyVector.length; i++) {
                    tempatTujuan =
harmonyVector[i];
                    demandcustomer =
chosendata[tempatTujuan].demand;
                    if (tempWeight <
demandcustomer) {
                        total +=
distances[tempatAsal][0];
                        tempWeight =
vehiclecapacity;
                        tempatAsal = 0;
                    }
                    tempWeight -=
demandcustomer;
                    total +=
distances[tempatAsal][tempatTujuan];
                    tempatAsal = tempatTujuan;
                }
                total +=
distances[tempatAsal][0];
                return 1.0 / total;
            }
        };
```

```
        clusterUtils = new iClusterAble() {
            @Override
            public double
getEuclidianDistanceBetween(int pointA, int
pointB) {
                return
distances[pointA][pointB];
            }

            @Override
            public double
getEuclidianDistanceBetween(int pointA, double
centerX, double centerY) {
                double x =
Math.pow(chosendata[pointA].numX - centerX, 2);
                double y =
Math.pow(chosendata[pointA].numY - centerY, 2);
                return Math.sqrt(x + y);
            }

            @Override
            public int getX(int customer) {
                return
chosendata[customer].numX;
            }

            @Override
            public int getY(int customer) {
                return
chosendata[customer].numY;
            }

            @Override
            public double
getEuclidianDistanceFromDepot(int pointA) {
                return distances[0][pointA];
            }

            @Override
            public double
getEuclidianDistanceFromDepot(double centerX,
double centerY) {
```

```
                return
getEuclidianDistanceBetween(0, centerX,
centerY);
            }
        };
    }

    public boolean CheckingDemand(Data[]
chosendata) {
        for (int i = 1; i < chosendata.length;
i++){
            if (chosendata[i].demand >
vehiclecapacity){
                JOptionPane optionPane = new
JOptionPane();
                optionPane.setMessage("Demand
of customer " +i+ " exceeds the vehicle
capacity!");

optionPane.setMessageType(JOptionPane.WARNING_M
ESSAGE);
                JDialog dialog =
optionPane.createDialog(null, "Warning
Message");
                dialog.setVisible(true);
                return false;
            }
        }
        return true;
    }

    public void RandomizeDemand() {
        Random random = new Random();
        int randomNumber;
        DefaultTableModel model =
(DefaultTableModel) data.getModel();
        Object rowData[] = new Object[4];
        for (int i = 1; i < chosendata.length;
i++) {
            randomNumber = random.nextInt((4) +
1) + 0;
            chosendata[i].demand =
randomNumber;
            rowData[0] = i;
```

```
            rowData[1] = chosendata[i].numX;
            rowData[2] = chosendata[i].numY;
            rowData[3] = chosendata[i].demand;
            model.addRow(rowData);
        }
    }

    public ArrayList<Integer> adaDemand() {
        int i, j;
        ArrayList<Integer> adaDemand = new
ArrayList<>();

        for (i = 0; i < chosendata.length; i++)
{
            if (chosendata[i].demand != 0) {
                adaDemand.add(i);
            }
        }
        return adaDemand;
    }

    public Double[][] hitungJarak() {
        distances = new
Double[chosendata.length][chosendata.length];
        for (int i = 0; i < chosendata.length;
i++) {
            for (int j = 0; j <= i; j++) {
                if (j == i) {
                    distances[i][j] = 0.0000;
                } else {
                    double x =
Math.pow(chosendata[j].numX -
chosendata[i].numX, 2);
                    double y =
Math.pow(chosendata[j].numY -
chosendata[i].numY, 2);
                    distances[i][j] =
Math.sqrt(x + y);
                    distances[j][i] =
Math.sqrt(x + y);
                }
                System.out.print("jarak " + i +
" ke " + j + " yaitu " + distances[i][j]);
                System.out.println();
```

```
            }
        }
        return distances;
    }

    private int[] getRawDemand() {
        int[] demands = new
int[chosendata.length];
        for (int i = 0; i < chosendata.length;
i++) {
            demands[i] = chosendata[i].demand;
        }
        return demands;
    }

    private Integer[] initGAxHarmony() {
        Integer[] result = new
Integer[adaDemand().size()];
        Double[][] jarak = distances;
        return result;
    }

    private Integer[]
generateFinalRoute(Integer[] rawRoute) {
        List<Integer> route = new
ArrayList<>();
        double total = 0;
        int tempWeight = vehiclecapacity;
        int tempatAsal = 0;
        int tempatTujuan;
        int demandcustomer;
        route.add(0);
        for (int i = 0; i < rawRoute.length;
i++) {
            tempatTujuan = rawRoute[i];
            demandcustomer =
chosendata[tempatTujuan].demand;
            if (tempWeight < demandcustomer) {
                total +=
distances[tempatAsal][0];
                route.add(0);
                tempWeight = vehiclecapacity;
                tempatAsal = 0;
            }
```

```
            tempWeight -= demandcustomer;
            total +=
distances[tempatAsal][tempatTujuan];
            tempatAsal = tempatTujuan;
            route.add(tempatTujuan);
        }
        total += distances[tempatAsal][0];
        route.add(0);
        System.out.println("Final Route: " +
route.toString());
        System.out.println("Total Distance: " +
total);
        return route.toArray(new Integer[0]);
    }
private void
case1chosenActionPerformed(java.awt.event.Actio
nEvent evt) {
        chosendata = Case1;
        addChoosenDataToTabel();
    }
private void
case2chosenActionPerformed(java.awt.event.Actio
nEvent evt) {
        chosendata = Case2;
        addChoosenDataToTabel();
    }
private void
clearActionPerformed(java.awt.event.ActionEvent
evt) {
        data.setModel(new
DefaultTableModel(null, new
String[]{"Customer", "Abscissa", "Ordinate",
"Demand"}));
        chosendata = null;
    }
private void
hhsActionPerformed(java.awt.event.ActionEvent
evt) {
        mutationValue =
Double.parseDouble(mutation.getText());
        crossoverValue =
Double.parseDouble(crossover.getText());
        iteration1 =
Integer.parseInt(ni1.getText());
```

```
        populationValue =
Integer.parseInt(population.getText());

        hmsValue =
Integer.parseInt(hms.getText());
        vehiclecapacity =
Integer.parseInt(capacity.getText());
        iteration2 =
Integer.parseInt(ni2.getText());
        hmcrValue =
Double.parseDouble(hmcr.getText());
        parValue =
Double.parseDouble(par.getText());
        hitungJarak();
        GAxHarmony gxh = new
GAxHarmony(getRawDemand(), vehiclecapacity,
iteration1, populationValue, mutationValue,
crossoverValue, iteration2, hmsValue,
                hmcrValue, parValue,
fitnessUtils, clusterUtils);
        Integer[] rawRoute =
gxh.getBestFitness();
        finalRoute =
generateFinalRoute(rawRoute);
    }

    private void
graphActionPerformed(java.awt.event.ActionEvent
evt) {
        Graph graph = new Graph(finalRoute,
chosendata);
    }
private void
graphhsActionPerformed(java.awt.event.ActionEve
nt evt) {
        Graph graph = new Graph(finalRoutehs,
chosendata);
    }
private void
graphgaActionPerformed(java.awt.event.ActionEve
nt evt) {
        Graph graph = new Graph(finalRoutega,
chosendata);
    }
```

```
private void
gaActionPerformed(java.awt.event.ActionEvent
evt) {
        mutationValuega =
Double.parseDouble(mutationga.getText());
        crossoverValuega =
Double.parseDouble(crossoverga.getText());
        iterationValuega =
Integer.parseInt(niga.getText());
        populationValuega =
Integer.parseInt(populationga.getText());
        vehiclecapacity =
Integer.parseInt(capacity.getText());
        hitungJarak();
        GeneticAlgorithm ga = new
GeneticAlgorithm(getRawDemand(),
vehiclecapacity,
                iterationValuega,
populationValuega, mutationValuega,
crossoverValuega, fitnessUtils, clusterUtils);
        Integer[] rawRoute =
ga.getBestFitness();
        finalRoutega =
generateFinalRoute(rawRoute);
    }

    private void
hsActionPerformed(java.awt.event.ActionEvent
evt) {
        hmsValuehs =
Integer.parseInt(hmshs.getText());
        vehiclecapacity =
Integer.parseInt(capacity.getText());
        iterationValuehs =
Integer.parseInt(nihs.getText());
        hmcrValuehs =
Double.parseDouble(hmcrhs.getText());
        parValuehs =
Double.parseDouble(parhs.getText());
        hitungJarak();
        HarmonySearch hs = new
HarmonySearch(getRawDemand(), iterationValuehs,
hmsValuehs, hmcrValuehs, parValuehs,
fitnessUtils, clusterUtils);
```

```
        Integer[] rawRoute =
hs.getBestFitness();
        finalRoutehs =
generateFinalRoute(rawRoute);
    }

    private void
chooserActionPerformed(java.awt.event.ActionEve
nt evt) {
        JFileChooser browse = new
JFileChooser();
        browse.setFileFilter(new FileFilter() {
            @Override
            public String getDescription() {
                return "Excel file (*.xlsx)";
            }
            @Override
            public boolean accept(File f) {
                if (f.isDirectory()) {
                    return true;
                } else {
                    String filename =
f.getName().toLowerCase();
                    return
filename.endsWith(".xlsx") ;
                }
            }
         });
        browse.showOpenDialog(null);
        File f = browse.getSelectedFile();
        String filename = f.getAbsolutePath();
        ExcelReader reader = new ExcelReader();
        Data[] data =
reader.transformToData(filename);
        vehiclecapacity =
Integer.parseInt(capacity.getText());
        if(CheckingDemand(data)) {
            chosendata = data;
            addChoosenDataToTabel();
        }
    }
```

```
    private void
templateButtonActionPerformed(java.awt.event.Ac
tionEvent evt) {
        JFileChooser savefile = new
JFileChooser();
int sf = savefile.showSaveDialog(null);
        if(sf == JFileChooser.APPROVE_OPTION){
            try {
                File source = new
File("template.xlsx");
                File dest =
savefile.getSelectedFile();
                System.out.println("source " +
source.toString());
                System.out.println("dest " +
dest.toString());
                if
(!dest.getName().endsWith(".xlsx")){
                    dest = new
File(savefile.getSelectedFile().getAbsolutePath
() + ".xlsx");
                }
                Files.copy(source.toPath(),
dest.toPath());

JOptionPane.showMessageDialog(null, "File has
been saved: " + dest.getAbsolutePath(),"File
Saved",JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed"
desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6)
is not available, stay with the default look
and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uisw
ing/lookandfeel/plaf.html
         */
        try {
```

```
            for
(javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels(
)) {
                if
("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getCl
assName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(simulation.c
lass.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(simulation.c
lass.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(simulation.c
lass.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);
        } catch
(javax.swing.UnsupportedLookAndFeelException
ex) {

java.util.logging.Logger.getLogger(simulation.c
lass.getName()).log(java.util.logging.Level.SEV
ERE, null, ex);
        }

        //</editor-fold>
        simulation msimulation = new
simulation();

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new
Runnable() {
            public void run() {
```

```
                msimulation.setVisible(true);
            }
        });
    }

    public void addChoosenDataToTabel() {
        DefaultTableModel model =
(DefaultTableModel) data.getModel();
        Object rowData[] = new Object[4];
        for (int i = 1; i < chosendata.length;
i++) {
            rowData[0] = i;
            rowData[1] = chosendata[i].numX;
            rowData[2] = chosendata[i].numY;
            rowData[3] = chosendata[i].demand;
            model.addRow(rowData);
        }
    }

    public Data[] Case1 = {
        new Data(-11, 0, 0),//DEPOT
        new Data(-13, 1, 2), //1
        new Data(-14, 4, 4),
        new Data(-14, 7, 3),
        new Data(-13, 9, 0),
        new Data(-11, 12, 0),
        new Data(-12, 14, 4),
        new Data(-13, 17, 1),
        new Data(-11, 19, 2),
        new Data(-7, 18, 3),
        new Data(-4, 19, 4),//10
        new Data(-7, 15, 1),
        new Data(-8, 10, 1),
        new Data(-9, 8, 2),
        new Data(-11, 5, 3),
        new Data(-11, 3, 0), //15

        new Data(-8, 4, 3),
        new Data(-7, 6, 0),
        new Data(-5, 8, 2),//18
        new Data(-5, 11, 0),
        new Data(-2, 14, 1),
        new Data(-2, 17, 2),
        new Data(2, 19, 2),
```

```
        new Data(3, 16, 1),
        new Data(7, 16, 2),
        new Data(11, 17, 0),//25
        new Data(14, 19, 4),
        new Data(14, 16, 1),
        new Data(12, 13, 2),
        new Data(11, 10, 1),
        new Data(8, 10, 4),
        new Data(5, 10, 0),
        new Data(3, 8, 1),
        new Data(1, 6, 3),
        new Data(-2, 4, 0),
        new Data(-6, 3, 1),//35

        new Data(-7, -1, 4), //1
        new Data(-4, 0, 4),
        new Data(-2, 1, 0),
        new Data(2, 2, 1),
        new Data(7, 3, 2),//40
        new Data(11, 5, 0),
        new Data(12, 3, 4),
        new Data(12, 0, 0),
        new Data(13, -3, 2),
        new Data(13, -9, 3),//45
        new Data(11, -7, 1), //1
        new Data(9, -5, 2),
        new Data(7, -4, 0),
        new Data(2, -3, 1),
        new Data(-2, -3, 0),//50
        new Data(-5, -3, 2),
        new Data(-8, -3, 4), //52

        new Data(-11, -4, 4), //1
        new Data(-6, -8, 3),
        new Data(-2, -9, 1), //55
        new Data(0, -11, 0),
        new Data(5, -12, 0),
        new Data(6, -13, 1),
        new Data(10, -14, 3),
        new Data(13, -18, 2),
        new Data(8, -18, 1),
        new Data(3, -18, 1),//62
        new Data(0, -16, 4), //63
        new Data(-8, -19, 3),//64
```

```
        new Data(-11, -17, 0),//65
        new Data(-14, -18, 1),//66
        new Data(-12, -13, 0),//67
        new Data(-12, -10, 2),//68
        new Data(-13, -8, 1),//69
        new Data(-14, -6, 3) //70
    };

    public Data[] Case2 = {
        new Data(0, 0, 0),//DEPOT
        new Data(1, 5, 4), //1
        new Data(2, 8, 3),
        new Data(5, 10, 2),
        new Data(4, 15, 2),
        new Data(3, 18, 4),//5
        new Data(6, 19, 1),
        new Data(7, 18, 2),
        new Data(10, 19, 3),
        new Data(13, 18, 0),
        new Data(10, 15, 2),//10
        new Data(7, 9, 3),
        new Data(3, 4, 4), //12

        new Data(3, 0, 4),
        new Data(5, 1, 1),
        new Data(9, 5, 4),//15
        new Data(12, 7, 1),
        new Data(14, 9, 0),
        new Data(14, 3, 0),
        new Data(10, 1, 1),
        new Data(11, -1, 2),
        new Data(14, -2, 0),
        new Data(14, -3, 3),//22
        new Data(10, -4, 2),
        new Data(12, -8, 3),
        new Data(14, -12, 1),
        new Data(10, -10, 0),
        new Data(8, -7, 4),
        new Data(5, -2, 1),
        new Data(3, -2, 3), //29

        new Data(1, -3, 4), //1
        new Data(3, -6, 1),
        new Data(2, -7, 4),
```

```
        new Data(4, -9, 0),
        new Data(2, -12, 3),
        new Data(3, -14, 1),//35
        new Data(3, -17, 0),
        new Data(1, -19, 1),
        new Data(-5, -19, 2),
        new Data(-6, -16, 0),//39
        new Data(-12, -18, 3), //1
        new Data(-7, -13, 2),
        new Data(-4, -13, 2),
        new Data(-2, -11, 4),
        new Data(-2, -6, 3), //44

        new Data(-3, -2, 4), //1
        new Data(-5, -5, 0),
        new Data(-8, -7, 3),
        new Data(-10, -10, 4),
        new Data(-13, -13, 3),
        new Data(-13, -10, 4),//50
        new Data(-11, -7, 2),
        new Data(-14, -2, 3),
        new Data(-14, 0, 0),
        new Data(-13, 1, 2),//54
        new Data(-11, 2, 1), //1
        new Data(-8, 2, 0),
        new Data(-5, 1, 4),
        new Data(-3, 0, 0), //58

        new Data(-3, 3, 3),
        new Data(-7, 5, 4),//60
        new Data(-8, 9, 1),
        new Data(-12, 13, 3),
        new Data(-13, 16, 3),
        new Data(-14, 19, 4),
        new Data(-10, 18, 0),//65
        new Data(-7, 16, 3),
        new Data(-3, 15, 0),
        new Data(-2, 12, 2),//68
        new Data(-1, 7, 4), //1
        new Data(-1, 5, 3)
    };
```

```
    // Variables declaration - do not modify
    private javax.swing.JTextField capacity;
    private javax.swing.JLabel capacitylabel;
    private javax.swing.JButton case1chosen;
    private javax.swing.JButton case2chosen;
    private javax.swing.JButton chooser;
    private javax.swing.JButton clear;
    private javax.swing.JTextField crossover;
    private javax.swing.JTextField crossoverga;
    private javax.swing.JLabel
crossovergalabel;
    private javax.swing.JLabel crossoverlabel;
    private javax.swing.JTable data;
    private javax.swing.JButton ga;
    private javax.swing.JButton graph;
    private javax.swing.JButton graphga;
    private javax.swing.JButton graphhs;
    private javax.swing.JButton hhs;
    private javax.swing.JTextField hmcr;
    private javax.swing.JTextField hmcrhs;
    private javax.swing.JLabel hmcrhslabel;
    private javax.swing.JLabel hmcrlabel;
    private javax.swing.JTextField hms;
    private javax.swing.JTextField hmshs;
    private javax.swing.JLabel hmshslabel;
    private javax.swing.JLabel hmslabel;
    private javax.swing.JButton hs;
    private javax.swing.JDialog jDialog1;
    private javax.swing.JScrollPane
jScrollPane5;
    private javax.swing.JScrollPane
jScrollPane6;
    private javax.swing.JScrollPane
jScrollPane7;
    private javax.swing.JTable jTable2;
    private javax.swing.JTextField mutation;
    private javax.swing.JTextField mutationga;
    private javax.swing.JLabel mutationgalabel;
    private javax.swing.JLabel mutationlabel;
    private javax.swing.JTextField ni1;
    private javax.swing.JLabel ni1label;
    private javax.swing.JTextField ni2;
    private javax.swing.JLabel ni2label;
    private javax.swing.JTextField niga;
```

```
    private javax.swing.JLabel nigalabel;
    private javax.swing.JTextField nihs;
    private javax.swing.JLabel nihslabel;
    private javax.swing.JTextField par;
    private javax.swing.JTextField parhs;
    private javax.swing.JLabel parhslabel;
    private javax.swing.JLabel parlabel;
    private javax.swing.JTextField population;
    private javax.swing.JTextField
populationga;
    private javax.swing.JLabel
populationgalabel;
    private javax.swing.JLabel populationlabel;
    private javax.swing.JTable tabeldata1;
    private javax.swing.JButton templateButton;
    private javax.swing.JOptionPane warning1;
    // End of variables declaration
}
```

b. *GeneticAlgorithm*

```
package purehs;
import java.util.*;
import javafx.geometry.Point2D;

public class GeneticAlgorithm {
    iFitnessAble fitnessDelegate;
    iClusterAble clusterDelegate;
    int practiceTime;
    double[] fitnessValues;
    double mutationProbability;
    double crossOverProbability;
    List<Integer> isTaken;
    HashMap<Integer, Integer> demands;
    List<Integer> instances;
    int[] fitnessValueRank;
    int maxDemand;
    int populationCount;
    List< List <Point2D> > centerPopulation;


    public GeneticAlgorithm(int[] demands, int
maxDemand, int GAPracticeTime, int
populationCount, double mutationRate, double
```

```
crossOverRate, iFitnessAble fitnessDelegate,
iClusterAble clusterDelegate) {
        this.practiceTime = GAPracticeTime;
        this.maxDemand = maxDemand;
        this.fitnessDelegate = fitnessDelegate;
        this.clusterDelegate = clusterDelegate;
        this.demands = new HashMap<>();
        this.instances = new ArrayList<>();
        this.centerPopulation = new
ArrayList<>();
        this.mutationProbability =
mutationRate;
        this.crossOverProbability =
crossOverRate;
        this.populationCount = populationCount;
        filterDemands(demands);
        fitnessValues = new
double[populationCount];
        isTaken = new ArrayList<>();
        fitnessValueRank = new
int[populationCount];
        initClusterPopulation();
        refreshFitness();
    }

    public Integer[] getBestFitness() {
        long startTime =
System.currentTimeMillis();
        doGA();
        int bestIndex =
getFitnessIndexAtRank(1);
        System.out.println("Best Fitness = " +
fitnessValues[bestIndex]);
        System.out.println("Execution time (s):
" + (System.currentTimeMillis() - startTime) /
1000.0);
        List< List<Integer>> bestRoute =
generateClustersFromCenters(centerPopulation.ge
t(bestIndex));
        return
makeVector(bestRoute).toArray(new Integer[0]);
    }

   private void doGA() {
```

```
        for(int i = 0; i<practiceTime; i++) {
            double rand1 = getRandom();
            int randRow1 =
getRowBasedOnProbability(rand1);
            int randRow2;
            do {
                double rand2 = getRandom();
                randRow2 =
getRowBasedOnProbability(rand2);
            } while (randRow2 == randRow1);
            List<Point2D> newCenters =
doCrossover(centerPopulation.get(randRow1),
centerPopulation.get(randRow2));
            newCenters =
doMutation(newCenters);
            List< List<Integer>> newClusters =
generateClustersFromCenters(newCenters);
            if(newClusters.size() !=
newCenters.size()) {
                i--;
                continue;
            }
            double lowestFitness =
fitnessValues[getLowestFitnessIndex()];
            double fitness =
getClusterFitness(newCenters);
            if(lowestFitness < fitness) {

centerPopulation.set(getLowestFitnessIndex(),
newCenters);

fitnessValues[getLowestFitnessIndex()] =
fitness;
                refreshFitnessRank();
            }
            System.out.println("#" + i + " |
GA-Best Fitness  = " +
fitnessValues[getFitnessIndexAtRank(1)]);
            System.out.println("#" + i + " |
GA-Worst Fitness = " +
fitnessValues[getLowestFitnessIndex()]);
        }
    }
```

```
    private void initClusterPopulation() {
        for (int i=0; i<populationCount; i++) {
            List<Integer> centers =
initClusterCenter();
            List<Point2D> centerPoints = new
ArrayList<>();
            for(int center: centers) {
                centerPoints.add(new
Point2D((double) clusterDelegate.getX(center),
(double) clusterDelegate.getY(center)));
            }
            List< List<Integer>> clusters =
generateClustersFromCenters(centerPoints);
            if(clusters.size() !=
centers.size()) {
                i--;
                continue;
            }
            centerPopulation.add(centerPoints);
        }
    }

    private List<Integer> initClusterCenter() {
        double totalDemand = 0;
        List<Integer> centers = new
ArrayList<>();
        for(int customer: instances) {
            totalDemand +=
demands.get(customer);
        }
        int k =
(int)(Math.ceil(totalDemand/(double)
maxDemand));
        for(int i=0; i<k; i++) {

centers.add(getNextRandomCustomerFor(centers));
        }
        return centers;
    }

    private void refreshFitness() {
        for(int i = 0; i < populationCount;
i++) {
```

```
            fitnessValues[i] =
getClusterFitness(centerPopulation.get(i));
        }
        refreshFitnessRank();
    }

    private void refreshFitnessRank() {
        for(int i = 0; i < populationCount;
i++) {
            int rank = 1;
            for(int j = 0; j < populationCount;
j++) {
                if(fitnessValues[i] <
fitnessValues[j]) {
                    rank++;
                } else if(fitnessValues[i] ==
fitnessValues[j] && i > j) {
                    rank++;
                }
            }
            fitnessValueRank[i] = rank;
        }
    }

    private int getLowestFitnessIndex() {
        return
getFitnessIndexAtRank(populationCount);
    }

    private int getFitnessIndexAtRank(int rank)
{
        int index = populationCount - 1;
        for(int i = 0; i < populationCount;
i++) {
            if(fitnessValueRank[i] == rank) {
                index = i;
                break;
            }
        }
        return index;
    }

    private int getRowBasedOnProbability(double
random) {
```

```
        int randRow = 0;

while(getUpperBoundProbabilityAt(randRow) <=
random) {
            randRow++;
        }
        return randRow;
    }

    private double getProbabilityAt(int index)
{
        double result;
        double total = 0;
        for (double fitness: fitnessValues) {
            total += fitness;
        }
        result = fitnessValues[index] / total;
        return result;
    }

    private double
getUpperBoundProbabilityAt(int index) {
        double bound = getProbabilityAt(0);
        for(int i = 1; i <= index; i++) {
            bound += getProbabilityAt(i);
        }
        return bound;
    }

    private void filterDemands(int[]
rawDemands) {
        for(int i=0; i < rawDemands.length;
i++) {
            if(rawDemands[i] > 0) {
                demands.put(i, rawDemands[i]);
                instances.add(i);
            }
        }
    }

    private List<Integer> makeVector(List< List
<Integer>> clusters) {
        List<Integer> vector = new
ArrayList<>();
```

```
        for(List<Integer> cluster: clusters) {
            for(int customer: cluster) {
                vector.add(customer);
            }
        }
        return vector;
    }


    private int
getNextRandomCustomerFor(List<Integer>
currentRoute) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(currentRoute);
        int rand = (new
Random()).nextInt(diff.size());
        int value = diff.get(rand);
        return value;
    }

    private double getRandom() {
        return (((new
Random()).nextInt(10000000)) / 10000000.0);
    }

    private List < List<Integer>>
generateClustersFromCenters(List<Point2D>
centers) {
        List<Point2D> centerTemp = new
ArrayList<>();
        List < List<Integer>> clusters = new
ArrayList<>();
        List<Integer> clusterSpace = new
ArrayList<>();
        List<Integer> sortedCustomers =
sortCustomerFromDistance(instances);
        centerTemp.addAll(centers);
        for(Point2D center: centers) {
            clusterSpace.add(maxDemand);
            clusters.add(new ArrayList<>());
        }
        for(int customer: sortedCustomers) {
            int nearestCluster = -1;
```

```
            double nearestDistance =
clusterDelegate.getEuclidianDistanceBetween(cus
tomer, centerTemp.get(0).getX(),
centerTemp.get(0).getY());
            for(int i=0; i<centers.size(); i++)
{
                double distance =
clusterDelegate.getEuclidianDistanceBetween(cus
tomer, centerTemp.get(i).getX(),
centerTemp.get(i).getY());
                if(distance <= nearestDistance
&& clusterSpace.get(i) >=
demands.get(customer)) {
                    nearestCluster = i;
                    nearestDistance = distance;
                }
            }
            if(nearestCluster == -1) {
                List<Integer> cluster = new
ArrayList<>();
                centerTemp.add(new
Point2D(clusterDelegate.getX(customer),
clusterDelegate.getY(customer)));
                cluster.add(customer);
                clusters.add(cluster);
                clusterSpace.add(maxDemand -
demands.get(customer));
            } else {
                List<Integer> cluster =
clusters.get(nearestCluster);
                cluster.add(customer);

clusterSpace.set(nearestCluster,
clusterSpace.get(nearestCluster) -
demands.get(customer));
            }
        }
        return clusters;
    }

    private List<Integer>
sortCustomerFromDistance(List<Integer>
customers) {
```

```
        List<Integer> sorted = new
ArrayList<>();
        List<Double> distances = new
ArrayList<>();
        for(int c: customers) {
            double distance =
clusterDelegate.getEuclidianDistanceFromDepot(c
);
            sorted.add(c);
            distances.add(distance);
        }
        for(int i=0; i<customers.size(); i++) {
            int rank = 0;
            for(int j=0; j<customers.size();
j++) {
                if(distances.get(i) <
distances.get(j)) {
                    rank++;
                } else
if(Objects.equals(distances.get(i),
distances.get(j)) && i > j) {
                    rank++;
                }
            }
            sorted.set(rank, customers.get(i));
        }
        return sorted;
    }

    private List<Point2D>
doCrossover(List<Point2D> chromosomeA,
List<Point2D> chromosomeB) {
        List<Point2D> childVector = new
ArrayList<>();
        for(int i = 0; i < chromosomeA.size();
i++) {
            if(getRandom() <
crossOverProbability) {

childVector.add(chromosomeA.get(i));
            } else {

childVector.add(chromosomeB.get(i));
            }
```

```
        }
        return childVector;
    }

private List<Point2D> doMutation(List<Point2D>
centers) {
        List<Point2D> newCenters = new
ArrayList<>();
        newCenters.addAll(centers);
        for(int i = 0; i < centers.size(); i++)
{
            if(getRandom() <
mutationProbability) {
                if(getRandom() <= 0.5) {
                    int rand = (new
Random()).nextInt(instances.size());
                    Point2D randValue = new
Point2D(clusterDelegate.getX(rand),
clusterDelegate.getY(rand));
                    newCenters.set(i,
randValue);
                } else {
                    int rand = (new
Random()).nextInt(centers.size());
                    Point2D randValue =
newCenters.get(rand);
                    newCenters.set(rand,
newCenters.get(i));
                    newCenters.set(i,
randValue);
                }
            }
        }
        return  newCenters;
    }

    private double
getClusterFitness(List<Point2D> centers) {
        double total = 0;
        List< List<Integer>> clusters =
generateClustersFromCenters(centers);
        for(int i=0; i<clusters.size(); i++) {
            List<Integer> cluster =
clusters.get(i);
```

```
            if(i < centers.size()) {
                Point2D center =
centers.get(i);
                double totalX = 0;
                double totalY = 0;
                for(int point: cluster) {
                    total +=
clusterDelegate.getEuclidianDistanceBetween(poi
nt, center.getX(), center.getY());
                    totalX +=
clusterDelegate.getX(point);
                    totalY +=
clusterDelegate.getY(point);
                }
                total +=
clusterDelegate.getEuclidianDistanceFromDepot(c
enter.getX(), center.getY());
            } else {
                for(int point: cluster) {
                    total +=
clusterDelegate.getEuclidianDistanceBetween(poi
nt, centers.get(0).getX(),
centers.get(0).getY());
                }
                total +=
clusterDelegate.getEuclidianDistanceFromDepot(c
enters.get(0).getX(), centers.get(0).getY());
            }
        }
        return 1.0/total;
  }
}
```

c. *HarmonySearch*

```
package purehs;
import java.util.*;

public class HarmonySearch {
    iFitnessAble delegate;
    iClusterAble clusterDelegate;
    int practiceTime;
    int HMS;
```

```
    double HMCR;
    double PAR;
    double[] fitnessValues;
    List<Integer> isTaken;
    List<Integer> instances;
    List< List<Integer> > harmonyMemory;
    int[] fitnessValueRank;

    public HarmonySearch(int[] demands, int
practiceTime, int HMS, double HMCR, double PAR,
iFitnessAble delegate, iClusterAble
clusterDelegate) {
        this.practiceTime = practiceTime;
        this.HMS = HMS;
        this.HMCR = HMCR;
        this.PAR = PAR;
        this.delegate = delegate;
        harmonyMemory = new ArrayList<>();
        fitnessValues = new double[HMS];
        isTaken = new ArrayList<>();
        fitnessValueRank = new int[HMS];
        this.clusterDelegate = clusterDelegate;
        instances = new ArrayList<>();
        filterDemands(demands);
    }

    private void filterDemands(int[]
rawDemands) {
        for(int i=0; i < rawDemands.length;
i++) {
            if(rawDemands[i] > 0) {
                instances.add(i);
            }
        }
    }

    private void initHarmonyMemory() {
        for(int i = 0; i < HMS; i++) {
            List<Integer> harmony = new
ArrayList<>();
            for (int j = 0; j <
instances.size(); j++) {
                harmony.add(instances.get(j));
            }
```

```
            Collections.shuffle(harmony);
            harmonyMemory.add(harmony);
        }
    }

    Integer[] getBestFitness() {
        doHarmony();
        refreshFitnessRank();
        int bestIndex =
getFitnessIndexAtRank(1);
        return
harmonyMemory.get(bestIndex).toArray(new
Integer[0]);
    }

    private void refreshFitness() {
        for(int i = 0; i < HMS; i++) {
            fitnessValues[i] =
delegate.getFitnessValue(harmonyMemory.get(i).t
oArray(new Integer[0]));
        }
        refreshFitnessRank();
    }

    private void refreshFitnessRank() {
        for(int i = 0; i < HMS; i++) {
            int rank = 1;
            for(int j = 0; j < HMS; j++) {
                if(fitnessValues[i] <
fitnessValues[j]) {
                    rank++;
                } else if(fitnessValues[i] ==
fitnessValues[j] && i > j) {
                    rank++;
                }
            }
            fitnessValueRank[i] = rank;
        }
    }

    private int getLowestFitnessIndex() {
        return getFitnessIndexAtRank(HMS);
    }
```

```
    private int getFitnessIndexAtRank(int rank)
{
        int index = HMS - 1;
        for(int i = 0; i < HMS; i++) {
            if(fitnessValueRank[i] == rank) {
                index = i;
                break;
            }
        }
        return index;
    }

    private double getRandom() {
        return (((new
Random()).nextInt(10000000)) / 10000000.0);
    }

    private int getNextRandom() {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(isTaken);
        Integer rand = (new
Random()).nextInt(diff.size());
        int value = diff.get(rand);
        return value;
    }

    private void resetNextRandom() {
        isTaken.clear();
    }

    private List<Integer>
tieBreakCrossover(List<Integer> vector) {
        for(int i = 0; i < vector.size() - 1;
i++) {
            for(int j = i+1; j < vector.size();
j++ ) {

if(vector.get(i).equals(vector.get(j))) {
                    Integer rand =
getNextRandom();
                    vector.set(j, rand);
                    isTaken.add(rand);
                }
```

```
            }
        }
        return vector;
    }

    private void doHarmony() {
        initHarmonyMemory();
        refreshFitness();
        for(int p=0; p<practiceTime; p++) {
            List<Integer> newVector = new
ArrayList<>();
            for(int i = 0; i <
instances.size(); i++) {
                Integer newValue;
                if(getRandom() < HMCR) {
                    newValue =
doMemoryConsideration(i);
                    if(getRandom() < PAR) {
                        newValue =
doPitchAdjustment(i);
                    }
                } else {
                    newValue =
doRandomSelection(newVector);
                }
                isTaken.add(newValue);
                newVector.add(newValue);
            }
            newVector =
tieBreakCrossover(newVector);
            double lowestFitness =
fitnessValues[getLowestFitnessIndex()];
            double fitness =
delegate.getFitnessValue(newVector.toArray(new
Integer[0]));
            if(lowestFitness < fitness) {

harmonyMemory.set(getLowestFitnessIndex(),
newVector);

fitnessValues[getLowestFitnessIndex()] =
fitness;
                refreshFitnessRank();
            }
```

```
            resetNextRandom();
            System.out.println("Iteration No. "
+ p + " | Best Fitness = " +
fitnessValues[getFitnessIndexAtRank(1)]);
            System.out.println("Iteration No. "
+ p + " | Worst Fitness = " +
fitnessValues[getLowestFitnessIndex()]);
        }
    }

    private int doMemoryConsideration(int
column) {
        double rand = getRandom();
        int randRow =
getRowBasedOnProbability(rand);
        int randValue =
harmonyMemory.get(randRow).get(column);
        return randValue;
    }

    private int doRandomSelection(List<Integer>
cluster) {
//        return getNextRandom();
        return
getNextRandomBasedOnDistance(cluster);
    }

    private int doPitchAdjustment(int column) {
        int rand =
doMemoryConsideration(column);
        int firstNearest = getNextNearest(rand,
new ArrayList<Integer>());
        int secondNearest =
getNextNearest(rand,
Arrays.asList(firstNearest));
        if(getRandom() < PAR) {
            if(getRandom() < 0.5) {
                return firstNearest;
            } else {
                return secondNearest;
            }
        }
        return rand;
    }
```

```
    private int getRowBasedOnProbability(double
random) {
        int randRow = 0;
        while(random >
getUpperBoundProbabilityAt(randRow)) {
            randRow++;
        }
        return randRow;
    }

    private double getProbabilityAt(int index)
{
        double result;
        double total = 0;
        for (double fitness: fitnessValues) {
            total += fitness;
        }
        result = fitnessValues[index] / total;
        return result;
    }

    private double
getUpperBoundProbabilityAt(int index) {
        double bound = getProbabilityAt(0);
        for(int i = 1; i <= index; i++) {
            bound += getProbabilityAt(i);
        }
        return bound;
    }

    private int
getNextRandomBasedOnDistance(List<Integer>
cluster) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(cluster);
        if(diff.size() == instances.size()) {
            return getNearestFromDepot(diff);
        }
        double totalDistance = 0;
        HashMap<Integer, Double> distances =
new HashMap<>();
        for(int i : diff) {
```

```
            double distance =
clusterDelegate.getEuclidianDistanceBetween(i,
cluster.get(cluster.size() - 1));
            totalDistance += distance;
            distances.put(i, distance);
        }
        double upperBound = 0;
        double rand = getRandom();
        int next = diff.get(0);
        for(int i : diff) {
            double probability =
distances.get(i)/totalDistance;
            upperBound += probability;
            if(rand <= upperBound) {
                next = i;
                break;
            }
        }
        return next;
    }

    private int
getNearestFromDepot(List<Integer> cluster) {
        int nearestIndex = 0;
        double nearest =
clusterDelegate.getEuclidianDistanceFromDepot(c
luster.get(0));
        for(int i = 1; i < cluster.size();
i++){
            double distance =
clusterDelegate.getEuclidianDistanceFromDepot(c
luster.get(i));
            if(distance < nearest) {
                nearestIndex = i;
                nearest = distance;
            }
        }
        return cluster.get(nearestIndex);
    }

    private int getNextNearest(Integer from,
List<Integer> exception) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
```

```
        diff.removeAll(exception);
        if(diff.isEmpty()) {
            return exception.get(0);
        }
        int nearestIndex = 0;
        double nearest =
clusterDelegate.getEuclidianDistanceBetween(dif
f.get(0), from);
        for(int i = 1; i < diff.size(); i++){
            double distance =
clusterDelegate.getEuclidianDistanceBetween(dif
f.get(i), from);
            if(distance < nearest) {
                nearestIndex = i;
                nearest = distance;
            }
        }
        return diff.get(nearestIndex);
    }
}
```

d. *GAxHarmony*

```
package purehs;
import java.util.*;
import javafx.geometry.Point2D;

interface iClusterAble {
    double getEuclidianDistanceBetween(int
pointA, int pointB);
    double getEuclidianDistanceBetween(int
pointA, double centerX, double centerY);
    double getEuclidianDistanceFromDepot(int
pointA);
    double getEuclidianDistanceFromDepot(double
centerX, double centerY);
    int getX(int customer);
    int getY(int customer);
}

public class GAxHarmony {
    iFitnessAble fitnessDelegate;
    iClusterAble clusterDelegate;
    int practiceTime;
```

```
    int harmonyPracticeTime;
    int HMS;
    double HMCR;
    double PAR;
    double[] fitnessValues;
    double mutationProbability;
    double crossOverProbability;
    List<Integer> isTaken;
    HashMap<Integer, Integer> demands;
    List<Integer> instances;
    int[] fitnessValueRank;
    int maxDemand;
    int populationCount;
    List< List <Point2D> > centerPopulation;

    public GAxHarmony(int[] demands, int
maxDemand, int GAPracticeTime, int
populationCount, double mutationRate, double
crossOverRate, int harmonyPracticeTime, int
HMS, double HMCR, double PAR, iFitnessAble
fitnessDelegate, iClusterAble clusterDelegate)
{
        this.practiceTime = GAPracticeTime;
        this.harmonyPracticeTime =
harmonyPracticeTime;
        this.maxDemand = maxDemand;
        this.mutationProbability =
mutationRate;
        this.crossOverProbability =
crossOverRate;
        this.populationCount = populationCount;
        this.HMS = HMS;
        this.HMCR = HMCR;
        this.PAR = PAR;
        this.fitnessDelegate = fitnessDelegate;
        this.clusterDelegate = clusterDelegate;
        this.demands = new HashMap<>();
        this.instances = new ArrayList<>();
        this.centerPopulation = new
ArrayList<>();
        filterDemands(demands);
        fitnessValues = new
double[populationCount];
        isTaken = new ArrayList<>();
```

```
        fitnessValueRank = new
int[populationCount];
        initClusterPopulation();
        refreshFitness();
    }

    public Integer[] getBestFitness() {
        long startTime =
System.currentTimeMillis();
        doHybridGA();
        int bestIndex =
getFitnessIndexAtRank(1);
         List< List<Integer>> bestRoute =
generateClustersFromCenters(centerPopulation.ge
t(bestIndex));
        List< List<Integer>> harmonyClusters =
doHybridHarmony(bestRoute);
        List<Integer> harmonyVector =
makeVector(harmonyClusters);
        System.out.println("Best distance
before Harmony = " + 1.0 /
fitnessDelegate.getFitnessValue(makeVector(best
Route).toArray(new Integer[0])));
        double fitness =
fitnessDelegate.getFitnessValue(harmonyVector.t
oArray(new Integer[0]));
        System.out.println("Best distance after
Harmony = " + 1.0 / fitness);
        System.out.println("Execution time (s):
" + (System.currentTimeMillis() - startTime) /
1000.0);
        return harmonyVector.toArray(new
Integer[0]);
    }

    private void doHybridGA() {
        for(int i = 0; i<practiceTime; i++) {
            double rand1 = getRandom();
            int randRow1 =
getRowBasedOnProbability(rand1);
            int randRow2;
            do {
                double rand2 = getRandom();
```

```
                randRow2 =
getRowBasedOnProbability(rand2);
            } while (randRow2 == randRow1);
            List<Point2D> newCenters =
doCrossover(centerPopulation.get(randRow1),
centerPopulation.get(randRow2));
            newCenters =
doMutation(newCenters);
            List< List<Integer>> newClusters =
generateClustersFromCenters(newCenters);
            if(newClusters.size() !=
newCenters.size()) {
                i--;
                continue;
            }
            double lowestFitness =
fitnessValues[getLowestFitnessIndex()];
            double fitness =
getClusterFitness(newCenters);
            if(lowestFitness < fitness) {

centerPopulation.set(getLowestFitnessIndex(),
newCenters);

fitnessValues[getLowestFitnessIndex()] =
fitness;
                refreshFitnessRank();
            }
            System.out.println("#" + i + " |
GA-Best Fitness  = " +
fitnessValues[getFitnessIndexAtRank(1)]);
            System.out.println("#" + i + " |
GA-Worst Fitness = " +
fitnessValues[getLowestFitnessIndex()]);
        }
    }

    private void initClusterPopulation() {
        for (int i=0; i<populationCount; i++) {
            List<Integer> centers =
initClusterCenter();
            List<Point2D> centerPoints = new
ArrayList<>();
            for(int center: centers) {
```

```
                centerPoints.add(new
Point2D((double) clusterDelegate.getX(center),
(double) clusterDelegate.getY(center)));
            }
            List< List<Integer>> clusters =
generateClustersFromCenters(centerPoints);
            if(clusters.size() !=
centers.size()) {
                i--;
                continue;
            }
            centerPopulation.add(centerPoints);
        }
    }

    private List<Integer> initClusterCenter() {
        double totalDemand = 0;
        List<Integer> centers = new
ArrayList<>();
        for(int customer: instances) {
            totalDemand +=
demands.get(customer);
        }
        int k =
(int)(Math.ceil(totalDemand/(double)
maxDemand));
        for(int i=0; i<k; i++) {

centers.add(getNextRandomCustomerFor(centers));
        }
        return centers;
    }

    private List< List<Integer>>
doHybridHarmony(List< List<Integer>> clusters)
{
        List< List<Integer>> harmonyCluster =
new ArrayList<>();
        for(int i=0; i<clusters.size(); i++) {
            if(clusters.get(i).isEmpty()) {
                clusters.remove(i);
            }
        }
        for(int i =0; i<clusters.size(); i++) {
```

```
            System.out.println("Harmony Search:
Optimizing cluster " + ((double)
i/clusters.size() * 100) + "%");
            List<Integer> cluster =
clusters.get(i);
            int index = i;
            HarmonySearchCluster hsc = new
HarmonySearchCluster(cluster,
harmonyPracticeTime, HMS, HMCR, PAR,
harmonyVector -> {
                List<Integer> newVector = new
ArrayList<>();
                for(int j=0; j<clusters.size();
j++) {
                    if(index == j) {

newVector.addAll(Arrays.asList(harmonyVector));
                    } else {

newVector.addAll(clusters.get(j));
                    }
                }
                return
fitnessDelegate.getFitnessValue(newVector.toArr
ay(new Integer[0]));
            }, clusterDelegate);
            List<Integer> newCluster =
hsc.getBestFitness();
            harmonyCluster.add(newCluster);
        }
        System.out.println("Harmony Search:
Optimizing cluster " + "100.0%");
        return harmonyCluster;
    }

    private void refreshFitness() {
        for(int i = 0; i < populationCount;
i++) {
            fitnessValues[i] =
getClusterFitness(centerPopulation.get(i));
        }
        refreshFitnessRank();
    }
```

```
    private void refreshFitnessRank() {
        for(int i = 0; i < populationCount;
i++) {
            int rank = 1;
            for(int j = 0; j < populationCount;
j++) {
                if(fitnessValues[i] <
fitnessValues[j]) {
                    rank++;
                } else if(fitnessValues[i] ==
fitnessValues[j] && i > j) {
                    rank++;
                }
            }
            fitnessValueRank[i] = rank;
        }
    }

    private int getLowestFitnessIndex() {
        return
getFitnessIndexAtRank(populationCount);
    }

    private int getFitnessIndexAtRank(int rank)
{
        int index = populationCount - 1;
        for(int i = 0; i < populationCount;
i++) {
            if(fitnessValueRank[i] == rank) {
                index = i;
                break;
            }
        }
        return index;
    }

    private int getRowBasedOnProbability(double
random) {
        int randRow = 0;

while(getUpperBoundProbabilityAt(randRow) <=
random) {
            randRow++;
        }
```

```
        return randRow;
    }

    private double getProbabilityAt(int index)
{
        double result;
        double total = 0;
        for (double fitness: fitnessValues) {
            total += fitness;
        }
        result = fitnessValues[index] / total;
        return result;
    }

    private double
getUpperBoundProbabilityAt(int index) {
        double bound = getProbabilityAt(0);
        for(int i = 1; i <= index; i++) {
            bound += getProbabilityAt(i);
        }
        return bound;
    }

    private void filterDemands(int[]
rawDemands) {
        for(int i=0; i < rawDemands.length;
i++) {
            if(rawDemands[i] > 0) {
                demands.put(i, rawDemands[i]);
                instances.add(i);
            }
        }
    }

    private List<Integer> makeVector(List< List
<Integer>> clusters) {
        List<Integer> vector = new
ArrayList<>();
        for(int i=0; i<clusters.size(); i++) {
            vector.addAll(clusters.get(i));
        }
        return vector;
    }
```

```
    private int
getNextRandomCustomerFor(List<Integer>
currentRoute) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(currentRoute);
        int rand = (new
Random()).nextInt(diff.size());
        int value = diff.get(rand);
        return value;
    }

    private double getRandom() {
        return (((new
Random()).nextInt(10000000)) / 10000000.0);
    }

    private List < List<Integer>>
generateClustersFromCenters(List<Point2D>
centers) {
        List<Point2D> centerTemp = new
ArrayList<>();
        List < List<Integer>> clusters = new
ArrayList<>();
        List<Integer> clusterSpace = new
ArrayList<>();
        List<Integer> sortedCustomers =
sortCustomerFromDistance(instances);
        centerTemp.addAll(centers);
        for(Point2D center: centers) {
            clusterSpace.add(maxDemand);
            clusters.add(new ArrayList<>());
        }
        for(int customer: sortedCustomers) {
            int nearestCluster = -1;
            double nearestDistance =
clusterDelegate.getEuclidianDistanceBetween(cus
tomer, centerTemp.get(0).getX(),
centerTemp.get(0).getY());
            for(int i=0; i<centers.size(); i++)
{
                double distance =
clusterDelegate.getEuclidianDistanceBetween(cus
```

```
tomer, centerTemp.get(i).getX(),
centerTemp.get(i).getY());
                if(distance <= nearestDistance
&& clusterSpace.get(i) >=
demands.get(customer)) {
                    nearestCluster = i;
                    nearestDistance = distance;
                }
            }
            if(nearestCluster == -1) {
                List<Integer> cluster = new
ArrayList<>();
                centerTemp.add(new
Point2D(clusterDelegate.getX(customer),
clusterDelegate.getY(customer)));
                cluster.add(customer);
                clusters.add(cluster);
                clusterSpace.add(maxDemand -
demands.get(customer));
            } else {
                List<Integer> cluster =
clusters.get(nearestCluster);
                cluster.add(customer);

clusterSpace.set(nearestCluster,
clusterSpace.get(nearestCluster) -
demands.get(customer));
            }
        }
        return clusters;
    }

    private List<Integer>
sortCustomerFromDistance(List<Integer>
customers) {
        List<Integer> sorted = new
ArrayList<>();
        List<Double> distances = new
ArrayList<>();
        for(int c: customers) {
            double distance =
clusterDelegate.getEuclidianDistanceFromDepot(c
);
            sorted.add(c);
```

```
            distances.add(distance);
        }
        for(int i=0; i<customers.size(); i++) {
            int rank = 0;
            for(int j=0; j<customers.size();
j++) {
                if(distances.get(i) <
distances.get(j)) {
                    rank++;
                } else
if(Objects.equals(distances.get(i),
distances.get(j)) && i > j) {
                    rank++;
                }
            }
            sorted.set(rank, customers.get(i));
        }
        return sorted;
    }

    private List<Point2D>
doCrossover(List<Point2D> chromosomeA,
List<Point2D> chromosomeB) {
        List<Point2D> childVector = new
ArrayList<>();
        for(int i = 0; i < chromosomeA.size();
i++) {
            if(getRandom() <
crossOverProbability) {

childVector.add(chromosomeA.get(i));
            } else {

childVector.add(chromosomeB.get(i));
            }
        }
        return childVector;
    }

    private List<Point2D>
doMutation(List<Point2D> centers) {
        List<Point2D> newCenters = new
ArrayList<>();
        newCenters.addAll(centers);
```

```
        for(int i = 0; i < centers.size(); i++)
{
            if(getRandom() <
mutationProbability) {
                if(getRandom() <= 0.5) {
                    int rand = (new
Random()).nextInt(instances.size());
                    Point2D randValue = new
Point2D(clusterDelegate.getX(rand),
clusterDelegate.getY(rand));
                    newCenters.set(i,
randValue);
                } else {
                    int rand = (new
Random()).nextInt(centers.size());
                    Point2D randValue =
newCenters.get(rand);
                    newCenters.set(rand,
newCenters.get(i));
                    newCenters.set(i,
randValue);
                }
            }
        }
        return  newCenters;
    }

      private double
getClusterFitness(List<Point2D> centers) {
        double total = 0;
        List< List<Integer>> clusters =
generateClustersFromCenters(centers);
        for(int i=0; i<clusters.size(); i++) {
            List<Integer> cluster =
clusters.get(i);
            if(i < centers.size()) {
                Point2D center =
centers.get(i);
                double totalX = 0;
                double totalY = 0;
                for(int point: cluster) {
                    total +=
clusterDelegate.getEuclidianDistanceBetween(poi
nt, center.getX(), center.getY());
```

```
                    totalX +=
clusterDelegate.getX(point);
                    totalY +=
clusterDelegate.getY(point);
                }
                total +=
clusterDelegate.getEuclidianDistanceFromDepot(c
enter.getX(), center.getY());
            } else {
                for(int point: cluster) {
                    total +=
clusterDelegate.getEuclidianDistanceBetween(poi
nt, centers.get(0).getX(),
centers.get(0).getY());
                }
                total +=
clusterDelegate.getEuclidianDistanceFromDepot(c
enters.get(0).getX(), centers.get(0).getY());
            }
        }
        return 1/total;
    }
}
```

e. *HarmonySearchCluster*

```
package purehs;
import java.util.*;

public class HarmonySearchCluster {
    iFitnessAble delegate;
    iClusterAble clusterDelegate;
    int practiceTime;
    int HMS;
    double HMCR;
    double PAR;
    double[] fitnessValues;
    List<Integer> isTaken;
    List<Integer> instances;
    List< List<Integer> > harmonyMemory;
    int[] fitnessValueRank;

    public HarmonySearchCluster(List<Integer>
instances, int practiceTime, int HMS, double
```

```
HMCR, double PAR, iFitnessAble delegate,
iClusterAble clusterDelegate) {
        this.practiceTime = practiceTime;
        this.HMS = HMS;
        this.HMCR = HMCR;
        this.PAR = PAR;
        this.delegate = delegate;
        this.instances = instances;
        harmonyMemory = new ArrayList<>();
        fitnessValues = new double[HMS];
        isTaken = new ArrayList<>();
        fitnessValueRank = new int[HMS];
        this.clusterDelegate = clusterDelegate;
    }

    private void initHarmonyMemory() {
        harmonyMemory.add(instances);
        for(int i = 1; i < HMS; i++) {
            List<Integer> harmony = new
ArrayList<>();
            for (int j = 0; j <
instances.size(); j++) {
                harmony.add(instances.get(j));
            }
            Collections.shuffle(harmony);
            harmonyMemory.add(harmony);
        }
    }

    List<Integer> getBestFitness() {
        doHarmony();
        refreshFitness();
        int bestIndex =
getFitnessIndexAtRank(1);
        return harmonyMemory.get(bestIndex);
    }

    private void refreshFitness() {
        for(int i = 0; i < HMS; i++) {
            fitnessValues[i] =
delegate.getFitnessValue(harmonyMemory.get(i).t
oArray(new Integer[0]));
        }
        refreshFitnessRank();
```

```
    }

    private void refreshFitnessRank() {
        for(int i = 0; i < HMS; i++) {
            int rank = 1;
            for(int j = 0; j < HMS; j++) {
                if(fitnessValues[i] <
fitnessValues[j]) {
                    rank++;
                } else if(fitnessValues[i] ==
fitnessValues[j] && i > j) {
                    rank++;
                }
            }
            fitnessValueRank[i] = rank;
        }
    }

    private int getLowestFitnessIndex() {
        return getFitnessIndexAtRank(HMS);
    }

    private int getFitnessIndexAtRank(int rank)
{
        int index = HMS - 1;
        for(int i = 0; i < HMS; i++) {
            if(fitnessValueRank[i] == rank) {
                index = i;
                break;
            }
        }
        return index;
    }

    private double getRandom() {
        return (((new
Random()).nextInt(10000000)) / 10000000.0);
    }

    private int getNextRandom() {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(isTaken);
```

```
        Integer rand = (new
Random()).nextInt(diff.size());
        int value = diff.get(rand);
        return value;
    }

    private void resetNextRandom() {
        isTaken.clear();
    }

    private List<Integer>
tieBreakCrossover(List<Integer> vector) {
        for(int i = 0; i < vector.size() - 1;
i++) {
            for(int j = i+1; j < vector.size();
j++ ) {

if(vector.get(i).equals(vector.get(j))) {
                    Integer rand =
getNextRandom();
                    vector.set(j, rand);
                    isTaken.add(rand);
                }
            }
        }
        return vector;
    }

    private void doHarmony() {
        initHarmonyMemory();
        refreshFitness();
        while(practiceTime-- > 0) {
            List<Integer> newVector = new
ArrayList<>();
            for(int i = 0; i <
instances.size(); i++) {
                Integer newValue;
                if(getRandom() < HMCR) {
                    newValue =
doMemoryConsideration(i);
                    if(getRandom() < PAR) {
                        newValue =
doPitchAdjustment(i);
                    }
```

```
                } else {
                    newValue =
doRandomSelection(newVector);
                }
                isTaken.add(newValue);
                newVector.add(newValue);
            }
            newVector =
tieBreakCrossover(newVector);
            double lowestFitness =
fitnessValues[getLowestFitnessIndex()];
            double fitness =
delegate.getFitnessValue(newVector.toArray(new
Integer[0]));
            if(lowestFitness < fitness) {

harmonyMemory.set(getLowestFitnessIndex(),
newVector);

fitnessValues[getLowestFitnessIndex()] =
fitness;
                refreshFitnessRank();
            }
            resetNextRandom();
        }
    }

    private int doMemoryConsideration(int
column) {
        double rand = getRandom();
        int randRow =
getRowBasedOnProbability(rand);
        int randValue =
harmonyMemory.get(randRow).get(column);
        return randValue;
    }

    private int doRandomSelection(List<Integer>
cluster) {
//        return getNextRandom();
        return
getNextRandomBasedOnDistance(cluster);
    }
```

```
    private int doPitchAdjustment(int column) {
        int rand =
doMemoryConsideration(column);
        int firstNearest = getNextNearest(rand,
new ArrayList<Integer>());
        int secondNearest =
getNextNearest(rand,
Arrays.asList(firstNearest));
        if(getRandom() < PAR) {
            if(getRandom() < 0.5) {
                return firstNearest;
            } else {
                return secondNearest;
            }
        }
        return rand;
    }

    private int getRowBasedOnProbability(double
random) {
        int randRow = 0;
        while(random >
getUpperBoundProbabilityAt(randRow)) {
            randRow++;
        }
        return randRow;
    }

    private double getProbabilityAt(int index)
{
        double result;
        double total = 0;
        for (double fitness: fitnessValues) {
            total += fitness;
        }
        result = fitnessValues[index] / total;
        return result;
    }

    private double
getUpperBoundProbabilityAt(int index) {
        double bound = getProbabilityAt(0);
        for(int i = 1; i <= index; i++) {
            bound += getProbabilityAt(i);
```

```
        }
        return bound;
    }

    private int
getNextRandomBasedOnDistance(List<Integer>
cluster) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(cluster);
        if(diff.size() == instances.size()) {
            return getNearestFromDepot(diff);
        }
        double totalDistance = 0;
        HashMap<Integer, Double> distances =
new HashMap<>();
        for(int i : diff) {
            double distance =
clusterDelegate.getEuclidianDistanceBetween(i,
cluster.get(cluster.size() - 1));
            totalDistance += distance;
            distances.put(i, distance);
        }
        double upperBound = 0;
        double rand = getRandom();
        int next = diff.get(0);
        for(int i : diff) {
            double probability =
distances.get(i)/totalDistance;
            upperBound += probability;
            if(rand <= upperBound) {
                next = i;
                break;
            }
        }
        return next;
    }

    private int
getNearestFromDepot(List<Integer> cluster) {
        int nearestIndex = 0;
        double nearest =
clusterDelegate.getEuclidianDistanceFromDepot(c
luster.get(0));
```

```
        for(int i = 1; i < cluster.size();
i++){
            double distance =
clusterDelegate.getEuclidianDistanceFromDepot(c
luster.get(i));
            if(distance < nearest) {
                nearestIndex = i;
                nearest = distance;
            }
        }
        return cluster.get(nearestIndex);
    }

    private int getNextNearest(Integer from,
List<Integer> exception) {
        List<Integer> diff = new ArrayList<>();
        diff.addAll(instances);
        diff.removeAll(exception);
        if(diff.isEmpty()) {
            return exception.get(0);
        }
        int nearestIndex = 0;
        double nearest =
clusterDelegate.getEuclidianDistanceBetween(dif
f.get(0), from);
        for(int i = 1; i < diff.size(); i++){
            double distance =
clusterDelegate.getEuclidianDistanceBetween(dif
f.get(i), from);
            if(distance < nearest) {
                nearestIndex = i;
                nearest = distance;
            }
        }
        return diff.get(nearestIndex);
    }
}
```

f. *Shuffle*

```
package purehs;

/**
 *
```

```
 * @author Dinan
 */
import java.util.Random;

public class Shuffle<T> {

    private final Random rnd;

    public Shuffle() {
        rnd = new Random();
    }

    public void shuffle(T[] ar) {
        for (int i = ar.length - 1; i > 0; i--)
{
            int index = rnd.nextInt(i + 1);
            T a = ar[index];
            ar[index] = ar[i];
            ar[i] = a;
        }
    }
}
```

g. *iFitnessable*

```
package purehs;

public interface iFitnessAble {
    public double getFitnessValue(Integer[]
harmonyVector); // higher better
}
```

h. *Graph*

```
package purehs;

import java.awt.*;
import java.util.Random;
import javax.swing.*;
import purehs.simulation.Data;
@SuppressWarnings("serial")
public class Graph extends JPanel {
```

```
  public static int WIDTH;
  public static int HEIGHT;
  public static final int PADDING        = 20;
  public static final int LINE_THICKNESS = 3;
  JFrame w;
  Integer[]route;
  Data[] chosendata;

  public Graph(Integer[]route, Data[]
chosendata) {
    setWidthAndHeight();
    setupWindow();
    this.route = route;
    this.chosendata = chosendata;
  }

 public Graph() {
    setWidthAndHeight();
    setupWindow();
  }

    // Sets the WIDTH and HEIGHT constants.
  private void setWidthAndHeight() {
    WIDTH  = 1100;
    HEIGHT = 600;
  }

  // Sets up the graph window
  private void setupWindow() {
    w = new JFrame("Best Solution");
    w.setLocationByPlatform(true);
    w.add(this);
    w.setSize(new Dimension(WIDTH + (2 *
PADDING), HEIGHT + (2 * PADDING)));
    w.setVisible(true);
  }

  protected void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    g2d.setStroke(new
BasicStroke(LINE_THICKNESS));

    for(int i = 0 ; i < chosendata.length;
i++){
```

```
            g2d.setColor(Color.LIGHT_GRAY);

g2d.drawOval((chosendata[i].numX*15+320),
(chosendata[i].numY*-15+300),10,10);

g2d.fillOval((chosendata[i].numX*15+320),
(chosendata[i].numY*-15+300), 10,10);
    }
    Random random = new Random();
    Color ruteColor = Color.BLUE;
    for(int i = 1 ; i < route.length; i++){
        g2d.setColor(ruteColor);

g2d.drawLine((chosendata[route[i]].numX*15+325)
, (chosendata[route[i]].numY*-15+305),
                    (chosendata[route[i-
1]].numX*15+325), (chosendata[route[i-
1]].numY*-15+305));
        if(route[i]==0){
           ruteColor = new
Color(random.nextInt(255), random.nextInt(255),
random.nextInt(255));
        }
        if(route[i] == 0){
                g2d.setColor(Color.RED);

g2d.drawOval((chosendata[route[i]].numX*15+320)
, (chosendata[route[i]].numY*-15+300),10,10);

g2d.fillOval((chosendata[route[i]].numX*15+320)
, (chosendata[route[i]].numY*-15+300), 10,10);
        }
        else{
                g2d.setColor(Color.BLACK);

g2d.drawOval((chosendata[route[i]].numX*15+320)
, (chosendata[route[i]].numY*-15+300),10,10);

g2d.fillOval((chosendata[route[i]].numX*15+320)
, (chosendata[route[i]].numY*-15+300), 10,10);
       }
    }
    for(int i = 0 ; i < chosendata.length;
i++){
```

```
            g2d.setColor(Color.WHITE);
            g2d.setFont(new Font("arial",
Font.BOLD, 11));

g2d.drawString(Integer.toString(chosendata[i].d
emand), chosendata[i].numX*15+323,
chosendata[i].numY*-15+309);
            g2d.setColor(Color.DARK_GRAY);
            g2d.drawString(Integer.toString(i),
chosendata[i].numX*15+322, chosendata[i].numY*-
15+321);
   }
}
}
```

i. Data

```
package purehs;

public class Data {

    public int numX, numY, demand;

    public Data(int numX, int numY, int demand)
{
        this.numX = numX;
        this.numY = numY;
        this.demand = demand;
    }
}
```

j. *ExcelReader*

```
package purehs;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

```
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import org.apache.poi.ss.usermodel.*;
import
org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ExcelReader {

    String DEPOT = "Depot";
    String PELANGGAN = "Pelanggan";

    public Data[] transformToData(String
filePath) {
        List<Data> data = new ArrayList<>();
        try {
            FileInputStream excelFile = new
FileInputStream(new File(filePath));
            System.out.println(filePath);
            Workbook workbook = new
XSSFWorkbook(excelFile);
            Sheet datatypeSheet =
workbook.getSheetAt(0);
            Iterator<Row> iterator =
datatypeSheet.iterator();

            while (iterator.hasNext()) {
                Row currentRow =
iterator.next();
                Iterator<Cell> cellIterator =
currentRow.iterator();
                boolean isDepot = false;
                boolean isPelanggan = false;
                int x = 0;
                int y = 0;
                int demand = 0;
                int column = 0;
                while (cellIterator.hasNext())
{
                    Cell currentCell =
cellIterator.next();
                    if
(currentCell.getCellTypeEnum() ==
CellType.STRING) {
```

```
                            isDepot =
currentCell.getStringCellValue().equals(DEPOT);
                            isPelanggan =
currentCell.getStringCellValue().equals(PELANGG
AN);

System.out.print(currentCell.getStringCellValue
() + " ");
                        } else if
(currentCell.getCellTypeEnum() ==
CellType.NUMERIC) {
                            int value = (int)
currentCell.getNumericCellValue();
                            System.out.print(value
+ " ");
                            switch(column) {
                                case 1:
                                    x = value;
                                    break;
                                case 2:
                                    y = value;
                                    break;
                                case 3:
                                    demand = value;
                                    break;
                                default:
                                    break;
                            }
                        }
                        column++;
                    }
                    if(isDepot || isPelanggan){
                        Data newData = new Data(x,
y, demand);
                        data.add(newData);
                    }

                     System.out.println("");
                }
            } catch (FileNotFoundException e) {
                JOptionPane optionPane = new
JOptionPane();

optionPane.setMessage(e.getMessage());
```

```
optionPane.setMessageType(JOptionPane.WARNING_M
ESSAGE);
            JDialog dialog =
optionPane.createDialog(null, "Warning
Message");
            dialog.setVisible(true);
        } catch (IOException e) {
            JOptionPane optionPane = new
JOptionPane();

optionPane.setMessage(e.getMessage());

optionPane.setMessageType(JOptionPane.WARNING_M
ESSAGE);
            JDialog dialog =
optionPane.createDialog(null, "Warning
Message");
            dialog.setVisible(true);
        }
        return data.toArray(new Data[0]);
    }
}
```

BIODATA PENULIS

Penulis bernama lengkap **Dinan Fakhrana Ramadhani**, lahir di Bekasi, 7 Februari 1996. Anak kedua dari pasangan Edy Wiratna dan Itun Supriatun, serta memiliki kakak laki-laki bernama Luthfan Aufar Ramadhan. Penulis menempuh pendidikan di SD Putradarma Global School, SMP Al-Muslim, dan SMA Negeri 1 Tambun Selatan. Selanjutnya, penulis melanjutkan pendidikan tingginya di Departemen Matematika Institut Teknologi Sepuluh Nopember (ITS) Surabaya dengan mengambil bidang minat Ilmu Komputer. Selama mengikuti perkuliahan di ITS, penulis turut aktif dalam beberapa kegiatan kemahasiswaan sebagai Pemandu FMIPA ITS, staff Departemen Dalam Negeri Himatika ITS Periode 2014/2015, Koordinator Sie. Acara Olimpiade Matematika ITS (OMITS), dll. Selain aktif dalam beberapa kegiatan kemahasiswaan, penulis juga mengikuti Kerja Praktek Perum Percetakan Uang Republik Indonesia (Perum PERURI) pada tahun 2016 dan ditempatkan di Departemen IT Solution. Informasi lebih lanjut mengenai Tugas Akhir ini dapat ditujukan ke penulis melalui email: dinanfak@gmail.com.