PENERAPAN *HYBRID* ALGORITMA GENETIKA UNTUK PERMASALAHAN *VEHICLE ROUTING PROBLEM WITH TIME WINDOWS* (VRPTW)

SKRIPSI

KONSENTRASI KOMPUTASI CERDAS DAN VISUALISASI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun oleh:

Diah Anggraeni Pitaloka

NIM. 125150209111003

KEMENTRIAN PENDIDIKAN DAN KEBUDAYAAN

UNIVERSITAS BRAWIJAYA

PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER

PROGRAM STUDI INFORMATIKA / ILMU KOMPUTER

MALANG

2014

LEMBAR PERSETUJUAN

PENERAPAN *HYBRID* ALGORITMA GENETIKA UNTUK PERMASALAHAN *VEHICLE ROUTING PROBLEM WITH TIME WINDOWS* (VRPTW)

SKRIPSI

KONSENTRASI KOMPUTASI CERDAS DAN VISUALISASI

Untuk memenuhi sebagian persyaratan mencapai gelar Sarjana Komputer

Disusun oleh:

Diah Anggraeni Pitaloka

NIM. 125150209111003

Skripsi ini telah disetujui oleh dosen pembimbing

pada tanggal 25 Agustus 2014

Dosen Pembimbing I	Dosen Pembimbing II
Wayan Firdaus Mahmudy, S.Si., M.T, Ph.D.	**Ir. Sutrisno, MT.**
NIP. 19720919 199702 1 001	**NIP. 195703251987011001**

LEMBAR PENGESAHAN

PENERAPAN *HYBRID* ALGORITMA GENETIKA UNTUK PERMASALAHAN *VEHICLE ROUTING PROBLEM WITH TIME WINDOWS* (VRPTW)

SKRIPSI

KONSENTRASI KOMPUTASI CERDAS DAN VISUALISASI

Untuk memenuhi sebagian persyaratan untuk mencapai gelar Sarjana Komputer

Disusun oleh:

Diah Anggraeni Pitaloka

NIM. 125150209111003

Setelah dipertahankan di depan Majelis Penguji Pada tanggal 25 Agustus 2014 dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana dalam bidang Ilmu Komputer

Penguji I,

Suprapto, ST., MT.
NIP. 19710727 199603 1 001

Penguji II,

Wijaya Kurniawan, ST., MT
NIK. 820125 16 1 1 0418

Penguji III,

Ahmad Afif Supianto, Ssi., M.Kom
NIK. 820623 16 1 1 0425

Mengetahui,
Ketua Program Studi Informatika / Ilmu Komputer

Drs. Marji, M.T.
19670801 199203 1 001

KATA PENGANTAR

Puji syukur penulis panjatkan atas kehadirat Allah SWT, karena atas rahmat dan hidayahNya lah, penulis dapat menyelesaikan skripsi yang berjudul "Penerapan *Hybrid* Algoritma Genetika untuk Permasalahan *Vehicle Routing Problem with Time Windows* (VRPTW)" ini dengan baik dan tepat pada waktunya. Adapun maksud dan tujuan dari penulisan skripsi ini adalah sebagai syarat untuk menyelesaikan studi di program sarjana Program Teknologi Informasi dan Ilmu Komputer (PTIIK) Universitas Brawijaya.

Menyadari penulisan skripsi ini tidak lepas dari bantuan berbagai pihak, maka penulis mengucapkan terimakasih yang sebesar-besarnya kepada pihak-pihak yang membantu dalam pengerjaan maupun proses penyusunan laporan skripsi ini yaitu kepada,

1. Bapak Wayan Firdaus Mahmudy, S.Si., M.T, Ph.D. selaku dosen pembimbing 1 dan Bapak Ir. Sutrisno, MT. Selaku dosen pembimbing 2 yang telah bersedia meluangkan waktu memberikan arahan, bimbingan serta saran selama penyusunan skripsi ini.
2. Ibu, Bapak, Adik dan seluruh keluarga, Penulis mengucapkan terimakasih atas dukungan, doa serta kasih sayang yang sangat luar biasa diberikan kepada Penulis.
3. Bapak Drs. Marji, MT, Selaku Ketua Program Studi Informatika/Ilmu Komputer Program Teknologi Informasi dan Ilmu Komputer (PTIIK) Universitas Brawijaya.
4. Bapak Drs. Achmad Ridok, M.Kom, selaku dosen penasehat akademik penulis.
5. Bapak dan Ibu Dosen serta seluruh Staf Program Teknologi Informasi dan Ilmu Komputer (PTIIK) Universitas Brawijaya yang telah memberikan ilmunya serta arahan selama masa perkuliahan.
6. Teman-teman Program Teknologi Informasi dan Ilmu Komputer (PTIIK) dan Program Pendidikan Vokasi Universitas Brawijaya atas motivasi, bantuan

serta rekan dalam berbagi pengetahuan, suka dan duka selama menempuh masa studi.

7. Rekan-rekan MOST (Most Obviously Science and Technology) yang telah bersedia memberikan wadah kepada penulis untuk belajar, mendapatkan pengalaman, serta berorganisasi.
8. Sahabat-Sahabat kos jalan MT.Haryono 56 Malang yang selama ini menghibur dan memberikan dukungan kepada penulis.
9. Sahabat-sahabat baikku serta teman-teman yang selalu menemani dan memberikan kritik dan saran kepada penulis.
10. Teman-teman Aikido Shizen Dojo Malang, atas dukungan dan doanya.
11. Seluruh staf perpustakaan Universitas Brawijaya, perpustakaan fakultas MIPA dan ruang baca PTIIK Universitas Brawijaya yang membantu penulis memperoleh refrensi.
12. Dan berbagai pihak yang tidak dapat disebut satu persatu yang telah memberikan bantuan dan dukungannnya baik secara langsung maupun tidak langsung.

Menyadari bahwa penulisan skripsi ini masih jauh dari sempurna, penulis mengharapkan kritik dan saran yang membangun dari semua pihak. Saran dan kritikan yang bersifat membangun dapat disampaikan melalui email penulis diahanggraeni91@gmail.com. Akhir kata, penulis mengucapkan banyak terimakasih dan semoga skripsi ini dapat memberikan manfaat bagi kita semua.

Malang, Juni 2014

Penulis

ABSTRAK

Diah Anggraeni Pitaloka. 2014. Penerapan *Hybrid* Algoritma Genetika Untuk Permasalahan *Vehicle Routing Problem with Time Windows* (VRPTW). Skripsi Program Studi Informatika/ Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer. Pembimbing: Wayan Firdaus Mahmudy, S.Si., M.T, Ph.D. dan Ir. Sutrisno, MT.

Vehicle Routing Problem With Time Windows (VRPTW) adalah sebuah permasalahan kombinatorial yang digunakan dalam menentukan rute distribusi barang dari sebuah depot (pusat distribusi) kepada pelanggan yang tersebar di berbagai titik lokasi. VRPTW merupakan perluasan dari VRP dimana terdapat batasan kapasitas barang dan setiap pelanggan memiliki waktu interval pelayanan $[a_i, b_i]$ atau *time windows*. Pada penelitian ini, metode *hybrid* algoritma genetika dengan *nearest insertion heuristic* digunakan sebagai salah satu alternatif pencarian solusi. Pada implementasinya, pembangkitan 50% solusi awal dibentuk menggunakan metode *nearest insertion heuristic* dan 50% sisanya dibentuk secara acak. Pengujian dilakukan pada permasalahan Solomon. Hasil dari penelitian ini kemudian dibandingkan dengan *best known solution of Solomon* menggunakan 2 set data uji dari masing-masing tipe data yaitu C1, C2, R1, R2, RC1, dan RC2 dengan jumlah pelangggan 25, 50, dan 100. Hasilnya menunjukkan bahwa penerapan *hybrid* algoritma genetika dan *nearest insertion heuristic* memberikan hasil yang mendekati optimal atau mendekati *best known solomon* untuk tipe data C101, C105, C201, R101, RC101 dan RC102.

Kata kunci: VRPTW, VRP, Algoritma Genetika, *nearest insertion heuristic*.

ABSTRACT

Diah Anggraeni Pitaloka. 2014. *The Implementation of Hybrid Genetic Algorithm for Vehicle Routing Problem with Time Windows (VRPTW)*. Undergraduate Thesis. Informatics/ Computer Science Department, Information Technology and Computer Science Program (PTIIK). Advisor: Wayan Firdaus Mahmudy, S.Si., M.T, Ph.D. dan Ir. Sutrisno, MT.

Vehicle Routing Problem with Time Windows (VRPTW) is a combinatorial problem which is used to determine the route of a goods distribution from depots (distribution center) to customers scattered in various points of the site. VRPTW is an extension of the VRP with constraints the limitations on the capacity of goods and every customer have time interval of service $[a_i, b_i]$ *or time windows. In this study, the hybrid genetic algorithm with nearest insertion heuristic is used as one of the search alternative solutions. The implementation for this case, 50% initial solution was generated using nearest insertion heuristic method and 50% rest are generated by randomly. This completion is tested in a set of Solomons problem. The results of this study are compared with the best known solution of Solomon using 2 sets of test data from each data type such as* C1, C2, R1, R2, RC1, dan RC2 *with 25, 50 and 100 customers. The results show that the implementation of hybrid genetic algorithm and the nearest insertion heuristic gives solutions which is nearly optimal or approaching the best known of Solomon for data type* C101, C105, C201, R101, RC101 dan RC102.

***Keyword**: VRPTW, VRP, Genetic Algorithm, nearest insertion heuristic.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR *SOURCE CODE*

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Distribusi barang merupakan suatu proses penyaluran barang dan jasa dari produsen kepada konsumen dengan jumlah dan waktu tertentu [1]. Distribusi menjadi kegiatan operasional yang penting bagi perusahaan untuk menunjang proses pemasaran. Pada prosesnya, penentuan rute terbaik dengan mengoptimalkan sumber daya yang tersedia dapat menekan biaya operasional perusahaan. Menurut Toth dan Vigo [2], Amerika Utara dan Eropa mengakui bahwa penggunaan prosedur secara komputerisasi untuk merencanakan proses distribusi dapat mengurangi biaya transportasi umumnya 5% hingga 20%. Artinya, penentuan rute distribusi berdasarkan operasi pencarian dan teknik *Mathematical Programming* sangat efektif dalam manajemen sistem pelayanan distribusi barang. Penentuan rute distribusi ini merupakan permasalahan kombinatorial yang dikenal dengan istilah matematisnya yaitu *Vehicle Routing Problem* (VRP). VRP merupakan permasalahan bagaimana distribusi barang dari sebuah depot (pusat distribusi) kepada pelanggan yang tersebar di berbagai titik lokasi dengan batasan kendaraan, jarak antar pelanggan dan depot, waktu, serta kapasitas kendaraan.

Vehicle Routing Problem with Time Window (VRPTW) merupakan perluasan dari VRP dimana terdapat batasan kapasitas barang dan setiap pelanggan i memiliki waktu interval $[a_i, b_i]$ yang disebut dengan time windows [2] artinya, setiap kendaraan melayani permintaan pelanggan dalam interval waktu tertentu.

Pada penelitian sebelumnya, solusi permasalahan VRPTW diperoleh melalui beberapa metode yaitu *Ant Colony* untuk permasalahan *Solomon* [3], Metode *Nearest Insertion* Heuristik [4] studi kasus Koran harian pagi Tribun Jabar, dan Algoritma Genetika studi kasus PT.MIF [5].

Pada metode *nearest insertion* heuristik studi kasus koran harian pagi Tribun Jabar, penentuan rute distribusi dilakukan dengan menentukan titik untuk

disisipkan dengan mencari lokasi titik bebas yang tersebar dan memiliki jarak paling dekat dengan suatu titik pada rute. Metode ini relatif lebih cepat dalam optimasi rute. Dari hasil penelitian, penggunaan metode ini terbukti dapat menghemat ongkos transportasi hingga 5% dibandingkan dengan rute yang sudah ada sebelumnya. Akan tetapi, solusi ini belum bisa dikatakan optimal karena ruang pencarian yang sempit dan proses pencarian solusi hanya dilakukan satu kali. Untuk memastikan suatu solusi mendekati optimal, maka diperlukan data pembanding serta ruang pencarian yang lebih luas.

Pencarian solusi optimum dengan menggunakan algoritma probabilitik seperti algoritma genetika, *simulated annealing*, *ant colony* serta *tabu search* dilakukan dengan menjelajahi ruang yang lebih besar dari solusi yang ada dengan harapan menemukan solusi yang mendekati optimum. Hal ini berbeda dengan algoritma deterministik seperti *nearest insertion heuristik* yang pada setiap eksekusinya hanya menghasilkan satu solusi. Artinya, pencarian selesai dilakukan terlepas dari solusi tersebut sudah optimum atau tidak [6]. Sampai saat ini, metode metaheuristik menghasilkan solusi yang sangat baik tetapi juga memiliki 2 permasalahan utama yaitu waktu yang lebih lama dan tantangan dalam menemukan transformasi yang tepat untuk mengubah solusi yang ada [2].

Algoritma genetika merupakan salah satu metode metaheuristik yang sangat efektif digunakan untuk permasalahan optimasi karena pencarian dilakukan dengan menjelajahi ruang solusi yang lebih luas. Algoritma genetika diilhami dari proses evolusi biologi dimana dalam suatu populasi, individu melakukan reproduksi untuk menghasilkan keturunan. Saat proses seleksi, individu terbaik umumnya akan mempunyai peluang hidup lebih besar. Menurut Gen dan Cheng [7], pada implementasinya algoritma genetika memungkinkan optimasi dilakukan pada ruang pencarian yang sangat luas dan kompleks sehingga memungkinkan daerah solusi didapatkan melampaui daerah optimum lokal.

Pada penelitian ini, penggunaan kombinasi algoritma genetika dan *nearest insertion* heuristik digunakan untuk menyelesaikan permasalahan *Solomon*. Kombinasi dua metode ini akan menghasilkan Algoritma Genetika hybrid. Teknik hybrid ini terbukti sukses dalam permasalahan kompleks [8]. Permasalahan Solomon merupakan standar permasalahan internasional untuk studi kasus

VRPTW. Penggunaan metode *nearest insertion heuristik* digunakan sebagai pembangkitan solusi awal sebanyak n% individu. Kemudian dilanjutkan dengan penggunaan algoritma genetika pada pencarian yang lebih luas. Dengan mempertimbangkan penggunaan setiap metode, diharapkan penelitian ini dapat menghasilkan solusi yang melampaui *local optimum solution*.

1.2 Rumusan Masalah

Permasalahan dari skripsi ini adalah bagaimana pengaruh penerapan metode *hybrid* algoritma genetika pada permasalahan *vehicle routing problem with time windows* (VRPTW) dalam meminimalkan jarak, jumlah kendaraan, dan waktu tempuh.

1.3 Batasan Masalah

Adapun batasan masalah pada penelitiaan ini adalah :

1. Data uji untuk studi kasus permasalahan ini adalah permasalahan Solomon (*Solomon Benchmark Problem*), URL : http://w.cba.neu.edu/~msolomon/problems.htm .
2. Terdapat sejumlah kendaraan dengan kapasitas angkut kendaraan seragam.
3. Setiap rute berangkat dan berakhir di depot.
4. Setiap pelanggan hanya dikunjungi tepat satu kali oleh suatu kendaraan.
5. Jumlah permintaan setiap rute tidak melebihi kapasitas suatu kendaraan.
6. Setiap pelanggan i memiliki interval waktu pelayanan (*time windows*) $[a_i, b_i]$.
7. Asumsi jarak antar pelanggan i ke pelanggan j adalah sama dengan jarak pelanggan j ke pelanggan i.
8. Setiap pelanggan hanya dikunjungi tepat satu kali oleh suatu kendaraan.

9. Waktu perjalanan hanya ditentukan oleh jarak dengan asumsi kecepatan kendaraan tertentu sehingga kendala waktu seperti macet, mogok, dan permasalahan lainnya diabaikan.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai pada penelitian ini adalah menerapkan metode *hybrid* algoritma genetika untuk mendapatkan solusi yang mendekati optimum rute perjalanan pada permasalahan *vehicle routing problem with time windows* (VRPTW).

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah mendapatkan solusi rute perjalanan yang optimal dengan menerapkan metode *hybrid* algoritma genetika pada permasalahan VRPTW sehingga dapat meminimalkan jarak, jumlah kendaraan dan waktu tempuh. Harapannya, solusi yang dihasilkan dapat menghemat biaya distribusi dan transportasi.

1.6 Sistematika Penulisan

Penulisan tugas akhir ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN

   Bab ini berisi tentang latar belakang permasalahan, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA

   Bab ini berisi tentang kajian teori yang digunakan untuk menyelesaikan permasalahan VRPTW seperti VRPTW, algoritma genetika, dan *nearest insertion heuristic*.

3. BAB III METODE PENELITIAN DAN PERANCANGAN

   Bab ini berisi tentang metode-metode, langkah kerja, serta perancangan sistem yang digunakan dalam menyelesaikan

permasalahan VRPTW. Metode penelitian terdiri dari studi literatur untuk dasar teori, metode pengambilan data, analisis dan perancangan sistem, implementasi sistem, pengujian dan analisis, dan penarikan kesimpulan. Sedangkan perancangan terdiri dari rangkaian proses kerja serta desain sistem pada permasalahan VRPTW menggunakan metode *hybrid* algoritma genetika.

4. BAB IV IMPLEMENTASI

   Bab ini berisi tentang implementasi sistem untuk permasalahan VRPTW dengan menggunakan metode *hybrid* algortima genetika.

5. BAB V PENGUJIAN DAN ANALISIS

   Bab ini berisi tentang analisa kerja sistem dan analisa data hasil pengujian.

6. BAB VI PENUTUP

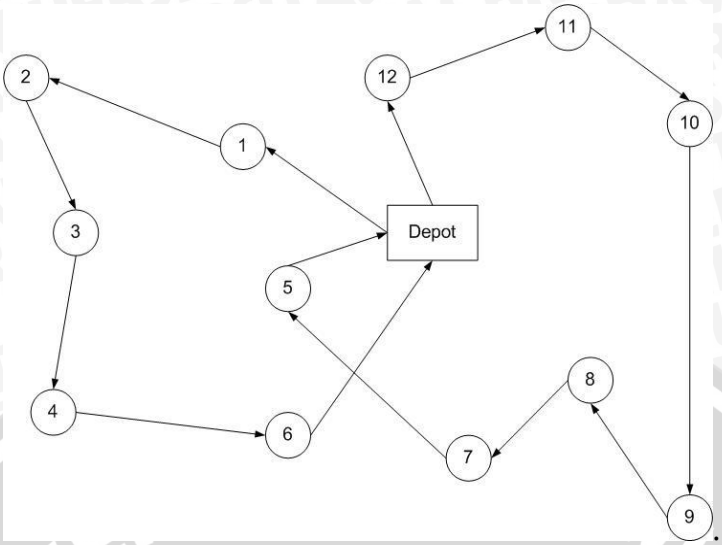
   Bab ini berisi tentang kesimpulan berdasarkan seluruh rangkaian pengujian dan analisis yang dilakukan di dalam penelitian serta saran untuk pengembangan sistem lebih lanjut.

BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Vehicle Routing Problem (VRP)

Menurut Toth dan Vigo [2], disebut *Vehicle Routing Problem* (VRP) karena digunakan untuk penentuan rute yang optimal oleh armada kendaraan melayani sejumlah pelanggan, dan ini merupakan suatu permasalahan optimasi kombinatorial yang penting untuk dipelajari. VRP pertama kali diperkenalkan oleh Dantzig dan Ramster pada tahun 1959. Mereka menggambarkan sebuah aplikasi *a real-world* mengenai pengiriman bensin untuk layanan stasiun dan untuk pertama kalinya mengenalkan formula matematika *programming* dan pendekatan algoritma. Beberapa tahun kemudian, tahun 1964, Clarke dan Wright mengusulkan metode greedy heuristik yang efektif untuk memperbaiki pendekatan Dantzig-Ramster. Berdasarkan dua penemuan tadi, ratusan model dan algoritma kemudian diusulkan untuk menemukan solusi yang optimal dan perkiraan solusi untuk versi yang berbeda dari VRP.

VRP merupakan permasalahan NP-hard dan generalisasi dari *Traveling Salesman Problem* (TSP). VRP membahas permasalahan mengenai distribusi barang dari depot kepada pelanggan dengan batasan-batasan tertentu seperti sejumlah pelanggan dapat dilayani dengan sejumlah kendaraan, adanya batas waktu pelayanan terhadap pelanggan, setiap pelanggan dikunjungi satu kali, serta rute yang dibentuk harus dimulai dan diakhiri di depot. Gambar 2.1 berikut ini adalah contoh penyelesaian VRP dalam bentuk *graph*.



Gambar 2.1 VRP sederhana dengan jumlah vehicle 2.
(Sumber : Tanujaya,W., dkk. 2011)

Dasar dari VRP adalah *Capacitated* VRP (CVRP). CVRP dapat digambarkan sebagai sebuah graph $G = (V, A)$ dimana $V = \{0, \dots, n\}$ adalah sekumpulan titik lokasi pelanggan dan A menyatakan jalan penghubung lokasi pelanggan dimana $i = 1, \dots, n$ adalah pelanggan awal, $j = 1, \dots, n$ adalah pelanggan tujuan. $i = 0$ dan $j = 0$ adalah depot yaitu tempat dimulai dan berakhirnya suatu rute kendaraan.c_{ij} adalah jarak antar konsumen, x_{ijk} menyatakan kendaraan k dari konsumen i ke konsumen j, Nilai k menunjukkan indeks kendaraan dengan Q_k adalah kapasitas kendaraan ke-k. Jumlah permintaan setiap pelanggan i bernilai q_i. Berikut ini, model VRP diformulasikan pada sebuah pemrograman linier integer:

$$\text{Min } z = \sum_{i=0}^{N} \sum_{j=0}^{N} (c_{ij} \sum_{k=1}^{K} x_{ijk}) \qquad (2.1)$$

Dengan kendala :

1. Setiap pelanggan hanya dapat dilayani oleh satu kendaraan saja.

$$\sum_{k=1}^{K} y_{ik} = 1 \qquad \forall i \in V \setminus \{0\} \qquad (2.2)$$

2. Setiap kendaraan berangkat dan berakhir didepot.

$$\sum_{k=1}^{K} y_{0k} = K \qquad (2.3)$$

3. Jika suatu kendaraan k mengunjungi suatu pelanggan, maka kendaraan k akan meninggalkan pelanggan tersebut menuju pelanggan lain:

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} \qquad \forall\, i \in V, k = 1, \dots, K, \qquad (2.4)$$

4. Setiap kendaraan k memiliki kapasitas barang sebesar Q_k dan permintaan pelanggan q_i tidak boleh melebihi kapasitas kendaraan:

$$\sum_{i \in V} q_i\, y_{ik} \leq Q_k \qquad \forall\ k = 1, \dots, K, \qquad (2.5)$$

5. Tiap konsumen dikunjungi oleh kendaraan yang sudah dijadwalkan untuk konsumen tersebut.

$$\sum_{i=0}^{N} x_{ijk} = y_{jk}; \sum_{j=0}^{N} x_{ijk} = y_{ik} \forall\, k = 1, \dots, K \qquad (2.6)$$

6. Variabel biner y_{ik} bernilai 1 jika pelanggan i dilayani oleh kendaraan k dan bernilai 0 jika tidak.

$$y_{ik} \in \{0,1\} \qquad \forall\, i \in V, k = 1, \dots, K, \qquad (2.7)$$

7. Variabel biner x_{ijk} bernilai 1 jika kendaraan k melayani j setelah melayani i, dan bernilai 0 jika tidak.

$$x_{ijk} \in \{0,1\} \qquad \forall\, i, j \in V, k = 1, \dots, K, \qquad (2.8)$$

Keterangan :

i = indeks untuk pelanggan awal, $i = 1, \dots, n$

j = indeks untuk pelanggan tujuan $j = 1, \dots, n$

k = indeks untuk kendaraan $k = 1, \dots, K$

c_{ij} = Jarak atau biaya perjalanan pelanggan i ke pelanggan j

q_i = Jumlah permintaan setiap pelanggan i

Q_k = Kapasitas kendaraan k

2.2 Vehicle Routing with Time Windows (VRPTW)

VRPTW merupakan perluasan dari VRP dimana layanan yang diberikan pelanggan berada pada interval waktu tertentu (*time windows*) dan setiap kendaraan harus tiba dilokasi pelanggan pada interval waktu tersebut. Pada kasus *soft time windows*, kendaraan yang datang melebihi *time windows* akan dilayani

tetapi dikenakan biaya tambahan atau penalti. Sementara untuk kasus *hard time windows*, pelayanan tidak akan dilakukan diluar interval waktu yang telah ditentukan [2].

Pada penelitian ini, digunakan beberapa asumsi Amini di dalam skripsi Astuti [9] dalam memodelkan VRPTW yaitu:

1. Terdapat satu depot dan sejumlah kendaraan di depot. Kendaraan-kendaraan tersebut digunakan untuk melayani pelanggan-pelanggan dan memiliki kapasitas yang sama.
2. Setiap pelanggan hanya dikunjungi satu kali.
3. Rute perjalanan setiap kendaraan harus dimulai dan kembali ke depot dengan tidak melanggar *time windows* depot.
4. Kecepatan kendaraan konstan, dan masalah kemacetan maupun gangguan lainnya dalam perjalanan (kecelakaan, ditilang, dan lain-lain) diabaikan.
5. Jumlah permintaan pelanggan-pelanggan pada suatu rute tidak boleh melebihi kapasitas kendaraan pada rute tersebut.
6. Setiap pelanggan harus dilayani dalam *time windows* yang telah ditentukan.

VRPTW didefinisikan sebagai sebuah graf $G = (V, A)$ dimana depot digambarkan dengan 2 node yaitu 0 dan $n + 1$. Semua kemungkinan rute kendaraan berangkat dari node 0 dan berakhir di node $n + 1$. *Time windows* disini juga berhubungan dengan node 0 dan $n + 1$. i.e., $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, dimana E dan L menunjukkan berturut-turut waktu berangkat dari depot dan waktu tiba atau kembali ke depot [2].

Misalkan terdapat n pelanggan $V = \{0, 1, \dots, n\}$ merupakan depot. Misalkan $i, j \in V$, jarak antar pelanggan i ke pelanggan j dinotasikan dengan d_{ij}, sedangkan waktu tempuh kendaraan dari pelanggan i ke pelanggan j dinotasikan dengan t_{ij}. Karena kecepatan kendaraan konstan, maka d_{ij} proporsional terhadap t_{ij}. *Time Window* pelanggan i dinotasikan dengan $[a_i, b_i]$dimana a_i merupakan waktu awal pelanggan i dan b_i merupakan waktu akhir pelanggan i, sedangkan *time window* pada depot dinotasikan dengan $[a_0, b_0]$ dan $a_0 = 0$. Setiap pelanggan i memiliki waktu pelayanan s_i dan permintaan-permintaan atas

barang/jasa sebanyak q_i. Kendaraan-kendaraan yang digunakan untuk melayani permintaan pelanggan-pelanggan memiliki kapasitas yang sama yaitu Q.

Misalkan K merupakan himpunan kendaraan yang tersedia di depot. Untuk setiap kendaraan k dan setiap jalur yang menghubungkan pelanggan i dan pelanggan j, dimana $i \neq j$ dan $i, j \neq 0$, didefinisikan variabel keputusan x_{ijk} sebagai berikut :

$$x_{ijk} \begin{cases} 1 \\ 0 \end{cases} \tag{2.9}$$

x_{ijk} bernilai 1 Jika Kendaraan k melayani pelanggan j setelah pelanggan i dan 0 jika lainnya. Selain itu didefinisikan pula y_{jk}, yaitu waktu pelanggan kendaraan k memulai pelayanan di pelanggan j. Karena diasumsikan $a_0 = 0$, maka $y_{0k} = 0$ untuk setiap kendaraan k.

Jika kendaraan k melayani pelanggan j setelah melayani pelanggan i, maka waktu mulai pelayanan pelanggan j bergantung pada waktu mulai pelayanan pelanggan i, lamanya waktu pelayanan pelanggan i, dan waktu tempuh dari pelanggan i ke pelanggan j. Jika kendaraan datang di pelanggan j di dalam *time window* pelanggan j, maka pelanggan j langsung dilayani, sedangkan jika kendaraan datang di pelanggan j sebelum waktu awal pelanggan j, maka kendaraan harus menunggu dan waktu mulai pelayanan akan sama dengan waktu awal pelanggan j. Jadi waktu mulai pelayanan pelanggan j, $y_{jp} = \max\{a_j, y_{ip} + s_i + t_{ij}\}$, dengan $y_{ip} + s_i + t_{ij}$ merupakan waktu kedatangan pelanggan j setelah melayani pelanggan i.

VRPTW dapat dijelaskan dengan formulasi matematis sebagai berikut :

$$\text{(VRPTW)}\ min \sum_{k \in K} \sum_{(i,j) \in V} c_{ij} x_{ijk} \tag{2.10}$$

Persamaan (2.10) merupakan total biaya keseluruhan dengan kendala berikut :

1. Setiap pelanggan dikunjungi tepat satu kali oleh suatu kendaraan.

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1 \qquad \forall i \in V - \{0\} \tag{2.11}$$

2. Setiap kendaraan memulai rute perjalanan dari depot.

$$\sum_{j \in V} x_{0jk} = 1 \qquad \forall k \in K \tag{2.12}$$

3. Setiap kendaraan yang mengunjungi pelanggan, maka akan meninggalkan pelanggan tersebut.

$$\sum_{i \in V} x_{ijk} - \sum_{j \in V} x_{jik} = 0 \qquad \forall\, k \in K, i, j \in N \qquad (2.13)$$

4. Setiap rute kendaraan berakhir di depot.

$$\sum_{i \in V} x_{i,0,k} = 1 \qquad \forall k \in K \qquad (2.14)$$

5. Jika kendaraan k mengunjungi pelanggan j setelah pelanggan i, maka kendaraan k akan melayani pelanggan j setelah setelah kendaraan melayani pelanggan i.

$$x_{ijk}\,(y_{ik} + s_i + t_{ij} - y_{jk}) \leq 0 \qquad \forall k \in K, (i,j) \in V \qquad (2.15)$$

6. Waktu dimulai pelayanan pelanggan i harus berada di dalam time window pelanggan i.

$$a_i \leq y_{ik} \leq b_i \qquad \forall k \in K, i \in V \qquad (2.16)$$

7. Total permintaan pelanggan dalam suatu rute tidak boleh melebihi kapasitas dari kendaraan yang melayani rute tersebut.

$$\sum_{i \in V} q_i \sum_{j \in V} x_{ijk} \leq Q \qquad \forall k \in K \qquad (2.17)$$

8. Variabel keputusan yang merupakan variabel biner.

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in V \qquad (2.18)$$

Keterangan :

y_{ik} = Waktu kendaraan k mulai melayani pelanggan i

K = Total kendaraan dengan kapasitas sama

x_{ijk} = Kendaraan k dari pelanggan i menuju pelanggan j

t_{ij} = waktu tempuh dari pelanggan i menuju pelanggan j

c_{ij} = Jarak tempuh perjalanan dari pelanggan i menuju pelanggan j

s_i= Waktu pelayanan pelanggan i

2.3 Algoritma Genetika

Algoritma genetik adalah model algoritmik yang dikembangkan untuk menyimulasikan sistem genetik. Algoritma ini diusulkan oleh Fraser, Bremermann dan Reed, dipopulerkan oleh Holland [10].

Algoritma genetik memodelkan evolusi genetik, dengan sifat individu dinyatakan dengan menggunakan genotype. Operator algoritma genetik terdiri atas seleksi dan rekombinasi. Seleksi digunakan untuk memodelkan ketahanan hidup dari yang paling *fit*, sedangkan rekombinasi digunakan untuk memodelkan reproduksi.

Algoritma genetik konvensional diusulkan oleh Holland, yang implementasinya sebagai berikut:

- Penyajiannya secara biner.
- Seleksi proporsional dengan tujuan untuk memilih induk untuk rekombinasi.
- *Crossover* satu-titik sebagai metode primer untuk menghasilkan *offspring*.
- Mutasi seragam sebagai operator yang kurang penting.

Menurut Widodo [10], Perlu dicatat bahwa pada algoritma genetik konvensional, mutasi bukan operator penting. Baru pada implementasi selanjutnya kemampuan eksploratif mutasi digunakan untuk meningkatkan kemampuan pencarian dari algoritma genetik. Beberapa variasi algoritma genetik telah dikembangkan yang berbeda dalam skema penyajian, operator seleksi, operator *crossover*, dan operator mutasi.

2.3.1 Penyajian Chromosome

Menurut Suyanto [11], Untuk dapat diproses menggunakan algoritma genetika, suatu permasalahan harus dikonversi dulu ke dalam bentuk individu yang diwakili oleh satu atau lebih kromosom dengan kode tertentu. Berbeda dengan teori genetika di dunia nyata yang merepresentasikan gen sebagai deretan *bases* A, C, T dan G, AG merepresentasikan gen (buatan), secara umum, sebagai bilangan real, decimal atau biner, yaitu :

- *Real-number encoding.* Pada skema ini, nilai gen berada dalam interval $[0, R]$, dimana R adalah bilangan real positif dan biasanya R =1.
- *Discrete decimal encoding*. Pada skema ini, setiap gen bisa berupa deretan bilangan bulat dalam interval [0,9].
- *Binary encoding*. Setiap gen bisa berupa deretan nilai 0 atau 1.

Pada penelitian ini, skema pengkodean yang digunakan dalam penyajian kromosom yaitu *permutation encoding*. Pada skema pengkodean ini, setiap individu berisi satu kromosom yang berisi gen-gen yang merepresentasikan nomor urut pelanggan. Contohnya saja terdapat 6 lokasi pelanggan, maka setiap individu berisi 6 gen yang nilainya berbeda (karena setiap pelanggan hanya boleh dikunjungi satu kali). Setiap nilai gen (*allele*) menyatakan nomor urut pelanggan dan posisi gen menyatakan urutan kunjungan. Berikut contoh individu menggunakan *permutation encoding*.

Individu1	3	2	5	1	4	6
Individu2	2	6	1	4	3	5

Gambar 2.2 Dua Contoh Individu Menggunakan *Permutation Encoding*.
(Sumber : Suyanto. 2011)

2.3.2 Populasi Awal

Pembangkitan populasi awal dilakukan dengan menetapkan nilai acak pada domain yang diinginkan untuk setiap *gene* dari setiap *chromosome*. Penetapan nilai acak menjamin populasi awal dapat disajikan seragam pada seluruh ruang pencarian [10].

Ukuran populasi awal menentukan kompleksitas komputasi dan kemampuan eksplorasi. Populasi yang besar akan menambah diversitas sehingga meningkatkan kemampuan eksplorasi. Namun, makin banyak individu akan menambah kompleksitas komputasi per generasi sehingga diperlukan sedikit generasi untuk melokasikan solusi yang dapat diterima. Populasi yang kecil, memberikan kompleksitas komputasi rendah, tetapi mewakili bagian yang kecil dari ruang pencarian sehingga diperlukan generasi yang banyak untuk konvergen.

2.3.3 Fungsi Fitness

Dalam model evolusi Darwin, individu dengan sifat terbaik memiliki peluang terbesar untuk bertahan hidup dan bereproduksi. Untuk menentukan kemampuan individu bertahan hidup, digunakan fungsi matematis yang disebut fungsi *fitness* untuk mengkuantifikasi seberapa baik solusi yang dinyatakan oleh *chromosome* [10].

Menurut Suyanto [11], individu dievaluasi berdasarkan suatu fungsi tertentu sebagai ukuran nilai *fitness*-nya. Pada masalah optimasi, jika solusi yang dicari adalah memaksimalkan sebuah fungsi h (dikenal sebagai masalah maksimasi), maka nilai *fitness* yang digunakan adalah nilai dari fungsi h tersebut, yakni $f = h$ (dimana f adalah nilai *fitness*). Tetapi jika masalahnya adalah meminimalkan fungsi h (masalah minimasi), maka fungsi h tidak bisa digunakan secara langsung. Hal ini desebabkan algoritma genetika menggunakan suatu aturan bahwa individu yang memiliki nilai *fitness* lebih tinggi akan memiliki kemampuan bertahan hidup lebih tinggi dari pada individu yang bernilai *fitness* rendah. Oleh karena itu, nilai *fitness* untuk masalah minimasi adalah $f = 1/h$, yang artinya semakin kecil nilai h semakin besar nilai f. Tetapi, fungsi ini akan bermasalah jika h bernilai 0, yang mengakibatkan f bisa bernilai tak hingga. Untuk mengatasi masalah tersebut, h perlu ditambah dengan sebuah bilangan yang dianggap sangat kecil, sehingga formula fungsi *fitness*-nya menjadi

$$f = \frac{1}{(h+a)'} \tag{2.19}$$

Di mana a adalah bilangan yang dianggap sangat kecil dan vervariasi sesuai dengan masalah yang akan diselesaikan.

2.3.4 Reproduksi

Menurut Widodo [10], reproduksi adalah proses untuk menghasilkan *offspring* dari induk terpilih dengan menerapkan operator *crossover* (persilangan) dan/atau mutasi. *Crossover* adalah proses untuk menciptakan satu atau lebih individu baru lewat kombinasi material genetik yang dipilih secara acak dari dua atau lebih induk (*parent*). Bila seleksi berfokus pada individu yang paling *fit* maka

tekanan seleksi dapat menyebabkan konvergensi prematur akibat diversitas berkurang pada populasi yang baru.

Mutasi adalah proses mengubah secara acak nilai *gene* pada *chromosome*. Tujuan utama mutasi adalah memunculkan material genetik baru ke dalam populasi sehingga memperbesar diversitas genetik. Mutasi harus diterapkan dengan cermat agar tidak merusak material genetik yang baik pada individu yang memiliki *fitness* tinggi. Karena alasan ini, mutasi biasanya diterapkan dengan probabilitas rendah. Dengan cara lain, probabilitas mutasi dapat dibuat proporsional dengan *fitness* individu, semakin individu kurang *fit* maka semakin sering dimutasi. Untuk mendukung eksplorasi pada generasi pertama, probabilitas mutasi dapat diinisialisasi pada nilai yang besar, yang kemudian dikurangi dari waktu ke waktu sehingga memungkinkan eksplorasi pada generasi akhir.

Reproduksi dapat diterapkan dengan penggantian, dalam kasus ini individu baru yang dihasilkan hanya mengganti individu induk bila fitness *offspring* yang baru lebih baik dari pada *fitness* induknya.

Operator *crossover* dan mutasi ini tergantung pada representasi dan paradigma komputasi evolusioner, apakah algoritma genetik, pemrograman genetik, atau pemrograman evolusioner.

Menurut Mahmudy [12], pada permasalahan ini, harus ditentukan tingkat *crossover* (*crossover rate* / p_c) dan tingkat mutasi (*mutation rate* / p_m). Nilai tersebut menyatakan rasio *offspring* yang dihasilkan pada setiap proses terhadap ukuran populasi sehingga akan dihasilkan *offspring* berturut-turut sebanyak $p_c \; x \; popSize$ untuk *crossover* dan $p_m \; x \; popSize$ untuk mutasi.

2.3.5 Crossover

Berdasar pada jumlah induk, operator *crossover* dapat dibedakan menjadi tiga kategori [10]:

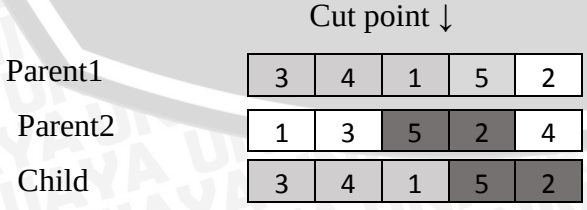
1. Aseksual: *offspring* dihasilkan dari satu induk.
2. Seksual : satu atau dua *offspring* dihasilkan dari dua induk.
3. Multi-rekombinasi : satu atau lebih *offspring* dihasilkan dari dua induk atau lebih.

Di dunia nyata, tidak mungkin ada dua individu yang sama. Hal ini disebabkan adanya evolusi yang didalamnya terdapat proses pindah silang (*crossover*). Pada proses pindah silang terjadi kombinasi pewarisan gen-gen dari induknya, gen-gen dari kedua induk dapat bercampur sehingga dihasilkan susunan kromosom yang baru. Dari proses tersebut akan dihasilkan variasi genetik [11].

Dengan suatu skema tertentu, dua individu dipilih sebagai orang tua. Setelah didapatkan dua individu orang tua, selanjutnya ditentukan titik pindah silang secara acak. Jika diasumsikan L adalah panjang kromosom, maka titik pindah silang berada antara 1 sampai $L - 1$. Kemudian beberapa bagian dari dua kromosom ditukar pada titik pindah silang yang dipilih. Titik pindah silang adalah titik terjadinya pertukaran gen antar dua individu orang tua. Pertukaran tersebut akan menghasilkan dua individu anak. Bagaimanapun, operasi pindah silang tidak selamanya berhasil. Peluang keberhasilan operasi pindah silang dinyatakan dengan probabilitas pindah silang atau p_c. Terdapat tiga skema pindah silang yang biasa digunakan, yaitu:

1. Pindah silang satu titik (*single-point crossover*)
2. Pindah silang banyak titik (*multi-point crossover*)
3. Pindah silang pola seragam (*uniform crossover*)

Menurut Mahmudy [12], metode *crossover* yang paling sederhana yang dapat digunakan pada permasalahan ini adalah dengan melakukan modifikasi *one-cut-point crossover*. Proses ini dilakukan dengan mengisi kromosom child dari potongan pertama parent1, kemudian selanjutnya diisi dengan gen pada Parent2 yang tidak mengandung gen pada potongan sebelumnya. Berikut adalah contoh implementasinya:



Gambar 2.3 Proses Crossover.
(Sumber : Mahmudy, Wayan Firdaus. 2013)

2.3.6 Mutasi

Menurut Kusumadewi [13], mutasi yang digunakan pada algoritma genetika sederhana dengan kromosom biner pada dasarnya akan mengubah secara acak nilai suatu bit pada posisi tertentu. Kemudian mengganti bit 1 dengan 0 atau mengganti bit 0 dengan 1. Pada mutasi ini sangat dimungkinkan munculkan kromosom baru yang semula belum muncul dalam populasi awal.

Misalkan :

$v_1' =$

1	0	1	0	0	1	1	0

Terkena mutasi pada gen ke-5, diperoleh :

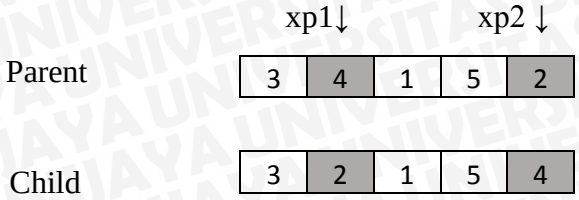
$v_2' =$

1	0	1	0	1	1	1	0

Gambar 2.4 Operator mutasi untuk penyajian biner.
(Sumber : Kusumadewi, Sri. 2003)

Pada mutasi ada satu parameter yang sangat penting yaitu peluang mutasi (p_m). Peluang mutasi menunjukkan prosentasi jumlah total gen pada populasi yang akan mengalami mutasi. Untuk melakukan mutasi, terlebih dahulu kita harus menghitung jumlah total gen pada populasi tersebut. Kemudian bangkitkan bilangan random yang akan menentukan posisi mana yang akan dimutasi (gen keberapa pada kromosom keberapa). Misalkan ukuran populasi (popsize = 100), setiap kromosom memiliki panjang 20 gen, maka total gen adalah 100 x 20 = 2000 gen. jika peluang mutasi ($p_m = 0{,}01$), berarti bahwa diharapkan ada (1/100) x 2000 = 20 gen akan mengalami mutasi.

Menurut Mahmudy [12], metode mutasi yang paling sederhana yang dapat digunakan pada permasalahan ini adalah *reciprocal exchange mutation*. Metode ini bekerja dengan memilih dua posisi (*exchange point* / XP) secara random kemudian menukarkan nilai pada posisi tersebut. Contoh penerapan *Reciprocal Exchange Mutation* adalah sebagai berikut:



Gambar 2.5 Reciprocal Exchange Mutation.
(Sumber : Mahmudy, Wayan Firdaus. 2013)

2.3.7 Seleksi

Menurut Widodo [10], Seleksi mempunyai relasi langsung dengan konsep Darwin tentang kebertahanan hidup (*survival*) dari individu yang paling *fit*. Tujuan operator seleksi adalah memperoleh solusi terbaik dalam memperoleh individu yang paling *fit* tersebut. Untuk memperolehnya diperlukan dua langkah dalam algoritma evolusi :

- Seleksi populasi baru

  Populasi baru sebagai kandidat solusi dipilih pada akhir setiap generasi sebagai populasi pada generasi selanjutnya. Populasi baru dapat dipilih dari *offspring* (turunan) saja atau dari induk dan *offspring*. Operator seleksi harus menjamin bahwa individu yang baik akan bertahan hidup pada generasi selanjutnya.
- Reproduksi

  Offspring diciptakan lewat penerapan operator *crossover* (persilangan) atau mutasi. Untuk *crossover*, individu yang lebih unggul harus mempunyai peluang yang lebih besar untuk berreproduksi untuk menjamin bahwa *offspring* berisi material genetik dari individu terbaik. Dalam kasus mutasi, mekanisme seleksi harus berfokus pada individu yang lemah. Harapan mutasi pada individu yang lemah akan menghasilkan sifat yang lebih baik sehingga menambah peluang bertahan hidup.

Pada penelitian ini, metode yang digunakan pada proses seleksi adalah *roulette wheel*. Menurut Mahmudy [12], pendekatan ini dilakukan dengan menghitung nilai probabilitas seleksi ($prob$) tiap individu berdasarkan nilai *fitnessnya*. Dari nilai $prob$ ini bisa dihitung probabilitas kumulatif ($probCum$) yang digunakan pada proses seleksi tiap individu.

Adapun langkah-langkah membentuk *roulette wheel* berdasarkan probabilitas kumulatif adalah:

- Misalkan *fitness*(P_k) adalah nilai *fitness* individu ke-k. Maka bisa dihitung total *fitness* sebagai berikut:

  $totalFitness = \sum_{k=1}^{popSize} fitness(P_k)$ (2.20)

- Hitung nilai probabilitas seleksi ($prob$) tiap individu:

  $prob_k = \frac{fitness(P_k)}{totalFitness}, \quad k = 1,2,\dots,popSize$ (2.21)

- Hitung nilai probabilitas kumulatif ($probCum$) tiap individu:

  $probCum_k = \sum_{j=1}^{k} prob_j, \quad k = 1,2,\dots,popSize$ (2.22)

Adapun contoh proses seleksi untuk masalah maksimasi dengan menggunakan *Roullette Wheel* untuk enam *chromosome* dengan *fitness* absolut masing-masing adalah sebagai berikut:

Chromosome	: 1	2	3	4	5	6
Fitness	: 6	4	15	7	2	10
Jumlah (*sum*) Fitness	: 6	10	25	32	34	44
Bilangan Acak ($1 \leq r \leq 44$) :			24			
Chromosome terseleksi			3			

Contoh penggunaan algoritma RWS adalah misalnya saja ada 5 individu (*chromosome*) $x_1, \dots, x_5$ dengan probabilitanya sebagai berikut :

$Prob(x_1) = 0{,}1$

$Prob(x_2) = 0{,}2$

$Prob(x_3) = 0{,}15$

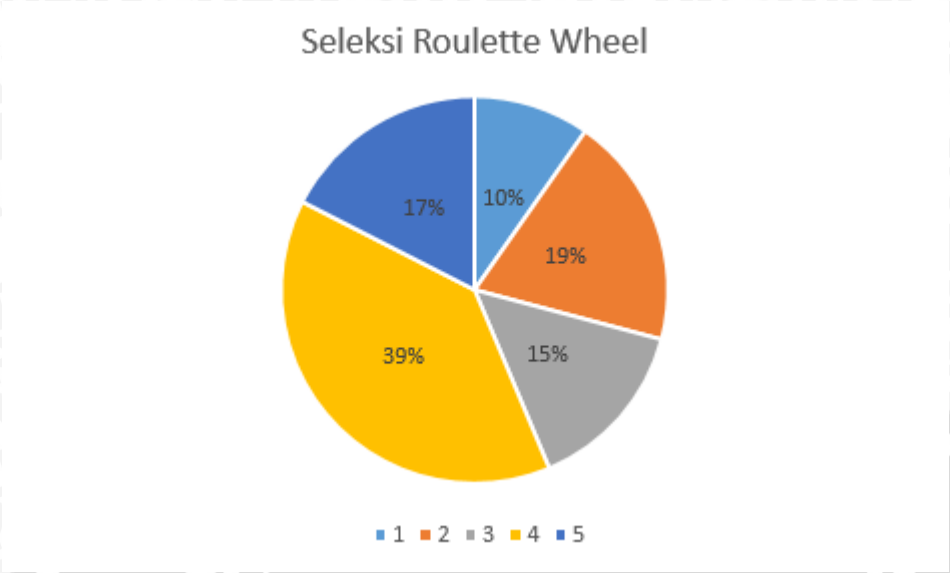
$Prob(x_4) = 0{,}4$

$Prob(x_5) = 0{,}18$

Maka ditentukan individu terseleksi dan berapa kali terseleksi untuk menghasilkan *offspring*. Didapatkan :

Bilangan Acak	= 0.9797	0.2714	0.2523	0.8757	0.7373
Index Chromosome	= 5	2	2	5	4

Disini terlihat bahwa *chromosome* x_2 dan x_5 terseleksi dua kali. Gambar berikut menunjukkan roulette wheel saat chromosome x_2 terseleksi:



Gambar 2.6 *Roulette Wheel Selection* sebagai peluang terpilihnya individu. (Sumber :Widodo, Sri Thomas. 2012)

2.3.8 Syarat Berhenti

Operator evolusioner diterapkan secara iterative pada algoritma evolusioner sampai syarat berhenti (*stopping* condition) dipenuhi. Menurut Widodo [10], syarat berhenti yang sederhana adalah membatasi jumlah generasi yang diperbolehkan, atau membatasi bilangan evaluasi fungsi fitness. Batas ini tidak boleh terlalu kecil, karena algoritma evalusioner tidak memiliki cukup waktu untuk eksplorasi ruang pencarian.

Di samping membatasi waktu eksekusi, kriteria konvergensi biasanya digunakan untuk mendeteksi apakah populasi telah konvergens. Konvergensi didefinisikan sebagai kejadian pada saat populasi menjadi stagnan atau dengan kata lain bila tidak ada perubahan *genotypic* didalam populasi. Kriteria konvergensi berikut dapat digunakan :

- Berhenti bila tidak ada perbaikan pada sejumlah generasi berikutnya.
- Berhenti bila tidak ada perubahan dalam populasi
- Berhenti bila solusi yang dapat diterima telah ditemukan.
- Berhenti bila lereng fungsi objektif mendekati nol.

2.4 Algoritma *Nearest Insertion Heuristic*

Pada *Nearest Insertion Heuristic*, pembentukan rute dilakukan dengan cara memilih pelanggan yang akan disisipkan kedalam suatu rute yang sudah ada. Proses penyisipan dilakukan hingga rute yang bersangkutan dinyatakan penuh, baik berdasarkan kapasitas kendaraan maupun jadwal waktu pelayanan di masing-masing pelanggan. Tujuannya adalah untuk membentuk satu atau beberapa rute pelayanan dengan total ongkos proporsional terhadap jarak dan waktu tempuh [4].

Langkah-langkah dalam memecahkan permasalahan dengan menggunakan algoritma *nearest insertion* dapat dilakukan dengan tahap berikut.

1. Buat matrik jarak dan waktu tempuh.
2. Tentukan seluruh node (tidak termasuk depot) yang belum masuk ke dalam rute sebagai node bebas. Pilih satu node bebas untuk dijadikan node awal dari rute yang akan dibentuk, nyatakan node tersebut sebagai node i. Pemilihan node awal dapat berdasarkan pada jarak node terhadap depot atau jadwal waktu pelayanan. Tetapkan rute awal sebagai $R = \{0, i, n+1\}$ dengan 0 dan $n+1$ adalah depot.
3. Tentukan node bebas yang dipertimbangkan untuk disisipkan dengan node u dimana $\mu \geq 0$. Tetapkan nilai parameter α_1 yaitu bobot yang diberikan terhadap total jarak yang terjadi akibat penyisipan node u dan parameter α_2 yaitu bobot yang diberikan terhadap perubahan waktu pelayanan akibat penyisipan node $u(\alpha_1 + \alpha_2 = 1)$. Tetapkan nilai parameter λ yaitu bobot yang diberikan bagi ongkos perjalanan dari depot ke node u jika node u tidak disisipkan kedalam rute ($\lambda \geq 0$).
4. Tentukan rute saat ini sebagai R = $\{0, i, \ldots\ldots, j\}$ dimana 0 dan j adalah depot. Untuk setiap node bebas u, hitung total tambahan jarak yang terjadi jika node u disisipkan dengan menggunakan formula : $Z_{11}(i,u,j) = d_{iu} + d_{ju} - \mu\, d_{ij} \geq 0$; dimana : d_{iu}, d_{ju} dan d_{ij} masing-masing adalah jarak antara node i dengan node u, node u dengan node j, dan node i dengan node j.
5. Hitung tambahan waktu untuk kendaraan tiba dan memulai pelayanan di node i jika node u disisipkan dengan menggunkan formula : $Z_{12}(i,u,j) =$

$t_{0u} + t_u + t_{ui} - t_{0i}$; dimana : t_{0u}, t_{ui}, t_{0i} masing-masing adalah waktu tempuh dari depot ke node u, dari node u ke node i, dan dari depot ke node i, sedangkan t_u adalah waktu pelayanan di node u.

6. Hitung besarnya ongkos penyisipan yang besarnya proporsional terhadap tambahan jarak dan tambahan waktu tempuh untuk tiba di node i jika node u disisipkan dengan menggunakan $Z_1(i,u,j) = \alpha_1 Z_{11}(i,u,j) + \alpha_2 Z_{12}(i,u,j)$; $\alpha_1 \geq 0$; $\alpha_2 \geq 0$; $\alpha_1 + \alpha_2 = 1$. Sisipkan node bebas u yang memiliki nilai $Z_1(i,u,j)$ minimum ke dalam rute diantara node i dan node j yang sudah ada.
7. Jika kapasitas kendaraan dan batas waktu pelayanan masih memungkinkan, maka lakukan penyisipan sebagai berikutnya dimana nilai $Z_2(i,u,j)$ maksimum, dimana ; $Z_2(i,u,j) = \lambda\, d_{0u} - Z_1(i,u,j)$; $\lambda \geq 0$; dimana : $Z_2(i,u,j)$ menyatakan selisih antara ongkos penyisipan yang terjadi jika node u ditempuh langsung dari depot dengan ongkos yang terjadi jika node u disisipkan kedalam rute. Sesuaikan dengan jumlah permintaan tiap node dari rute yang terbentuk dengan kapasitas angkut.
8. Jika masih terdapat node bebas maka ulangi dengan dengan dimulai dari langkah 3 hingga keseluruhan node masuk kedalam rute.

BAB III
METODE PENELITIAN DAN PERANCANGAN

3.1 Metode Penelitian

Metode penelitian menjelaskan mengenai langkah-langkah yang akan digunakan dalam pembuatan aplikasi optimasi pada permasalahan *vehicle routing problem with time windows* (VRPTW) dengan menerapkan *hybrid* algoritma genetika. Gambar 3.1 berikut ini menunjukkan langkah-langkah yang dilakukan dalam penelitian skripsi.

Gambar 3.1 *Flowchart* langkah-langkah penelitian.

3.1.1 Studi Literatur

Pada tahap ini, peneliti mengumpulkan studi literatur terkait dengan penelitian yang dilakukan. Studi literatur ini digunakan sebagai landasan teori untuk dipelajari dan dianalisis terkait dengan permasalahan yang ada guna menunjang penelitian. Adapun teori – teori yang terkait yaitu:

1. *Vehicle Routing Problem* (VRP)
2. *Vehicle Routing Problem with Time Windows* (VRPTW)
3. Algoritma genetika
4. *Nearest Insertion Heuristic*
5. Pemrograman C#
6. Database MySql

3.1.2 Pengambilan Data

Data yang digunakan dalam penelitian ini adalah data sekunder dimana data tidak diperoleh secara langsung dari objek penelitian. Data yang digunakan merupakan data yang sudah tersedia yaitu permasalahan Solomon sebagai standar permasalahan internasional studi kasus VRPTW. Data diperoleh di alamat URL : http://w.cba.neu.edu/~msolomon/problems.htm.

3.1.3 Analisis dan Perancangan Sistem

Pada tahap ini dilakukan analisis dan perancangan sistem yang bertujuan menganalisis permasalahan serta perancangan aplikasi optimasi pada permasalahan VRPTW dengan menggunakan *hybrid* algoritma genetika yang bertujuan meminimalkan jumlah kendaraan dan waktu tempuh.

Pada tahap analisis permasalahan, pemecahan kasus dimodelkan sedemikian rupa sehingga dapat dipahami dan selanjutnya akan digunakan untuk menyelesaikan studi kasus VRPTW. Selanjutnya, dilakukan analisis kebutuhan yang menunjang proses penelitian. Berikut ini adalah kebutuhan perangkat lunak dan perangkat keras serta data yang dibutuhkan dalam penelitian:

1. Kebutuhan Perangkat Keras, meliputi:
   - *Processor Intel® Core™2 Duo* 2,26 *Ghz*
   - *RAM* 4096 *MB*
   - *Harddisk* 300 GB
   - *Monitor* 14*".*
2. Kebutuhan Perangkat lunak, meliputi:
   - *Operating System Windows 7 64 bit*.
   - *Visual Studio 2010*
   - Bahasa Pemrograman C#
   - *Database* MySql
3. Data yang dibutuhkan adalah permasalahan VRPTW studi kasus Solomon yang meliputi
   - Lokasi pelanggan
   - Jumlah kendaraan
   - Kapasitas kendaraan
   - Jumlah permintaan pelanggan
   - Interval waktu buka dan waktu tutup pelanggan
   - Waktu pelayanan pelanggan

Tahap perancangan sistem merupakan alur kerja penyelesaian permasalahan VRPTW menggunakan metode *hybrid* algoritma genetika, penggunaan variable, penggunaan data uji kedalam algoritma dan formula matematika, desain aplikasi yang menampilkan hasil permasalahan dan solusi, serta desain uji coba dan evaluasi sistem sesuai dengan variable-variabel yang berpengaruh terhadap kinerja sistem.

3.1.4 Implementasi Sistem

Implementasi merupakan tahap pembuatan aplikasi sesuai dengan analisis dan perancangan sistem yang dilakukan pada tahap sebelumnya. Permasalahan VRPTW dengan menerapkan *hybrid* algoritma genetika diimplementasikan di

dalam suatu aplikasi sistem agar siap dioperasikan untuk selanjutnya diuji dan dianalisis terhadap permasalahan yang ada.

3.1.5 Pengujian dan Analisiss

Pada tahap ini, permasalahan VRPTW yang sudah diimplementasikan dalam bentuk program aplikasi akan diuji tingkat keakuratannya. Jika pada proses uji coba ditemukan kesalahan dan kekurangan pada sistem, maka dilakukan proses evaluasi dan perbaikan program agar sesuai dengan tujuan yang diharapkan. Pengujian disini meliputi:

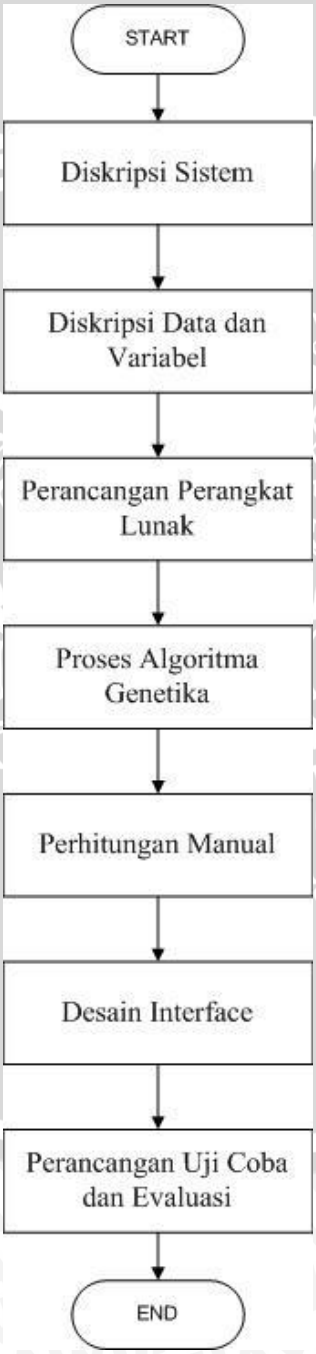
- Penerapan *hybrid* algoritma genetika untuk optimasi pada permasalahan VRPTW
- Probabilitas *crossover*
- Probabilitas mutasi
- Perbandingan kinerja antara penerapan *hybrid* algoritma genetika dengan *best known Solomon* of VRPTW.

3.1.6 Penarikan Kesimpulan

Penarikan kesimpulan merupakan tahap akhir dari seluruh rangkaian proses pembuatan aplikasi dimana dilakukan penilaian dan penarikan kesimpulan terhadap hasil yang didapatkan dari proses pengujian dan analisis. Kesimpulan tersebut kemudian dianalisis oleh pembaca sebagai saran untuk perbaikan serta pengembangan aplikasi selanjutnya.

3.2 Perancangan Sistem

Adapun tahap perancangan sistem pada permasalahan VRPTW terdiri dari diskripsi umum, diskripsi data dan variabel, perancangan perangkat lunak, proses algoritma genetika, perhitungan manual, desain interface, serta perancangan uji coba dan evaluasi. Gambar 3.2 merupakan tahap perancangan yang dilakukan pada proses penelitian.



Gambar 3.2 *Flowchart* tahap perancangan sistem.

3.2.1 Diskripsi Sistem

Pada penelitian ini, akan dibangun perangkat lunak dengan menerapkan *hybrid* algoritma genetika pada permasalahan VRPTW. VRPTW merupakan permasalahan optimasi rute kendaraan dan perluasan dari VRP dimana terdapat batasan kapasitas barang untuk setiap kendaraan dan setiap pelanggan i memiliki waktu interval $[a_i, b_i]$. Jadi, setiap pelanggan dilayani tepat satu kali dengan jumlah permintaan tertentu dengan interval waktu tertentu.

Penggunaan *hybrid* algoritma genetika pada permasalahan VRPTW bertujuan meminimalkan jumlah kendaraan, total waktu perjalanan dan jumlah *penalty*. Jumlah *penalty* adalah nilai hukuman yang didapatkan jika suatu kendaraan melayani pelanggan diluar waktu interval yang sudah ditentukan.

3.2.2 Diskripsi Data

Data yang digunakan sebagai objek pada penelitian ini adalah studi kasus Solomon yang merupakan standar permasalahan internasional untuk studi kasus VRP. Data didapat dari alamat URL: http://w.cba.neu.edu/~msolomon/problems.htm. Rincian data yang digunakan untuk penyelesaian permasalahan VRPTW adalah sebagai berikut:

- Lokasi pelanggan
- Jumlah kendaraan
- Kapasitas kendaraan
- Jumlah permintaan pelanggan
- Interval waktu buka dan waktu tutup pelanggan
- Waktu pelayanan
- Jumlah pelanggan
- *Time frame* kendaraan (Waktu kendaraan berangkat dan waktu akhir kendaraan melakukan pelayanan)

Dari kumpulan data tersebut, kemudian diolah dengan menerapkan *hybrid* algoritma genetika untuk memperoleh solusi yang optimal baik dari jumlah kendaraan, waktu pelayanan, dan total *penalty*. Solusi yang diperoleh pada permasalahan ini dipengaruhi oleh beberapa faktor yaitu:

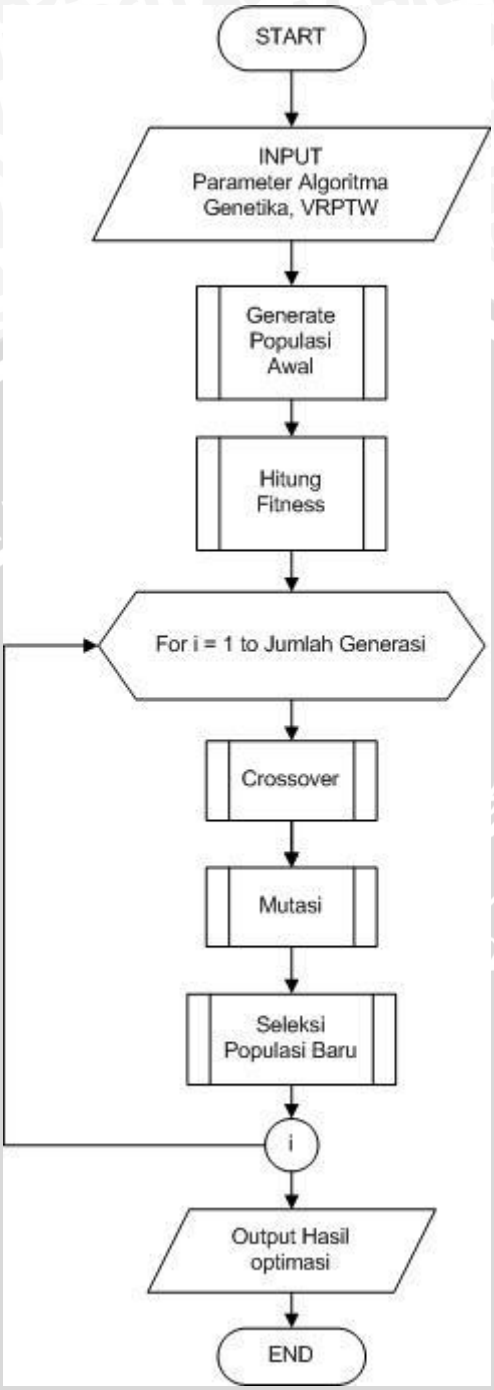
- Asumsi kecepatan rata-rata kendaraan
- Probabilitas mutasi
- Probabilitas *crossover*
- Banyaknya generasi
- Ukuran populasi

3.2.3 Perancangan Perangkat Lunak

Pada tahap perancangan perangkat lunak, proses penerapan *hybrid* algoritma genetika untuk permasalahan VRPTW yang berjalan pada aplikasi dijabarkan dalam sebuah diagram alir sesuai dengan analisis yang dilakukan sebelumnya. Pada prosesnya, terdapat beberapa parameter yang dibutuhkan yaitu :

1. Parameter genetika yang terdiri dari :
   - Nilai *pop_size* (ukuran populasi)
   - Banyaknya generasi
   - Probabilitas *crossover* dan mutasi
2. Parameter VRPTW yang terdiri dari :
   - Jumlah pelanggan
   - Permintaan pelanggan
   - Asumsi kecepatan rata-rata kendaraan
   - Jumlah kendaraan
   - Kapasitas kendaraan
   - Waktu pelayanan kendaraan
   - Interval waktu pelayanan pelanggan
   - Lokasi pelanggan

Gambar 3.3 berikut ini menunjukkan tahapan – tahapan proses *hybrid* algoritma genetika yang berjalan pada aplikasi.



Gambar 3.3 *Flowchart* penerapan *hybrid* algoritma genetika dalam perangkat lunak.

3.2.4 Proses *Hybrid* Algoritma Genetika

Secara garis besar, proses *hybrid* algoritma genetika diilustrasikan pada Gambar 3.3. prosesnya meliputi:

1. Inisialisasi parameter yang terdiri dari parameter algoritma genetika, dan VRPTW.
2. Membangkitkan sejumlah 50% populasi secara acak dan 50% berdasarkan *nearest insertion heuristic* sebagai individu awal.
3. Menghitung nilai fitness setiap individu.
4. Melakukan proses *crossover* yang bertujuan menghasilkan *offspring* sesuai dengan nilai probabilitas *crossover*.
5. Melakukan proses mutasi berdasarkan nilai probabilitas mutasi.
6. Melakukan proses seleksi untuk menentukan populasai baru dengan metode seleksi *roulette wheel*.

Pada setiap proses yang di ilustrasikan pada Gambar 3.3, terdapat *predefined process* dimana proses tersebut akan dijelaskan secara rinci dan lebih mendetail pada sub bab selanjutnya.

3.2.4.1 Representasi Individu

Pada permasalahan ini, representasi kromosom menggunakan pengkodean nilai atau permutasi. Setiap kendaraan yang melayani sejumlah pelanggan ditandai dengan nilai 0. Artinya, nilai 0 disini menunjukkan batasan pelanggan yang dilayani oleh suatu kendaraan dengan batasan permintaan pelanggan tidak melebihi daya angkut kendaraan.

Pada penelitian ini, 1 individu merupakan sebuah solusi yang menunjukkan sejumlah rute distribusi ke sejumlah pelanggan. Individu ini menunjukkan suatu kendaraan yang melayani suatu pelanggan dengan batasan setiap pelanggan hanya dilayani satu kali oleh satu kendaraan.

Sebagai ilustrasi, dimisalkan terdapat 10 pelanggan dengan jumlah permintaan tertentu. Diasumsikan jumlah kendaraan adalah 3 dan kapasitas setiap kendaraan adalah 100 unit barang. Maka model penyelesaiannya jika diketahui jumlah pelanggan yang tertera pada Tabel 3.1 adalah:

Tabel 3.1 Jumlah Permintaan Pelanggan

Pelanggan	1	2	3	4	5	6	7	8	9	10
Permintaan	55	35	20	15	60	30	10	25	10	25

1. Representasi individu awal berdasarkan data yang diambil secara acak dari data pelanggan. Misalnya saja : 3 – 1 – 2 – 4 – 6 – 7 – 10 – 5 – 9 – 8
2. Individu tersebut menunjukkan urutan pelanggan yang harus dilayani oleh setiap kendaraan. Dimulai dari indeks pelanggan yang pertama, akan dilayani oleh kendaraan 1. Jika jumlah permintaan tidak melebihi kapasitas kendaraan, maka kendaraan tersebut dapat melayani pelanggan selanjutnya. Jika permintaan selanjutnya melebihi kapasitas kendaraan, maka pelanggan tersebut akan dilayani oleh kendaraan berikutnya.
3. Dari kromosom yang dibentuk sebelumnya, maka didapatkan solusi pembagian kendaraan sebagai berikut:

Tabel 3.2 Solusi Pembagian Kendaraan

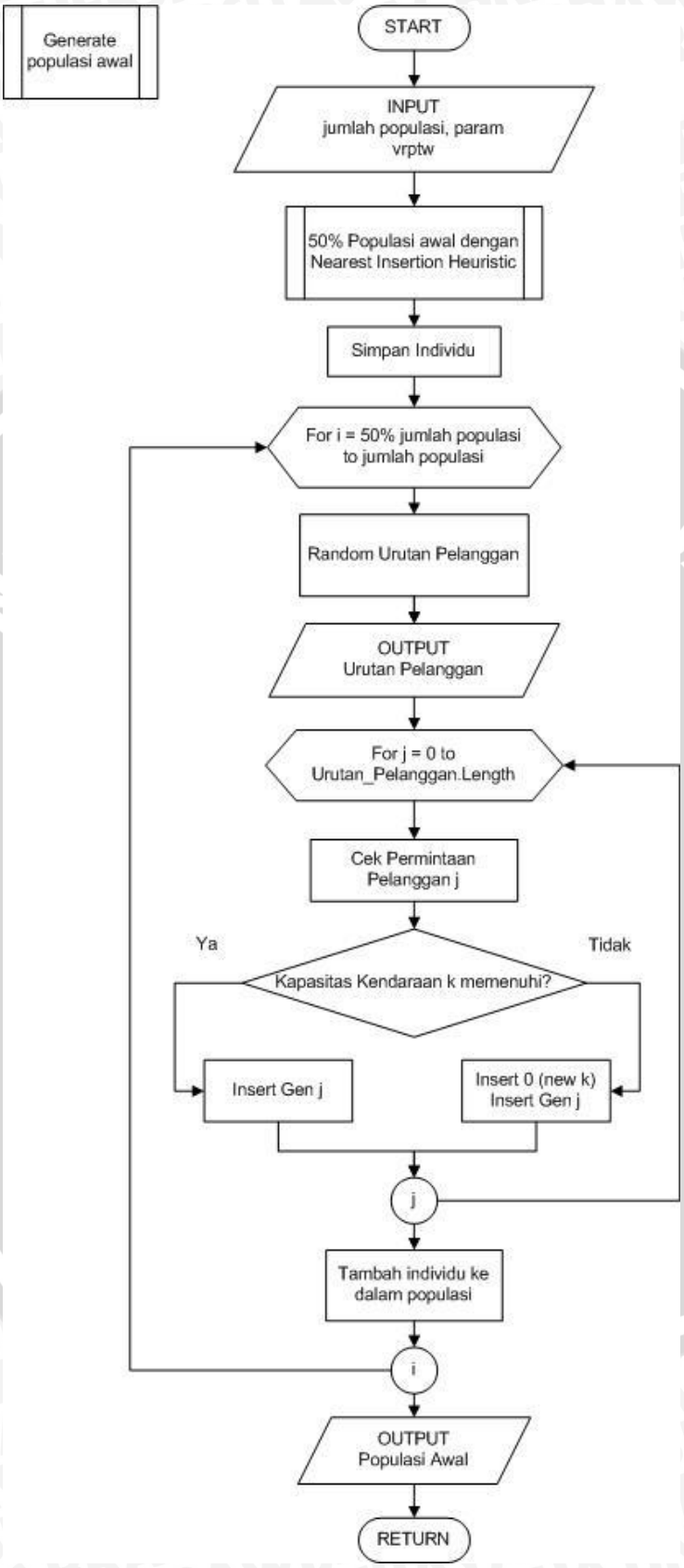
Kendaraan	Pelanggan
1	3,1
2	2,4,6,7
3	10,5,9
4	8

4. Dari solusi pembagian kendaraan yang tertera di Tabel 3.2, jumlah permintan pelanggan tidak melebihi kapasitas kendaraan. Maka didapatkan representasi kromosomnya adalah sebagai berikut:
5. 3 – 1 – 0 –2 – 4 – 6 – 7 – 0 – 10 – 5 – 9 – 0 – 8

3.2.4.2 Generate Populasi Awal

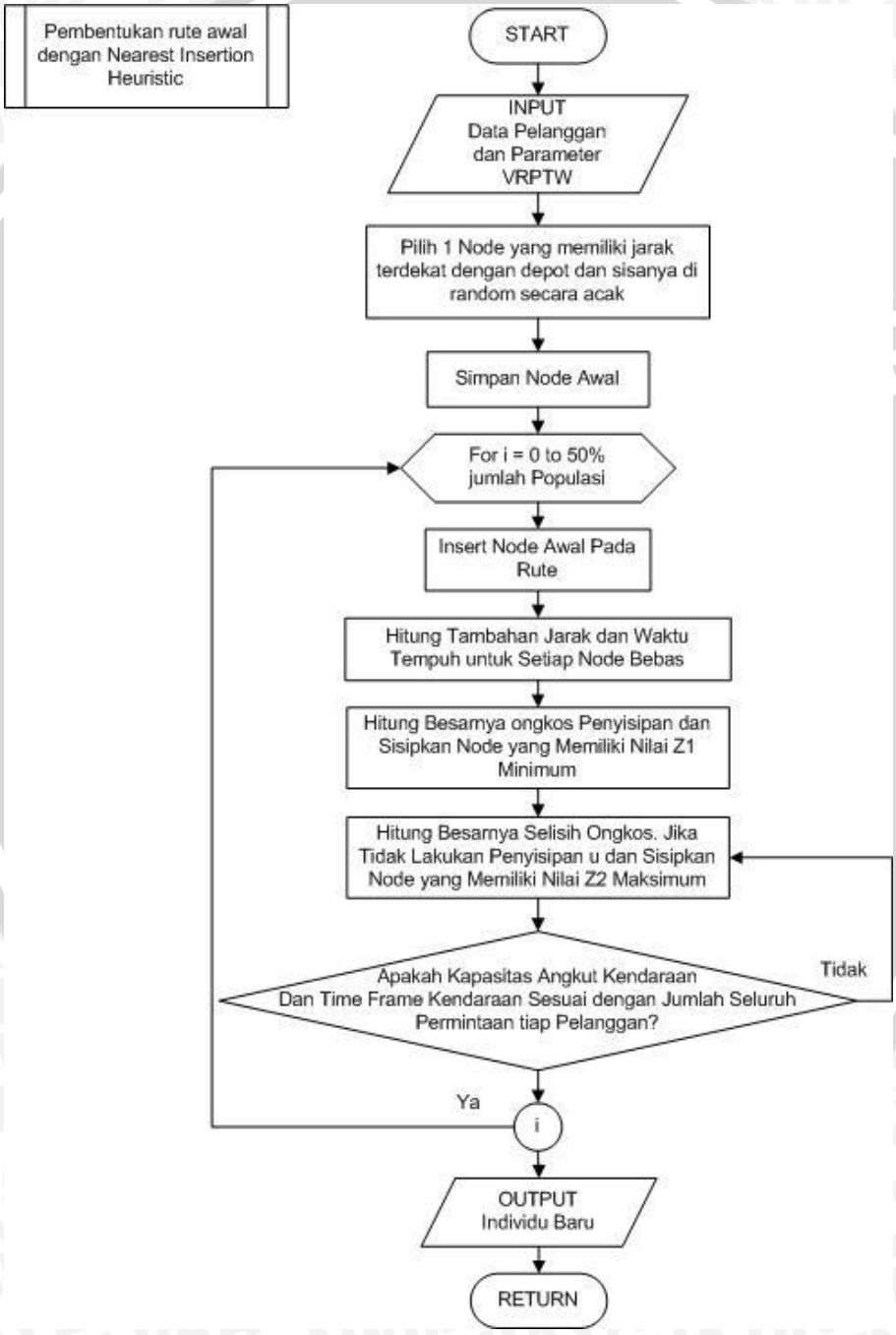
Pada proses pembangkitan populasi awal, populasi dibangkitkan 50% secara acak dan 50% berdasarkan *nearest insertion heuristic*. Berikut langkah-langkah pembangkitan populasi awal yang dilakukan dalam penelitian:

1. Pembentukan rute awal dengan *nearest insertion heuristic* yang bertujuan menghasilkan solusi awal dengan meminimumkan jarak serta memenuhi kendala time windows dan kapasitas kendaraan tidak melebihi permintaan pelanggan.
2. Proses pemilihan individu secara acak dengan mempertimbangkan kendala kapasitas kendaraan tetapi tidak memperhitungkan kendala time windows dan nominal jarak yang dihasilkan. *Flowchart* pembangkitan populasi baru, dapat dilihat pada Gambar 3.4 berikut:



Gambar 3.4 *Flowchart* Pembangkitan Populasi Awal.

Seperti yang dijelaskan sebelumnya, *nearest insertion heuristic* merupakan algoritma pembentukan 50 % dari solusi awal sebagai solusi layak yang bertujuan meminimalkan jarak serta memenuhi kendala time windows dan kapasitas kendaraan. *Flowchart* proses *nearest insertion heuristic* ditunjukkan oleh Gambar 3.5 berikut:



Gambar 3.5 *Flowchart* Penerapan *Nearest Insertion Heuristic*.

3.2.4.3 Nilai Fitness

Fitness bertujuan menentukan kemampuan individu untuk bertahan hidup. Semakin besar nilai fitness maka peluang individu untuk bertahan hidup juga semakin besar. Pada penelitian ini, fungsi fitness bertujuan menentukan rute yang paling optimal pada permasalahan VRPTW.

Fungsi fitness yang digunakan pada permasalahan ini yaitu fungsi minimasi. Fitness didapatkan dari nilai *penalty* dan total jarak. *Penalty* diberikan jika suatu kendaraan melayani pelanggan diatas jam pelayanan atau diluar waktu interval yang sudah ditentukan. Jumlah dari seluruh penalty dan total jarak akan menjadi tolak ukur mengetahui nilai fitness. Berdasarkan fungsi minimasi, semakin besar nilai *penalty* dan total jarak, maka fitness yang didapatkan semakin kecil begitu pula sebaliknya. Semakin kecil nilai *penalty* dan total jarak maka nilai fitness semakin tinggi yang berarti solusi tersebut layak dijadikan solusi yang mendekati optimal. Pada penelitian ini, nilai fitness yang dihasilkan akan dikalikan C dimana C merupakan nilai konstan yang ditetapkan sebelumnya [12]. Nilai C bertujuan memudahkan proses perhitungan dan mencegah ketika fitness yang dihasilkan bernilai sangat kecil. Persamaan 3.1 berikut merupakan persamaan fungsi fitness yang digunakan pada penelitian.

$$f = \frac{1}{(\sum_{i=0}^{n} p_i + \sum_{k \in K} \sum_{(i,j) \in V} c_{ij} x_{ijk}) + 1} \; x \; C \qquad (3.1)$$

Diketahui:

f = nilai fitness

p_i = *penalty* yang diberikan pada pelanggan ke-i

c_{ij} = jarak antara konsumen i ke konsumen j

x_{ijk} = Kendaraan k dari konsumen i ke konsumen j

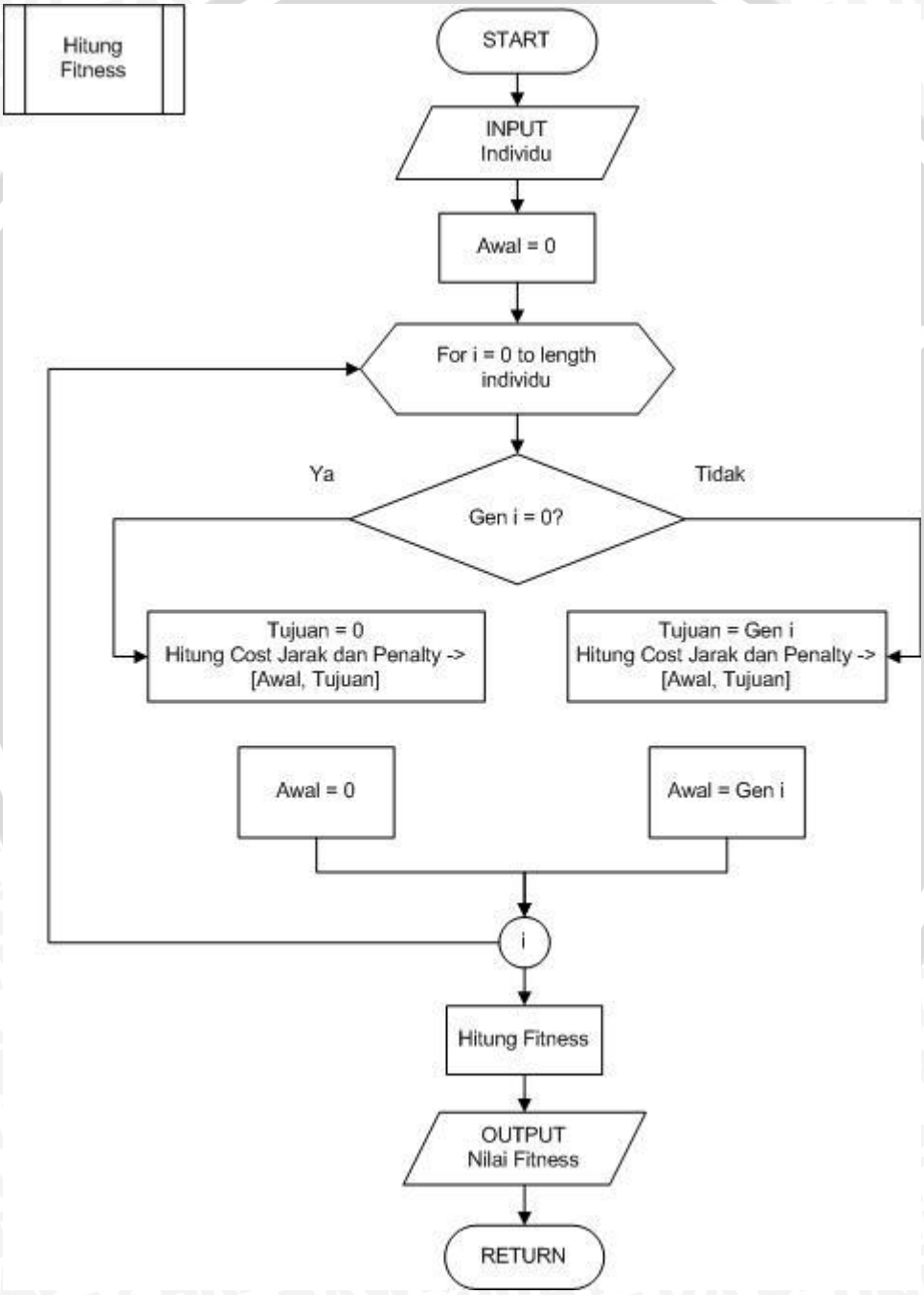
C = Nilai konstan yang ditetapkan

Adapun langkah-langkah yang dilakukan dalam pencarian nilai fitness yaitu:

1. Inisialisasi daftar rute yang direpresentasikan dalam bentuk kromosom.
2. Setiap kendaraan melakukan perjalanan dimulai dari gen 0 yang merupakan depot dan berakhir pula di depot.

3. Menghitung *cost* perjalanan serta *penalty* dari rangkaian gen yang ada pada kromosom. Pemilihan rute pada sejumlah gen dalam kromosom bergantung pada kapasitas kendaraan yang melayani permintaan pelanggan.

Flowchart perhitungan nilai fitness dapat dilihat pada Gambar 3.6 berikut:



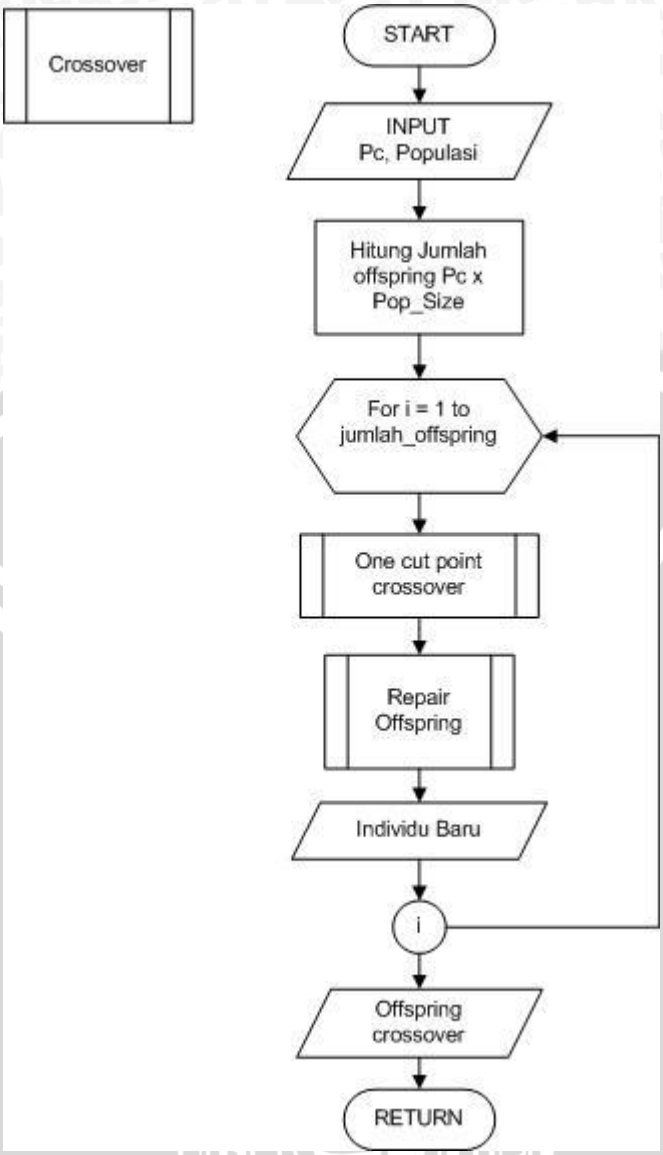
Gambar 3.6 *Flowchart* Hitung Nilai Fitness.

3.2.4.4 Crossover

Pada penelitian ini, proses *crossover* dilakukan dengan memilih dua induk untuk menghasilkan *offspring* (kategori seksual). Metode yang digunakan yaitu *crossover* satu titik (*one-cut-point*) dimana titik tersebut dipilih secara acak sebagai titik potong untuk kemudian saling dipertukarkan antara 2 induk. Selanjutnya *repair offspring* dimana akan dilakukan penggabungan gen 2 kendaraan yang memiliki jumlah pelanggan paling sedikit.

Fungsi *crossover* dipengaruhi oleh nilai probabilitas *crossover* (p_c). Pada permasalahan ini, nilai p_c menunjukkan probabilitas jumlah *offspring* yang dihasilkan pada proses *crossover*. Berdasarkan probabilitas tersebut, maka didapatkan jumlah *offspring* $p_c \; x \; ukuran \; populasi$ ($p_c \; x \; pop\_size$). Adapun langkah-langkah proses crossover yang ditunjukkan pada gambar 3.7 adalah:

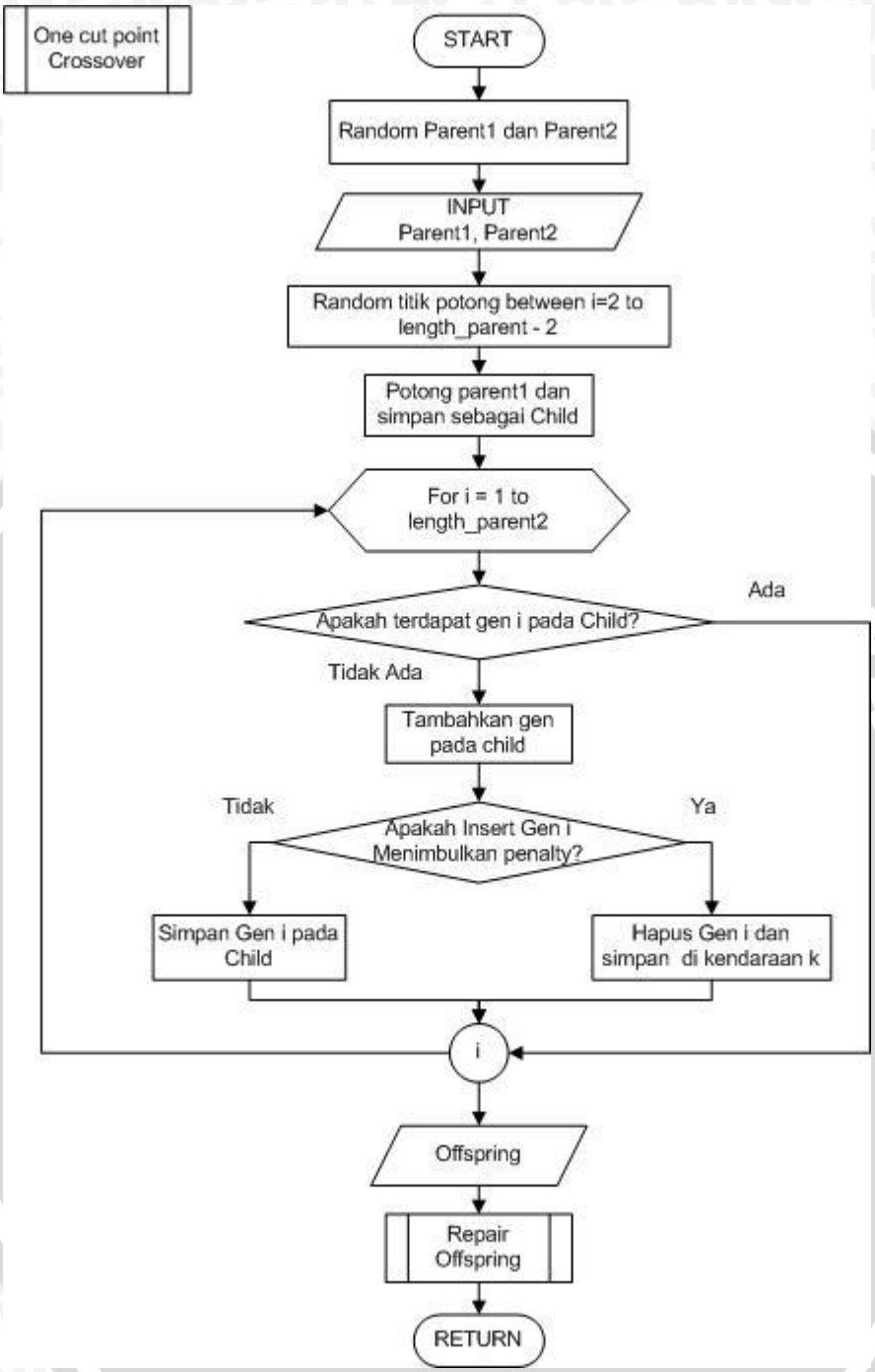
1. Inisialisasi jumlah populasi dan nilai probabilitas crossover (p_c).
2. Menghitung jumlah offspring yang dihasilkan sebanyak $p_c \; x \; pop\_size$.
3. Melakukan proses *one cut point crossover* hingga jumlah *offspring* yang dihasilkan = $p_c \; x \; pop\_size$
4. Melakukan proses *repair offspring*.



Gambar 3.7 *Flowchart* Proses Crossover.

Adapun langkah-langkah proses *one-cut-point crossover* yang ditunjukkan pada Gambar 3.8 adalah sebagai berikut:

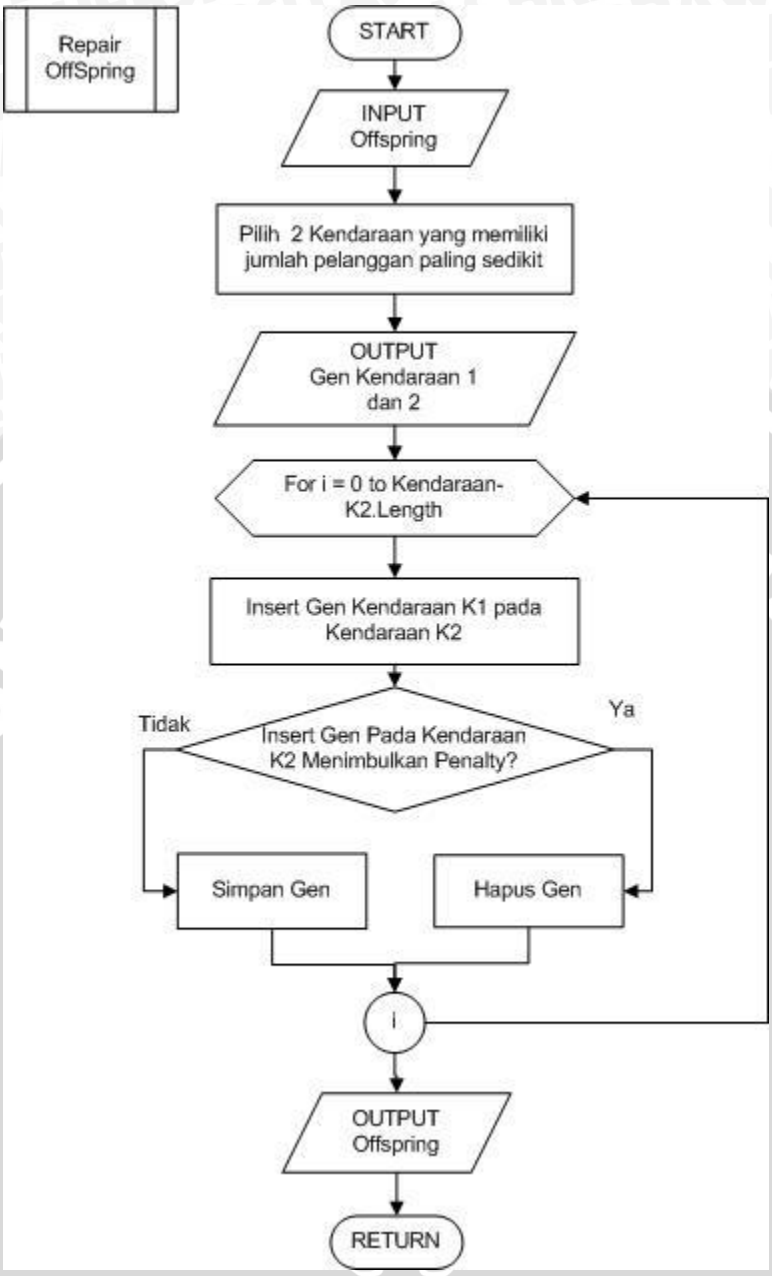
1. Memilih 2 *parent* secara acak dalam satu populasi.
2. Memilih titik potong secara random antara indeks ke-2 sampai dengan *length_parent* – 2. Hal ini bertujuan agar *offspring* yang dihasilkan lebih varian. Melakukan *crossover* atau kawin silang antara 2 *parent* yang dipilih sebelumnya.
3. Melakukan proses *crossover* dengan metode *one-cut-point crossover* yaitu dengan mengambil potongan dari induk pertama. Potongan induk pertama kemudian disimpan sebagai calon *offspring* atau *child*.
4. Gen *child* yang diisi dengan potongan induk pertama kemudian ditambahkan dengan gen pada induk 2 yang belum terkandung pada gen *child*.
5. Dilakukan pengecekan terhadap gen parent 2 yang diinsertkan. Jika proses insert menimbulkan penalty pada kendaraan *k*, maka gen tersebut akan disimpan dikendaraan lain. Sebaliknya, jika tidak menimbulkan *penalty*, maka gen akan disimpan pada kendaraan *k* pada child.
6. Proses ini akan menghasilkan offspring yang kemudian akan dilakukan proses *repair*.



Gambar 3.8 *Flowchart* Proses One Cut Point Crossover.

Pada proses *repair offspring*, akan dipilih kendaraan yang memiliki jumlah pelanggan paling sedikit. Tahapan dari proses *repair* berdasarkan Gambar 3.9 adalah sebagai berikut :

1. Dipilih 2 kendaraan yang memiliki jumlah pelanggan paling sedikit pada suatu offspring.
2. Menyisipkan atau penggabungan gen-gen yang ada pada kendaraan $k1$ (kendaraan yang memiliki jumlah pelanggan paling sedikit) pada kendaraan $k2$. Jika proses penyisipan menimbulkan *penalty*, maka gen tersebut batal disisipkan. Sebaliknya, jika proses penyisipan tidak menimbulkan penalty, maka gen yang disisipkan disimpan di kendaraan $k2$.



Gambar 3.9 *Flowchart* Proses *Repair Offspring*.

3.2.4.5 Mutasi

Proses mutasi dilakukan dengan mengubah secara acak nilai dari suatu bit. Pada permasalahan ini, ada 3 tahap mutasi yang dilakukan yaitu:

1. *Insertion Mutation.*

   Pada tahap ini, mutasi dilakukan dengan memilih kendaraan yang memiliki jumlah pelanggan paling sedikit. Setelah itu, ditentukan titik

mutasi dimana penyisipan dilakukan. Penyisipan dari gen kendaraan paling sedikit dimulai dari titik terpilih hingga kendaraan terakhir.

2. *Insertion Mutation* Kendaraan.

   Pada mutasi tahap ini, dipilih 2 kendaraan secara acak untuk kemudian dilakukan proses penyisipan kendaraan diantara keduanya. Proses penyisipan dilakukan jika tidak menimbulkan *penalty*.

3. *Reciprocal Exchange Mutation* Gen Kendaraan.

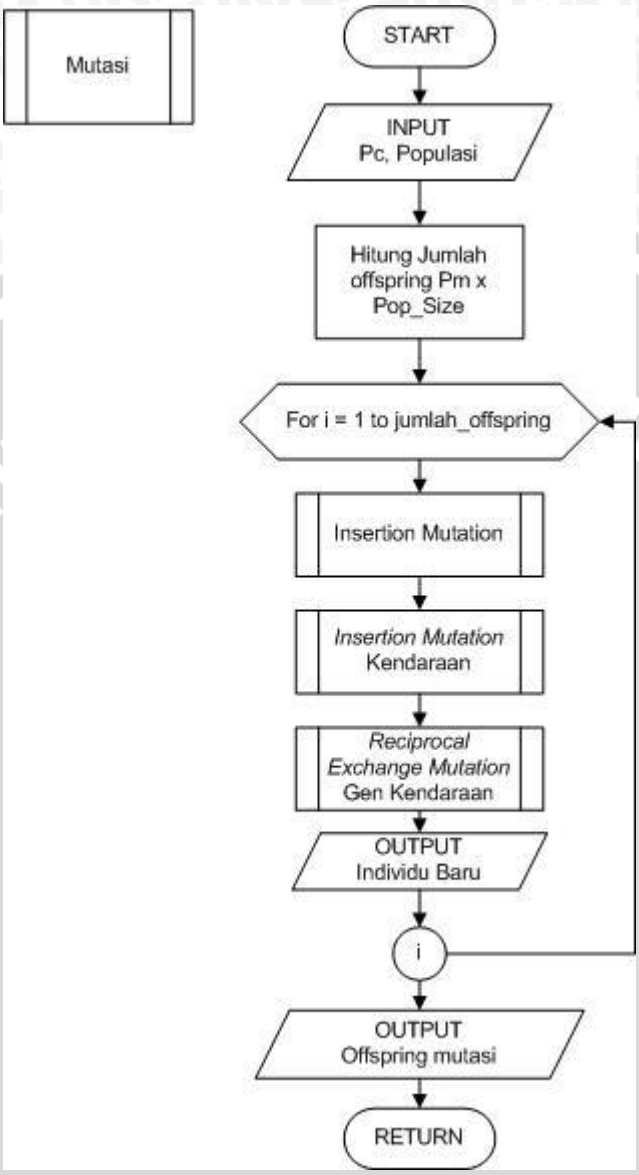
   Mutasi tahap ini dilakukan dengan cara memilih secara acak suatu kendaraan pada individu. Dari kendaraan tersebut, dipilih secara acak 2 gen untuk ditukarkan. Selanjutnya dilakukan pengecekan apakah proses penukaran menimbulkan penalty atau tidak. Jika tidak, maka akan dicek apakah dengan penukaran gen tersebut dapat mengurangi total jarak kendaraan. Jika pada proses tersebut dapat mengurangi jarak tanpa menimbulkan penalty yang lebih besar, maka gen tersebut disimpan sebagai hasil dari proses mutasi.

Parameter yang sangat penting yaitu peluang mutasi (p_m). Peluang mutasi disini menunjukkan persentasi jumlah individu pada populasi yang akan mengalami mutasi.

Adapun langkah-langkah proses mutasi yang dilakukan pada penelitian ini adalah sebagai berikut:

1. Inisialisasi jumlah populasi dan nilai probabilitas mutasi (p_m).
2. Menghitung jumlah *offspring* yang dihasilkan sebanyak $p_m \; x \; pop\_size$.
3. Melakukan proses *insertion mutation*.
4. Menyimpan offspring hasil dari proses *insertion mutation*.
5. Melakukan proses *insertion mutation* Kendaraan.
6. *Reciprocal exchange mutation* Gen Kendaraan.
7. Membandingkan total jarak yang dihasilkan antara *offspring* dari proses *insertion mutation* dengan tambahan metode *insertion mutation* kendaraan dan *reciprocal exchange mutation* gen kendaraan.
8. Memilih *offspring* yang dihasilkan berdasarkan total jarak terkecil untuk disimpan sebagai individu baru.

Flowchart proses mutasi dapat dilihat pada Gambar 3.10 berikut:



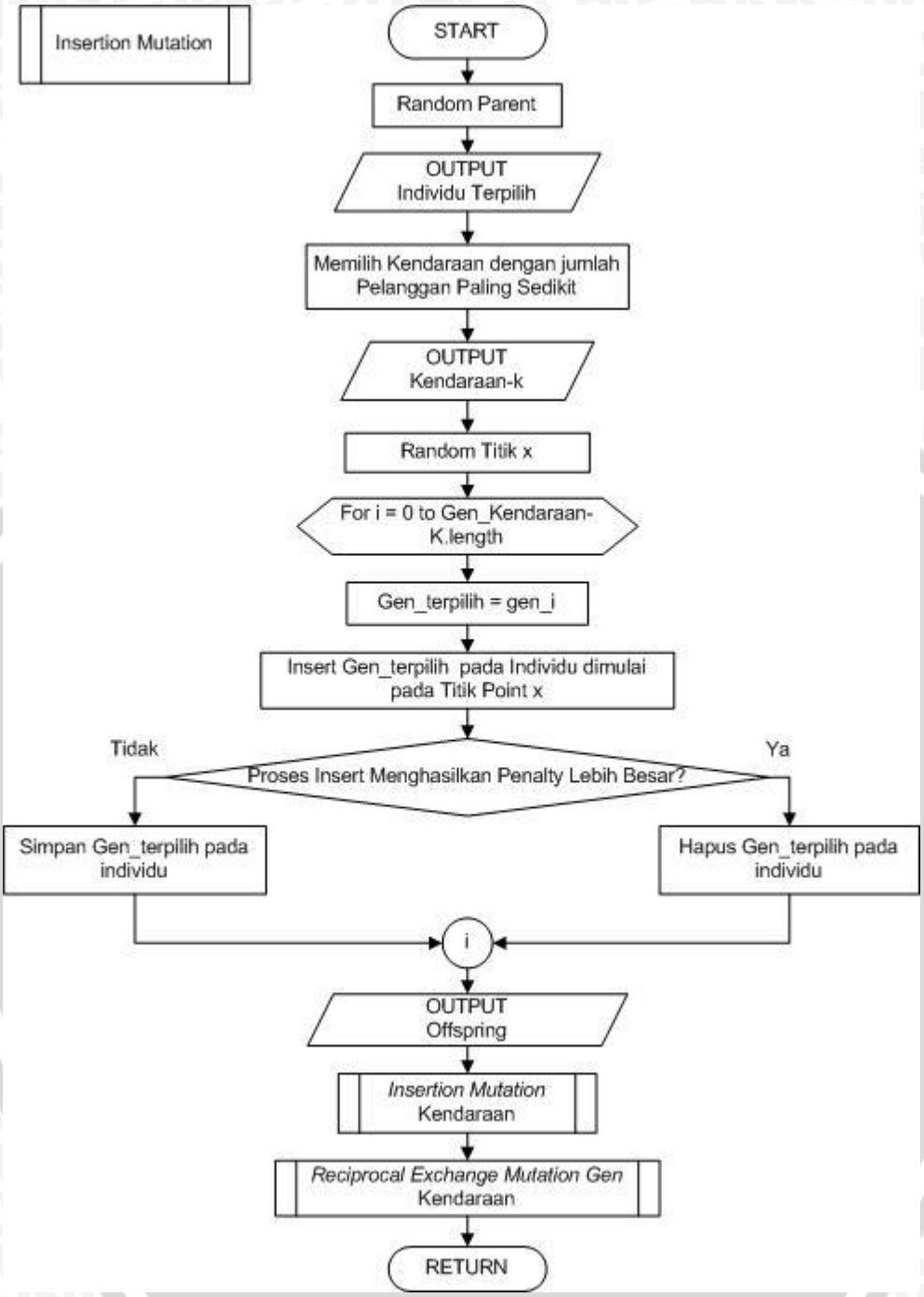
Gambar 3.10 *Flowchart* Proses Mutasi.

Metode *insertion mutation* dapat dilakukan dengan langkah-langkah sebagai berikut:

1. Memilih random *parent* dalam suatu populasi.
2. Memilih kendaraan yang memiliki jumlah pelanggan paling sedikit.
3. Memilih titik *point* dimana akan dilakukan proses penyisipan.

4. Dilakukan penyisipan gen kendaraan dengan jumlah pelanggan paling sedikit dimulai dari titik tersebut.
5. Dilakukan proses pengecekan apakah penyisipan menimbulkan *penalty* atau tidak. Gen tidak disisipkan jika menimbulkan *penalty*.

Detail proses mutasi dengan metode *insertion mutation* dapat dilihat pada *flowchart* gambar 3.11 berikut:

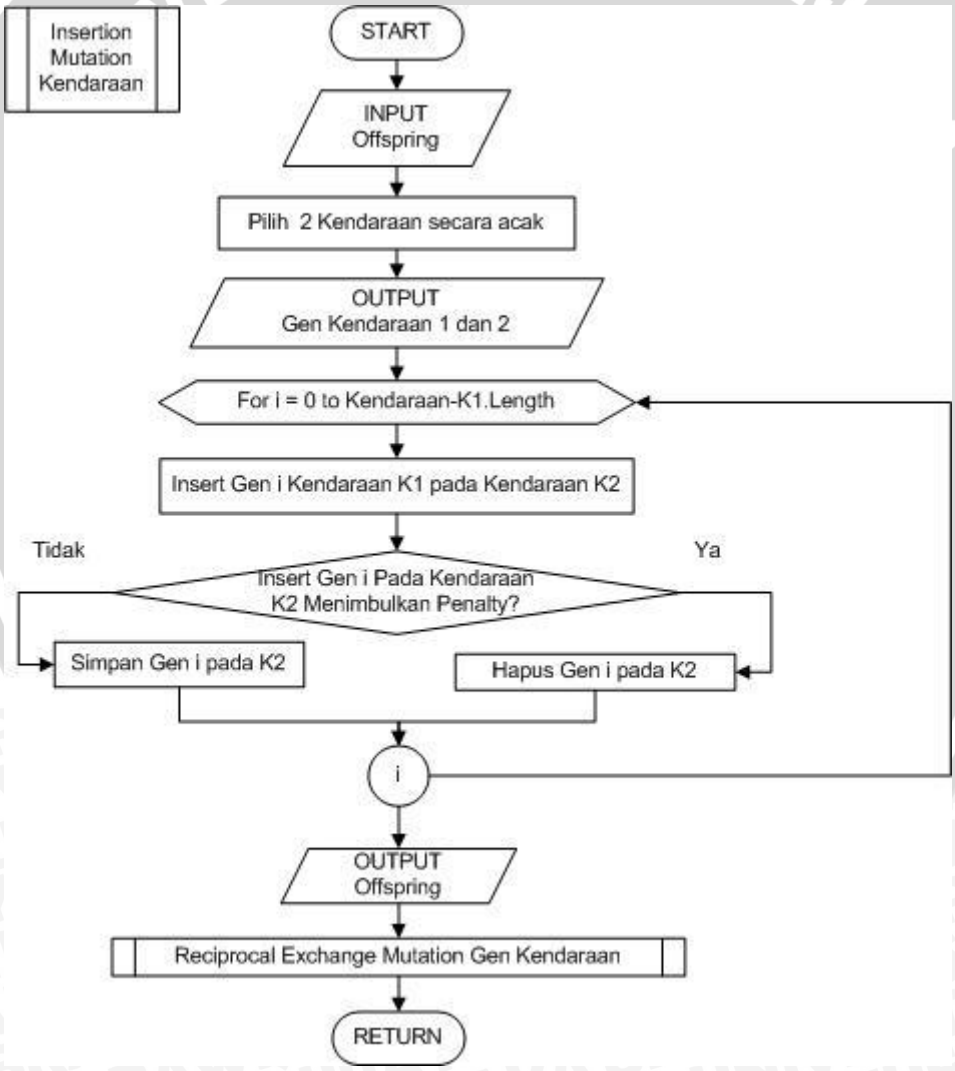


Gambar 3.11 *Flowchart* Proses *Insertion Mutation*.

Selanjutnya, langkah-langkah yang dilakukan pada proses mutasi dengan menggunakan metode i*nsertion mutation* kendaraan adalah sebagai berikut:

1. Menyimpan *offspring* dari proses sebelumnya.
2. Memilih 2 kendaraan secara acak pada *offspring* tersebut.
3. Setelah terpilih 2 kendaraan, misalnya saja kendaraan-1 dan kendaraan-2, dilakukan penyisipan antara 2 kendaraan tersebut. Proses penyisipan dilakukan jika tidak menimbulkan *penalty*.

Detail proses mutasi dengan metode i*nsertion mutation* kendaraan dapat dilihat pada *flowchart* gambar 3.12 berikut:

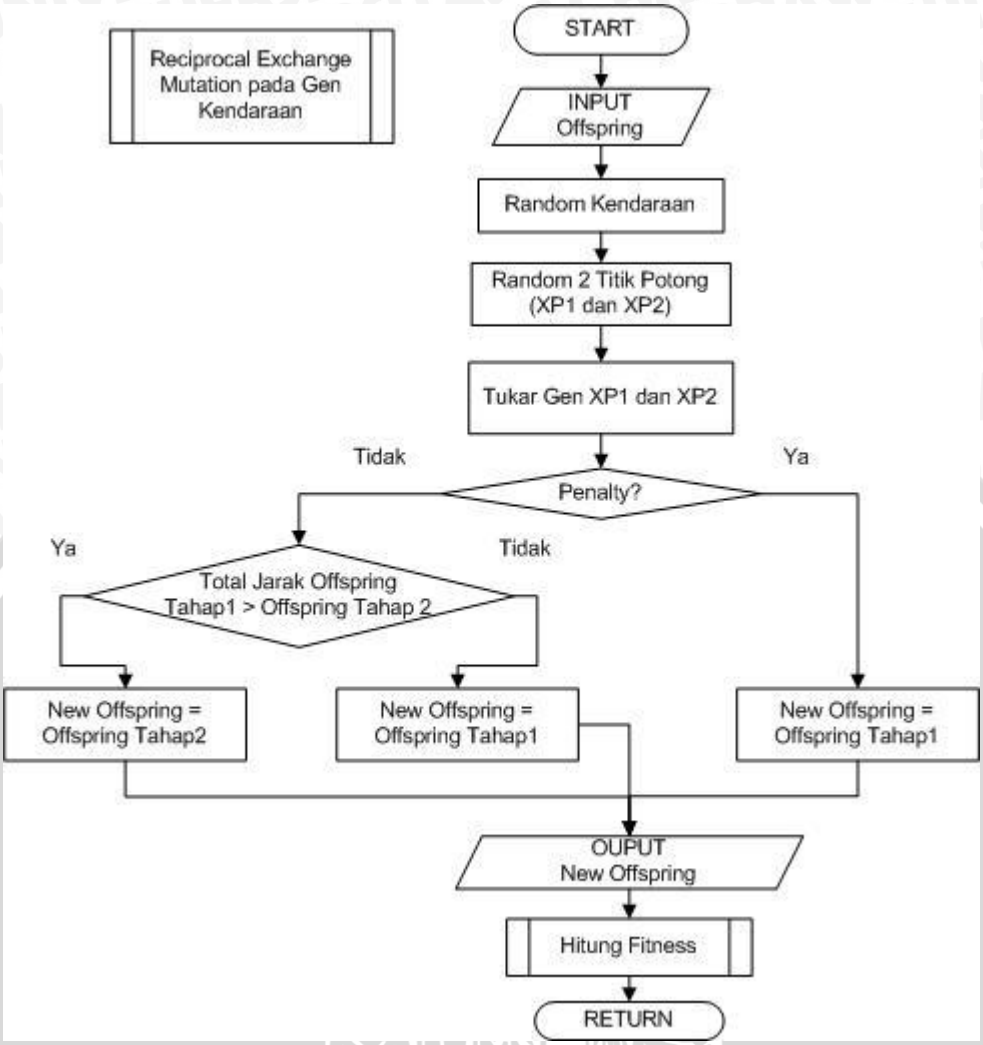


Gambar 3.12 *Flowchart* Proses *Insertion Mutation* Kendaraan.

Mutasi terakhir yang dilakukan adalah *reciprocal exchange mutation* gen kendaraan yang dilakukan dengan cara:

1. Memilih kendaraan secara acak pada *offspring* tersebut.
2. Random *exchange point* 1 (XP1) dan *exchange point* 2 (XP2) untuk dilakukan proses pertukaran posisi gen pada kendaraan.
3. Tukarkan kedua posisi gen pada kendaraan.
4. Lakukan evaluasi terhadap total jarak dari offspring yang dihasilkan dari mutasi tahap 1 dengan kombinasi mutasi dari tahap 1, tahap 2 dan tahap 3.

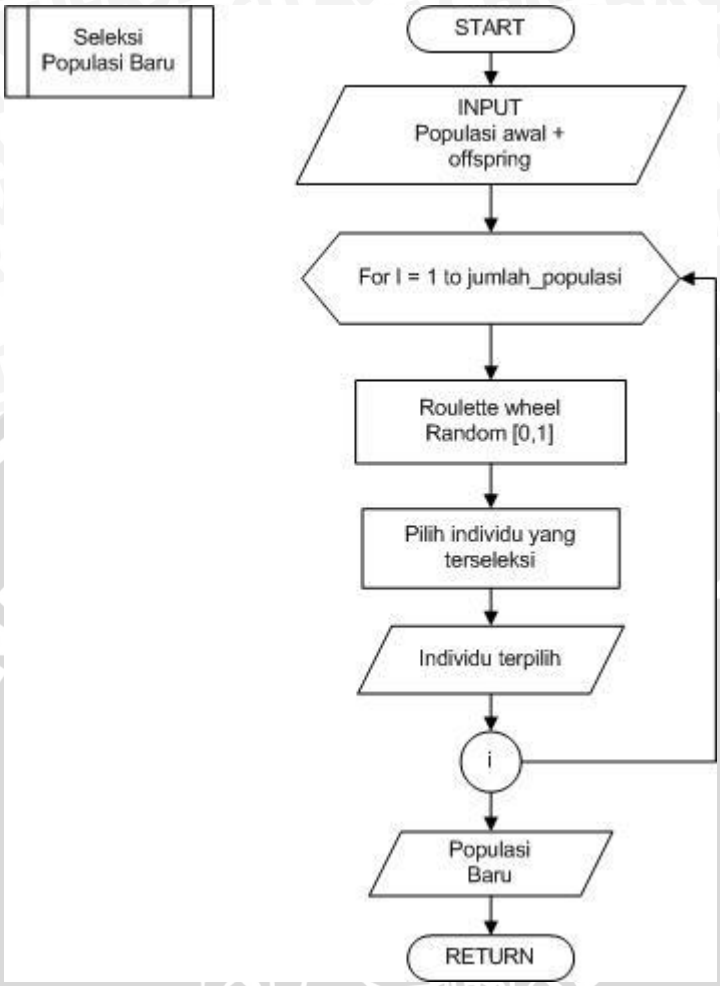
Detail proses mutasi dengan metode *reciprocal exchange mutation* dapat dilihat pada *flowchart* gambar 3.13 berikut:



Gambar 3.13 *Flowchart* Proses *Reciprocal Exchange Mutation* pada Gen Kendaraan.

3.2.4.6 Seleksi

Proses seleksi merupakan tahap dimana dipilih sejumlah individu yang memiliki peluang hidup lebih besar. Pada penelitian ini metode solusi yang digunakan adalah *roulette wheel*. Metode *roulette wheel* merupakan bagian dari seleksi proporsional probabilitas dimana setiap individu memiliki peluang hidup sejumlah n % yang dihitung berdasarkan nilai fitnessnya. Fungsi fitness yang digunakan pada penelitian ini berdasarkan fungsi minimasi yang telah dijelaskan sebelumnya.



Gambar 3.14 *Flowchart* Seleksi Populasi Baru.

Langkah-langkah proses seleksi menggunakan metode *roulette wheel* yang ditunjukkan pada *flowchart* Gambar 3.14 adalah sebagai berikut:

1. Inputkan populasi dan *offspring*.
2. Individu dipilih sebanyak jumlah populasi dengan menggunakan metode *roulette wheel*. Metode ini memilih secara random nilai antara 0 dan 1. Kemudian dipilih individu berdasarkan probabilitas kumulatif yang dihasilkan.

3.2.5 Perhitungan Manual

Perhitungan manual menunjukkan contoh model perhitungan pada permasalahan VRPTW dengan menggunakan metode *hybrid* algoritma genetika. Dimisalkan ada 6 pelanggan yang harus dikunjungi dan setiap pelanggan memiliki sejumlah permintaan dan *time frame* pelayanan masing-masing. Berikut diketahui:

Kapasitas kendaraan : 50
Jumlah kendaraan : 5
Kecepatan rata-rata : 60
Waktu pelayanan : 90
Time frame kendaraan : 0 – 1236

Tabel 3.3 Data Pelanggan

Customer No	XCoord.	YCoord.	Demand	Ready Time	Due Date	Service
0	40	50	0	0	1236	0
1	45	68	15	912	967	90
2	18	75	25	825	870	90
3	40	66	20	65	146	90
4	35	69	15	727	782	90
5	42	68	20	15	67	90
6	45	70	30	621	702	90

Berikut Tabel 3.4 menunjukkan parameter algoritma genetika yang digunakan:

Tabel 3.4 Parameter Algoritma Genetika

Parameter	Nilai
Pop Size	6
Pc	0.4
Pm	0.2
Banyaknya generasi	10

Dari Tabel 3.3, dibuat matrik jarak antar konsumen dan depot. Perhitungan jarak dicari dengan rumus euclidean yaitu $= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Misalnya saja dicari jarak dari depot (0) ke pelanggan 1, maka:

Jarak [0,1] = $\sqrt{(40 - 45)^2 + (50 - 68)^2}$

$= \sqrt{25 + 324} = 18{,}68$

Dari contoh perhitungan tersebut, maka didapatkan matrik jarak sebagai berikut:

Tabel 3.5 Matrik Jarak

Cust.	0	1	2	3	4	5	6
0	0	18,68	33,3	16	19,65	18,11	20,62
1	18,68	0	27,89	5,39	10,05	3	2
2	33,3	27,89	0	23,77	18,03	25	27,46
3	16	5,39	23,77	0	5,83	2,83	6,4
4	19,65	10,05	18,03	5,83	0	7,07	10,05
5	18,11	3	25	2,83	7,07	0	3,61
6	20,62	2	27,46	6,4	10,05	3,61	0

Dari tabel 3.5 maka dibuat matriks waktu. Waktu tempuh diperoleh dari perhitungan jarak dibagi kecepatan rata-rata. Berikut contoh perhitungan:

Waktu tempuh [0,1] = $\frac{18{,}68}{60}$ = 0,311 Jam = 18,68 menit = 19 menit.

Tabel 3.6 Matrik Waktu

Cust.	0	1	2	3	4	5	6
0	0	19	33	16	20	18	21
1	19	0	28	5	10	3	2
2	33	28	0	24	18	25	27
3	16	5	24	0	6	3	6
4	20	10	18	6	0	7	10
5	18	3	25	3	7	0	4
6	21	2	27	6	10	4	0

3.2.5.1 Generate Populasi Awal

Pembangkitan populasi awal dilakukan dengan membangkitkan 50% solusi dari metode *nearest insertion heuristic* dan 50 % solusi yang diambil secara acak.

Adapun tahap perhitungan yang dilakukan pada metode *nearest insertion heuristic* untuk membangkitkan solusi awal dilakukan dengan cara sebagai berikut:

1. Memilih *node* awal untuk penentuan rute. *Node* awal sebanyak n dipilih sebanyak 50% dari *pop_size*. Maka $n = 0.5\ x\ 6 = 3$. Dari matriks jarak di

ambil 1 titik terdekat dari depot yaitu *node* 3, dan sisanya diambil secara acak. Pada kasus ini, misalnya 2 *node* tersisa yang dipilih secara acak adalah *node* 4 dan *node* 5.

2. Menentukan individu awal berdasarkan jarak terdekat misalnya saja *node* 3 sehingga $R_1 = \{0,3,0\}$.
3. Perhitungan penghematan jarak dan waktu tempuh terhadap seluruh *node* untuk menentukan *node* sisipan antara *node* 3 dan *node* 0 disesuaikan dengan kapasitas angkut dan *time windows*.
4. Contoh perhitungan penghematan jarak (Z_{11}) pada rute 1:
   a. Penghematan jarak di *node* (3,1)
      - Jika jarak dari *node* 3 ke *node* 1 adalah 5,39
      - Jika jarak dari *node* 1 ke *node* 0 adalah 18,68
      - Jika jarak dari *node* 3 ke *node* 0 adalah 16
      - Maka (3,1) = $d(3,1) + d(1,0) - d(3,0) = 5,39 + 18,68 - 16 = 8,07$
   b. Penghematan jarak di *node* (3,2), (3,4), (3,5), dan (3,6) dilakukan dengan cara yang sama seperti sebelumnya.
5. Hasil perhitungannya yaitu:

Tabel 3.7 Penghematan Jarak Untuk Rute 1

I	U	d(i,u)	d(u,0)	d(i,0)	Z11
3	1	5,39	18,68	16	8,07
	2	23,77	33,3	16	41,07
	4	5,83	19,65	16	9,48
	5	2,83	18,11	16	4,94
	6	6,4	20,62	16	11,02

6. Contoh untuk perhitungan penghematan waktu tempuh (Z_{12}) pada rute 1:
   a. Penghematan waktu di *node* (3,1)
      - Jika waktu tempuh dari *node* 3 ke *node* 1 adalah 6
      - Jika waktu tempuh dari *node* 1 ke *node* 0 adalah 19
      - Jika waktu tempuh dari *node* 3 ke *node* 0 adalah 16
      - Maka (3,1) = $t(3,1) + waktu\ pelayanan + t(1,0) - t(3,0) = 5 + 90 + 19 - 16 = 98$

- Penghematan waktu tempuh di *node* (3,2), (3,4), (3,5), dan (3,6) dilakukan dengan cara yang sama seperti sebelumnya.
- Hasil perhitungannya yaitu:

Tabel 3.8 Penghematan Waktu Tempuh Untuk Rute 1

i	U	t(i,u)	t(u)	t(u,0)	t(0,i)	Z12
3	1	5	90	19	16	98
	2	24	90	33	16	131
	4	6	90	20	16	100
	5	3	90	18	16	95
	6	6	90	21	16	101

c. Penghematan waktu tempuh di *node* (3,2), (3,4), (3,5), dan (3,6) dilakukan dengan cara yang sama seperti sebelumnya.

7. Perhitungan dalam menentukan *node* terbaik pada rute 1 sebagai berikut :
   a. Contoh perhitungan pada $Z_1(i,u,j)$
      1. $Z_1(3,1) = (\alpha_1 * Z_{11}) + (\alpha_2 * Z_{12}) = (0{,}9 * 8{,}07) + (0{,}1 * 98) = 17{,}063$
      2. $Z_1(3,2) = (\alpha_1 * Z_{11}) + (\alpha_2 * Z_{12}) = (0{,}9 * 42{,}07) + (0{,}1 * 131) = 50{,}063$
   b. Contoh perhitungan pada $Z_2(i,u,j)$
      1. $Z_2(3,1) = d(0,u) - Z_1(3,1) = 18{,}68 - 17{,}063 = 1{,}617$
      2. $Z_2(3,2) = d(0,u) - Z_1(3,2) = 33{,}3 - 50{,}063 = -16{,}763$
   c. Hasil perhitungan bisa dilihat pada Tabel 3.9.
8. Untuk menentukan posisi *node* terbaik di rute 1 yang akan disisipkan diantara *node* 3 dan *node* 0 (depot) adalah sebagai berikut:
   a. Mengambil nilai minimum pada kolom $Z_1(i,u,j)$

      Berdasarkan Tabel 3.9, *node* 5 terpilih. Kemudian di cek apakah *node* 5 memenuhi kendala kapasitas dan *time windows*. Berdasarkan data pelanggan, *demand node* 3 = 20 dan *node* 5 = 15 sehingga memenuhi kendala kapasitas maksimal angkut kendaraan. Akan tetapi, *node* 5 tidak memenuhi kendala *time windows*. Alternatif selanjutnya yang dilakukan adalah dengan menyisipkan *node* 5 sebelum *node* 3. Pada

proses ini, *node* 5 memenuhi kendala *time windows* sehingga layak untuk diinsertkan pada rute 1 sehingga $R1 = \{0,5,3,0\}$.

b. Kapasitas angkut tersisa

Karena kapasitas angkut tersisa maka disisipkan node berikutnya dengan mengambil nilai maksimum secara berurutan pada kolom $Z_2(i,u,j)$ dengan batasan kapasitas angkut dan *time windows*. Nilai maksimum yang ada pada $Z_2(i,u,j)$ yang belum diinsertkan pada rute 1 adalah *node* 1. Kapasitas angkut kendaraan pada rute 1 adalah 5 sedangkan *demand node* 1 = 15 sehingga, *node* 1 tidak layak di sisipkan pada rute 1 karena tidak memenuhi kendala kapasitas kendaraan. Pencarian *node* dilakukan sampai kapasitas tercukupi.

Tabel 3.9 Node Terbaik yang Disisipkan ke dalam Rute 1

I	U	Z1(i,u,j)	i(u)	j(u)	Z2(i,u,j)
3	1	17,063	3	0	1,617
	2	50,063	3	0	-16,763
	4	18,532	3	0	1,118
	5	13,946	3	0	4,164
	6	20,018	3	0	0,602

Berdasarkan perhitungan yang dilakukan sebelumnya, maka didapatkan sebanyak 3 individu awal dari metode *nearest insertion heuristic* dan 3 individu dibangkitkan secara acak dengan mempertimbangkan kapasitas kendaraan. Perlu diperhatikan bahwa nilai 0 yang ada pada rangkaian kromosom menunjukkan batas akhir pelanggan pada suatu kendaraan. Berikut ini 6 individu yang dibangkitkan sebagai populasi awal:

Tabel 3.10 Populasi Awal

Individu	Kromosom
1	5-3-0-6-1-0-4-2
2	4-2-0-5-3-0-6-1
3	5-3-0-6-1-0-4-2
4	2-3-0-5-6-0-4-1
5	4-6-0-5-2-0-1-3
6	6-1-0-5-2-0-4-3

3.2.5.2 Hitung Fitness

Setelah rute terbentuk sebagai suatu individu, kemudian dihitung nilai *fitness* nya. Pada penelitian ini, nilai *fitness* didapatkan berdasarkan total *penalty* dan total jarak yang dihasilkan. Berdasarkan contoh kasus yang ada pada Tabel 3.3, perhitungan nilai *penalty* dan total jarak dilakukan dengan detail perhitungan sebagai berikut:

Tabel 3.11 Perhitungan Total Jarak dan Penalty Individu 1

K	Rute	Jarak	W. Tempuh	W. Tiba	Pelayanan	W. Berangkat	Penalty
1	0-5	18,11	18	18	90	108	0
	5-3	2,83	3	111	90	201	0
	3-0	16	3	217	90	0	0
2	0-6	20,62	21	21	90	711	0
	6-1	2	2	713	90	1002	0
	1-0	18,68	2	1021	90	0	0
3	0-4	19,65	20	20	90	817	0
	4-2	18,03	18	835	90	925	0
	2-0	33,3	33	958	90	0	0
Tot. Jarak		149,22				**Total Penalty**	0

Diasumsikan setiap kendaraan berangkat dan berakhir didepot dengan waktu perjalanan pelanggan di mulai dari 0 sampai dengan 1236. *Penalty* dihitung jika kendaraan melayani pelanggan diluar waktu interval yang diberikan oleh pelanggan. Pada *node* 0 menuju *node* 6, kendaraan meninggalkan pelanggan 6 dimenit 711 karena kendaraan harus menunggu *time open* pelanggan 6 yaitu pada menit ke 621. Jadi pelayanan dapat dilakukan pada menit ke 621. Pada kasus ini, nilai $C = 1000$, maka didapatkan nilai fitness yaitu:

$$f = \frac{1}{149{,}22 + 0 + 1} x\ 1000 = 6{,}657$$

Tabel 3.12 Perhitungan Total Jarak dan Penalty Individu 6

K	Rute	Jarak	W. Tempuh	W. Tiba	Pelayanan	W. Berangkat	Penalty
1	0-6	20,62	21	21	90	711	0
	6-1	2	2	713	90	1002	0
	1-0	18,68	2	1021	90	0	0
2	0-5	18,11	18	18	90	108	0
	5-2	25	25	133	90	915	0
	2-0	33,3	25	948	90	0	0
3	0-4	19,65	20	20	90	817	0
	4-3	5,83	6	823	90	913	767
	3-0	16	16	929	90	0	0
Tot. Jarak		159,19				**Total Penalty**	0

Pada Tabel 3.12, rute kendaraan 3 dari *node* 4 menuju *node* 3 dikenakan *penalty* karena kendaraan tiba di pelanggan 3 pada menit ke 823 yang berarti hal ini diluar *time windows* dari pelanggan 3 (65-146). Berdasarkan perhitungan total jarak dan *penalty* seluruh individu pada populasi awal, maka didapatkan nilai fitness yang terdapat pada Tabel 3.13 berikut:

Tabel 3.13 Nilai Fitness Individu

Individu	Jumlah Kendaraan	Tot. Jarak	Penalty	Fitness
1	3	149,22	0	6,657
2	3	149,22	0	6,657
3	3	149,22	0	6,657
4	3	163,79	883	0,954
5	3	166,8	1166	0,750
6	3	159,19	767	1,079

Berdasarkan Tabel 3.13, nilai fitness yang paling besar terdapat pada individu 1, 2 dan 3 yang berarti individu tersebut memiliki peluang hidup lebih besar dibandingkan dengan individu lainnya.

3.2.5.3 Crossover

Seperti yang telah dijelaskan sebelumnya, proses *crossover* dilakukan dengan metode *one-cut-point*. Diketahui probabilitas *crossover* (p_c) = 0.4 dan *pop_size* = 6. maka jumlah offspring yang dihasilkan = 0.4 x 6 = 2 *offspring*. Pemilihan *parent* dilakukan secara acak dari ukuran populasi. Misalnya saja terpilih individu 2 dan 3 sebagai *parent*, maka proses *crossover* nya adalah sebagai berikut:

Individu-2	4	2	0	5	3	0	6	1
Individu-3	5	3	0	6	1	0	4	2

Gambar 3.15 Pemilihan Individu Secara Acak Proses Crossover.

Setelah dipilih Parent, maka dipilih titik potong secara acak. Misalnya titik potong atau *cut point* yang terpilih adalah 5 maka:

	Cut Point ↓							
Parent-1	4	2	0	5	3	0	6	1
Parent-2	5	3	0	6	1	0	4	2

Child-1	4	2	0	5	3	0	6	1

Gambar 3.16 Proses *Crossover* Menghasilkan *Offspring*.

Pada proses penyisipan, diketahui bahwa sisa demand pada kendaraan 2 yang memiliki pelanggan 5 dan 3 adalah 10. Gen tersisa yang ada pada *Parent 2* yaitu pelanggan 6 dan 1 ternyata tidak memenuhi kendala kapasitas kendaraan untuk disisipkan pada kendaraan 2. maka *node* 6 dan 1 disimpan dalam suatu barisan *node* yang nanti akan di proses kembali dengan menggunakan metode *nearest insertion heuristic*. Tahap selanjutnya yang dilakukan adalah *repair offspring crossover* dimana akan dipilih 2 kendaraan yang memiliki jumlah pelanggan paling sedikit. Pada Gambar 3.14, seluruh kendaraan memiliki jumlah pelanggan yang sama dan setiap kendaraan memenuhi kapasitas kendaraan sehingga tidak dapat disisipkan oleh *node* lainnya. *Offspring* yang dihasilkan pada

proses *crossover* kemudian dihitung nilai fitnessnya masing-masing. Hasil perhitungan nilai fitness yaitu:

Tabel 3.14 Nilai Fitness *Offspring* Proses *Crossover*

Individu	Jumlah Kendaraan	Tot. Jarak	Penalty	Fitness
C1	3	149,22	0	6,657
C2	3	149,22	0	6,657

3.2.5.4 Mutasi

Pada tahap awal, Proses mutasi dilakukan dengan metode *insertion mutation*. Diketahui probabilitas mutasi (p_m) = 0.2, maka jumlah *offspring* yang dihasilkan adalah 0.2 x 6 = 1 *offspring*. Sama seperti *crossover*, kandidat parent dipilih secara acak dari populasi yang ada. Misalnya saja, terpilih individu 3 sebagai *Parent*, maka proses mutasinya adalah:

Individu-3	5	3	0	6	1	0	4	2

Gambar 3.17 Pemilihan Individu Secara Acak Proses Mutasi.

Pada metode ini, dipilih kendaraan yang memiliki jumlah pelanggan paling sedikit. Misalnya saja terpilih kendaraan 1 yang berisi pelanggan 5 dan 3. *Node* 5 dan 3 dikeluarkan dari kromosom. Kemudian dipilih 1 titik sebagai posisi awal untuk penyisipan. Misalnya titik yang terpilih yaitu titik 0. Maka proses penyisipannya yaitu:

	↓ Insert Point				
Individu-3	6	1	0	4	2

Child-3	6	1	0	4	2	0	5	3

Gambar 3.18 Proses *Insertion Mutation*.

Sama seperti kasus sebelumnya pada *crossover, node* 5 dan 3 tidak dapat disisipkan karena semua kendaraan memenuhi kendala kapasitas kendaraan dan *time windows*.

Mutasi yang selanjutnya dilakukan adalah *insertion mutation* pada kendaraan. Dari o*ffspring* yang dihasilkan pada tahap mutasi sebelumnya, dipilih 2 kendaraan secara acak. Misalnya saja terpilih kendaraan $k-1$ dan $k-3$ Kemudian dilakukan proses penyisipan gen pada kendaraan 3 ke kendaraan 1. Proses penyisipan ini mempertimbangkan kendala kapasitas kendaraan dan *time windows*. Gen yang ada pada kendaraan 3 tidak dapat disisipkan pada kendaraan 1 karena tidak memenuhi kapasitas angkut kendaraan sehingga *offspring* yang dihasilkan sama seperti sebelumnya yaitu:

Child-3	6	1	0	4	2	0	5	3

Gambar 3.19 Proses *Insertion Mutation* Kendaraan.

Langkah terakhir yang dilakukan pada proses mutasi adalah *Reciprocal Exchange Mutation* Gen Kendaraan. Pada tahap ini, dipilih 1 kendaraan secara acak. Misalnya saja $k-1$ terpilih. Dipilih 2 *exchange point* secara random pada kendaraan tersebut untuk dilakukan proses pertukaran. Misalnya saja XP1 = 1 dan XP2 = 2. Maka rangkaian kromosom pada kendaraan 1 berubah menjadi 1-6. Dari hasil penukaran, dilakukan evaluasi apakah proses mutasi ini menimbulkan *penalty* atau tidak. Perubahan susunan gen pada kendaraan 1 ternyata menimbulkan *penalty* sehingga proses mutasi dibatalkan. Jika kombinasi mutasi pada tahap 2 dan 3 dapat mengurangi jarak tempuh dan memenuhi *time windows*, maka solusi tersebut akan dijadikan sebagai individu baru (*offspring*).

Tabel 3.15 Nilai Fitness Offspring Proses Mutasi

Individu	Jumlah Kendaraan	Tot. Jarak	Penalty	Fitness
C3	3	149,22	0	6,657

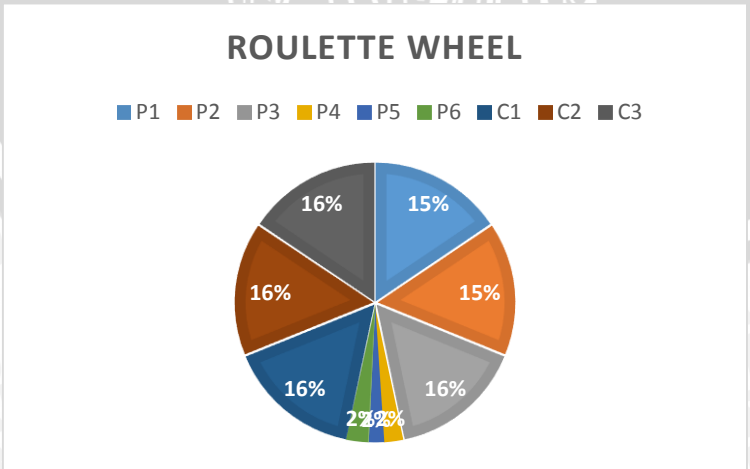
3.2.5.5 Seleksi Populasi Baru

Pada tahap seleksi, individu terpilih sebagai generasi selanjutnya dipilih dengan menggunakan metode *roulette wheel*. Jumlah individu dipilih sebanyak n *pop_size* dimana pada contoh kasus nilai *pop_size* = 6. Individu dipilih dengan menghitung nilai probabilitas kumulatif seluruh individu. Oleh karena itu, individu yang memiliki nilai *fitness* yang lebih besar memiliki peluang hidup yang lebih besar juga. Berikut metode seleksi menggunakan metode *roulette wheel*.

Tabel 3.16 Kumpulan Individu

Individu	Jumlah Kendaraan	Tot. Jarak	Penalty	Fitness	Prob	Prob Kum
P1	3	149,22	0	6,657	0,1558	0,1558
P2	3	149,22	0	6,657	0,1558	0,3116
P3	3	149,22	0	6,657	0,1558	0,4674
P4	3	163,79	883	0,954	0,0223	0,4898
P5	3	166,8	1166	0,750	0,0175	0,5073
P6	3	159,19	767	1,079	0,0252	0,5326
C1	3	149,22	0	6,657	0,1558	0,6884
C2	3	149,22	0	6,657	0,1558	0,8442
C3	3	149,22	0	6,657	0,1558	1,0000
Total		1385,1		42,724		

Jika digambarkan dalam bentuk *roulette wheel*, maka peluang masing-masing individu dapat digambarkan seperti Gambar 3.20 berikut ini:



Gambar 3.20 *Roulette Wheel* Peluang Terpilihnya Individu

Individu dipilih dengan mengacak bilangan random antara 0 dan 1. Dari nilai tersebut, akan dipilh peluang berdasarkan probabilitas kumulatif atau berdasarkan dimana perputaran *roulette wheel* berhenti. Untuk mencegah individu terbaik tereliminasi pada proses seleksi, maka individu terbaik dari kumpulan individu disimpan sebagai individu terpilih untuk generasi selanjutnya. Individu terpilih berdasarkan metode *roulette wheel* adalah sebagai berikut:

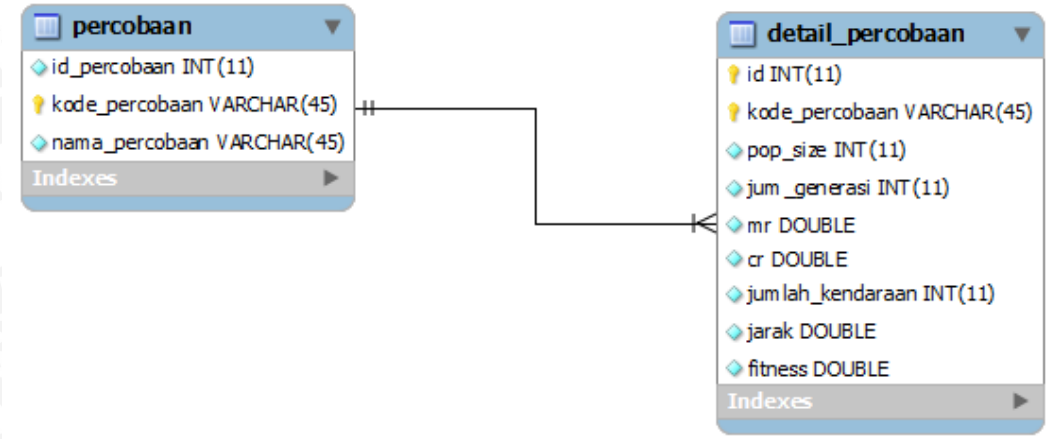
Tabel 3.17 Proses Pemilihan Individu dengan *Roulette Wheel*

Individu	Random	Individu Terpilih	Jum Kendaraan	Tot. Jarak	Penalty	Fitness
1	0,8948	C3	3	149,22	0	6,657
2	0,2658	P2	3	149,22	0	6,657
3	0,0408	P1	3	149,22	0	6,657
4	0,2214	P2	3	149,22	0	6,657
5	0,6641	C1	3	149,22	0	6,657
6	0,6114	C1	3	149,22	0	6,657

Berdasarkan Tabel 3.17, individu terbaik untuk generasi pertama merupakan seluruh individu karena memiliki nilai fitness yang sama yaitu 6,657. Pada permasalahan ini, dimisalkan indiviu C3 terpilih sebagai yang terbaik karena berada pada urutan pertama dari kumpulan individu. Untuk generasi selanjutnya, pembangkitan populasi diambil dari 6 individu terbaik pada seleksi sebelumnya. Pada penelitian ini, pengujian dilakukan sebanyak n generasi kemudian pada tahap akhir, dipilih individu terbaik dari kumpulan generasi tersebut.

3.2.6 Perancangan *Database*

Pada tahap perancangan *database*, data yang disimpan berupa hasil uji coba percobaan agar memudahkan pada proses perhitungan tahap perancangan uji coba dan evaluasi. Desain *database* pada penelitian ini ditunjukkan pada Gambar 3.20 berikut ini:



Gambar 3.21 Desain *Database* Sistem.

Masing-masing entitas yaitu Percobaan dan Detail Percobaan memiliki atribut masing-masing yaitu:

Tabel 3.18 Tabel Percobaan

Nama Atribut	Tipe	Keterangan
kode_percobaan	Varchar	Kode percobaan sebagai *primary key*
nama_percobaan	Varchar	Nama percobaan
id_percobaan	Integer	id dari suatu percobaan

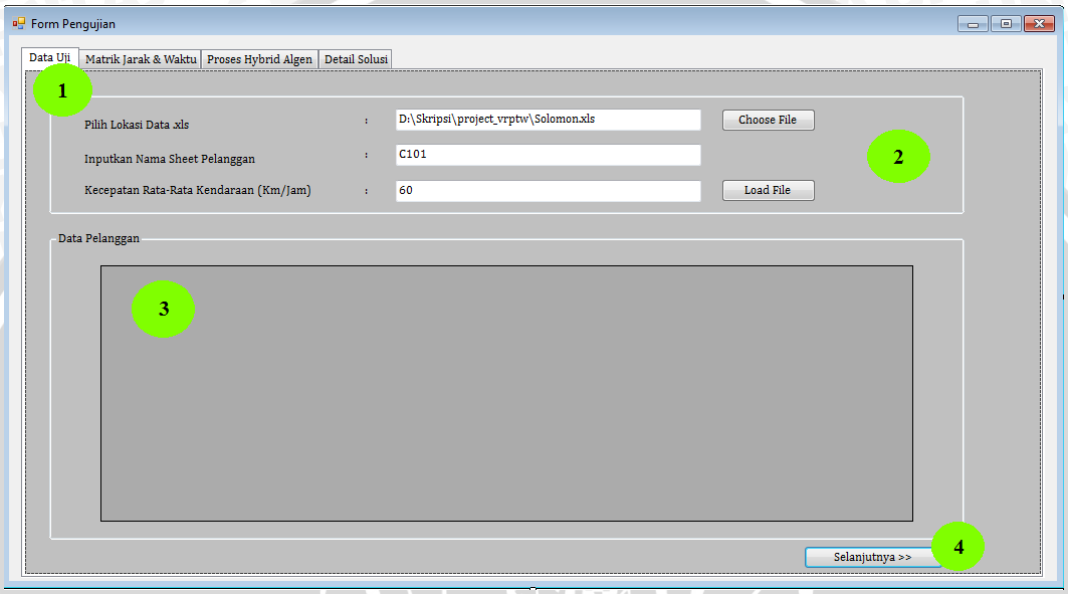
Tabel 3.19 Tabel Detail Percobaan

Nama Atribut	Tipe	Keterangan
id	Integer	ID sebagai *primary key* dan bersifat *auto increment*
kode_percobaan	Varchar	Kode percobaan sebagai *primary key* dan *foreign key*
pop_size	Integer	Ukuran populasi
jum_generasi	Interger	Jumlah generasi
cr	Double	Probabilitas *Crossover*
mr	Double	Probabilitas Mutasi
jumlah_kendaraan	Integer	Jumlah kendaraan yang digunakan
jarak	Double	Total jarak yang ditempuh
fitness	Double	Nilai fitness

3.2.7 Desain *Interface*

Perancangan antarmuka pada permasalahan *vehicle routing problem with time windows* menggunakan metode *hybrid* algoritma genetika terdiri dari proses *load* data uji, proses input parameter VRPTW, proses pengujian *hybrid* algoritma genetika, detail solusi dan data hasil pengujian.

a. Proses *load* data uji



Gambar 3.22 Rancangan Antarmuka *Load* Data Uji.

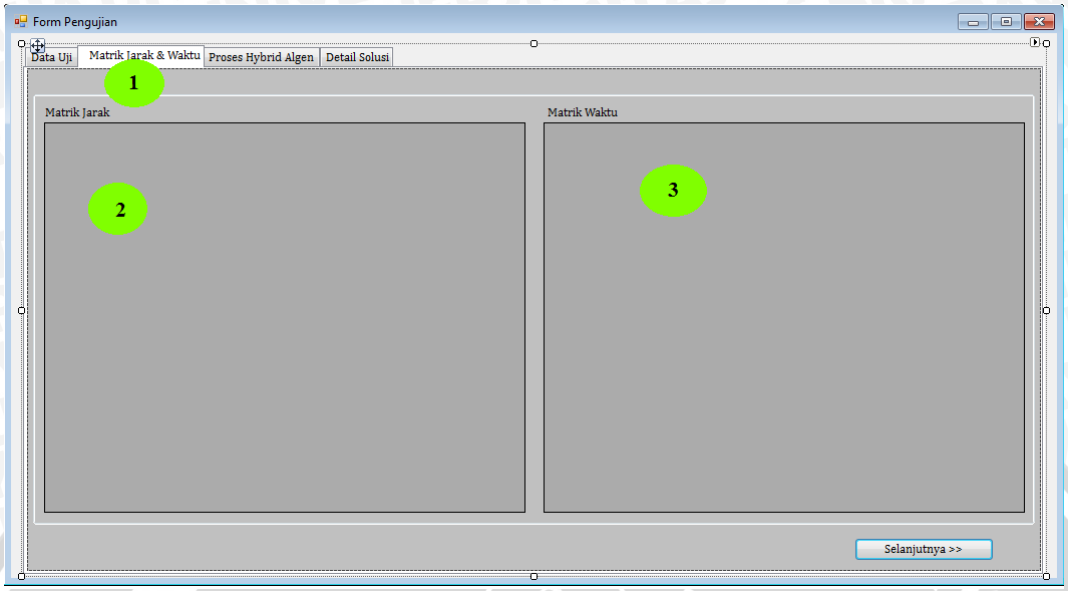
Keterangan Gambar 3.22:

1. *Tab* data uji dimana data pelanggan di inputkan
2. *Groupbox load* data.

   Groupbox ini berisi tentang lokasi data dalam file, nama *sheet* yang diujikan serta salah satu parameter VRPTW yaitu asumsi kecepatan rata-rata kendaraan.
3. *Groupbox* data pelanggan.

   Groupbox ini berisi *datagridview* yang akan menampilkan data pelanggan dan parameter VRPTW seperti jumlah kendaraan dan kapasitas kendaraan.
4. *Button* selanjutnya bertujuan menyimpan data dan menampilkan proses selanjutnya.

b. Matrik Jarak dan Waktu

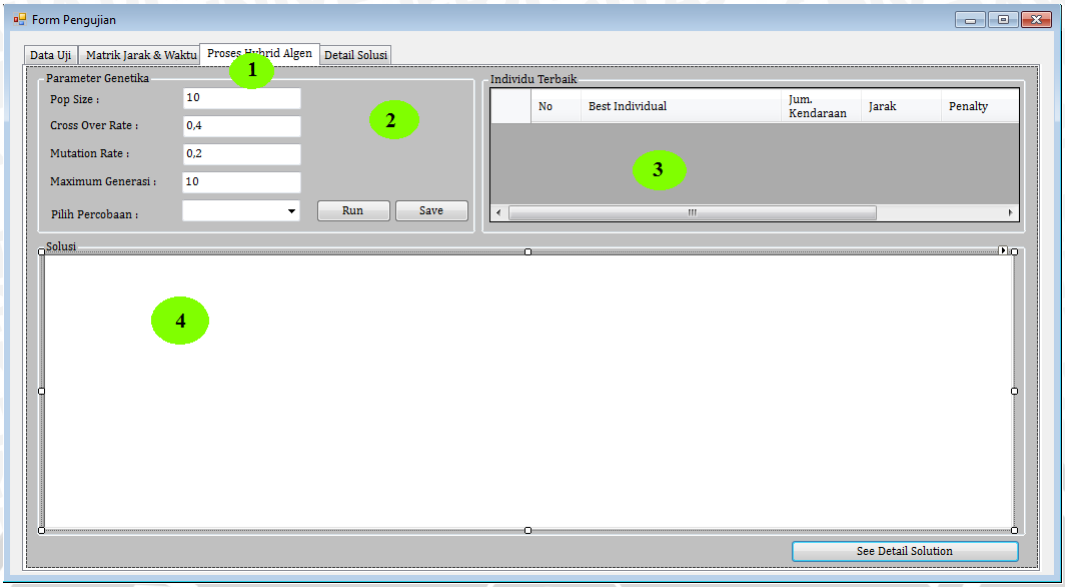


Gambar 3.23 Rancangan Antarmuka Matrik Jarak dan Waktu.

Keterangan Gambar 3.23:

1. *Tab* parameter Matrik Jarak dan Waktu merupakan proses output perhitungan jarak dan waktu antar pelanggan.
2. *Datagridview* matrik jarak berisi informasi tentang jarak antar pelanggan.
3. *Datagridview* matrik waktu berisi tentang waktu tempuh antar pelanggan.
4. *Button* selanjutnya bertujuan menyimpan data dan menampilkan proses selanjutnya.

c. Proses *hybrid* algoritma genetika

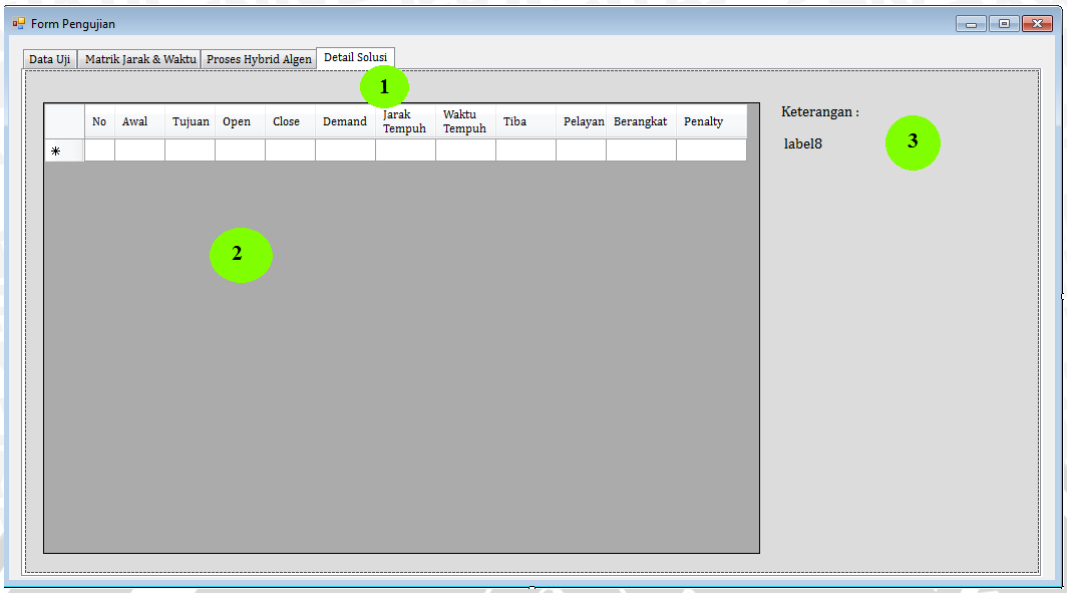


Gambar 3.24 Rancangan Antarmuka Penerapan *Hybrid* Algoritma Genetika.

Keterangan Gambar 3.24:

1. *Tab* proses *hybrid* algoritma genetika berisi proses penerapan metode *hybrid* algoritma genetika pada permasalahan *vehicle routing problem with time windows.*
2. *Group box* Parameter algoritma genetika terdiri dari proses penginputan parameter genetika.
3. *Group box* individu terbaik bertujuan menampilkan individu terbaik dari setiap generasi yang dicantumkan dalam *data grid view*.
4. *Group box* solusi bertujuan menampilkan rincian solusi perhitungan menggunakan metode hybrid algoritma genetika.

d. Detail Solusi

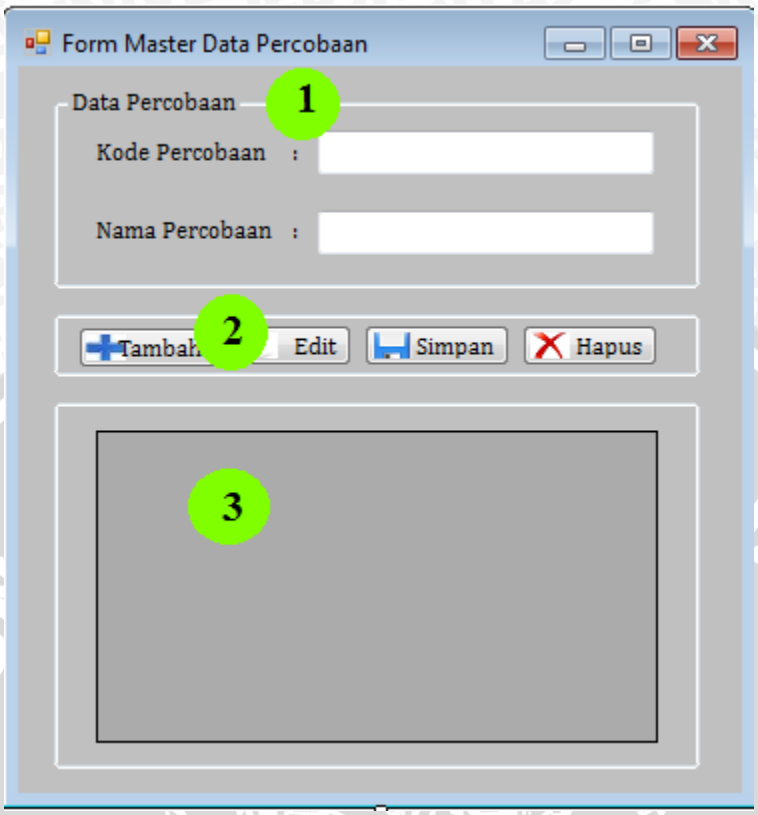


Gambar 3.25 Rancangan Antarmuka Detail Solusi.

Keterangan Gambar 3.25:

1. *Tab* Detail Solusi akan menampilkan proses perhitungan *cost* jarak dan *penalty* dari individu terbaik.
2. Detail solusi dari individu terbaik dicantumkan di dalam *datagridview*.
3. Label pada detail solusi menampilkan keterangan tambahan untuk *the best solution*.

e. Data master percobaan

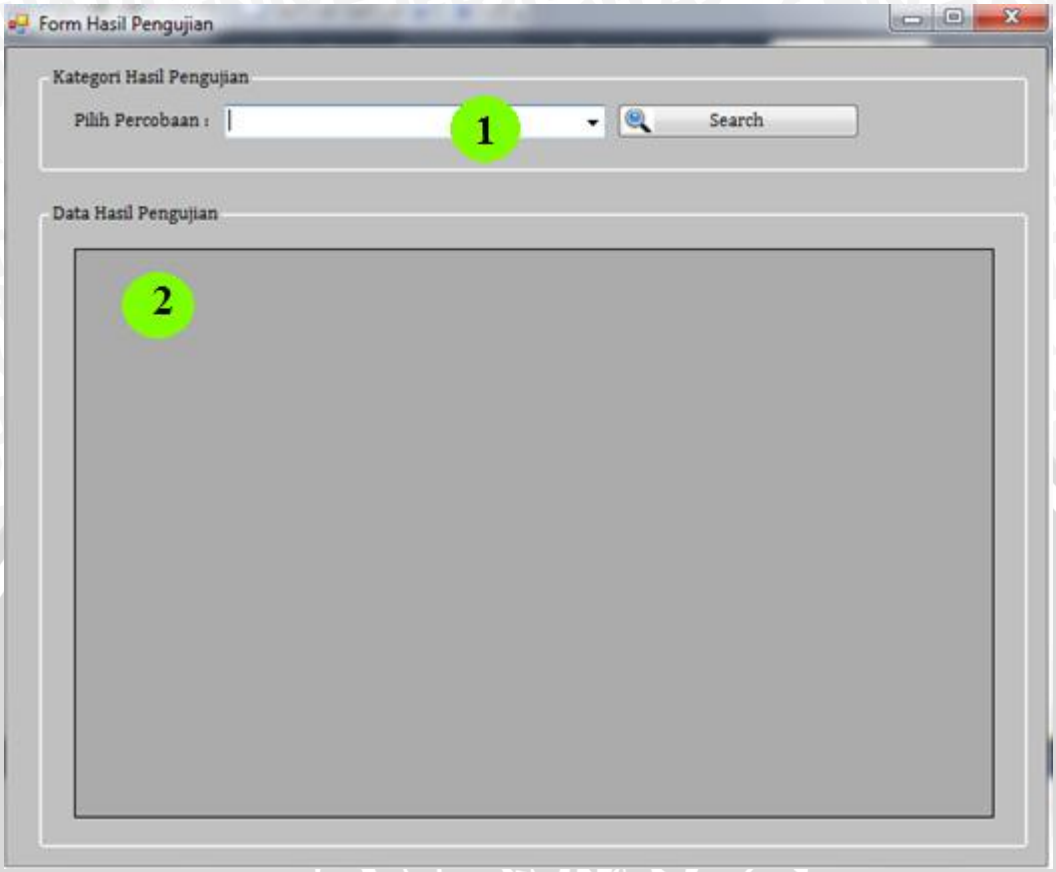


Gambar 3.26 Rancangan Antarmuka Master Data Percobaan.

Keterangan Gambar 3.26:

1. *Group box* data percobaan berisi inputan data percobaan.
2. Kumpulan *button function* yang bertujuan menambah, mengubah, menyimpan dan menghapus data.
3. *Datagridview* yang menampilkan master data percobaan.

f. Hasil Pengujian



Gambar 3.27 Rancangan Antarmuka Hasil Pengujian.

Keterangan Gambar 3.27:

1. *Group box* kategori hasil pengujian bertujuan memilih kategori pengujian yang akan dicari.
2. *Group box* data hasil pengujian bertujuan menampilkan data hasil pengujian ke dalam *datagridview* sesuai dengan katagori yang dipilih.

3.2.8 Perancangan Uji Coba dan Evaluasi

Pada permasalahan ini, proses perancangan uji coba dan evaluasi dilakukan dengan menggunakan kasus Solomon dengan tipe data C1, C2, R1, R2, RC1, dan RC2. Dari masing-masing tipe data tersebut, akan diambil 2 set data uji yang kemudian akan dibandingkan dengan *best solution of Solomon*.

Metode hybrid algoritma genetika memiliki beberapa parameter yang mempengaruhi solusi seperti *crossover rate* (P_c), *mutation rate* (P_m), jumlah

populasi, dan jumlah generasi. Pengujian dilakukan dengan menguji parameter genetika dengan menginputkan nilai yang bervariasi kemudian mengkombinasikan parameter genetika untuk mengetahui keterkaitan antara parameter tersebut. Hal ini bertujuan untuk mengetahui pengaruh parameter genetika untuk pencarian solusi optimum pada permasalahan VRPTW.

Paremeter yang digunakan pada pengujian tahap 1 yaitu pengaruh probabilitas *crossover* dan mutasi terhadap pencarian solusi. Total persentasi pengujian untuk masing-masing nilai probabilitas *crossover* dan mutasi yang diujikan adalah 100%. Sedangkan untuk penggunaan parameter lainnya digunakan nilai yang sama sesuai dengan yang akan ditetapkan. Nilai fitness yang diambil dari seluruh tahap percobaan ini adalah nilai rata-rata fitness dari 10 kali percobaan untuk tiap-tiap nilai parameter. Tabel 3.20 berikut ini menunjukkan rancangan pengujian dari pengaruh probabilitas *crossover* dan mutasi terhadap pencarian solusi dan fitness yang didapatkan:

Tabel 3.20 Rancangan Pengujian Pengaruh p_c dan p_m terhadap Fitness Rata-Rata

No	Prob. Crossover	Prob. Mutation	Fitness Rata-Rata
1	0%	1	
2	10%	90%	
3	20%	80%	
4	30%	70%	
5	40%	60%	
6	50%	50%	
7	60%	40%	
8	70%	30%	
9	80%	20%	
10	90%	10%	
11	100%	0%	

Rancangan pengujian tahap 2 dilakukan untuk mengetahui pengaruh ukuran populasi terhadap pencarian solusi dan nilai *fitness* yang dihasilkan. Nilai dan parameter probabilitas *crossover* dan mutasi yang digunakan adalah nilai terbaik yang didapatkan dari pengujian tahap sebelumnya sedangkan untuk parameter lainnya menggunakan nilai yang sama sesuai dengan yang

ditetapkan.Tabel 3.21 berikut ini menunjukkan rancangan pengujian dari pengaruh ukuran populasi terhadap pencarian solusi dan fitness yang didapatkan:

Tabel 3.21 Rancangan Pengujian Pengaruh Ukuran Populasi Terhadap Fitness Rata-Rata

No	Ukuran Populasi	Fitness Rata-Rata
1	10	
2	30	
3	60	
4	90	
5	120	
...	...	
N	N	

Rancangan pengujian tahap 3 dilakukan untuk mengetahui pengaruh banyaknya generasi terhadap pencarian solusi dan nilai *fitness* yang dihasilkan. Nilai dan parameter probabilitas *crossover*, probabilitas mutasi dan ukuran populasi yang digunakan adalah nilai terbaik yang didapatkan dari pengujian tahap sebelumnya. Tabel 3.22 berikut ini menunjukkan rancangan pengujian dari pengaruh banyaknya generasi terhadap pencarian solusi dan fitness yang didapatkan:

Tabel 3.22 Rancangan Pengujian Pengaruh Banyaknya Generasi Terhadap Fitness Rata-Rata

No	Banyaknya Generasi	Rata-Rata Nilai Fitness
1	10	
2	50	
3	100	
...	...	
N	N	

Rancangan pengujian tahap 4 dilakukan untuk membandingkan solusi penggunaan metode *hybrid* algoritma genetika dengan *best known* Solomon. Tahap ini bertujuan untuk mengetahui sejauh mana keberhasilan sistem terhadap

permasalahan VRPTW. Parameter yang digunakan sebagai acuan keberhasilan sistem adalah Relative Percentage Deviation (RPD) atau derajat kesalahan. Proses perhitungan RPD yaitu:

$$RPD = \frac{BestTestVRPTW - BestKnownSolomon}{BestKnownSolomon} \; x \; 100\% \qquad (3.2)$$

Tabel 3.23 berikut ini menunjukkan rancangan pengujian perbandingan *best test* dan *best known* Solomon:

Tabel 3.23 Rancangan Pengujian Perbandingan *Best Test* Hybrid Algoritma Genetika dan *Best Known* Solomon

No	Problem	Best Known Solomon			Algoritma Genetika		RPD (%)
		NV	Distance	Authors	NV	Distance	
1							
2							
3							
4							
5							
						Rata - Rata RPD	

NV = *Number Of Vehicles*

BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Pada bab ini, akan dibahas mengenai implementasi perangkat lunak berupa penerapan *hybrid* algoritma genetika untuk permasalahan VRPTW. Adapun lingkungan implementasi perangkat keras (*hardware*) dan perangkat lunak (*software*) yang digunakan adalah:

4.1.1 Lingkungan Perangkat Keras

Spesifikasi perangkat keras yang digunakan dalam perancangan sistem ini adalah sebagai berikut:

1. *Processor Intel® Core™ 2 Duo* 2,26 *Ghz*
2. *RAM* 4096 *MB*
3. *Harddisk* 300 GB
4. *Monitor* 14*".*
5. *Mouse*
6. *Keyboard*

4.1.2 Lingkungan Perangkat Lunak

Adapun spesifikasi perangkat lunak yang digunakan pada penelitian ini adalah:

1. *Operating System Windows 7 64bit*.
2. *Visual Studio 2010*
3. Bahasa Pemrograman C#
4. Microsoft Office 2013
5. Database MySql
6. Microsoft Visio 2007

4.2 Implementasi Program

Pada subbab ini, akan dibahas implementasi berdasarkan perancangan sistem yang dilakukan pada bab sebelumnya yaitu implementasi penerapan *hybrid* algoritma genetika untuk permasalahan VRPTW. Adapun kelas yang dibangun pada program ini dapat dilihat pada Tabel 4.1.

Tabel 4.1 Kelas-kelas dalam Program

Kelas	Deskripsi Kelas
Program.cs	Kelas ini bertujuan menjalankan atau *running* program.
Menu_Utama.cs	Kelas ini mengimplementasikan tampilan awal dari antar muka yang bertujuan memanggil kelas-kelas lainnya.
Form_Master_Percobaan.cs	Kelas ini menyimpan master data dari percobaan yang akan dilakukan
Form_Hasil_Pengujian.cs	Kelas ini bertujuan menampilkan hasil dari pengujian sesuai dengan kategori percobaan.
Inisiasi.cs	Kelas ini bertujuan untuk membangkitkan solusi awal atau *parent* pada proses algoritma genetika
Hitung_Fitness.cs	Kelas ini bertujuan untuk melakukan perhitungan nilai fitness pada setiap individu.
Koneksi.cs	Kelas ini menyimpan data *connection string* yang bertujuan membuat koneksi antara program dengan database MySql
Form.Pengujian.cs	Kelas ini mengimplementasikan proses perhitungan *hybrid* algoritma genetika terhadap sejumlah data uji Solomon

4.2.1 Implementasi Kelas Koneksi

Kelas koneksi merupakan tempat penyimpanan data *connection string* yang bertujuan membuat koneksi antara program dengan database MySql. Adapun *listing code* kelas koneksi dapat dilihat pada *source code* 4.1 berikut:

```
1   string                          strConnString                   =
2   "Server=localhost;Port=3306;UID=root;PWD=;Database=db_vrptw;Allow
3   Zero Datetime=true";
4
5   publicstring connString
6           {
7   get { return strConnString; }
8   set { strConnString = value; }
9           }
```

Source Code 4.1 *Listing Code* Kelas Koneksi

Penjelasan *source code* 4.1 adalah:

1. Baris 1-3 menjelaskan variabel *string connection* antara program dan database MySql.
2. Baris 5-9 menjelaskan *method* connString yang digunakan untuk mengambil data koneksi.

4.2.2 Implementasi Proses Penyimpanan Master Data

Pada tahap ini, dilakukan proses penyimpanan data master percobaan pada database MySql. Data yang disimpan yaitu berupa kode percobaan dan nama percobaan itu sendiri. Adapun *listing code* dapat dilihat pada *source code* 4.2 berikut:

```
1           MySqlConnection sqlConn;
2   koneksi kon = newkoneksi();
3
4   public Form_Master_Percobaan()
5           {
6               sqlConn = newMySqlConnection(kon.connString);
7               InitializeComponent();
8           }
9
10  privatevoid btnSimpan_Click(object sender, EventArgs e)
11          {
12              strKode = txtKodePercobaan.Text;
13              strNama = txtNamaPercobaan.Text;
14
15              sqlConn.Open();
16  string sql = "insert into percobaan(kode_percobaan, nama_percobaan)
```

```
values (?kode_percobaan, ?nama_percobaan)";
MySqlCommand cmd = newMySqlCommand(sql, sqlConn);
            cmd.Parameters.Add("?kode_percobaan", strKode);
            cmd.Parameters.Add("?nama_percobaan", strNama);
            cmd.Prepare();
            cmd.ExecuteNonQuery();
            sqlConn.Close();

MessageBox.Show("Tambah Data Percobaan Sukses");
        }
```

Source Code 4.2 *Listing Code* Proses Penyimpanan Master Data

Penjelasan *source code* 4.2 adalah:

1. Baris 1-2 menjelaskan pemanggilan kelas koneksi dan inisialisasi *mysqlconnection* yang merupakan *library* yang disediakan untuk koneksi program dengan database.
2. Baris 15 menjelaskan proses *open connection* pada database MySql.
3. Baris 16-17 menjelaskan *query* yang akan dijalankan yaitu insert data percobaan pada database.
4. Baris 15-22 menjelaskan perintah pada database dalam menjalankan *query* yang diminta.
5. Baris 23 menjelaskan proses *close* koneksi pada database.

4.2.3 Implementasi proses *Load* Data

Proses *load* data pelanggan dilakukan pada *class* Form_Pengujian.cs. pada tahap ini data yang di *load* adalah data pelanggan yang diambil dari *microsoft excel* dengan format *.xls*. Adapun *listing code load* data dapat dilihat pada *source code* 4.3 berikut:

```
privatevoid btnLoadFile_Click(object sender, EventArgs e)
        {
string pathConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
txtPath.Text + ";Extended Properties=\"Excel 8.0;HDR=Yes;\";";
OleDbConnection conn = newOleDbConnection(pathConn);
OleDbDataAdapter myDataAdapter = newOleDbDataAdapter("select * from
[" + txtSheetDataUji.Text + "$]", conn);
DataTable dt = newDataTable();
            myDataAdapter.Fill(dt);
            dgvPelanggan.DataSource = dt;
        }
```

Source Code 4.3 *Listing Code* Proses *Load* Data

Penjelasan *source code* 4.3 adalah:

1. Baris 3-5 merupakan data koneksi antara program dan data *excel*.
2. Baris 6-9 merupakan proses pemanggilan data dari *excel* ke program.
3. Baris 10-13 merupakan proses membaca data dan menampilkan data ke dalam *data grid view*.

4.2.4 Implementasi Proses Inisialisasi

Kelas inisialisasi merupakan proses pembangkitan solusi awal pada proses algoritma genetika. Pembangkitan solusi awal dilakukan dengan dua tahap yaitu 50% populasi berdasarkan *nearest insertion heuristic* dan 50% populasi yang diambil secara acak. Tabel 4.2 dibawah ini memaparkan method pembentuk kelas inisialisasi:

Tabel 4.2 Method class Inisialisasi

Method	Deskripsi *Method*
Inisiasi()	Method ini merupakan konstruktor yang bertujuan mengambil data dari *class* Form_Pengujian seperti data pelanggan, *pop_size*, dan *service time*.
InisialisasiAwal()	Melakukan proses inisialisasi awal
nearestInsertion()	Melakukan proses inisialisasi awal berdasarkan perhitungan *nearest insertion heuristic*.
kromosomRandom()	Melakukan proses inisialisasi awal secara acak tetapi tetap dengan mempertimbangkan kendala kapasitas dan *time windows* pelanggan.
nodeTerdekatDepot()	Mencari *node* terdekat dari depot

Adapun *listing code* pada proses pembangkitan solusi awal dengan *nearest insertion heuristic* dapat dilihat pada *source code* 4.4 berikut:

```
1   publicvoid nearestInsertion()
2           {
3   ArrayList tempGen;
4               tempKromosomResult = newArrayList();
5
6               x = 0;
7   while (nodeBebas.Count != 0)
8               {
9                   tempGen = newArrayList();
10                  nodeRemove = newArrayList();
11  int node0, nodeI, nodeU;
12                  node0 = 0;
13
14                  nodeTerdekatDepot2();
15                  nodeI = Convert.ToInt32(tempTerdekat);
16                  tempGen.Add(nodeI.ToString());
17                  nodeBebas.Remove(nodeI.ToString());
18
19  ArrayList z1 = newArrayList();
20  ArrayList z2 = newArrayList();
21  double tempZ1n, tempZ2n, tempZ1, tempZ2;
22  double diu, du0, di0;
23  double tiu, tu0, ti0;
24
25  for (int j = 0; j < nodeBebas.Count; j++)
26                  {
27                      nodeU = Convert.ToInt32(nodeBebas[j].ToString());
28                      diu = matrikJarak[nodeI, nodeU];
29                      du0 = matrikJarak[nodeU, node0];
30                      di0 = matrikJarak[nodeI, node0];
31                      tempZ1n = (diu + du0) - di0;
32
33                      tiu = matrikWaktu[nodeI, nodeU];
34                      tu0 = matrikWaktu[nodeU, node0];
35                      ti0 = matrikWaktu[nodeI, node0];
36                      tempZ2n = (tiu + tu0 + serviceTime) - ti0;
37
38                      tempZ1 = (0.9 * tempZ1n) + (0.1 * tempZ2n);
39                      z1.Add(tempZ1.ToString());
40                      tempZ2 = du0 - tempZ1;
41                      z2.Add(tempZ2.ToString());
42                  }
43                  cekOngkosPenyisipanTerkecil();
44                  evaluasiPenyisipan();
45              }
46          }
```

Source Code 4.4 *Listing Code* Proses Pembangkitan Solusi dengan *Nearest Insertion Heuristic*.

Penjelasan *source code* 4.4 adalah:

1. Baris 3 menyimpan gen sementara pada suatu rute kendaraan
2. Baris 4 menyimpan hasil dari proses pembangkitan solusi dengan *nearest insertion heuristic*.
3. Baris 7 menunjukkan proses *nearest insertion heuristic* akan terus dilakukan selama *node* bebas yang belum dimasukkan kedalam rute masih ada.
4. Baris 9-23 merupakan proses penentuan *node*-0, *node*-i, *node*-u dan pembentukan variabel *nearest insertion heuristic*.
5. Baris 25-42 merupakan proses perhitungan penghematan jarak dan waktu tempuh.
6. Baris 43 merupakan *method* pemilihan *node* dengan *cost* terkecil dari proses perhitungan penghematan jarak dan waktu.
7. Baris 44 merupakan *method* evaluasi penyisipan *node* pada suatu kendaraan.

Sedangkan *listing code* pada proses pembangkitan solusi awal secara acak dapat dilihat pada *source code* 4.5 berikut:

```
1   publicvoid kromosomRandom()
2           {
3               int valRandom1, valRandom2;
4   for (int i = valPopSize/2; i < valPopSize; i++)
5               {
6                   x = 0;
7   while (x < 100)
8                   {
9                       valRandom1 = r.Next(0, totGen / 2);
10                      valRandom2 = r.Next(totGen / 2, totGen);
11
12  string temp = "";
13  if (valRandom1 != valRandom2)
14                      {
15                          temp = tempKromosom[valRandom2].ToString();
16                          tempKromosom[valRandom2] =
17  tempKromosom[valRandom1];
18                          tempKromosom[valRandom1] = temp;
19                      }
20                      x++;
21                  }
22  for (int j = 0; j < totGen; j++)
23                  {
24                      kromosomParents[i, j] =
```

```
25  tempKromosom[j].ToString();
26                  }
27              }
28          }
```

Source Code 4.5 *Listing Code* Proses Pembangkitan Solusi dengan *Nearest Insertion Heuristic*.

Penjelasan *source code* 4.5 adalah:

1. Baris 7-21 merupakan proses acak gen yang dilakukan pada kromosom dengan cara menukarkan posisi setiap gen yang ditandai dengan pemilihan 2 posisi secara random yaitu variabel valRandom1 dan valRandom2.
2. Baris 22-26 merupakan proses penyimpanan data individu dari proses acak yang dilakukan.

4.2.5 Implementasi Proses Hitung Fitness

Proses perhitungan *fitness* dilakukan pada kelas Hitung_Fitness.cs. Seperti dijelaskan pada bab sebelumnya, parameter yang mempengaruhi nilai fitness adalah *penalty* dan total jarak. *Source code* 4.6 berikut ini menunjukkan *listing code* pada proses hitung fitness.

```
1  publicvoid cariNilaiFitness()
2          {
3              tempFitness = (100000 / ((totalPenalti) + tempTotJarak
4  + 1));
           }
```

Source Code 4.6 *Listing Code* Proses Hitung Fitness.

Penjelasan *source code* 4.6 adalah:

1. Baris 1-4 Merupakan proses perhitungan fitness dimana fitness didapatkan dari 100 ribu dibagi dengan total *penalty* dan total jarak.

Listing Code untuk proses menghitung total *penalty* dan total jarak pada suatu individu dapat dilihat pada *Source Code* 4.7 berikut:

```
1   publicvoid hitungPenaltyDanJarak()
2           {
3               totalPenalti = 0;
4   int tempTotData = tempRute.Count;
5   int readyTimeDepot = readyTime[0];
6   int dueDateDepot = dueDate[0];
7               tempTotJarak = 0;
8   int waktuDatang = 0, waktuBerangkat = readyTimeDepot, waktuTempuh =
9   0, tempReadyTime, tempDueDate;
10              lokasiAwal = 0;
11              tempjumlahKendaraan = 1;
12  for (int i = 0; i < tempTotData; i++){
13  if (tempRute[i] == "0"){
14                      lokasiTujuan = 0;
15                      tempTotJarak += matrikJarak[lokasiAwal,
16  lokasiTujuan];
17  if (waktuBerangkat > dueDateDepot){
18                              totalPenalti += waktuBerangkat -
19  dueDateDepot;
20                              tempPenal = waktuBerangkat - dueDateDepot;}
21  else{tempPenal = 0;}
22                      waktuDatang = waktuBerangkat +
23  matrixWaktu[lokasiAwal, lokasiTujuan];
24                      tempjumlahKendaraan += 1;
25                      readyTimeDepot = readyTime[0];
26                      dueDateDepot = dueDate[0];
27                      waktuBerangkat = readyTimeDepot;
28                  lokasiAwal = 0;}
29  else{
30                      lokasiTujuan =
31  Convert.ToInt32(tempRute[i].ToString());
32                      waktuTempuh = matrixWaktu[lokasiAwal,
33  lokasiTujuan];
34                      waktuDatang = waktuBerangkat + waktuTempuh;
35                      tempReadyTime = readyTime[lokasiTujuan];
36                      tempDueDate = dueDate[lokasiTujuan];
37  if (waktuDatang < tempDueDate){
38  if (waktuDatang > tempReadyTime){
39                              waktuBerangkat = waktuDatang +
40  serviceTime;
41                              totalPenalti += 0;
42                      tempPenal = 0;}
43  else{                               waktuBerangkat = tempReadyTime +
44  serviceTime;
45                              totalPenalti += 0;
46                  tempPenal = 0;}}
47  else{
48                              waktuBerangkat = waktuDatang + serviceTime;
49                              totalPenalti += waktuBerangkat - tempDueDate;
50                              tempPenal = waktuBerangkat - tempDueDate;}
51                      tempTotJarak += matrikJarak[lokasiAwal,
52  lokasiTujuan];
53                      lokasiAwal =
54  Convert.ToInt32(tempRute[i].ToString());
55                  }}}
```

Source Code 4.7 *Listing Code* Proses Hitung Penalty dan Total Jarak.

Penjelasan *source code* 4.7 adalah:

1. Baris 3-10 adalah variabel awal untuk perhitungan *penalty* dan jarak.
2. Baris 13-28 merupakan proses perhitungan kendaraan berakhir menuju depot. Pada representasi kromosom, terdapat nilai 0 yang menunjukkan batas akhir pelayanan suatu kendaraan.
3. Baris 29-55 merupakan proses perhitungan *penalty* dan total jarak sesuai dengan kendala waktu *time windows* yaitu jika kendaraan datang diluar interval waktu pelayanan. Baris 37 menunjukkan kendala jika kendaraan datang sebelum jam buka pelanggan dan baris 47 menunjukkan kendala jika kendaraan datang terlambat atau melebihi jam pelayanan pelanggan.

4.2.6 Implementasi Proses *Crossover*

Proses *crossover* yang dilakukan pada penelitian ini yaitu *one cut point crossover*. Proses *crossover* dilakukan pada kelas Form_Pengujian.cs. Adapun *Listing Code* pada proses ini dapat dilihat pada *source code* 4.8:

```
1   publicvoid crossover()
2           {
3               noParent1 = r.Next(0, valPopSize);
4               noParent2 = r.Next(0, valPopSize);
5               tempKromosom1 = newArrayList();
6               tempKromosom2 = newArrayList();
7               tempKromosomResult = newArrayList();
8
9   if (noParent1 != noParent2){
10  for (int i = 0; i < kromosom[noParent1].Length; i++){
11                      tempKromosom1.Add(kromosom[noParent1][i]);
12                  }
13  for (int i = 0; i < kromosom[noParent2].Length; i++){
14  if (kromosom[noParent2][i] != "0"){
15                          tempKromosom2.Add(kromosom[noParent2][i]);
16                      }}
17                  valRandom1                  =                   r.Next(2,
18  Math.Min((kromosom[noParent1].Length                -                   2),
19  (kromosom[noParent2].Length - 2)));
20                  tempKromosomResult = newArrayList();
21                  tempGen = newArrayList();
22                  tempRecycle = newArrayList();
23                  remainDemand = kapasitasKendaraan;
24  for (int i = 0; i < valRandom1; i++){
25  if (tempKromosom1[i] == "0"){
26                          remainDemand = kapasitasKendaraan;
27                          tempGen = newArrayList();
28                      }
29  else{
```

```
30                          remainDemand      =      remainDemand      -
31  demand[Convert.ToInt32(tempKromosom1[i])];
32                          tempGen.Add(tempKromosom1[i]);
33                      }
34                      tempKromosomResult.Add(tempKromosom1[i]);
35                  }
36  int cekPenalty1 = 0, cekPenalty2 = 0;
37  int tempVal = 0;
38  for (int i = 0; i < tempKromosom2.Count; i++){
39                      tempVal = 0;
40  for (int j = 0; j < tempKromosomResult.Count; j++){
41  if (tempKromosom2[i].ToString() == tempKromosomResult[j].ToString()){
42                              j = tempKromosomResult.Count;
43                              tempVal = 1;
44                          }}
45  if (remainDemand == 0){
46  if (tempKromosomResult[tempKromosomResult.Count - 1] != "0")
47                          {
48                              tempKromosomResult.Add("0");
49                          }
50                          remainDemand = kapasitasKendaraan;
51                          tempGen = newArrayList();
52                      }
53  if (tempVal == 0){
54  int tempTujuan = Convert.ToInt32(tempKromosom2[i]);
55  if (remainDemand >= demand[tempTujuan]){
56                              remainDemand     =     remainDemand     -
57  demand[tempTujuan];
58
59  //Penalty sebelum diinsertkan
60                              cekPenalty1 = cekPenalty;
61
62  //Penalty sesudah diinsertkan
63                              tempGen.Add(tempTujuan.ToString());
64                              cekPenalty2 = cekPenalty;
65
66  if (cekPenalty2 > cekPenalty1){
67                                  remainDemand += demand[tempTujuan];
68
69  tempGen.Remove(tempTujuan.ToString());
70  tempRecycle.Add(tempTujuan.ToString())
71                              }
72  else{
73
74  tempKromosomResult.Add(tempTujuan.ToString());
75                              }}}}}}
76
77
78
```

Source Code 4.8 *Listing Code* Proses *Crossover*.

Penjelasan *source code* 4.8 adalah:

1. Baris 10-16 merupakan proses penyimpanan *parent* terpilih untuk proses *crossover* pada variabel sementara yaitu kendaraan1 dan kendaraan2.

2. Baris 17-19 merupakan proses pemilihan titik potong *crossover*.
3. Baris 24-35 merupakan proses pengambilan rangkaian kromosom yang telah dipotong dan disimpan pada kromosom hasil.
4. Baris 38-78 merupakan proses pengambilan gen yang ada pada kendaraan 2 yang belum tersimpan di kromosom hasil. Pada tahap ini penyisipan gen pada kromosom hasil dilakukan dengan memperhatikan kapasitas kendaraan dan *penalty* yang dihasilkan. Kendala ini bisa dilihat pada *code* baris 58 dan 69.

4.2.7 Implementasi Proses *Mutasi*

Proses mutasi yang dibahas pada implementasi ini adalah *reciprocal exchange mutation* gen kendaraan. Proses mutasi dilakukan pada kelas Form_Pengujian.cs. Adapun *Listing Code* pada proses ini dapat dilihat pada *source code* 4.9:

```
1   valRandom1 = 0;
2               valRandom2 = 0;
3
4   while (valRandom1 == valRandom2)
5               {
6                   valRandom1 = r.Next(0, tempGenKendaraan1.Count);
7                   valRandom2 = r.Next(0, tempGenKendaraan1.Count);
8               }
9
10  string tempString;
11              tempString = tempGenKendaraan1[valRandom1].ToString();
12              tempGenKendaraan1[valRandom1] =
13  tempGenKendaraan1[valRandom2];
14              tempGenKendaraan1[valRandom2] = tempString;
```

Source Code 4.9 *Listing Code* Proses Mutasi.

Penjelasan *source code* 4.9 adalah:

1. Baris 1-8. Pada tahap mutasi sebelumnya, telah dipilih *parent* secara acak. Kemudian dari *parent* tersebut dipilih 1 kendaraan secara acak untuk dilakukan proses mutasi gen pada kendaraan. Baris 1-8 tersebut merupakan proses pemilihan nilai *exchange point*.
2. Baris 10-14 merupakan proses penukaran gen yang ada pada kendaraan tersebut.

4.2.8 Implementasi Proses Seleksi dengan *Roulette Wheel*

Proses seleksi pada tahap implementasi ini dilakukan dengan proses *roulette wheel*. Adapun *Listing Code* pada proses ini dapat dilihat pada *Source Code* 4.10.

```
1   totFitness = 0;
2   for (int i = 0; i < totPopSize; i++)
3               {
4                   totFitness += fitness[i];
5               }
6               tempProb = 0;
7               tempProbKum = 0;
8
9               prob = newdouble[totPopSize];
10              probKum = newdouble[totPopSize];
11
12  for (int i = 0; i < totPopSize; i++)
13              {
14                  tempProb = fitness[i] / totFitness;
15              prob[i] = tempProb;
16                  tempProbKum += tempProb;
17                  probKum[i] = tempProbKum;
18              }
```

Source Code 4.10 *Listing Code* Proses Seleksi dengan *Roulette Wheel*.

Penjelasan *source code* 4.10 adalah:

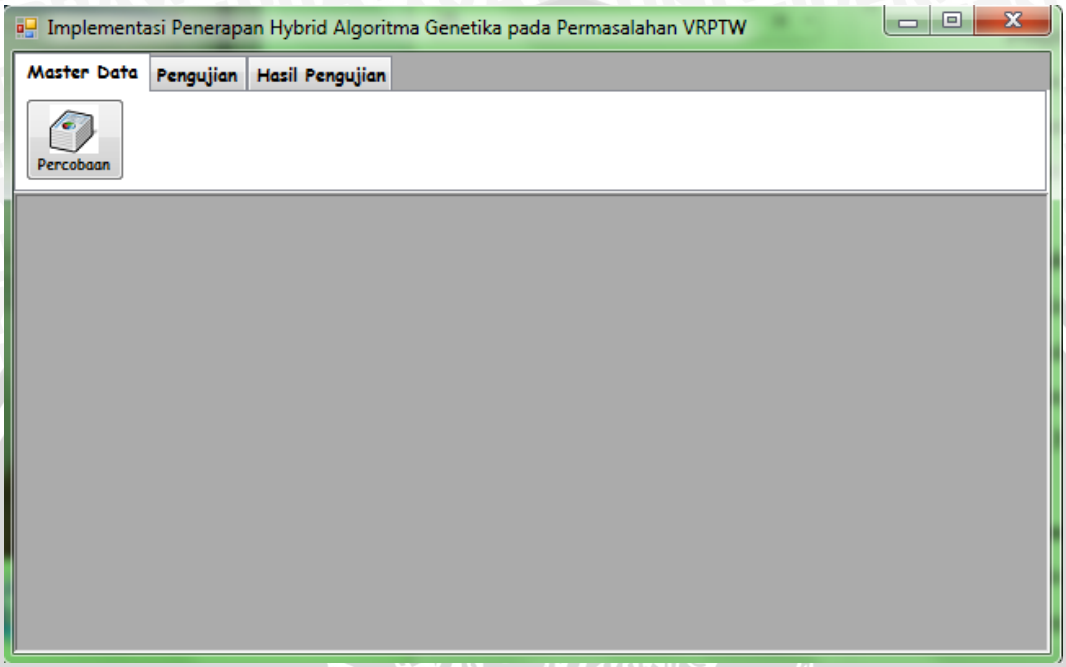
1. Baris 1-5 merupakan proses perhitungan total fitness seluruh individu baik *parent* dan *child*.
2. Baris 9-18 merupakan proses perhitungan probabiltas dan probabilitas kumulatif.

4.3 Implementasi Antarmuka

Implementasi antarmuka merupakan hasil implementasi perancangan yang telah dilakukan pada tahap sebelumnya yaitu perancangan antarmuka sistem. Adapun tampilan antarmuka terdiri dari Menu Utama dan Sub Menu seperti Percobaan, Pengujian dan Hasil.

4.3.1 Menu Utama

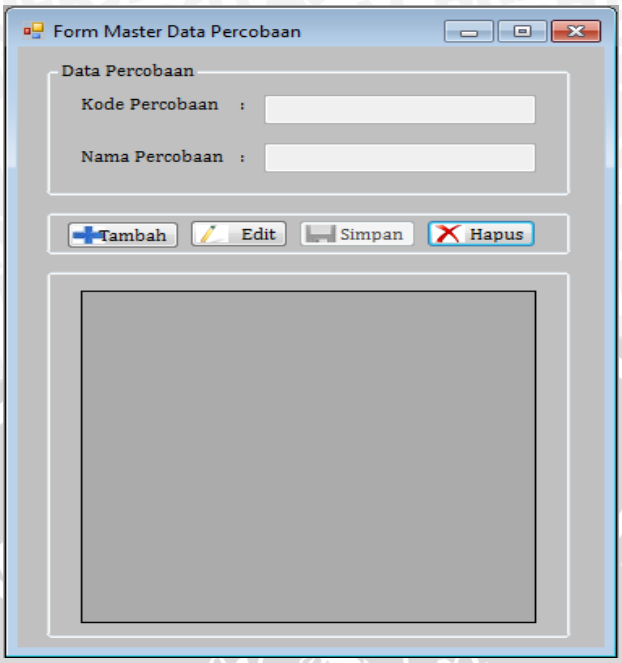
Pada menu utama terdapat beberapa sub menu yang disediakan yaitu Master Data, Pengujian, dan Hasil Pengujian. Tampilan Menu Utama dari sistem ini dapat dilihat pada Gambar 4.1 berikut:



Gambar 4.1 Menu Utama.

4.3.2 Master Data Percobaan

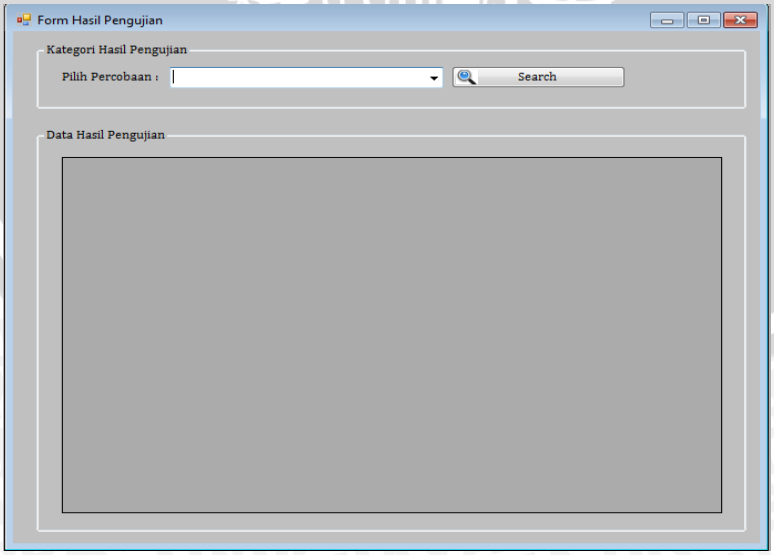
Antar muka master data percobaan digunakan sebagai proses tambah, *update*, dan hapus data master percobaan. Master Data Percobaan ditunjukkan pada Gambar 4.2 berikut:



Gambar 4.2 Master Data Percobaan.

4.3.3 Hasil Pengujian

Hasil pengujian dari penerapan proses *hybrid* algoritma genetika ditampilkan pada antarmuka hasil pengujian. Pencarian hasil pengujian dipilih berdasarkan kategori percobaan. Gambar 4.3 berikut menunjukkan antarmuka dari hasil pengujian.



Gambar 4.3 Master Data Percobaan.

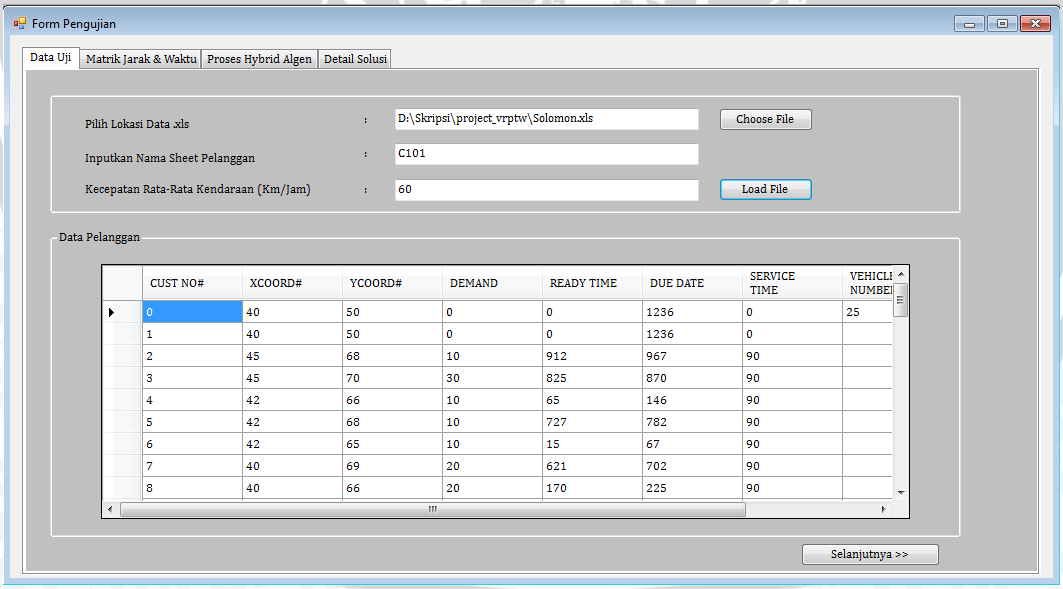
4.3.4 Pengujian

Sub menu pengujian merupakan antarmuka dimana proses perhitungan penerapan *hybrid* algoritma genetika untuk permasalahan VRPTW dilakukan. Antarmuka pengujian terdiri dari 4 *tab* control yaitu Data Uji, Matriks Jarak dan Waktu, Proses Hybrid Algen, dan Detail Solusi.

Pada tahap pertama yaitu pada *tab* Data Uji, proses yang dilakukan adalah

1. Pilih lokasi data bertipe .xls dengan klik *button choose* file.
2. Inputkan nama sheet tempat data pelanggan tersebut disimpan didalam file excel.
3. Inputkan asumsi kecepatan rata-rata yang digunakan pada permasalahan ini.
4. *Button* Selanjutnya merupakan *link* menuju proses berikutnya yaitu *tab* Matrik Jarak dan Waktu dimana perhitungan jarak tempuh dan waktu tempuh antar pelanggan dilakukan.

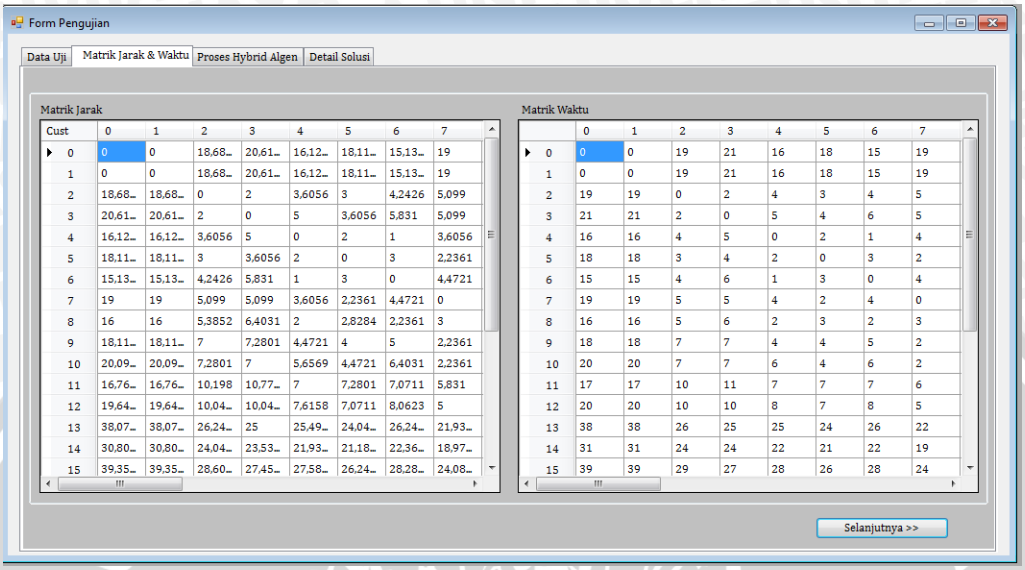
Antarmuka pengujian pada *tab* Data Uji ditunjukkan pada Gambar 4.4 berikut:



Gambar 4.4 Antarmuka Pengujian *Tab* Data Uji.

Tahap kedua yaitu menampilkan *tab* Matriks Jarak dan Waktu. Antarmuka ini menampilkan hasil perhitungan jarak tempuh dan waktu tempuh antar pelanggan. *Button* selanjutnya pada antarmuka ini merupakan *action* menuju ke

tab selanjutnya yaitu Proses *hybrid* Algen. Gambar 4.5 berikut ini menunjukkan antarmuka untuk *tab* Matriks Jarak dan Waktu Tempuh.

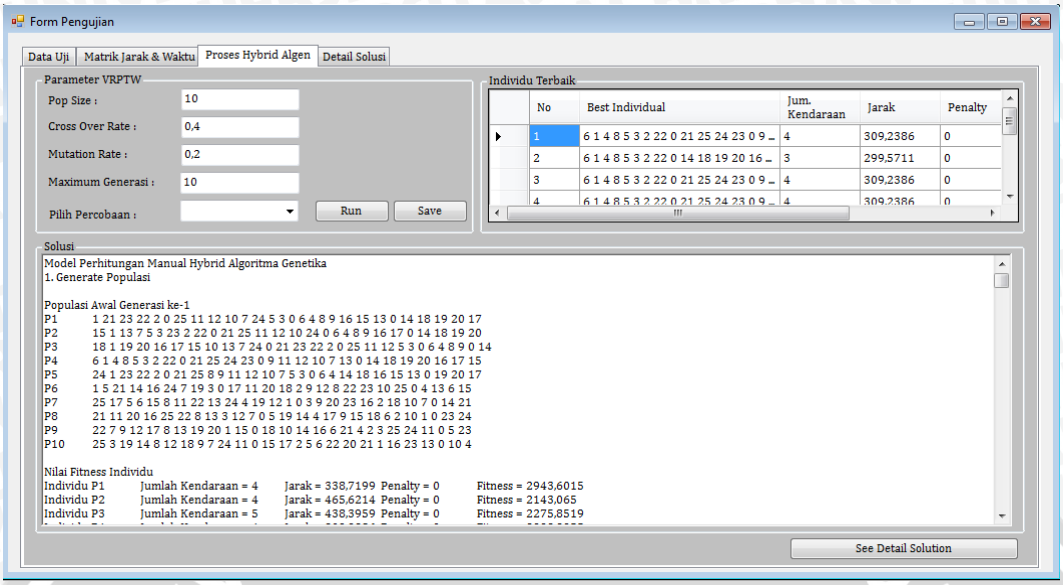


Gambar 4.5 Antarmuka Pengujian *Tab* Matriks Jarak dan Waktu.

Tahap ketiga pada proses pengujian yaitu penerapan *hybrid* algoritma genetika dimana proses ini dilakukan pada *tab* Penerapan *Hybrid* Algen. Rincian proses yang dilakukan pada tahap ini adalah:

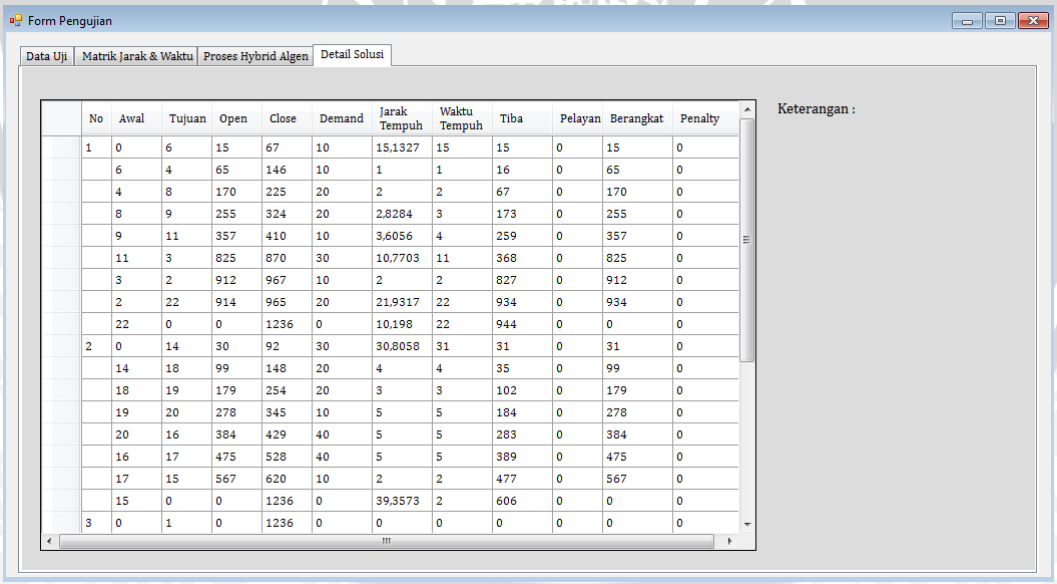
1. Inputkan nilai *pop size*.
2. Inputkan probabilitas *crossover*.
3. Inputkan probabilitas mutasi.
4. Inputkan *Maximum* generasi.
5. Run program untuk mengetahui proses perhitungan dan *best solution* yang dihasilkan pada proses perhitungan. Detail perhitungan dan *best solution* ditampilkan pada *group box* Solusi sedangkan *group box* individu terbaik menampilkan solusi terbaik yang dihasilkan pada setiap generasi.
6. *Button see detail solution* merupakan link menuju *tab* selanjutnya yaitu Detail Solusi dimana proses perhitungan jarak dan waktu dari *best solution* di tampilkan.

Antarmuka pada tahap pengujian untuk proses perhitungan ditunjukkan pada Gambar 4.6 berikut ini.



Gambar 4.6 Antarmuka Pengujian *Tab* Proses *Hybrid* Algen.

Tahap terakhir yang dilakukan pada antarmuka pengujian yaitu menampilkan detail solusi dari *the best solution* pada *tab* detail solusi yaitu berupa *cost* jarak, waktu tempuh, dan *penalty* kendaraan. Adapun antarmuka pada proses ini dapat dilihat pada Gambar 4.7.



No	Awal	Tujuan	Open	Close	Demand	Jarak Tempuh	Waktu Tempuh	Tiba	Pelayan	Berangkat	Penalty
1	0	6	15	67	10	15,1327	15	15	0	15	0
	6	4	65	146	10	1	1	16	0	65	0
	4	8	170	225	20	2	2	67	0	170	0
	8	9	255	324	20	2,8284	3	173	0	255	0
	9	11	357	410	10	3,6056	4	259	0	357	0
	11	3	825	870	30	10,7703	11	368	0	825	0
	3	2	912	967	10	2	2	827	0	912	0
	2	22	914	965	20	21,9317	22	934	0	934	0
	22	0	0	1236	0	10,198	22	944	0	0	0
2	0	14	30	92	30	30,8058	31	31	0	31	0
	14	18	99	148	20	4	4	35	0	99	0
	18	19	179	254	20	3	3	102	0	179	0
	19	20	278	345	10	5	5	184	0	278	0
	20	16	384	429	40	5	5	283	0	384	0
	16	17	475	528	40	5	5	389	0	475	0
	17	15	567	620	10	2	2	477	0	567	0
	15	0	0	1236	0	39,3573	2	606	0	0	0
3	0	1	0	1236	0	0	0	0	0	0	0

Gambar 4.7 Antarmuka Pengujian *Tab* Detail Solusi.

BAB V
PENGUJIAN DAN ANALISIS

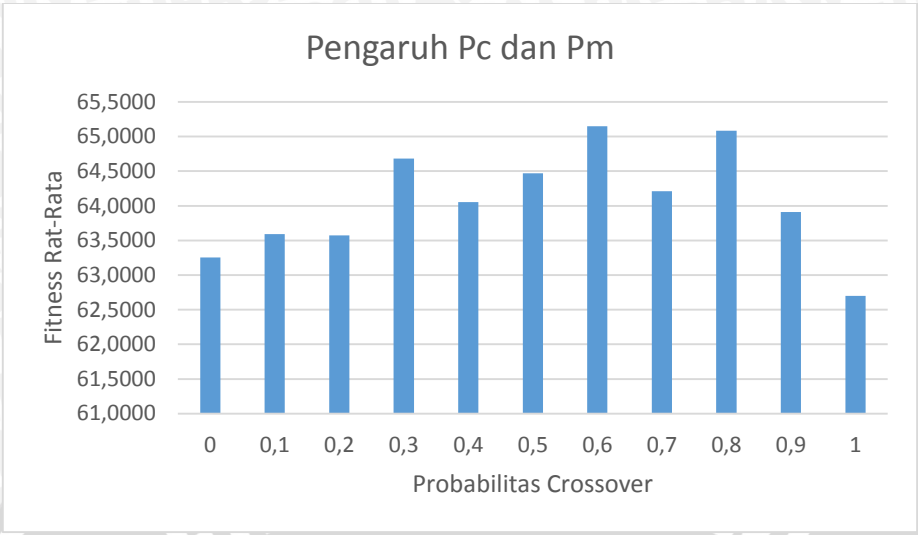
Bab ini akan membahas mengenai analisis dan pengujian yang telah diimplementasikan pada tahap sebelumnya. Analisis pengujian meliputi pengaruh probabilitas *crossover* dan mutasi terhadap *fitness* rata-rata, pengaruh ukuran populasi terhadap *fitness* rata-rata, pengaruh banyaknya generasi terhadap *fitness* rata-rata, dan analisis perbandingan *best test* dan *best known* Solomon.

5.1 Pengujian Pengaruh Probabilitas *Crossover* dan Mutasi terhadap Fitness Rata-Rata

Pada pengujian ini, akan dianalisis pengaruh probabilitas *crossover* (p_c) dan probabilitas mutasi (p_m) terhadap *fitness* rata-rata. Nilai yang digunakan untuk p_c dan p_m adalah antara 10% sampai dengan 100% dengan total probabilitas keduanya adalah 100%. Ukuran populasi dan banyaknya generasi yang diujikan pada tahap ini masing-masing 30 individu dalam suatu populasi dan banyaknya generasi adalah 100. Untuk masing-masing parameter, dilakukan 10 kali percobaan kemudian nilai *fitness* diambil berdasarkan rata-rata *fitness* dari percobaan tersebut. Hasil pengujian dari pengaruh p_c dan p_m dapat dilihat pada Tabel 5.1 berikut:

Tabel 5.1 Pengujian Pengaruh p_c dan p_m terhadap *Fitness* Rata-Rata

No	Prob. Crossover	Prob. Mutation	Fitness Rata-Rata
1	0%	1	63,2520
2	10%	90%	63,5924
3	20%	80%	63,5710
4	30%	70%	64,6799
5	40%	60%	64,0536
6	50%	50%	64,4681
7	60%	40%	65,1493
8	70%	30%	64,2111
9	80%	20%	65,0864
10	90%	10%	63,9096
11	100%	0%	62,6976



Gambar 5.1 Diagram Pengaruh p_c dan p_m terhadap *Fitness* Rata-Rata.

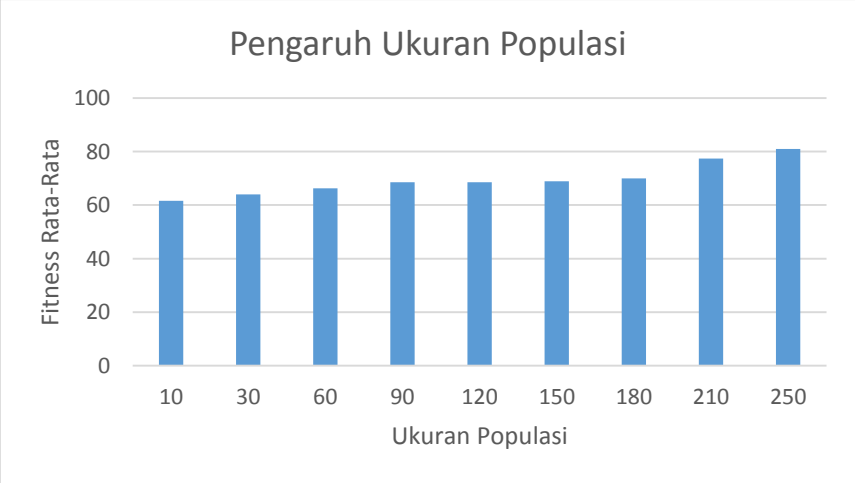
Berdasarkan pengujian terhadap pengaruh p_c dan p_m dan diagram pada Gambar 5.1, terlihat bahwa probabilitas crossover memberikan pengaruh yang lebih baik terhadap nilai *fitness*. Jadi, jika nilai probabilitas *crossover* lebih besar dari probabilitas mutasi, maka memiliki kecendrungan menghasilkan nilai fitness yang lebih tinggi. Hal ini karena tujuan mutasi adalah memunculkan material genetik baru ke dalam populasi sehingga memperbesar diversitas genetik. Jadi, penggunaan probabilitas mutasi yang tinggi dikhawatirkan dapat merusak material genetik yang baik pada individu yang memiliki *fitness* tinggi [10]. *Fitness* tertinggi dihasilkan dengan nilai p_c = 0,6 dan p_m = 0,4. Sedangkan fitness terendah dihasilkan dengan nilai p_c = 1 dan p_m = 0.

5.2 Pengujian Pengaruh Ukuran Populasi Terhadap Fitness Rata-Rata

Ukuran populasi merupakan parameter penting dalam algoritma genetika. Ukuran populasi menentukan jumlah individu dalam suatu populasi. Pada pengujian tahap ini, ukuran populasi yang digunakan adalah 10 sampai dengan N *pop_size*, banyaknya generasi adalah 500, nilai p_c = 0,6 dan p_m = 0,4. Pengujian dilakukan sebanyak 10 kali untuk masing-masing ukuran populasi. Hasil pengujian dapat dilihat pada Tabel 5.2 berikut.

Tabel 5.2 Pengujian Pengaruh Ukuran Populasi Terhadap Fitness Rata-Rata

No	Ukuran Populasi	Fitness Rata-Rata
1	10	61,59327
2	30	63,9899
3	60	66,21779
4	90	68,56718
5	120	68,55262
6	150	68,87978
7	180	69,91288
8	210	77,42599
9	250	80,93933



Gambar 5.2 Diagram Pengaruh Ukuran Populasi Terhadap Fitness Rata-Rata.

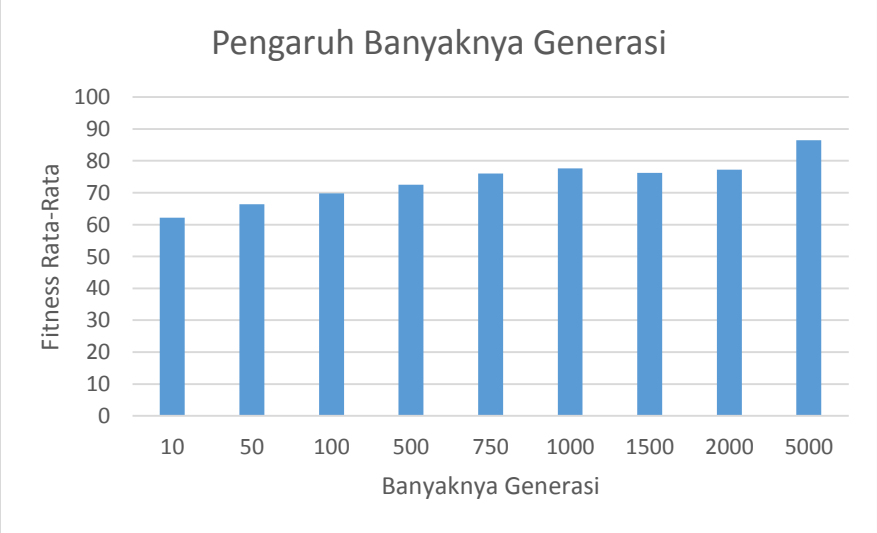
Berdasarkan pengujian pengaruh ukuran populasi terhadap fitness rata-rata pada Tabel 5.2 dan diagram Gambar 5.2, maka dapat dianalisis bahwa semakin besar ukuran populasi maka fitness yang dihasilkan cendrung meningkat. Hal ini disebabkan meningkatnya proses genetika seperti *crossover* dan mutasi sehingga menghasilkan individu yang lebih beragam. Penggunaan ukuran populasi pada tahap pengujian dengan mempertimbangkan waktu komputasi karena ukuran populasi yang semakin tinggi membutuhkan waktu komputasi yang semakin lama.

5.3 Pengujian Pengaruh Banyaknya Generasi Terhadap Fitness Rata-Rata

Rancangan pengujian tahap ini dilakukan untuk mengetahui pengaruh banyaknya generasi terhadap pencarian solusi dan nilai *fitness* yang dihasilkan. Parameter banyaknya generasi yang digunakan pada pengujian yaitu 10 sampai dengan 5000. Pengujian dilakukan sebanyak 10 kali untuk kemudian dihitung fitness rata-rata. Untuk ukuran populasi yaitu 100, nilai p_c = 0,6 dan p_m = 0,4. Pengujian terhadap pengaruh banyaknya generasi terhadap fitness rata-rata dapat dilihat pada Tabel 5.3 berikut.

Tabel 5.3 Pengujian Pengaruh Banyaknya Generasi Terhadap Fitness Rata-Rata

No	Banyaknya Generasi	Rata-Rata Nilai Fitness
1	10	62,16792
2	50	66,34275
3	100	69,79339
4	500	72,55016
5	750	75,98849
6	1000	77,68348
7	1500	76,20943
8	2000	77,24768
9	5000	86,45934



Gambar 5.3 Diagram Pengaruh Banyaknya Generasi Terhadap Fitness Rata-Rata.

Berdasarkan Gambar 5.3, dapat dianalisis bahwa nilai fitness rata-rata cendrung berbanding lurus dengan peningkatan banyaknya generasi. Artinya, semakin tinggi banyaknya generasi, maka nilai fitness yang dihasilkan cendrung meningkat. Sama seperti pengaruh ukuran populasi, tingginya generasi juga mempengaruhi waktu komputasi sehingga pada tahap pengujian, banyaknya generasi dipilih dengan mempertimbangkan waktu komputasi.

5.4 Pengujian *Best Test* dan *Best Known* Solomon

Pengujian tahap ini dilakukan dengan melakukan pencarian solusi dan membandingkan hasil terbaik dengan *best solution of Solomon*. *Best solution of Solomon* dihasilkan oleh para pakar dengan berbagai macam metode. Sebuah metode mungkin memberikan hasil terbaik untuk satu kasus tetapi bisa dikalahkan oleh metode yang lainnya pada kasus berbeda. Tabel 5.4 berikut ini menunjukkan metode yang digunakan pada *best solution of Solomon*.

Tabel 5.4 Metode Pakar dari *Best Solution of Solomon*

No	Problem	Authors	Methods
1	C101.25	KDMSS	2-Path Cuts Algorithm
2	C105.25	KDMSS	2-Path Cuts Algorithm
3	C201.25	CR+L	A Parallel Cutting Plane and Parallelization Algorithm
4	C202.25	CR+L	A Parallel Cutting Plane and Parallelization Algorithm
5	C101.50	KDMSS	2-Path Cuts Algorithm
6	C105.50	KDMSS	2-Path Cuts Algorithm
7	C201.50	CR+L	A Parallel Cutting Plane and Parallelization Algorithm
8	C202.50	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
9	C101.100	KDMSS	2-Path Cuts Algorithm
10	C105.100	KDMSS	2-Path Cuts Algorithm
11	C201.100	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
12	C202.100	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
13	R101.25	KDMSS	2-Path Cuts Algorithm
14	R102.25	KDMSS	2-Path Cuts Algorithm
15	R201.25	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization

16	R202.25	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
17	R101.50	KDMSS	2-Path Cuts Algorithm
18	R102.50	KDMSS	2-Path Cuts Algorithm
19	R201.50	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
20	R202.50	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
21	R101.100	KDMSS	2-Path Cuts Algorithm
22	R102.100	KDMSS	2-Path Cuts Algorithm
23	R201.100	KLM	Lagrangean Duality and Non-Differentiable Optimization
24	R202.100	RGP	Using Constraint-Based Operators
25	RC101.25	KDMSS	2-Path Cuts Algorithm
26	RC102.25	KDMSS	2-Path Cuts Algorithm
27	RC201.25	CR+L	A Parallel Cutting Plane and Parallelization Algorithm
28	RC202.25	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
29	RC101.50	KDMSS	2-Path Cuts Algorithm
30	RC102.50	KDMSS	2-Path Cuts Algorithm
31	RC201.50	L+KLM	Parallelization Algorithm + Lagrangean Duality and Non-Differentiable Optimization
32	RC202.50	IV+C	The Shortest Path Problem with K-Cycle Elimination (k≥3) : Improving A Branch and Price Algorithm and Elementary Shortest Path Based Column Generation
33	RC101.100	KDMSS	2-Path Cuts Algorithm
34	RC102.100	CR+KLM	A Parallel Cutting Plane Algorithm and Lagrangean Duality and Non-Differentiable Optimization
35	RC201.100	KLM	Lagrangean Duality and Non-Differentiable Optimization
36	RC202.100	IV+C	The Shortest Path Problem with K-Cycle Elimination (k≥3) : Improving A Branch and Price Algorithm and Elementary Shortest Path Based Column Generation

Pengujian dilakukan dengan mengambil 2 set data uji dari masing-masing tipe data yaitu C1, C2, R1, R2, RC1, dan RC2. Adapun parameter genetika yang digunakan yaitu:

- Ukuran populasi = 1000
- Banyaknya Generasi = 250
- Probabilitas *crossover* = 0,6

- Probabilitas mutasi = 0,4
- Kecepatan kendaraan dianggap konstan yaitu 60 km/jam

Adapun Tabel 5.5 merupakan hasil percobaan untuk 25 *customers* pada 2 set data uji dari masing-masing tipe data.

Tabel 5.5 Hasil Percobaan 25 *Customers*

No	Problem	Best Known Solomon			Algoritma Genetika		RPD (%)
		NV	Distance	Authors	NV	Distance	
1	C101	3	191,30	KDMSS	3	191,81	0,267
2	C105	3	191,30	KDMSS	3	191,83	0,277
						Rata - Rata RPD	0,272
3	C201	2	214,70	CR+L	2	215,54	0,393
4	C202	2	214,70	CR+L	1	223,3103	4,010
						Rata - Rata RPD	2,201
5	R101	8	617,10	KDMSS	8	618,3296	0,199
6	R102	7	547,10	KDMSS	7	561,2715	2,590
						Rata - Rata RPD	1,395
7	R201	4	463,00	CR+KLM	4	539,9472	16,619
8	R202	4	410,50	CR+KLM	2	521,9274	27,144
						Rata - Rata RPD	21,882
9	RC101	4	461,10	KDMSS	4	467,481	1,384
10	RC102	3	351,80	KDMSS	3	354,0117	0,629
						Rata - Rata RPD	1,006
11	RC201	3	360,20	CR+L	4	467,5	29,789
12	RC202	3	338,00	CR+KLM	3	354,0117	4,737
						Rata - Rata RPD	17,263

NV = *Number Of Vehicles*

Percobaan untuk 50 *customers* pada 2 set data uji dari masing-masing tipe data dapat dilihat pada Tabel 5.6.

Tabel 5.6 Hasil Percobaan 50 *Customers*

No	Problem	Best Known Solomon			Algoritma Genetika		RPD (%)
		NV	Distance	Authors	NV	Distance	
1	C101	5	362,40	KDMSS	5	363,247	0,234
2	C105	5	362,40	KDMSS	5	363,247	0,234
						Rata - Rata RPD	0,234
3	C201	3	360,20	CR+L	3	361,797	0,443
4	C202	3	360,20	CR+KLM	3	479,2886	33,062
						Rata - Rata RPD	16,753
5	R101	12	1044,00	KDMSS	13	1133,292	8,553
6	R102	11	909,00	KDMSS	11	1023,077	12,550
						Rata - Rata RPD	10,551
7	R201	6	791,90	CR+KLM	3	1032,490	30,381
8	R202	5	698,50	CR+KLM	3	979,483	40,227
						Rata - Rata RPD	35,304
9	RC101	8	944,00	KDMSS	4	979,072	3,715
10	RC102	7	822,50	KDMSS	3	958,178	16,496
						Rata - Rata RPD	10,106
11	RC201	5	684,80	L+KLM	4	945,959	38,137
12	RC202	5	613,60	IV+C	3	855,402	39,407
						Rata - Rata RPD	38,772

NV = *Number Of Vehicles*

Percobaan untuk 100 *customers* pada 2 set data uji dari masing-masing tipe data dapat dilihat pada Tabel 5.7.

Tabel 5.7 Hasil Percobaan 100 *Customers*

No	Problem	Best Known Solomon			Algoritma Genetika		RPD (%)
		NV	Distance	Authors	NV	Distance	
1	C101	10	827,30	KDMSS	10	850,402	2,792
2	C105	10	827,30	KDMSS	11	1177,485	42,329
						Rata - Rata RPD	22,561
3	C201	3	589,10	CR+KLM	3	591,557	0,417
4	C202	3	589,10	CR+KLM	3	946,282	60,632
						Rata - Rata RPD	30,525
5	R101	20	1637,70	KDMSS	20	1953,457	19,281
6	R102	18	1466,60	KDMSS	19	1846,208	25,884
						Rata - Rata RPD	22,582
7	R201	8	1143,20	KLM	7	1924,042	68,303

8	R202	3	1191,70	RGP	5	1875,888	57,413
						Rata - Rata RPD	62,858
9	RC101	15	1619,80	KDMSS	18	2073,977	28,039
10	RC102	14	1457,40	CR+KLM	16	1928,914	32,353
						Rata - Rata RPD	30,196
11	RC201	9	1261,80	KLM	8	2005,036	58,903
12	RC202	8	1092,30	IV+C	9	2023,394	85,242
						Rata - Rata RPD	72,072

NV = *Number Of Vehicles*

Tabel 5.8 Rata-Rata RPD 25 *Customers*

No	Tipe	Rata-Rata RPD (%)
1	C1	0,272
2	C2	2,201
3	R1	1,395
4	R2	21,882
5	RC1	1,006
6	RC2	17,263
Rata-Rata RPD (%)		7,337

Tabel 5.9 Rata-Rata RPD 50 *Customers*

No	Tipe	Rata-Rata RPD (%)
1	C1	0,234
2	C2	16,753
3	R1	10,551
4	R2	35,304
5	RC1	10,106
6	RC2	38,772
Rata-Rata RPD (%)		18,620

Tabel 5.10 Rata-Rata RPD 100 *Customers*

No	Tipe	Rata-Rata RPD (%)
1	C1	22,561
2	C2	30,525
3	R1	22,582
4	R2	62,858
5	RC1	30,196
6	RC2	72,072
Rata-Rata RPD (%)		40,132

Berdasarkan hasil pengujian terhadap 25 *customers*, sistem menghasilkan solusi yang mendekati *best known* Solomon pada tipe data C1, C2, R1, dan RC1 dengan rata-rata RPD kurang dari 3%. Sedangkan hasil terburuk dengan rata-rata RPD 17,263% untuk tipe data RC2 dan 21,882% untuk tipe data R2.

Pengujian terhadap 50 *customers*, sistem menghasilkan solusi yang mendekati *best known* Solomon untuk tipe data C1 dengan rata-rata RPD kurang dari 1%. Untuk tipe data R1 dan RC1 dengan rata-rata RPD kurang dari 11% dan rata-rata RPD lebih dari 11% untuk tipe data C2, R2, dan RC2.

Pengujian terhadap 100 *customers*, sistem menghasilkan solusi yang dihasilkan dengan rata-rata RPD dibawah 25% untuk tipe data C1 dan R1. Dan sisanya menghasilkan rata-rata RPD lebih besar dari 25%.

Penerapan *hybrid* algoritma genetika untuk permasalahan Solomon menghasilkan solusi yang mendekati *best known* solomon dengan RPD kurang dari 1% terhadap 25 *customers* untuk tipe data C101, C105, C201, R101, dan RC102. Pengujian terhadap 50 *customers*, sistem menghasilkan solusi yang mendekati *best known* Solomon dengan RPD kurang dari 1% untuk tipe data C101, C105, dan C201. Pengujian terhadap 100 *customers*, sistem menghasilkan solusi yang mendekati *best known* Solomon dengan RPD kurang dari 1% untuk tipe data C201.

Hasil tersebut menunjukkan bahwa penerapan algoritma ini lebih cocok untuk permasalahan data skala kecil dan lebih cendrung menghasilkan solusi mendekati Solomon untuk tipe data C (data yang terletak secara kluster) dibandingkan tipe data R (data yang tersebar secara random). Pada kasus dengan

25 *customers*, beberapa tipe data random menghasilkan solusi mendekati Solomon dengan RPD kurang dari 1%, hal ini tidak lagi berlaku ketika dilakukan pengujian terhadap tipe data R dengan 50 dan 100 *customers*.

Berikut ini disajikan *Sample* dari *output* pengujian sistem terhadap rute hasil optimasi untuk tipe data C101 dengan 25 *customers*.

```
Generasi terbaik didapatkan pada
Generasi = 160
Kromosom = 13 17 18 19 15 16 14 12 0 20 24 25 23 22 21 0 5 3 7 8 10 11 9 6 4 2 1
Kendaraan = 3
Jarak = 191,8136
Penalti = 0
Fitness = 5186,3561
Rata-rata = 3784,4723
```

Gambar 5.6 *Output* Hasil Pengujian Sistem terhadap 25 *Customers* untuk Tipe Data C101

Solusi tersebut menghasilkan rute yang mendekati solusi optimal berdasarkan solusi Solomon yang menghasilkan NV = 3 dan total jarak 191,3. Adapun detail rute untuk 25 *customers* pada tipe data C101 adalah

- R1 = 0 – 13 – 17 – 18 – 19 – 15 – 16 – 14 – 12 – 0
- R2 = 0 – 20 – 24 – 25 – 23 – 22 – 21 – 0
- R3 = 0 – 5 – 3 – 7 – 8 – 10 – 11 – 9 – 6 – 4 – 2 – 1 – 0

Rute tersebut menghasilkan rute yang sama dengan penelitian yang dilakukan sebelumnya oleh Astuti [9] untuk tipe data C101 dengan 25 *Customers*.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil penelitian, maka dapat disimpulkan bahwa:

1. Implementasi metode *hybrid* algoritma genetika pada kasus Solomon tidak dapat memberikan hasil yang lebih baik dari *best known solomon*. Akan tetapi solusi yang dihasilkan mendekati *best known solomon* yaitu dengan RPD kurang dari 1% terhadap 25 *customers* untuk tipe data C101, C105, C201, R101, dan RC102. RPD kurang dari 1% untuk tipe C101, C105, dan C201 terhadap 50 *customers*, dan RPD kurang dari 1% untuk tipe data C201 terhadap 100 *customers*.
2. Implementasi metode *hybrid* algoritma genetika pada kasus Solomon memberikan nilai RPD yang sangat besar pada tipe data R2, dan RC2.
3. Penerapan *hybrid algoritma genetika* lebih cocok untuk permasalahan data skala kecil dan lebih cendrung menghasilkan solusi mendekati Solomon untuk tipe data C (data yang terletak secara kluster) dibandingkan tipe data R (data yang tersebar secara random).
4. Metode *hybrid* algoritma genetika dapat menjadi salah satu alternatif dalam menyelesaikan permasalahan VRPTW.

6.2 Saran

Berdasarkan penelitian yang telah dilakukan, penerapan *hybrid* algoritma genetika pada permasalahan VRPTW dapat dikembangkan dengan mencoba menerapkan algoritma lainnya atau mengkombinasikan antara algoritma genetika dengan metode heuristik lainnya untuk menghasilkan solusi yang lebih baik. Penambahan jumlah generasi dan jumlah populasi memungkinkan menghasilkan solusi yang lebih optimal akan tetapi membutuhkan proses komputasi yang lebih lama. Untuk itu, pengembangan terhadap penerapan *hybrid* algoritma genetika untuk penelitian selanjutnya dapat mengurangi permasalahan waktu pada proses komputasi. Selain itu, diharapkan aplikasi ini dapat diterapkan lebih lanjut pada

permasalahan distribusi secara nyata pada suatu perusahaan atau instansi karena penelitian yang dilakukan oleh penulis masih berupa simulasi.

DAFTAR PUSTAKA

[1] Danfar. 2009. *Definisi/Pengertian Distribusi*. URL: http://dansite.wordpress.com/2009/03/25/pengertian-distribusi/, diakses tanggal 30 Januari 2014.

[2] Toth, Paolo dan Daniel Vigo. 2001. *The Vehicle Routing Problem*. Philadelphia : Society for Industrial and Applied Mathematics.

[3] Ningsih, Dewi Shinta. 2008. *Implementasi Algoritma Ant Colony System Pada Masalah Vehicle Routing Problem With Time Windows*. Komputasi Cerdas dan Virtual. Universitas Brawijaya. Malang.

[4] Purnomo, Agus. 2010. *Analisis Rute Pendistribusian Dengan Menggunakan Metode Nearest Insertion Heuristic Persoalan The Vehicle Routing Problem With Time Windows (VRPTW)*. http://digilib.unpas.ac.id/files/disk1/17/jbptunpaspp-gdl-driraguspu-801-1-18vrptw-).pdf, 27 Januari 2014.

[5] Tanujaya, W., Dian Retno S.D. dan Dini Endah. 2011. *Penerapan Algoritma Genetik untuk Penyelesaian Masalah Vehicle Routing di PT.MIF*. http://journal.wima.ac.id/index.php/teknik/article/download/163/159, 27 Januari 2014.

[6] Suyanto. 2010. *Algoritma Optimasi Deterministik atau Probabilitik*. Graha Ilmu : Yogyakarta.

[7] Gen, M. dan Runwei Cheng. 2000. *Genetic Algorithms and Engineering Optimization*. New York : John Wiley & Sons.

[8] Mahmudy, WF, Marian, RM & Luong, LHS 2014, 'Hybrid genetic algorithms for part type selection and machine loading problems with alternative production plans in flexible manufacturing system', *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, vol. 8, no. 1, pp. 80-93.

[9] Astuti, Sri. 2012. *Aplikasi Algoritma Genetika Hibrida pada Vehicle Routing Problem With Time Windows*. Matematika. Universitas Indonesia. Depok.

[10] Widodo, Thomas Sri. 2012. *Komputasi Evolusioner*. Yogyakarta : Graha Ilmu.

[11] Suyanto. 2011. *Artificial Intelegence*. Bandung : Informatika.

[12] Mahmudy, Wayan Firdaus. 2013. Modul Mata Kuliah Algoritma Evolusi. Universitas Brawijaya. Malang.

[13] Kusumadewi, Sri. 2003. *Artificial Intelligence*. Jogjakarta : Graha Ilmu.

[14] Cinantya, Celia. 2010. *Implementasi Algoritma Genetika untuk Penyelesaian Capacitated Vehicle Routing with Time Windows*. Komputasi Cerdas dan Virtual. Universitas Brawijaya. Malang.

LAMPIRAN

Lampiran 1. *Authors Best Solution* of Solomon

Legend:

C - A. Chabrier, "Vehicle Routing Problem with Elementary Shortest Path based Column Generation." Forthcoming in: ***Computers and Operations Research*** (2005).

CR - W. Cook and J. L. Rich, "A parallel cutting plane algorithm for the vehicle routing problem with time windows," Working Paper, Computational and Applied Mathematics, Rice University, Houston, TX, 1999.

IV - S. Irnich and D. Villeneuve, "The shortest path problem with k-cycle elimination ($k \geq 3$): Improving a branch-and-price algorithm for the VRPTW." Forthcoming in: **INFORMS Journal of Computing** (2005).

KDMSS - N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis, "2-Path Cuts for the Vehicle Routing Problem with Time Windows," ***Transportation Science***, Vol. 33 (1), 101-116 (1999).

KLM - B. Kallehauge, J. Larsen, and O.B.G. Madsen. "Lagrangean duality and non-differentiable optimization applied on routing with time windows - experimental results." Internal report IMM-REP-2000-8, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 2000.

L - J. Larsen. "Parallelization of the vehicle routing problem with time windows." Ph.D. Thesis IMM-PHD-1999-62, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmar k, 1999.

RGP - L.M. Rousseau, M. Gendreau and G. Pesant, "Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows," ***Journal of Heuristics,*** forthcoming.