TUGAS AKHIR – IS184853

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

OPTIMIZATION OF CITY BUS ROUTES IN SURABAYA USING MEMETIC ALGORITHM AS A SOLUTION TO SOLVE ORIENTEERING PROBLEM

FAUZI RAKHMAD FIRDAUSI
NRP 0521 14 4000 0035

Dosen Pembimbing:
Prof. Ir. Arif Djunaidy, M.Sc., Ph.D.
Ahmad Mukhlason, S.Kom., M.Sc., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – IS184853

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

FAUZI RAKHMAD FIRDAUSI
NRP 0521 14 4000 0035

Dosen Pembimbing:
Prof. Ir. Arif Djunaidy, M.Sc., Ph.D.
Ahmad Mukhlason, S.Kom., M.Sc., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

"Halaman ini sengaja dikosongkan"

FINAL PROJECT – IS184853

OPTIMIZATION OF CITY BUS ROUTES IN SURABAYA USING MEMETIC ALGORITHM AS A SOLUTION TO SOLVE ORIENTEERING PROBLEM

FAUZI RAKHMAD FIRDAUSI
NRP 0521 14 4000 0035

Dosen Pembimbing:
Prof. Ir. Arif Djunaidy, M.Sc., Ph.D.
Ahmad Mukhlason, S.Kom., M.Sc., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

"Halaman ini sengaja dikosongkan"

LEMBAR PENGESAHAN

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

FAUZI RAKHMAD FIRDAUSI
NRP 0521 14 4000 0035

Surabaya, 9 Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

Mahendrawathi ER., S.T., M.Sc., Ph.D.
NIP 19761011 200604 2 001

"Halaman ini sengaja dikosongkan"

LEMBAR PERSETUJUAN

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

FAUZI RAKHMAD FIRDAUSI
NRP 0521 14 4000 0035

Disetujui Tim Penguji: Tanggal Ujian: 20 Mei 2019
Periode Wisuda: September 2019

Prof. Ir. Arif Djunaidy, M.Sc., Ph.D. (Pembimbing I)

Ahmad Mukhlason, S.Kom., M.Sc., Ph.D. (Pembimbing II)

Wiwik Anggraeni S.Si., M.Kom. (Penguji I)

Edwin Riksakomara, S.Kom., MT. (Penguji II)

"Halaman ini sengaja dikosongkan"

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

Nama Mahasiswa	**: Fauzi Rakhmad Firdausi**
NRP	**: 0521 14 4000 0035**
Departemen	**: Sistem Informasi FTIK-ITS**
Pembimbing 1	**: Prof. Ir. Arif Djunaidy, M.Sc., Ph.D.**
Pembimbing 2	**: Ahmad Mukhlason, S.Kom., M.Sc., Ph.D.**

ABSTRAK

Kota Surabaya merupakan kota metropolitan terbesar di Jawa Timur dan memiliki beragam objek wisata, seperti pusat perbelanjaan, taman hiburan, monumen, museum, dan tempat-tempat bersejarah. Surabaya sebagai kota terbesar kedua di Indonesia tidak luput dari kemacetan lalu lintas. Kemacetan terebut diperparah oleh banyaknya wisatawan luar kota yang berkunjung ke kota Surabaya menggunakan kendaraan pribadi. Untuk itu, Pemerintah Kota Surabaya mengupayakan berbagai cara untuk mengurangi kemacetan tersebut melalui perbaikan sarana dan layanan transportasi umum serta upaya peningkatan kesadaran masyarakat terhadap pentingnya penggunaan transportasi umum. Untuk mendukung upaya tersebut, perlu dibuat sebuah layanan untuk mengoptimalkan pemilihan jalur perjalanan ke berbagai objek wisata sesuai dengan ketersediaan waktu yang dimiliki oleh wisatawan.

Dalam tugas akhir ini, pemilihan jalur bis kota dipandang sebagai orienteering problem (OP) untuk menentukan tempat-tempat wisata yang akan dikunjungi. Algoritma memetika digunakan untuk menyelesaikan OP guna mencari solusi yang optimal. Algoritma memetika memanfaatkan algoritma genetika dengan satu tambahan subalgoritma berupa pencarian lokal untuk memperoleh solusi optimal yang lebih efisien.

Algoritma memetika (AM) yang diimplementasikan menggunakan bahasa pemrograman Java diuji coba pada jalur bis kota yang terdiri dari 10 trayek. Uji coba dilakukan dengan membandingkan hasil implementasi solusi OP menggunakan AG dasar dan solusi OP menggunakan AM yang dilengkapi dengan subalgoritma pencarian berbasis hill climbing (AMHC) dan simulated annealing (AMSA). Dengan menggunakan parameter terbaik untuk ketiga implementasi AG, AMHC, dan AMSA, hasil uji coba menunjukkan bahwa baik AMHC dan AMSA mampu menghasilkan nilai fitnes yang lebih baik dibandingkan AG. Tetapi dari kedua hasil implementasi AM, AMSA memberikan solusi yang lebih baij dibandingkan AMHC.

Kata Kunci: Optimasi, Orienteering Problem, Algoritma Genetika, Algoritma Memetika, Hill Climbing, Simulated Annealing

OPTIMIZATION OF CITY BUS ROUTES IN SURABAYA USING MEMETIC ALGORITHM AS A SOLUTION TO SOLVE ORIENTEERING PROBLEM

Name	**: Fauzi Rakhmad Firdausi**
NRP	**: 0521 14 4000 0035**
Department	**: Information Systems FTIK -ITS**
Supervisor 1	**: Prof. Ir. Arif Djunaidy, M.Sc., Ph.D.**
Supervisor 2	**: Ahmad Mukhlason, S.Kom., M.Sc., Ph.D.**

ABSTRACT

The City of Surabaya is the largest metropolitan city in East Java that has a variety of tourist attractions, such as shopping malls, amusement parks, monuments, museums, and historical places. Surabaya as the second largest city in Indonesia, indeed it still has traffic jam problem. Such a problem is worsened by many domestic tourists who visit Surabaya using their private vehicles. For this reason, the Municipality Government of Surabaya seeks various ways to reduce the traffic jam problem through improvements in public transportation facilities and services and attempts to increase public awareness of the importance of using public transportation. To support these efforts, it is necessary for the government to provide a service to optimize the selection of travel routes to various tourist objects in accordance with the availability of time of the tourists.

In this final project, the selection of city bus lines is regarded as an orienteering problem (OP) to determine the tourist objects to be visited. The memetic algorithm is used to solve the OP to find the optimal solution. Memetic algorithms utilizes genetic algorithms with one additional local search algorithm to obtain a more efficient optimal solution.

The memetic algorithm (MA) which was implemented using the Java programming language was tested on the city bus line

consisting of 10 routes. The experiment was carried out by comparing the results of the implementation of the OP solution using basic GA and the OP solution using MA which is improved with a hill climbing search (HCMA) and simulated annealing algorithm (SAMA). By using the best parameters for each of these three implementations (i.e., GA, HCMA, and SAMA), the experimental results show that both HCMA and SAMA produced better fitness values than the original GA. However, among the two MA implementation results, SAMA provides a better solution than HCMA.

Keywords: Optimization, Orienteering Problem, Genetic Algorithm, Memetic Algorithm, Hill Climbing, Simulated Annealing

KATA PENGANTAR

Puji syukur atas nikmat, rahmat, dan karunia yang telah diberikan Allah SWT kepada penulis, sehingga penulis diberikan kelancaran dalam menyelesaikan tugas akhir dengan judul:

OPTIMASI TRAYEK BIS KOTA DI SURABAYA MENGGUNAKAN ALGORITMA MEMETIKA SEBAGAI SOLUSI PENYELESAIAN ORIENTEERING PROBLEM

Sebagai salah satu syarat kelulusan pada Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember, Surabaya.

Terima kasih kepada pihak-pihak yang telah mendukung, memberikan saran, motivasi, semangat, dan doanya kepada penulis. Secara khusus penulis menyampaikan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Kedua orang tua penulis, yaitu Bapak Budi Siswanto dan Ibu Sri Indah Sudaryanti yang tidak pernah lelah memberikan dukungan, motivasi dan doanya kepada penulis sehingga dapat menyelesaikan pendidikan S1 ini dengan baik.
2. Kedua kakak penulis, Arif dan Tiara yang selalu memberikan bantuan moril maupun materiil.
3. Bapak Prof. Ir. Arif Djunaidy, M.Sc., Ph.D. selaku dosen wali sekaligus dosen pembimbing I yang selalu memberikan ilmu, bimbingan, dan nasehat sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
4. Bapak Ahmad Mukhlason, S.Kom., M.Sc., Ph.D. selaku dosen pembimbing II yang sudah banyak direpotkan oleh penulis dan selalu sabar dalam meluangkan waktunya untuk memberikan ilmu, bimbingan, dan motivasi untuk segera menyelesaikan Tugas Akhir ini dengan baik.
5. Ibu Wiwik Anggraeni, S.Si., M.Kom. dan Bapak Faizal Mahananto S.Kom., M.Eng., Ph.D. selaku dosen penguji

yang telah memberikan kritik dan sarannya untuk membantu menyempurnakan Tugas Akhir ini.

6. Ibu Mahendrawathi ER, S.T., M.Sc., Ph.D dan Bapak Nisfu Asrul Sani S.Kom., M.Sc selaku Kepala Departemen Sistem Informasi dan Kepala Program Studi S-1 Sistem Informasi, ITS, yang telah memberikan banyak bantuan kepada penulis selama kuliah di Jurusan Sistem Informasi FTIK, ITS Surabaya.
7. Seluruh dosen pengajar beserta staf Tata Usaha yang telah memberikan ilmu dan bantuan kepada penulis selama menempuh pendidikan di Jurusan Sistem Informasi, FTIK ITS Surabaya.
8. Teman-teman kontrakan: Ekadhana, Radyan, Almas, Graha, Ropek, Yoga, Januar, Dendi, Akbar yang selalu ada disaat suka dan duka.
9. Teman-teman OSIRIS dan adik-adik LANNISTER yang telah memberi segudang kenangan kepada penulis.
10. Berbagai pihak yang telah membantu dalam pengerjaan Tugas Akhir ini yang belum mampu penulis sebutkan di atas.

Penulis menyadari bahwa penyusunan laporan ini masih jauh dari kata sempurna, untuk itu penulis selalu menerima apabila terdapat kritik dan saran yang dapat membangun untuk masukan penulis dan perbaikan penelitian di masa mendatang. Semoga buku tugas akhir ini dapat memberikan manfaat pembaca.

Surabaya, 9 Juli 2019

Penulis

DAFTAR ISI

"Halaman ini sengaja dikosongkan"

DAFTAR TABEL

DAFTAR GAMBAR

"Halaman ini sengaja dikosongkan"

DAFTAR PERSAMAAN

"Halaman ini sengaja dikosongkan"

DAFTAR KODE

"Halaman ini sengaja dikosongkan"

BAB I
PENDAHULUAN

Dalam bab ini menjelaskan mengenai identifikasi permasalahan penelitian meliputi latar belakang masalah, rumusan masalah, batasan masalah, tujuan tugas akhir, manfaat tugas akhir, dan relevansi terhadap pengerjaan tugas akhir. Berdasarkan uraian dalam bab ini, diharapkan dapat memberikan gambaran mengenai permasalahan yang dibahas serta dapat memahami tujuan dan manfaat tugas akhir.

1.1. Latar Belakang

Kota Surabaya merupakan kota metropolitan terbesar di Jawa Timur dan kota terbesar kedua setelah Jakarta. Tidak heran apabila kota ini menjadi pusat bisnis dan bahkan pariwisata bagi penduduk asli maupun wisatawan luar kota Surabaya. Sekian dari banyaknya tempat wisata di kota Surabaya, pusat perbelanjaan merupakan salah satu objek yang sangat popular dikalangan masyarakat dan wisatawan luar kota dikarenakan banyaknya pilihan pusat perbelanjaan di kota tersebut. Mayoritas wisatawan luar kota menggunakan transportasi pribadi untuk mengunjungi objek-objek tersebut sehingga memperparah kemacetan lalu lintas yang terjadi. Kurangnya kesadaran masyarakat untuk menggunakan transportasi umum menjadi salah satu penyebab kemacetan lalu lintas, hal ini disebabkan karena menurut mereka transportasi umum di kota Surabaya belum sepenuhnya memenuhi standar, belum menjangkau ke seluruh daerah dan masih banyak yang tidak layak digunakan sehingga masyarakat lebih memilih menggunakan transportasi pribadi[1].

Seiring dengan seringnya terjadi kemacetan lalu lintas di Surabaya menyebabkan penggunaan Bahan Bakar Minyak (BBM) meningkat karena mesin menyala lebih lama sehingga pengendara harus mengeluarkan biaya yang lebih banyak untuk pembelian BBM[2].

Perihal tersebut yang melatarbelakangi penulis untuk membuat optimasi jalur transportasi umum di kota Surabaya. Hal ini dilakukan dengan cara membuat sebuah model dan solusi yang bertujuan untuk memberikan rekomendasi jalur wisata yang optimal bagi masyarakat dan wisatawan luar kota yang akan berwisata di kota Surabaya dengan menggunakan transportasi umum. Transportasi umum yang akan digunakan sebagai objek penelitian adalah bis kota di Surabaya. Dengan adanya rekomendasi jalur wisata ini, diharapkan masyarakat dan wisatawan luar kota Surabaya yang akan berwisata dapat beralih menggunakan transportasi umum sebagai pilihan utamanya. Sehingga seiring dengan peningkatan penggunaan transportasi umum dapat mengurangi kemacetan lalu lintas di kota Surabaya.

Orienteering Problem (OP) merupakan penyelesaian permasalahan optimasi jalur wisata yang akan digunakan, dikarenakan banyaknya keberhasilan penggunaan OP untuk menyelesaikan permasalahan optimasi jalur transportasi. *Orienteering Problem* (OP) merupakan gabungan dari pemilihan simpul dan menentukan jalur terpendek dari tiap simpul yang telah dipilih dengan jangka waktu yang telah disediakan. Sehingga, OP dapat dilihat sebagai kombinasi antara *Knapsack Problem* (KP) dengan *Travelling Salesman Problem* (TSP). Dalam TSP, kita mencari jalur terpendek untuk mengunjungi semua simpul, sedangkan dalam OP kita tidak perlu mengunjungi semua simpul yang ada sebelum mencapai tujuan[3].

Solusi permasalahan OP akan diselesaikan dengan pendekatan heuristik menggunakan suatu algoritma yang secara interaktif akan menghasilkan solusi yang akan mendekati optimal. Pendekatan heuristik menghasilkan perhitungan yang cepat karena dilakukan dengan membatasi pencarian dengan mengurangi jumlah alternatif yang ada[4]. Salah satu algoritma heuristik yang dapat digunakan adalah Algoritma Genetika (AG). AG sendiri sudah sering digunakan dalam menyelesaikan

permasalahan optimasi. Kelebihan dari AG adalah pada cara kerjanya yang paralel. AG bekerja dalam ruang pencarian yang menggunakan banyak individu sekaligus, sehingga kemungkinan AG untuk terjebak pada ekstrim lokal lebih kecil dibandingkan metode lain. Kekurangan dari AG adalah dalam hal waktu komputasi karena harus melakukan evaluasi *fitness* pada semua solusi di setiap iterasinya, sehingga AG bisa lebih lambat dibandingkan dengan metode lain[5]. Untuk mengatasi kekurangan pada AG, Moscato mengembangkannya menjadi Algoritma Memetika[6].

Algoritma Memetika (AM) merupakan turunan dari Algoritma Genetika. Karakteristik dasar dari AM menyerupai karakteristik dari AG. Hanya saja AM terdapat proses lanjutan berupa pencarian lokal pada seluruh individu dalam populasi. Peran pencarian lokal di dalam AM adalah menemukan lokal optimum lebih efisien dari Algoritma Genetika[7]. Diharapkan dengan menggunakan Algoritma Memetika, solusi yang dihasilkan untuk mendapatkan rekomendasi jalur wisata optimal menjadi lebih baik.

Berdasarkan kelebihan dari Algoritma Memetika maka dibuatlah tugas akhir dengan menggunakan Orienteering Problem sebagai model permasalahan untuk membuat solusi menggunakan Algoritma Memetika untuk mencari solusi yang optimal.

1.2. Perumusan Masalah

Rumusan masalah yang akan diselesaikan dalam tugas akhir ini adalah:

a. Penyelesaian permasalahan *Orienteering Problem* rute perjalanan wisata dengan bis kota di Kota Surabaya.
b. Desain Algoritma Memetika yang sesuai untuk permasalahan *Orienteering Problem*.
c. Perbandingan hasil Algoritma Memetika dengan Algoritma Genetika.

1.3. Batasan Tugas Akhir

Beberapa batas yang diberlakukan dalam Tugas Akhir ini adalah:

a. Objek yang digunakan dalam tugas akhir ini adalah transportasi umum bis kota di kota Surabaya.
b. Trayek yang digunakan sebagai objek tugas akhir sebanyak 10 trayek bis kota Surabaya dengan daerah asal-tujuan beserta kodenya, yaitu jalur Purabaya–Ngagel–Semut (A), Purabaya–Bratang (D), Purabaya–Joyoboyo (E), Purabaya–Kupang–Raden Saleh–JMP (F), Purabaya–Darmo–Indrapura–Perak (P1), Sidoarjo–Dupak–Rajawali–JMP (P3), Purabaya–Dupak–Perak (P4), Purabaya–Dupak–JMP (P5), Purabaya–Kupang–Demak–Tambak Oso Wilangun (P6), Purabaya–Tambak Oso Wilangun (P8).
c. Data uji coba berupa jarak dan waktu pada setiap simpul/titik diperoleh dengan menggunakan aplikasi Google Maps, dimana simpul/titik merepresentasikan tempat yang berpotensi ramai oleh para wisatawan, seperti pusat perbelanjaan, terminal, taman, museum, dan tempat wisata lainnya.
d. Waktu tempuh oleh bis kota hanya dipengaruhi oleh waktu perjalanan dari simpul/titik ke simpul/titik lainnya yang didapatkan dari waktu tempuh *Google Maps* dan tidak dipengaruhi oleh kondisi jalur, tingkat kemacetan, dan waktu saat bis kota berhenti pada sebuah simpul .
e. Data skor pada setiap simpul/titik diperoleh dengan menggunakan aplikasi *Google Trends*, dimana skor berada pada rentang nilai 0 hingga 100 yang merupakan hasil pencarian rata-rata dalam 1 minggu selama tahun 2018.
f. Tugas Akhir ini tidak melibatkan sisi psikologis dan finansial serta banyaknya jumlah penumpang yang menggunakan bis kota pada masing-masing trayek.

1.4. Tujuan Tugas Akhir

Tujuan dari tugas akhir ini adalah mengimplementasikan optimasi trayek bis kota di surabaya menggunakan algoritma memetika sebagai solusi penyelesaian *Orienteering Problem*.

1.5. Manfaat Tugas Akhir

Manfaat yang diharapkan dari tugas akhir ini adalah tersedianya sebuah aplikasi yang mampu memberikan rekomendasi rute perjalanan yang optimal ke berbagai tempat wisata di kota Surabaya menggunakan bis kota.

1.6. Relevansi

Tugas akhir ini dapat dijadikan solusi yang tepat untuk mengatasi permasalahan kemacetan yang terjadi di kota Surabaya. Adanya tugas akhir ini diharapkan dapat membantu masyarakat untuk melakukan perjalanan wisata di kota Surabaya, khususnya bagi wisatawan luar kota Surabaya yang ingin menggunakan bis kota sebagai transportasi namun belum memiliki pengetahuan yang cukup mengenai jalur transportasi bis kota yang ada di Surabaya. Sehingga dapat meningkatkan penggunaan transportasi umum dan mengurangi kemacetan lalu lintas di kota Surabaya. Selain itu dengan menggunakan model dari Orienteering Problem dan Algoritma Memetika diharapkan dapat menjadi sumber pustaka untuk penelitian terkait optimasi jalur transportasi dengan metode sejenis. Topik tugas akhir ini juga berkaitan dengan bidang Pemodelan dan Analisis Sistem di laboratorium Rekayasa Data dan Inteligensi Bisnis (RDIB) Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi.

"Halaman ini sengaja dikosongkan"

BAB II
DASAR TEORI DAN TINJAUAN PUSTAKA

Bab ini menjelaskan beberapa teori yang digunakan sebagai dasar pengerjaan Tugas Akhir. Selain itu, bab ini juga menjelaskan tinjauan singkat tehadap beberapa literatur terkait.

2.1. Dasar Teori

Berikut ini dijabarkan dasar-dasar teori yang digunakan dalam pengerjaan tugas akhir, yaitu:

2.1.1 Optimasi

Optimasi adalah sebuah disiplin ilmu dalam matematika terapan yang berhubungan dengan model pengoptimalan, sifat matematisnya (teori optimasi), dan pengembangan dan implementasi algoritma (analisis numerik dan desain algoritmik). Menentukan dan merumuskan sebuah masalah, membangun model matematika yang cocok dan mendapatkan solusi yang sesuai, menguji dan memodifikasi model, dan akhirnya mengimplementasikan solusi model ke dalam situasi masalah sebenarnya. Fase-fase ini merupakan pendekatan riset operasi[8].

Optimalisasi adalah disiplin ilmu dalam matematika terapan yang berhubungan dengan masalah optimasi, atau yang biasa disebut program matematis. Dalam masalah optimasi atau program matematis kita berusaha untuk meminimalkan atau memaksimalkan fungsi (nilai fakta) lebih dari satu set variabel (keputusan) dengan berdasarkan pada batasan yang telah ditentukan[8].

Dalam suatu problem optimasi diusahakan untuk memaksimalkan ataupun meminimalkan suatu besaran spesifik sebagai “tujuan” (objective), yang tergantung dari input sejumlah variabel keputusan[9].

Setiap problem optimasi memiliki dua bagian penting yaitu fungsi tujuan (*objective function*) serta serangkaian kendala (*constraints*). Fungsi tujuan menjelaskan kriteria yang ingin dicapai oleh sistem. Sedangkan kendala menjelaskan proses atau sistem yang sedang didesain atau dianalisis[10].

Keputusan yang optimal menurut Mays dan Tung (1992) berupa serangkaian nilai variabel keputusan yang memberikan respons optimal terhadap fungsi tujuan serta masih memenuhi kendala[10].

2.1.2 Algoritma Dijkstra

Algoritma ini ditemukan oleh E. W. Dijkstra pada tahun 1959 dengan tujuan untuk menyelesaikan *minimal spanning tree problem* dan *shortest path* problem[11]. Algoritma ini digunakan untuk menyelesaikan permasalahan dalam mencari rute terpendek dari titik awal (sumber) ke seluruh titik tujuan dalam sebuah *weighted graph* baik itu memiliki arah (*directed*) atau tidak (*undirected*). Ketentuan dalam menggunakan algoritma ini adalah graph yang digunakan tidak boleh memiliki bobot negatif di setiap garis penghubung antar titik[12].

2.1.3 Orienteering Problem

Kata *Orienteering* awalnya merupakan sebuah permainan luar ruangan yang biasanya dimainkan di area hutan. Dalam permainan ini, titik/simpul awal dan titik/simpul akhir ditentukan bersama dengan lokasi lainnya. Lokasi-lokasi ini mempunyai nilai yang telah ditentukan. Pemain diharuskan untuk mendatangi lokasi-lokasi dari titik awal hingga akhir dengan waktu yang telah disediakan yang bertujuan untuk mendapatkan total skor yang maksimal[13].

Orienteering Problem merupakan sebuah kombinasi dari pemilihan simpul dan penentuan jalur tercepat antara simpul-simpul yang terpilih. Tujuannya yaitu memaksimalkan total skor yang dikumpulkan dari simpul yang dikunjungi. Permasalahannya tidak semua simpul yang tersedia dapat

dikunjungi dikarenakan terbatasnya anggaran waktu. Oleh karena itu, OP dapat dilihat sebagai kombinasi antara dua masalah kombinatorial klasi, yaitu *Knapsack Problem* dan *Traveling Salesman Problem*[3].

Versi sederhana dari OP dapat dijelaskan sebagai berikut : Diberikan sejumlah *n* simpul di sebuah pesawat, dimana setiap simpul memiliki skor $s(i) \geq 0$ dan skor simpul awal = skor simpul akhir yaitu 0 ($s(1) = s(n) = 0$), maka temukan rute dengan skor maksimal dengan bermula pada simpul awal dan berakhir pada simpul akhir selama durasi yang telah dilalui dan tidak boleh melewati batas total durasi maksimal (TMAX)[13].

Fungsi tujuan dari OP yaitu memaksimalkan total skor yang dikumpulkan. Fungsi tujuan dapat dirumuskan dalam bentuk Persamaan 2.1. Selanjutnya terdapat batasan-batasan yang harus dipenuhi. Pada permasalahan OP batasan (1) yaitu memastikan bahwa permainan dimulai dari simpul awal dan berakhir pada simpul akhir |N|. Lalu batasan (2) menjamin bahwa setiap jalur terhubung dan setiap simpul hanya dapat dikunjungi 1 kali. Batasan (3) yakni total waktu tempuh dari simpul awal hingga akhir harus lebih kecil dari total waktu tempuh maksimal. Batasan (4) dan (5) mencegah terjadinya *subtours*, yaitu kondisi dimana lintasan dimulai dan berakhir pada simpul yang sama sehingga lintasan berbentuk sirkuit. Batasan (1), (2), (3), (4), dan (5) secara berturut-turut dapat dilihat pada Persamaan 2.2, 2.3, 2.4, 2.5, dan 2.6.

$$Maximize \sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} S_i X_{ij}$$

Persamaan 2.1 Fungsi Tujuan Orienteering Problem

$$\sum_{j=2}^{|N|} X_{1j} = \sum_{i=1}^{|N|-1} X_{i|N|} = 1$$

Persamaan 2.2 Batasan (1)

$$\sum_{i=1}^{|N|-1} X_{ik} = \sum_{j=2}^{|N|} X_{kj} \leq 1; \forall k = 2, \dots, (|N| - 1)$$

Persamaan 2.3 Batasan (2)

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max},$$

Persamaan 2.4 Batasan (3)

$$2 \leq u_i \leq N;\ \forall i = 2, \dots, N,$$

Persamaan 2.5 Batasan (4)

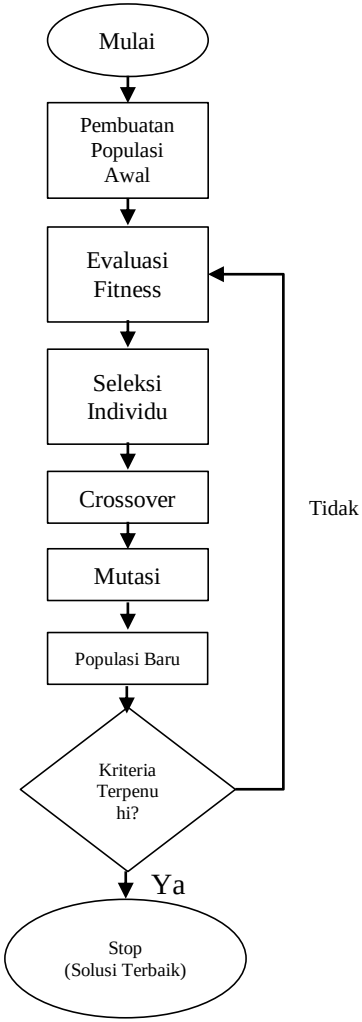
$$u_i - u_j + 1 \leq (N - 1)\left(1 - x_{ij}\right);\ \forall i, j = 2, \dots, N,$$

Persamaan 2.6 Batasan (5)

2.1.4 Algoritma Genetika

Algoritma Genetika adalah sebuah algoritma *meta-heuristic* yang didasarkan pada teori evolusi Darwin. Algoritma ini dimodelkan pada interpretasi yang relatif sederhana dari proses evolusi; meskipun begitu, penggunaan algoritma ini telah terbukti mampu menjadi teknik optimasi yang kuat di berbagai penerapan[14].

Sebagai salah satu teknik optimasi, Algoritma Genetika secara serentak memeriksa dan memanipulasi satu set kemungkinan solusi. Selama 1 iterasi algoritma, proses seleksi, reproduksi, dan mutasi masing-masing berlangsung untuk menghasilkan solusi pada generasi berikutnya. Gambar 2.1 merupakan siklus dari Algoritma Genetika.



Gambar 2.1 Siklus Algoritma Genetika

Algoritma Genetika dimulai dengan memilih secara acak populasi yang terdiri dari beberapa individu/kromosom. Algoritma ini menggunakan populasi saat ini untuk membuat

populasi baru sehingga individu/kromosom dalam generasi baru lebih baik dari populasi sebelumnya. Proses seleksi menentukan individu/kromosom saat ini yang akan digunakan untuk proses perkawinan silang. Lalu proses perkawinan silang menentukan pembentukan individu baru untuk generasi selanjutnya. Pada proses tersebut dipilih 2 individu yang dipasangkan sebagai induk. Sebuah probabilitas mutasi ditentukan saat akan menjalankan algoritma ini. Proses perkawinan silang dan mutasi ini memastikan bahwa Algoritma Genetika dapat menjelajahi solusi yang kemungkinan belum ada pada populasi. Membuat seluruh ruang pencarian dapat dijangkau, meskipun ukuran populasi terbatas[14]. Berikut merupakan penjelasan lebih lanjut mengenai Algoritma Genetika[15].

a. Pembangkitan Populasi Awal

Proses ini merupakan tahapan untuk membangkitkan sejumlah individu (solusi) secara acak atau dengan prosedur tertentu. Ukuran dari populasi awal tergantung pada kondisi permasalahan yang ingin diselesaikan.

b. Evaluasi Fitness

Berdasarkan nilai fitness, pada tahapan ini akan dilakukan evaluasi pada populasi awal maupun populasi baru yang nantinya terbentuk. Untuk mengetahui pencapaian nilai optimum dapat dilihat dari seberapa tinggi nilai fitness.

c. Seleksi Individu

Proses ini digunakan untuk memilih individu-individu dalam suatu populasi untuk dijadikan induk dalam proses kawin silang (*crossover*). Terdapat beberapa teknik yang digunakan dalam proses seleksi, yaitu seleksi dengan Roda Roulette (*Roulette Wheel Selection*), seleksi berdasarkan Ranking Fitness (*Rank-based Fitness*), seleksi *Stochastic Universal Sampling*, seleksi Lokal (*Local Selection*), seleksi dengan Pemotongan (*Truncation Selection*), seleksi dengan Turnamen (*Tournament Selection*).

d. Perkawinan Silang (*Crossover*)

Proses ini melibatkan dua induk yang akan melakukan perkawinan silang (menggunakan operasi pertukaran, aritmatika) untuk membentuk individu (solusi) baru. Individu baru (keturunan) diharapkan dapat memiliki kualitas yang lebih baik dibandingkan induknya. Proses ini disebut juga sebagai proses rekombinasi. Dalam proses ini memiliki beberapa tahapan.

a) Menentukan probabilitas dari perkawinan silang (*crossover*).
b) Memunculkan bilangan acak sebanyak i.
c) Membandingkan bilangan acak tersebut dengan probabilitas *crossover* yang telah ditentukan.
d) Jika bilangan acak ke-i kurang dari nilai probabilitas *crossover* maka terpilih calon induk dan *crossover* dapat dilakukan.

Ada beberapa jenis proses perkawinan silang yang sering digunakan diantaranya yaitu *Ordered Based Crossover*, *One-Point Crossover*, *Multi-Point Crossover*, *N-Point Crossover*, *Injection Crossover*.

e. Mutasi

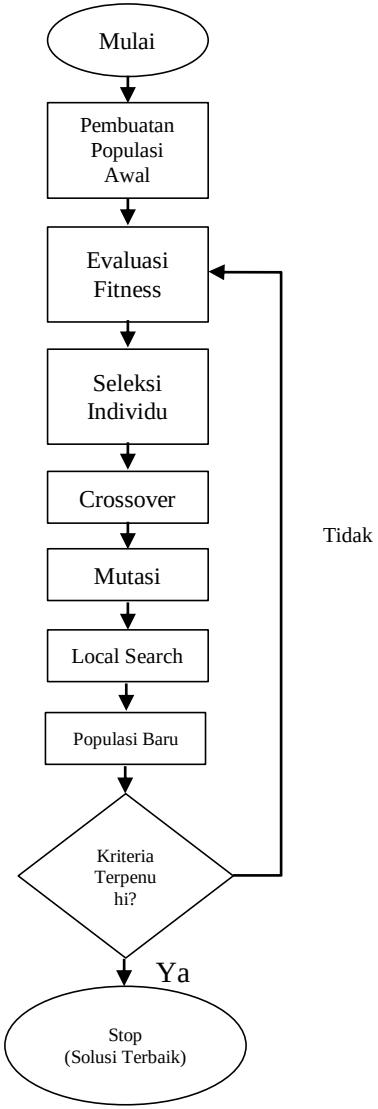
Proses ini berperan untuk menggantikan gen yang hilang akibat dari proses seleksi sehingga dapat juga memungkinkan munculnya kembali gen yang tidak muncul pada inisialisasi populasi. Individu dimutasi dengan menambahkan nilai acak yang sangat kecil dengan probabilitas yang rendah. Jika probabilitas mutasi terlalu kecil, banyak gen yang mungkin berguna tidak pernah dievaluasi. Namun bila probabilitas mutasi terlalu besar, maka akan terlalu banyak gangguan acak, sehingga individu baru dapat kehilangan kemiripan dengan induknya. Ada beberapa jenis mutasi diantaranya yaitu mutasi biner, mutasi bilangan real, dan mutasi kromosom permutasi.

Setelah menyelesaikan semua proses di atas, maka akan terbentuk populasi baru. Populasi baru tersebut akan menjadi populasi awal untuk generasi (iterasi) selanjutnya. Proses tersebut akan terus berulang hingga jumlah parameter generasi sudah sesuai dengan yang ditentukan.

2.1.5 Algoritma Memetika

Algoritma Memetika merupakan kombinasi antara teknik global berbasis populasi dan pencarian lokal yang dilakukan masing-masing individu. Seperti Algoritma Genetika, Algoritma Memetika juga merupakan pendekatan berbasis populasi. Karena Algoritma Memetika merupakan suatu pendekatan, maka algoritma ini termasuk dalam metode heuristik yakni suatu metode yang digunakan ketika ukuran ruang pencarian solusi sulit dikontrol secara eksas dan belum ada algoritma yang dapat mencari solusi optimal secara efektif[14][16].

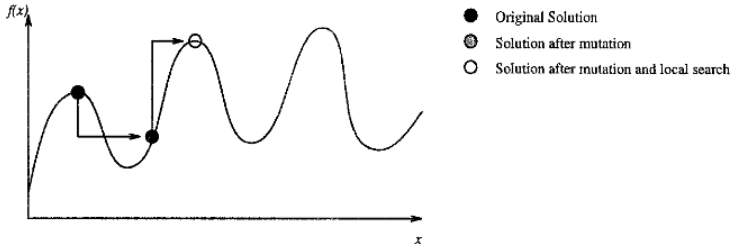
Penerapan Algoritma Memetika mempunyai dasaran yang sama dengan Algoritma Genetika, akan tetapi setelah proses mutasi dilanjutkan dengan pencarian lokal pada seluruh individu dalam populasi. Peran pencarian lokal di dalam Algoritma Memetika adalah menemukan lokal optimum lebih efisien dari Algoritma Genetika. Penambahan pencarian lokal ke dalam operasi genetika akan menambah beban komputasi, namun hal tersebut dapat diatasi dengan mengurangi ruang pencarian yang harus dieksplorasi untuk menemukan solusi optimum[7]. Gambar 2.2 merupakan siklus dari Algoritma Memetika.



Gambar 2.2 Siklus Algoritma Memetika

Seperti pada Gambar 2.3, dicontohkan solusi awal yang dibangkitkan selanjutnya dilakukan mutasi terhadap solusi yang

ada sehingga keluar dari lokal optimum (fungsi objektif lebih buruk), lalu dilakukan perbaikan lokal pada solusi tersebut sehingga didapatkan solusi lokal optimum[7].



Gambar 2.3 Contoh Operasi Memetika

2.1.6 Algoritma Pencarian Lokal

Sebuah algoritma pencarian lokal dimulai dari konfigurasi solusi yang telah dihasilkan secara acak atau dibangun dari algoritma lain, dalam hal ini hasil hasil mutasi. Selanjutnya, solusi tersebut dilakukan perulangan menggunakan setiap langkah transisi berdasarkan lingkungan solusi. Transisi yang mengarah pada hasil konfigurasi solusi yang lebih baik akan diterima. Konfigurasi solusi baru ini akan dijadikan konfigurasi solusi awal pada langkah selanjutnya. Jika tidak, maka konfigurasi saat ini akan disimpan. Proses ini akan diulang hingga kriteria penghentian tertentu telah terpenuhi. Kriteria penghentian yang biasanya dipakai adalah jumlah pengulangan yang telah ditentukan, tidak menemukan peningkatan pada pengulangan terakhir, atau mekanisme yang lebih kompleks berdasarkan dari perkiraan kemungkinan menjadi solusi lokal optimal[17].

2.1.7 Algoritma *Hill Climbing*

Hill Climbing adalah salah satu algoritma pencarian lokal. Cara kerjanya adalah menentukan langkah berikutnya dengan menempatkan simpul yang akan muncul sedekat mungkin dengan sasarannya. Proses pengujian dilakukan dengan menggunakan fungsi heuristik. Pembangkitan keadaan berikutnya sangat tergantung pada feedback dari prosedur

pengetesan. Tes yang berupa fungsi heuristik ini akan menunjukkan seberapa baiknya nilai terkaan yang diambil terhadap keadaan keadaan lainnya yang mungkin[18].

Terdapat dua jenis *Hill Climbing*, yaitu *Simple Hill Climbing* dan *Steepest-Ascent Hill Climbing*. Pada *Simple Hill Climbing*, *next state* akan ditentukan dengan membandingkan *current state* dengan satu penerus. Apabila ditemukan penerus baru yang lebih baik dari kondisi saat itu maka penerus tersebut akan menjadi *next state*. Sedangkan pada *Steepest-Ascent Hill Climbing* dalam menentukan *next state*, *current state* langsung dibandingkan dengan semua penerus yang ada di dekatnya, sehingga *next state* yang diperoleh merupakan penerus yang paling baik serta mendekati hasil optimasi yang diharapkan. Adapun tahap-tahap penerapan *Hill Climbing* sebagai berikut[19]:

1. Evaluasi state awal, jika state awal sama dengan tujuan, maka proses berhenti. Jika tidak, maka lanjutkan proses dengan membuat state awal sebagai state sekarang.
2. Mengerjakan langkah berikut sampai solusi ditemukan atau sampai tidak ada lagi operator baru yang dapat digunakan dalam state sekarang:
    a. Mencari sebuah operator yang belum pernah digunakan dalam state sekarang dan gunakan operator tersebut untuk membentuk state baru.
    b. Evaluasi state baru. Jika state baru adalah tujuan, maka proses berhenti. Jika state baru merupakan state yang lebih baik dari state sekarang, maka buat state baru menjadi state sekarang. Jika state baru tidak lebih baik dari state sekarang, maka lanjutkan ke langkah 2.

2.1.8 Optimasi

Simulated Annealing adalah algoritma metaheuristik yang dapat menganalogikan teori fisika ketika proses annealing baja, yaitu ketika baja dipanaskan hingga mencapai titik didihnya, atom dalam baja akan bergerak bebas. Kemudian baja didinginkan

bertahap hingga mencapai titik tentu dengan tujuan energinya berkurang secara perlahan. Semakin rendah suhu baja tersebut, atom dalam baja akan lebih terstruktur hingga membentuk kristal yang memiliki energi paling optimum[20].

Metode *Simulated Annealing* dimulai dengan solusi awal yang dihasilkan secara acak. Kemudian dilakukan proses evaluasi lingkungan solusi awal. Apabila solusi baru lebih optimal daripada solusi awal, maka solusi akan diterima sebagai solusi sementara. Jika tidak lebih optimal, maka solusi akan melalui fungsi pengontrol. Fungsi pengontrol dalam Simulated Annealing dinyatakan dengan persamaan Boltzman, yaitu $P = e^{-\frac{c}{t}}$ dimana P adalah Probabilitas Boltzman, e adalah bilangan eksponensial, c adalah perbedaan evaluasi fungsi tujuan antara solusi dengan kandidat solusi, dan t adalah parameter suhu.

Simulated Annealing bergantung pada beberapa parameter dalam setiap iterasi, diantara adalah,

a. Suhu awal dan akhir, yaitu suhu tertinggi dan terendah saat baja dipanaskan. Semakin lebar selisih antara suhu awal dan akhir, maka semakin luas area solusi.
b. Jumlah iterasi setiap penurunan suhu, yaitu banyaknya iterasi setiap kondisi suhu tertentu. Setelah mencapai jumlah iterasi tersebut, maka suhu akan diturunkan secara berkala.
c. *Cooling rate*, yaitu nilai koefisien α yang berfungsi untuk menurunkan suhu. Berdasarkan [21], nilai koefisien α bervariasi dengan nilai $0{,}8 \leq \alpha < 1$.

2.2. Tinjauan Pustaka

Dalam Tugas Akhir ini dijelaskan lima tinjauan pustaka yang terkait dengan penyelesaian *Orienteering Problem*. Yang pertama adalah tesis dengan judul "Advanced Traveler Information Systems: Optimasi Rencana Perjalanan dengan Model Orienteering Problem dan Genetic Algorithm (Studi

Kasus: Trayek Angkot Surabaya)" oleh I Wayan Angga Kusuma Yoga pada tahun 2017)[22]. Dalam penelitian ini digunakan Algoritma Genetika sebagai penyelesaian permasalahan Orienteering Problem untuk mencari solusi rute perjalanan menggunakan transportasi angkot di kota Surabaya. Algoritma diimplementasikan melalui bahasa pemrograman Java. Solusi yang diinginkan penulis yaitu mendapatkan rute perjalanan yang optimal dengan memaksimalkan skor di setiap simpul yang dilalui dengan batasan waktu tempuh yang telah ditentukan sebelumnya. Tesis ini sangat berkaitan dengan tugas akhir yang dikerjakan penulis dikarenakan tugas akhir ini merupakan pengembangan dari tesis tersebut. Dalam tesis tersebut, permasalahan yang diambil sama yakni *Orienteering Problem* dan implementasi Algoritma Genetika digunakan untuk membandingkan hasilnya dengan algoritma yang diterapkan pada tugas akhir ini.

Yang kedua adalah artikel jurnal dengan judul "Orienteering Problem: A Survey of Recent Variants, Solution Approaches and Applications" oleh Aldy Gunawan, Hoong Chuin Lau, dan Pieter Vansteenwegen pada tahun 2016[3]. Paper ini menjelaskan sebuah penelitian tentang variasi dan perluasan model Orienteering Problem yang tidak pernah dilakukan survey sebelumnya selama 10 tahun terakhir. Paper ini mencakup studi literatur dari banyak variasi penelitian OP. Selain itu paper ini membahas mengenai solusi model dari tiap-tiap metode OP yang digunakan, membandingkan dengan solusi model lain, dan contoh pengaplikasian praktikalnya. Sehingga pada paper ini penulis mendapatkan gambaran solusi model dari OP yang digunakan pada tugas akhir.

Yang ketiga adalah artikel jurnal dengan judul "A Memetic Algorithm for The Orienteering Problem with Mandatory Visits and Exclusionary Constraints" oleh Yongliang Lu, Una Benlic, dan Qinghua Wu pada tahun 2017[23]. Penelitian ini membahas mengenai solusi penyelesaian Orienteering Problem khusus yaitu Orienteeri Penelitian ini membahas mengenai solusi

penyelesaian Orienteering Problem khusus yaitu Orienteering Problem with Mandatory Visits and Exclusionary Constraints (OPMVEC) yang diatasi dengan menggunakan Algoritma Memetika. Hasil komputasi menggunakan metode ini dimana mereka menggunakan 16 kelas dari 340 contoh benchmark menunjukkan bahwa Algoritma Memetika lebih unggul dari algoritma Variable Neighborhood Search (VNS) yang ada.ng Problem with Mandatory Visits and Exclusionary Constraints (OPMVEC) yang diatasi dengan menggunakan Algoritma Memetika. Hasil komputasi menggunakan metode ini dimana mereka menggunakan 16 kelas dari 340 contoh benchmark menunjukkan bahwa Algoritma Memetika lebih unggul dari algoritma Variable Neighborhood Search (VNS) yang ada. Penelitian ini dapat menjadi referensi untuk membandingkan hasil solusi dari AM dengan hasil solusi dari AG.

Yang keempat adalah paper dengan judul "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem" oleh Peter Merz dan Bernd Freisleben pada tahun 1999[24]. Penelitian ini membahas mengenai implementasi Algoritma Memetika untuk memecahkan beberapa contoh permasalahan QAP (Quadratic Assignment Problem) Parameter, operator, dan perbaikan lokal dideskripsikan. Kinerja Algoritma Memetika diselidiki melalui sebuah set QAP *instances* dan performanya dibandingkan dengan 3 pendekatan heuristic yang sangat baik untuk QAP: *reactive tabu search*, *robust tabu search*, dan *fast ant colony system*. Hasilnya, Algoritma Memetika mampu mengungguli semua pendekatan heuristic untuk semua QAP *instances*. Selain itu, pendekatan ini terbukti sangat kuat, karena semua solusi terbaik yang diketahui dapat ditemukan dengan rata-rata waktu *running* yang pendek. Penelitian ini dapat menjadi tambahan bukti kuat bahwa Algoritma Memetika memungkinkan dapat menemukan solusi optimal lebih cepat dalam permasalahan Orienteering Problem.

Yang terakhir adalah artikel jurnal dengan judul "A Comparison between Memetic algorithm and Genetic algorithm for the Cryptanalysis of Simplified Data Encryption Standard Algorithm" oleh Poonam Garg pada tahun 2009[14]. Penelitian ini membahas mengenai implementasi Algoritma Genetika dan Algoritma Memetika untuk *cryptanalysis* dari algoritma standar enkripsi data. Tujuan penelitian ini yaitu membandingkan performa Algoritma Memetika dengan Algoritma Genetika. Hasil eksperimen menunjukkan bahwa Algoritma Memetika sedikit lebih unggul dalam menemukan jumlah kunci secara akurat dibandingkan dengan Algoritma Genetika. Dengan melakukan perbandingan hasil dari kedua algoritma tersebut, terlihat bahwa pendekatan Algoritma Memetika masih lebih baik dari Algoritma Genetika. Meskipun permasalahan dalam penelitian tersebut bukan *Orienteering Problem*, namun patut dicoba untuk membuktikan apakah Algoritma Memetika juga menghasilkan solusi optimal lebih baik dengan permasalahan *Orienteering Problem*.

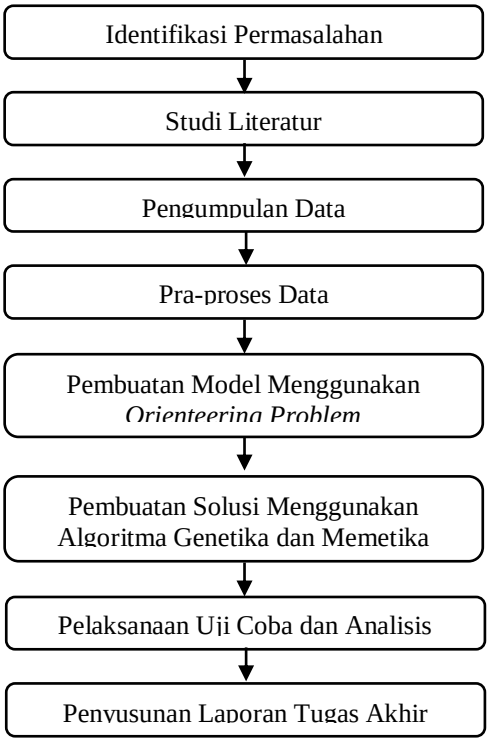
"Halaman ini sengaja dikosongkan"

BAB III
METODOLOGI PENELITIAN

Bab ini menjelaskan mengenai metodologi dan tahapan-tahapan apa saja yang dilakukan dalam pengerjaan tugas akhir ini.

3.1 Tahapan Pelaksanaan Tugas Akhir

Pada sub bab ini akan menjelaskan mengenai metodologi dalam pelaksanaan tugas akhir. Metodologi ini dapat dilihat pada Gambar 3.1.



Gambar 3.1 Metodologi Pengerjaan Tugas Akhir

3.1.1 Identifikasi Permasalahan

Pada tahap ini dilakukan identifikasi permasalahan dengan melakukan observasi kondisi lalu lintas saat ini dan mencari isu-isu mengenai kemacetan di Indonesia khususnya di kota Surabaya. Permasalahan kemacetan yang ditemukan terkait dengan besarnya peningkatan jumlah kendaraan yang ada di kota Surabaya. Ditemukan bahwa faktor utama yang menjadi penyebab kemacetan adalah pertambahan kendaraan bermotor di Surabaya yang bertambah sebanyak 17 ribu unit per bulan dengan rincian lebih dari 13 ribu unit sepeda motor dan 4 ribu unit mobil sedangkan setiap bulannya bahkan setiap tahunnya ruas jalan raya di kota Surabaya belum tentu bertambah[4]. Maka dari itu didapatkan topik permasalahan berupa optimalisasi jalur transportasi guna mengurangi kemacetan di kota Surabaya. Pihak pemerintah Surabaya telah membuat rencana untuk revitalisasi transportasi umum, maka digunakanlah objek penelitian bis kota pada tugas akhir ini. Tugas akhir ini dibatasi pada trayek bis kota di Surabaya sebanyak sepuluh jalur trayek yang didapatkan dari website kota Surabaya.

3.1.2 Studi Literatur

Studi literatur dilakukan dengan mengumpulkan berbagai referensi seperti penelitian sebelumnya, paper, buku, dan dokumen terkait. Pencarian referensi didasarkan pada topik permasalahan yang telah didapatkan pada proses sebelumnya. Pada tugas akhir ini diusulkan topik optimasi penentuan jalur terpendek dengan menggunakan angkutan umum bis kota dengan menggunakan model optimasi tertentu. Selanjutnya dilakukan studi literatur kembali untuk memahami konsep dasar dari optimasi. Lalu dilakukan gap analysis untuk mencari kelebihan dan kekurangan dari masing-masing referensi dengan topik permasalahan serupa. Sehingga didapatkan metode terbaik yang dapat digunakan sebagai acuan pengerjaan tugas akhir ini. Sehingga diusulkanlah model yang akan digunakan yaitu Orienteering Problem. Selanjutnya dilakukan studi literatur lebih mendalam untuk menentukan solusi dari

pemodelan yang dibuat. Sehingga didapatkanlah solusi dari model optimasi ini dengan membandingkan 3 model menggunakan Algoritma Genetika, Algoritma Memetika Hill Climbing, dan Algoritma Memetika Simulated Annealing.

3.1.3 Pengumpulan Data

Setelah menemukan metode yang sesuai, tahapan selanjutnya adalah mengumpulkan data yang dibutuhkan dalam pengerjaan Tugas Akhir ini karena data merupakan komponen utama dalam melakukan penelitian ini. Data yang dibutuhkan harus sesuai dengan topik dan batasan permasalahan yang telah ditetapkan. Data yang digunakan yaitu data trayek berupa enam jalur bis kota Surabaya yang didapatkan dari website Pemerintah Kota Surabaya dengan rincian sebagai berikut :

1. Bis Kota jurusan Purabaya – Ngagel – Semut dengan kode trayek (A)
2. Bis Kota jurusan Purabaya – Bratang dengan kode trayek (D)
3. Bis Kota jurusan Purabaya – Joyoboyo dengan kode trayek (E)
4. Bis Kota jurusan Purabaya – Jembatan Merah Plaza dengan kode trayek (F)
5. Bis Kota jurusan Purabaya – Darmo – Indrapura – Perak dengan kode trayek (P1)
6. Bis Kota jurusan Sidoarjo – Dupak – Rajawali – JMP via tol dengan kode trayek (P3)
7. Bis Kota jurusan Purabaya – Dupak – Perak via tol dengan kode trayek (P4)
8. Bis Kota jurusan Purabaya – Jembatan Merah Plaza via tol dengan kode trayek (P5)
9. Bis Kota jurusan Purabaya – Kupang – Demak – Tambak Oso Wilangun dengan kode trayek (P6).
10. Bis Kota jurusan Purabaya – Tambak Oso Wilangun via tol dengan kode trayek (P8)

Selanjutnya diperlukan data lain yang didapatkan menggunakan mesin pencari *Google Trends* dan aplikasi *Google Maps* sebagai berikut:

a. Data skor berasal dari tingkat popularitas dari setiap simpul yang didapatkan menggunakan aplikasi *Google Trends*. Nilai skor untuk setiap simpul bervariasi mulai dari 0 hingga 100 yang didapatkan dari hasil rata-rata pencarian per minggu dalam satu tahun (2018).
b. Data waktu tempuh dari setiap simpul yang terhubung didapatkan berdasarkan asumsi yang telah dibuat dan sesuai dengan waktu tempuh yang didapat melalui aplikasi *Google Maps*.

3.1.4 Pra Proses Data

Data yang didapatkan pada tahap sebelumnya masih belum diolah, sehingga untuk dapat digunakan dalam proses pemodelan, maka data tersebut harus diolah. Data diolah menggunakan aplikasi Paint untuk memberikan informasi visual mengenai jalur yang akan digunakan untuk proses pemodelan yakni model jaringan, tidak luput juga pemberian simpul-simpul pada jalur di peta. Selain itu input untuk pemodelan berupa catatan, data skor, dan waktu tempuh dibuat dalam aplikasi Microsoft Excel berupa sebuah matriks.

3.1.5 Pembuatan Model Menggunakan OP

Sebelum melakukan pemodelan, diperlukan asumsi terlebih dahulu untuk menentukan skor tiap simpul, waktu dan jarak tempuh antar simpul.

1. Simpul (*node*) adalah representasi dari tempat-tempat pemberhentian bis kota yang berpotensi padat pengunjung seperti tempat wisata, pusat perbelanjaan, museum, terminal, dan taman.
2. Jarak dari suatu titik ke titik lainnya yang membentuk sebuah garis disebut busur (arc). Busur mempunyai satuan yaitu meter (m)

3. Skor pada tiap simpul/titik direpresentasikan dalam hasil pencarian tempat tersebut melalui mesin pencari *Google Trends*. Semakin besar skor pada titik tersebut maka semakin terkenal tempat tersebut.
4. Waktu berhenti bis kota saat berada di simpul diasumsikan tidak ada (nol).

Lalu selanjutnya menentukan variabel-variabel yang bersangkutan seperti fungsi tujuan, batasan, dan variabel keputusan. Penjabarannya sebagai berikut:

a. Variabel Keputusan
   - Kunjungan dari satu tempat ke tempat tertentu sesuai dengan simpul yang dapat dilewati.

b. Fungsi Tujuan
   - Memaksimalkan jumlah skor yang dicapai selama melakukan perjalanan menggunakan bis kota Surabaya dari simpul awal hingga simpul akhir.

c. Batasan
   - Penetapan simpul awal dan akhir.
   - Tiap simpul harus saling terhubung dengan simpul lainnya dan simpul hanya dapat dikunjungi satu kali.
   - Perjalanan ditempuh tidak boleh melewati batas waktu tempuh maksimal.
   - Menghindari terjadinya subtours dimana pengguna kembali ke simpul awal.

3.1.6 Pembuatan Solusi Model Menggunakan Algoritma Genetika dan Memetika

Tahap ini merupakan tahap konstruksi solusi dari model yang telah dibuat pada tahap sebelumnya. Pada tahap ini, terdapat pembaruan waktu tempuh pada data yang telah diproses menggunakan Algoritma Dijkstra, lalu Algoritma Genetika dan Algoritma Memetika diaplikasikan dengan cara computing menggunakan bahasa pemrograman. Untuk tahapan ini akan dijelaskan sebagai berikut:

3.1.6.1 Pembaruan Nilai Waktu Tempuh Menggunakan Algoritma Dijkstra

Data waktu tempuh antar simpul dalam pembuatan solusi model AG dan AM hanya dapat digunakan apabila nilai waktu tempuh untuk seluruh simpul telah terisi. Pada kenyataannya pra-proses data dari data waktu tempuh hanya mencari waktu tempuh dari simpul-simpul yang terhubung secara langsung, sedangkan simpul yang tidak terhubung secara langsung tidak dapat dihitung nilai waktu tempuhnya sehingga dinyatakan dengan nilai 0. Hal ini dapat diantisipasi dengan melakukan perbaikan nilai waktu tempuh dengan cara mengisi nilai waktu tempuh simpul-simpul yang sebelumnya bernilai 0. Merubah nilai simpul yang tidak terhubung secara langsung dapat dilakukan dengan cara menambahkan nilai waktu tempuh dari simpul-simpul yang terhubung secara langsung di dalam jalur simpul yang tidak terhubung secara langsung. Penambahan nilai waktu tempuh tersebut didasarkan pada jalur dengan waktu tempuh tercepat yang dilalui. Pada implementasinya, digunakan Algoritma Dijkstra untuk mencari waktu tempuh tercepat. Hasil pencarian waktu tempuh akan diperbarui dalam matriks yang sudah ada menggantikan nilai 0. Hasil dari pembaruan nilai waktu tempuh dapat dilihat pada sub bab 6.3 dan LAMPIRAN D.

3.1.6.2 Pembuatan Solusi Model

a. Pendefinisian Individu dan Parameter Pembuatan Solusi

Sebelum menentukan populasi awal, dilakukan terlebih dahulu penentuan individu, dimana individu/kromosom direpresentasikan sebagai solusi rute bis kota dimana setiap gennya mewakili tempat-tempat (simpul) yang dikunjungi. Berdasarkan karakteristik dari model OP, maka setiap kromosom memiliki simpul awal dan akhir yang sama. Selain itu ditentukan juga parameter-parameter dalam AG dan AM seperti jumlah populasi, jumlah generasi, probabilitas perkawinan silang, probabilitas

mutasi, dan juga Tmax. Setiap parameter tersebut akan mempengaruhi optimumnya solusi akhir yang ditemukan.

b. Pembuatan Populasi Awal

Untuk membentuk populasi, dipilih secara acak sejumlah individu berdasarkan total waktu tempuh yang tidak boleh melebihi kapasitas waktu tempuh (Tmax)

c. Evaluasi Fitness

Berdasarkan nilai fitness, pada tahapan ini akan dilakukan evaluasi pada populasi awal maupun populasi baru yang nantinya terbentuk. Apabila populasi masih berbentuk populasi awal, maka tidak perlu dilakukan evaluasi terlebih dahulu. Untuk mengetahui pencapaian nilai optimum dapat dilihat dari seberapa tinggi nilai fitness. Individu yang memiliki nilai fitness tinggi akan bertahan hidup, sedangkan individu dengan nilai fitness rendah akan gugur.

d. Seleksi Individu

Seleksi dilakukan dengan memilih sejumlah individu dari populasi yang akan melanjutkan ke proses berikutnya, untuk dijadikan induk pada proses perkawinan silang. Proses seleksi menggunakan metode *roulette wheel selection* dimana akan diambil sejumlah individu yang *unique* pada populasi. Seleksi tersebut akan memilih individu dari populasi berdasarkan nilai fitness. Semakin tinggi nilai fitness maka semakin besar peluang individu tersebut terpilih. Individu-individu yang terbaik dalam kelompok ini akan diseleksi untuk menjadi induk.

e. Proses Perkawinan Silang (Crossover)

Setiap individu yang terpilih pada proses seleksi akan menjadi induk dimana setiap induk akan ditukar gennya dengan cara tertentu untuk mendapat individu baru yang disebut keturunan. Dalam tahap ini dilakukan *single-point crossover* pada sejumlah individu yang terpilih di tahap sebelumnya untuk mendapatkan keturunan (solusi) baru berdasarkan probabilitas crossover yang telah ditentukan sebelumnya. Proses dilakukan dengan memilih 2 induk, lalu memecah kromosom pada setiap induk menjadi 2

bagian yaitu kepala dan ekor. Kepala induk 1 dan ekor induk 2 digabung menjadi 1 begitu pula dengan kepala induk 2 dan ekor induk 1 digabung menjadi 1 untuk mendapatkan 2 proto child. Proto child tersebut akan disesuaikan ukurannya sehingga didapatkan 2 keturunan hasil perkawinan silang (offspring).

f. Proses Mutasi

Proses ini dilakukan pada keturunan hasil perkawinan silang dengan cara mengubah nilai gen pada kromosom menggunakan operator yang disediakan. Operator yang digunakan yaitu add, add-swap, add-remove, dan remove. Dari proses ini dihasilkan sejumlah keturunan hasil mutasi berdasarkan probabilitas mutasi.

g. Pencarian Lokal (Local Search)

Tahap inilah yang membuat perbedaan dari pembuatan solusi Algoritma Genetika dengan Algoritma Memetika. Setelah proses mutasi, pada AG langsung dilakukan proses pembentukan populasi baru, namun pada AM, akan dilakukan proses pencarian lokal sebelum pada akhirnya terjadi proses pembentukan populasi baru. Proses pencarian lokal dilakukan perulangan menggunakan setiap langkah transisi berdasarkan lingkungan solusi. Solusi yang digunakan adalah solusi hasil mutasi. Transisi yang mengarah pada hasil konfigurasi solusi yang lebih baik akan diterima. Konfigurasi solusi baru ini akan dijadikan konfigurasi solusi awal pada langkah selanjutnya. Jika tidak, maka konfigurasi saat ini akan disimpan. Proses ini akan diulang hingga kriteria pemberhentian tertentu telah terpenuhi. Kriteria penghentian yang dipakai adalah jumlah pengulangan yang telah ditentukan.

h. Pembentukan Populasi Baru

Gabungan dari individu pada populasi awal dan hasil pencarian lokal akan diseleksi kembali dengan mengambil individu terbaik (berdasarkan nilai fitness) sejumlah parameter jumlah populasi yang telah ditentukan sebelumnya. Populasi baru akan menjadi populasi awal dimana sub bab 3.1.6 proses c sampai proses g akan

diulang sebanyak parameter jumlah generasi yang telah ditentukan sebelumnya.

i. Pengambilan Solusi Terbaik

Individu terbaik berdasarkan nilai fitness hasil populasi dari perulangan terakhir (generasi terakhir) akan terpilih menjadi solusi akhir dari studi kasus permasalahan ini.

3.1.7 Pelaksanaan Uji Coba dan Analisis

Uji coba dilakukan dengan memvalidasi solusi yang dihasilkan oleh Algoritma Genetika dan Algoritma Memetika, mencari parameter terbaik untuk masing-masing algoritma, membandingkan performa algoritma dengan penggunaan parameter terbaiknya. Hasil dari uji coba akan ditampilkan dalam bentuk tabel dan grafik.

3.1.8 Penyusunan Laporan Tugas Akhir

Pada tahap ini, semua proses di atas akan disusun dalam bentuk laporan tugas akhir sebagai bentuk dokumentasi atas terlaksananya tugas akhir ini. Laporan Tugas Akhir ini mencakup:

a. Bab I Pendahuluan

Bab ini menjelaskan mengenai identifikasi permasalahan penelitian meliputi latar belakang masalah, rumusan masalah, batasan masalah, tujuan tugas akhir, manfaat tugas akhir, dan relevansi terhadap pengerjaan tugas akhir.

b. Bab II Dasar Teori dan Tinjauan Pustaka

Bab ini menjelaskan mengenai penelitian sebelumnya dan dasar teori yang dijadikan acuan atau landasan dalam pengerjaan tugas akhir ini.

c. Bab III Metodologi Penelitian

Bab ini menjelaskan mengenai tahapan – tahapan apa saja yang dilakukan dalam pengerjaan tugas akhir ini, input-process-

output dari setiap tahapan, serta deskripsi dan penjelasan dari tiap tahapan.

d. Bab IV Desain

Bab ini menjelaskan mengenai rancangan penelitian, rancangan bagaimana penelitian akan dilakukan, pemilihan objek penelitian, dan sebagainya.

e. Bab V Implementasi

Bab ini menjelaskan proses pelaksanaan penelitian, bagaimana penelitian dilakukan, penerapan strategi pelaksanaan, hambatan dan rintangan dalam pelaksanaan, dan sebagainya.

f. Bab VI Analisis dan Pembahasan

Bab ini menjelaskan pembahasan tentang penyelesaian permasalahan yang dikerjakan pada penelitian tugas akhir.

g. Bab VII Kesimpulan dan Saran

Bab ini menjelaskan kesimpulan dari apa yang telah dikerjakan dalam tugas akhir, serta saran yang ditujukan untuk kelengkapan penyempurnaan tugas akhir.

BAB IV
DESAIN

Pada bab ini menjelaskan bagaimana rancangan dari penelitian tugas akhir yang meliputi subjek dan objek dari tugas akhir, pemilihan subjek dan objek tugas akhir serta bagaimana tugas akhir akan dilakukan.

4.1. Pengumpulan dan Deskripsi Data

Untuk memudahkan proses pemodelan menggunakan Orinteering Problem diperlukan pengumpulan data yang merupakan data trayek bis kota Surabaya yang didapatkan dari website resmi Dinas Perhubungan Kota Surabaya[25]. Deskripsi data trayek yang digunakan berupa 10 kode bus beserta rutenya. Data trayek bis kota dijelaskan pada Tabel 4.1.

Tabel 4.1 Kode Bis Kota dan Rutenya

Kode	Rute
A	Purabaya/Bungurasih – Ahmad Yani – Stasiun Wonokromo – Ngagel – Raya Gubeng – Stasiun Gubeng – Kusuma Bangsa – D. Gembong – Kalianyar – Pengampon - Bunguran – Putar Kya Kya (Kembang Jepun-Kapasan) – Gembong – Pecindilan – Kembali dengan rute yang sama
D	Berangkat : Purabaya/Bungurasih – Ahmad Yani – Jemursari – Prapen – (D. Panjang Jiwo) – Nginden – Bratang Kembali : Bratang – Bratang Jaya – Barata Jaya XIX – Barata Jaya XVII – Nginden - (D. Panjang Jiwo) – Prapen – Jemursari – Ahmad Yani – Purabaya/Bungurasih
E	Purabaya/Bungurasih – Ahmad Yani – Wonokromo – Joyoboyo – Kembali dengan rute yang sama
F	Berangkat : Purabaya/Bungurasih – Ahmad Yani – Wonokromo – Diponegoro – (D. Kupang) – Pasar

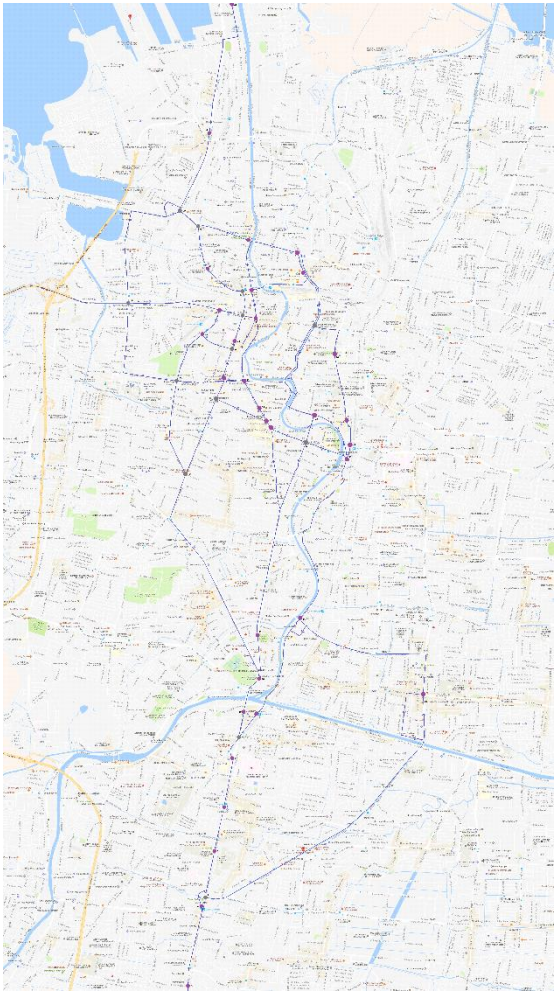
	Kembang – Arjuno – Semarang – Stasiun Pasar Turi - Raden Saleh – Bubutan – Indrapura – Rajawali – Jembatan Merah Plasa (JMP) Kembali : Jembatan Merah Plasa (JMP) – Jembatan Merah – Veteran – Pahlawan – Gemblongan – Siola – Praban – Bubutan – Raden Saleh – Stasiun Pasar Turi – Semarang – Arjuno – Pasar Kembang - (D. Kupang) – Diponegoro – Wonokromo – Ahmad Yani – Purabaya/Bungurasih
P1	Berangkat : Purabaya/Bungurasih – Ahmad Yani – Wonokromo – Darmo – Urip Sumoharjo – Basuki Rahmat – (D. Tunjungan / TP) – Embong Malang – Blauran – Bubutan – Indrapura – Rajawali – Perak Barat – Prapat Kurung – Kalimas Baru – Perak (Pelabuhan) Kembali : Perak (Pelabuhan) – Kalimas Baru – Prapat Kurung – Perak Timur – Rajawali – Jembatan Merah Plasa (JMP) - Jembatan Merah – Veteran – Pahlawan – Kramat Gantung – Siola – Tunjungan – Gubernur Suryo – Panglima Sudirman – Bambu Runcing – Panglima Sudirman – Urip Sumoharjo – Darmo – Wonokromo – Ahmad Yani – Purabaya/Bungurasih
P3	Berangkat : Terminal Sidoarjo – Pahlawan – (Gor Delta Sidoarjo) – Pahlawan – Masuk Tol Sidoarjo Keluar Tol Dupak – Pasar Loak - Dupak – Pasar Turi – Bubutan – Indrapura – Rajawali – Jembatan Merah Plasa (JMP) Kembali : Jembatan Merah Plasa (JMP) – Jembatan Merah – Veteran – Pahlawan – Tembaan – Pasar Turi – Dupak – Pasar Loak – Masuk Tol Dupak Keluar Tol Sidoarjo - Pahlawan – (Gor Delta Sidoarjo) – Pahlawan – Gajah Mada – Terminal Sidoarjo
P4	Berangkat : Purabaya/Bungurasih – Masuk Tol Waru Keluar Tol Dupak - Pasar Loak – Dupak –

	Demak – Gresik Gadukan – Gresik - Perak Barat – Prapat Kurung – Kalimas Baru – Perak (Pelabuhan) Kembali : Perak (Pelabuhan) – Kalimas Baru – Prapat Kurung – Perak Timur – Gresik – Gresik Gadukan – Demak – Dupak – Pasar Loak – Masuk Tol Dupak Keluar Tol Waru – Purabaya/Bungurasih
P5	Berangkat : Purabaya/Bungurasih – Masuk Tol Waru Keluar Tol Dupak - Pasar Loak – Dupak - Pasar Turi – Bubutan – Indrapura – Rajawali – Jembatan Merah Plasa (JMP) Kembali : Jembatan Merah Plasa (JMP) – Jembatan Merah – Veteran – Pahlawan – Tembaan – Pasar Turi – Dupak – Pasar Loak – Masuk Tol Dupak Keluar Tol Waru – Purabaya/Bungurasih
P6	Purabaya/Bungurasih – Ahmad Yani – Wonokromo – Diponegoro – (D. Kupang) – Pasar Kembang – Arjuno – Tembok Dukuh – Demak – Dupak – Pasar Loak – Masuk Tol Dupak Keluar Tol Tandes – Margomulyo – Tambak Osowilangun – Kembali dengan rute yang sama
P8	Purabaya - Tambak Oso Wilangun (Lewat Tol) Purabaya/Bungurasih – Masuk Tol Waru Keluar Tol Tandes – Margomulyo – Tambak Osowilangun – Kembali dengan rute yang sama

4.2. Perancangan Model Jejaring

Pembuatan model jejaring digunakan untuk mempermudah pemodelan *Orienteering Problem*. Diawali dengan penggambaran rute sesuai dengan trayek bis kota pada masing-masing jalur yang digunakan. Dengan menggunakan aplikasi Paint, setiap rute trayek digambarkan di atas peta digital dengan warna jalur berwarna biru. Simpul sebagai tempat pemberhentian bis yang digambarkan dalam bentuk lingkaran, untuk objek wisata digambarkan dengan lingkaran berwarna ungu, dan untuk pertemuan antar rute digambarkan dengan

lingkaran berwarna abu-abu. Untuk membedakan jalur satu arah/dua arah yaitu jalur satu arah digambarkan dengan garis biru dengan anak panah pada jalurnya sedangkan jalur dua arah digambarkan dengan garis biru tanpa anak panah. Hasil dari penggambaran jalur tersebut dapat dilihat pada Gambar 4.1. Untuk penggambaran jalur yang lebih lengkap dan jelas, ada pada LAMPIRAN E.



Gambar 4.1 Penggambaran Rute Bis Kota di Surabaya

4.2.1. Penentuan Simpul dan Skor

Setelah penggambaran rute dilakukan, maka hal yang dilakukan selanjutnya adalah menentukan titik simpul. Penentuan titik simpul berdasarkan asumsi bahwa titik tersebut merupakan tempat yang padat pengunjung seperti jalur yang melewati objek wisata dan pusat perbelanjaan. Setiap titik yang dijadikan simpul memiliki nilai skor agar dapat diimplementasikan menggunakan *Orienteering Problem*. Nilai skor ditentukan melalui rata-rata hasil pencarian objek yang dijadikan titik simpul menggunakan aplikasi *Google Trends* dengan jangka waktu 1 tahun (2018).

Berikut Tabel 4.2 yang merupakan penggalan lokasi titik simpul beserta nilai skornya. Untuk keseluruhan lokasi titik simpul dan skor dapat dilihat pada halaman LAMPIRAN A.

Tabel 4.2 Lokasi dan Skor Simpul

Titik	Lokasi	Skor	Skor Total
1	Terminal Purabaya	43	43
2	City of Tomorrow Mall	36	36
3	Masjid Al-Akbar	35	35
4	Taman Pelangi	44	44
5	Pertigaan Ahmad Yani Jemur Andayani	0	0
6	Taman Flora	27	27
7	DBL Arena	29	29
…	…	…	…
45	Pertigaan Pasar Kembang Arjuno	0	0
46	Perempatan Embong Malang Blauran	0	0
47	Pelabuhan Tanjung Perak	50	50
48	Terminal Tambak Osowilangun	46	46

4.2.2. Penentuan Waktu Tempuh

Setelah jalur digambarkan dan titik simpul ditentukan, maka garis jalur antar dua titik simpul yang berhubungan akan diberikan waktu tempuh. Waktu tempuh antara 2 titik simpul yang sama dapat berbeda dikarenakan waktu tempuh antara titik A ke titik B (berangkat) bisa saja berbeda dengan waktu tempuh antara titik B ke titik A (pulang) dikarenakan jalur yang ditempuh berbeda. Selanjutnya jarak antara dua simpul disebut dengan busur.

Seluruh nilai waktu tempuh untuk tiap busur didapatkan menggunakan aplikasi *Google Maps* yang diakses melalui *web browser*.

Tabel 4.3 merupakan tabel nilai waktu tempuh antara dua simpul yang ada di dalam model jejaring. Untuk keseluruhan tabel waktu tempuh tiap simpul dapat dilihat pada halaman LAMPIRAN B.

Tabel 4.3 Nilai Waktu Tempuh Tiap Busur

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
1	2	6
2	1	11
2	3	3
3	2	3
3	4	2
4	3	7
4	5	2
4	7	2
…	…	…
46	33	4
46	45	4
47	40	7
48	34	17

4.2.3. Asumsi

Terdapat beberapa tempat/objek yang tidak tepat bersebelahan dengan jalan raya yang dilalui oleh bus kota, maka dibuat beberapa asumsi untuk menentukan titik simpul yang berada pada jalur bus kota namun jauh dengan objek wisata. Asumsi titik simpul yang digunakan pada tugas akhir ini dapat dilihat pada tabel 4.4.

Tabel 4.4 Asumsi Titik Simpul

1	**Asumsi:** Masjid Al-Akbar Surabaya (Simpul No. 3)
	Alasan: Masjid Al-Akbar Surabaya tidak bersebelahan langsung dengan jalur yang dilalui bus kota, sehingga diasumsikan bahwa titik pemberhentian berada pada Carefour Ahmad Yani.
2	**Asumsi:** Taman Flora (Simpul No. 6)
	Alasan: Karena taman flora berada di utara terminal bratang, jadi setelah sampai di terminal bratang, wisatawan harus berjalan ke utara sejauh 100 meter.

Terdapat pula beberapa jalur yang tidak terdapat dalam trayek bis Surabaya namun akan diikut sertakan dalam pembuatan model jejaring. Hal tersebut dilakukan sehingga model jejaring yang terbentuk lebih fleksibel dan lebih banyak alternatif jalur lainnya. Penambahan jalur dilakukan dengan bantuan aplikasi *Google Street* untuk meninjau apakah lebar jalur jalan tersebut dapat dilewati oleh bis kota Surabaya atau tidak. Sehingga asumsi untuk penambahan jalur yang digunakan pada tugas akhir ini dapat dilihat pada Tabel 4.5.

Tabel 4.5 Asumsi Penambahan Jalur

1	**Asumsi:** Jalur Taman Flora (Simpul No. 6) → Marvel City (Simpul No. 12)
	Deskripsi: Taman Flora – Jl. Manyar – Jl. Ngagel Jaya Selatan – Jl. Bung Tomo – Jl. Ratna – Jl. Raya Ngagel – Marvel City

2	**Asumsi:** Jalur Tunjungan Plaza (Simpul No. 22) → Perempatan Pemuda (Simpul No. 42)
	Deskripsi: Tunjungan Plaza – Jl. Gubernur Suryo – Perempatan Pemuda
3	**Asumsi:** Jalur Monumen Kapal Selam (Simpul No. 13) → Perempatan Pemuda (Simpul No. 42)
	Deskripsi: Monumen Kapal Selam – Jl. Pemuda – Perempatan Pemuda
4	**Asumsi:** Jalur Perempatan Pemuda (Simpul No. 42) → Balai Kota Surabaya (Simpul No. 43)
	Deskripsi: Perempatan Pemuda – Jl. Yos Sudarso – Balai Kota Surabaya
5	**Asumsi:** Jalur Balai Kota Surabaya (Simpul No. 43) → Monumen Kapal Selam (Simpul No. 13)
	Deskripsi: Balai Kota Surabaya – Jl. Walikota Mustajab – Monumen Kapal Selam
6	**Asumsi:** Jalur Balai Kota Surabaya (Simpul No. 43) → Perempatan Kalianyar Pengampon (Simpul No. 44)
	Deskripsi: Balai Kota Surabaya – Jl. Walikota Mustajab – Jl. Genteng Kali – Jl. Undaan Wetan – Perempatan Kalianyar Pengampon
7	**Asumsi:** Jalur Balai Kota Surabaya (Simpul No. 43) → Gedung Siola (Simpul No. 25)
	Deskripsi: Balai Kota Surabaya – Jl. Walikota Mustajab – Jl. Genteng Kali – Museum Surabaya (Gedung Siola)
8	**Asumsi:** Jalur Stasiun Surabaya Kota (Simpul No. 35) → Pasar Atom dan ITC (Simpul No. 17)
	Deskripsi: Stasiun Surabaya Kota – Jl. Stasiun Kota – Jl. Gembong – Pasar Atom dan ITC
9	**Asumsi:** Jalur Stasiun Surabaya Kota (Simpul No. 35) → Pertigaan Bubutan Kebun Rojo (Simpul No. 41)
	Deskripsi: Stasiun Surabaya Kota – Jl. Kebun Rojo – Pertigaan Bubutan Kebun Rojo
10	**Asumsi:** Jalur Jembatan Merah Plaza (Simpul No. 37) → Wisata Religi Ampel (Simpul No. 18)

	Deskripsi: Jembatan Merah Plaza – Jl. Kembang Jepun – Wisata Religi Ampel
11	**Asumsi:** BG Junction (Simpul No. 26) → Perempatan Kranggan Tembok Dukuh (Simpul No. 33)
	Deskripsi: BG Junction – Jl. Kranggan – Perempatan Kranggan Tembok Dukuh
12	**Asumsi:** Jalur Pertigaan Pasar Kembang Arjuno (Simpul No. 45) → Perempatan Embong Malang Blauran (Simpul No. 46)
	Deskripsi: Pertigaan Pasar Kembang Arjuno – Jl. Kedung Doro – Perempatan Embong Malang Blauran

4.3. Pra Proses Data

Pada tahap ini data nilai skor dan nilai busur yang terbentuk belum dapat diolah, sehingga diperlukan transformasi data agar data dapat digunakan sebagai masukan (input) untuk pencarian solusi dari model Orienteering Problem dengan menggunakan algoritma Memetika yang diimplementasikan menggunakan bahasa pemrograman Java. Data yang nantinya dihasilkan berupa data dalam bentuk matriks yang terdiri dari matriks skor dan matriks waktu tempuh.

4.3.1. Pembentukan Matriks Skor

Pembentukan matriks skor dilakukan menggunakan *Microsoft Office Excel* dengan mengubah bentuk sama seperti sebelumnya, menggunakan data skor setiap simpul yang ada pada Tabel 4.2 dan dilakukan transformasi data dengan mengubah bentuknya ke dalam matriks satu dimensi agar dapat terbaca oleh *Java*. Matriks ini berisi 48 baris sesuai dengan jumlah simpul. Setiap baris pada matriks ini berisikan skor pada masing-masing simpul.

Tabel 4.6 merupakan potongan matriks dengan total 48 baris, dimana matriks ini hanya berisi 1 kolom atau 1 dimensi dan setiap barisnya merupakan skor pada setiap simpul. Kotak

merah pada tabel dibawah merupakan contoh bahwa skor pada simpul 9 adalah 54.

Tabel 4.6 Potongan Matriks Skor

simpul	
1	0
2	36
3	35
4	44
5	0
6	27
7	29
8	24
9	54
10	0
11	39
12	61
13	87
…	…
47	50
48	46

4.3.2. Pembentukan Matriks Waktu Tempuh

Sama seperti sebelumnya, dengan menggunakan *Microsoft Office Excel* Tabel 4.3 yang berisi waktu tempuh untuk setiap simpul ditransformasi menjadi matriks dengan ukuran baris dan kolom sebanyak 48. Matriks tersebut merepresentasikan hubungan tiap simpul. Setiap *cell* merepresentasikan waktu tempuh antara 2 simpul yang saling bersinggungan pada matriks. *Cell* hanya dapat terisi nilai waktu tempuh apabila simpul saling terhubung secara langsung sesuai dengan Tabel 4.3. Apabila simpul tidak terhubung secara langsung maka *cell* akan berisi nilai waktu tempuh 0. Tabel 4.7 merupakan potongan dari matriks waktu tempuh yang telah dibuat.

Tabel 4.7 Potongan Matriks Waktu Tempuh

simpul	1	2	3	4	5	6	…	47	48
1	0	5	0	0	0	0	…	0	0
2	5	0	3	0	0	0	…	0	0
3	0	3	0	4	0	0	…	0	0
4	0	0	4	0	2	0	…	0	0
5	0	0	0	2	0	12	…	0	0
6	0	0	0	0	12	0	…	0	0
…	…	…	…	…	…	…	…	…	…
47	0	0	0	0	0	0	…	0	0
48	0	0	0	0	0	0	…	0	0

Tabel 4.7 menunjukkan potongan matriks yang berisi hubungan antar simpul 1 hingga 48 dan berisi waktu tempuh dalam satuan menit. Matriks dapat dibaca dari baris ke kolom. Untuk simpul yang berhubungan langsung dengan simpul lain akan berisi waktu tempuh dan apabila simpul tidak berhubungan maka akan berisi waktu tempuh 0. Contoh pada kotak merah dalam matriks yaitu simpul 5 dengan simpul 6 berisi waktu tempuh sebesar 12 menit. Contoh untuk *cell* berisi nilai 0 yaitu simpul 1 dengan simpul 3 yang tidak terhubung secara langsung dan harus melalui simpul lain untuk menghubungkan simpul tersebut. Untuk mengisi setiap nilai 0 pada *cell* setiap simpul akan melalui proses transformasi data menggunakan algoritma Djikstra.

4.4. Formulasi Model Orienteering Problem

Setelah model jejaring dan pra proses data dilakukan, maka selanjutnya dibentuklah model yang sesuai untuk menyelesaikan permasalahan Orienteering Problem dengan menentukan variabel keputusan, fungsi tujuan, dan batasan sebagai berikut.

4.4.1. Variabel Keputusan

Variable keputusan yang digunakan dapat mempengaruhi nilai dari fungsi tujuan. Dalam *Orienteering Problem* variable keputusannya yaitu kunjungan yang harus ditempuh dari suatu simpul ke simpul berikutnya. Sehingga variabel keputusan dapat didefinisikan kunjungan dari simpul i ke simpul j. (x_{ij}). Nilai dari variabel keputusan tidak dapat bernilai negatif dan hanya dapat bernilai 0 atau 1. Variabel keputusan $x_{ij} = 1$ jika kunjungan ke simpul i diikuti dengan kunjungan ke simpul j, dan jika tidak maka $x_{ij} = 0$.

Tabel 4.8 Contoh Variabel Keputusan

No	Variabel	Deskripsi
1	x1.2	Kunjungan dari simpul 1 ke simpul 2
2	x2.1	Kunjungan dari simpul 2 ke simpul 1
3	x2.3	Kunjungan dari simpul 2 ke simpul 3
4	x3.2	Kunjungan dari simpul 3 ke simpul 2
5	x3.4	Kunjungan dari simpul 3 ke simpul 4
6	x4.3	Kunjungan dari simpul 4 ke simpul 3
7	x4.5	Kunjungan dari simpul 4 ke simpul 5
8	x4.7	Kunjungan dari simpul 4 ke simpul 7
9	x5.4	Kunjungan dari simpul 5 ke simpul 4
10	x5.6	Kunjungan dari simpul 5 ke simpul 6
11	x6.5	Kunjungan dari simpul 6 ke simpul 5
12	x7.5	Kunjungan dari simpul 7 ke simpul 5
13	x7.8	Kunjungan dari simpul 7 ke simpul 8
14	x8.7	Kunjungan dari simpul 8 ke simpul 7
15	x8.9	Kunjungan dari simpul 8 ke simpul 9
16	x9.8	Kunjungan dari simpul 9 ke simpul 8
…	…	…
114	x47.40	Kunjungan dari simpul 47 ke simpul 40
115	x48.34	Kunjungan dari simpul 48 ke simpul 34

Untuk melihat keseluruhan tabel variabel keputusan, dapat dilihat pada halaman LAMPIRAN C.

4.4.2. Fungsi Tujuan

Dalam penelitian ini, tujuan yang ingin dicapai oleh *Orienteering Problem* adalah mencari rute dengan cara memaksimalkan jumlah skor. Dalam pembuatan fungsi tujuan, diperlukan penentuan simpul awal dan simpul akhir terlebih dahulu. Total simpul yang digunakan pada tugas akhir sebanyak 48 simpul. Simpul awal dinotasikan dengan simbol i dan simpul akhir dinotasikan dengan simbol j. Dalam pengaplikasiannya, simpul awal dan simpul akhir dapat ditempatkan pada simpul manapun sesuai dengan keinginan calon pengguna, namun simpul awal dan simpul akhir tetap tidak boleh berada dalam satu simpul yang sama. Sehingga anggota dari i dan j adalah simpul 1 hingga 48. Sebagai contoh, dengan menggunakan salah satu rute bis kota yaitu P1, simpul awal dengan nomor 1 (Terminal Purabaya) dan simpul akhir dengan nomor 47 (Pelabuhan Tanjung Perak). Sehingga fungsi tujuan dalam contoh model OP tersebut adalah sebagai berikut.

$$Max\ (\sum_{i=1}^{N-1}\sum_{j=2}^{N} S_i x_{ij} + S_j)$$

Sehingga pada rute P1 ini, fungsi tujuan tersebut ditulis dalam persamaan berikut.

$$Max\ (\sum_{i=1}^{47}\sum_{j=1}^{48} S_i x_{ij} + S_j)$$
$$0 \leq x_{ij} \leq 1$$

Dimana,
S_i = Skor dari simpul i
x_{ij} = Kunjungan dari simpul i ke simpul j
i = 1, 2, 3, …, 48
j = 1, 2, 3, …, 48

4.4.3. Perumusan Batasan

Batasan-batasan yang ada dalam model antara lain sebagai berikut.

Batasan 1: Rute dimulai dari simpul 1 dan berakhir di simpul 47. Rute yang terpilih pastinya merupakan jalur yang terhubung dengan simpul awal dan simpul akhir, sehingga variabel keputusan yang terkait dengan kunjungan dari simpul 1 dan kunjungan ke simpul 47 pasti bernilai satu. Batasan 1 dalam model OP adalah sebagai berikut.

$$\sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1$$

Sehingga pada contoh rute P1 ini, batasan tersebut ditulis dalam persamaan berikut.

- Batasan simpul awal

$$\sum_{j=2}^{47} x_{1j} = 1$$

- Batasan simpul akhir

$$\sum_{i=1}^{46} x_{i47} = 1$$

Batasan 2: Semua simpul harus saling terhubung dan setiap simpul hanya dapat dikunjungi maksimum satu kali.

Terhubungnya semua simpul dalam suatu *network model* ditunjukkan dari adanya percabangan dan pertemuan antar simpul. Batasan 2 dalam model OP adalah sebagai berikut.

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1;\ \forall k = 2, \dots, N-1,$$

Sehingga pada rute P1 ini, batasan tersebut ditulis dalam persamaan berikut.

- Batasan untuk pertemuan antar simpul

$$\sum_{i=1}^{46} x_{ik} \leq 1;\ \forall k = 2, \dots, 46$$

- Batasan untuk percabangan antar simpul

$$\sum_{j=2}^{47} x_{kj} \leq 1;\ \forall k = 2, \dots, 46$$

Batasan 3: Jumlah total waktu tempuh tidak boleh melebihi batasan total waktu yang dialokasikan.
Total waktu tempuh (tMax) yang akan digunakan sebenarnya relatif berdasarkan keinginan dari pengguna yang akan menggunakan transportasi umum bis. Sebagai contoh, pengguna ingin berkeliling Surabaya menggunakan bis dari Terminal Purabaya (simpul 1) ke Pelabuhan Tanjung Perak (simpul 47) dengan alokasi waktu sebesar 80 menit. Batasan 3 dalam model OP adalah sebagai berikut.

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max},$$

Maka pada contoh kasus, batasan tersebut ditulis dalam persamaan berikut.

$$\sum_{i=1}^{46} \sum_{j=2}^{47} t_{ij} x_{ij} \leq 80$$

Batasan 4 dan 5: Tidak boleh ada *subtours* dimana lintasan dimulai dan berakhir pada simpul yang sama sehingga membentuk seperti sirkuit. Batasan ini sudah tergambarkan

dalam batasan lainnya. Sehingga secara tidak langsung, jika batasan lain terpenuhi, maka batasan 5 juga terpenuhi.

4.5. Penentuan Parameter Variabel AG dan AM

Pada tahap ini merupakan proses menentukan parameter untuk beberapa variabel yang mempengaruhi solusi model OP dari Algoritma Genetika (AG) dan Algoritma Memetika (AM). Proses penentuan parameter tersebut antara lain nilai dari probabilitas perkawinan silang (crossover) dan probabilitas mutasi. Perbedaan penentuan parameter dari kedua algoritma ini terletak pada nilai probabilitasnya sesuai dengan konfigurasi terbaik dari masing-masing algoritma. Konfigurasi tersebut didapatkan dari hasil penelitian yang dilakukan oleh Garg [14] secara empiris untuk Algoritma Genetika dan Algoritma Memetika yang digunakan untuk proses optimasi *Cryptanalysis*.

4.5.1. Probabilitas Perkawinan Silang (Crossover)

Probabilitas ini akan mempengaruhi apakah individu tersebut akan mengalami perkawinan silang atau tidak, sehingga jumlah proses perkawinan silang yang terjadi antara setiap individu pada populasi setiap generasinya dapat berbeda. Nilai dari parameter ini berkisar antara 0 hingga 1. Menurut Garg [14] pengaturan yang terbaik untuk probabilitas perkawinan silang AG adalah sebesar 0.95 dan AM sebesar 0.5

4.5.2. Probabilitas Mutasi

Probabilitas ini akan mempengaruhi apakah individu yang terpilih akan mengalami mutasi atau tidak, sehingga jumlah proses mutasi yang terjadi antara setiap individu setiap generasinya dapat berbeda. Nilai dari parameter ini berkisar antara 0 hingga 1. Menurut Garg [14] pengaturan yang terbaik untuk probabilitas mutasi AG adalah sebesar 0.05 dan AM yaitu sebesar 0.5.

4.6. Penentuan Komponen AG dan AM

Tahap ini dilakukan dengan menentukan beberapa komponen AG dan AM yaitu:

4.6.1. Populasi

Populasi pada studi kasus ini adalah sekumpulan kromosom/solusi berupa rute (individu) mulai dari simpul awal sampai simpul akhir yang telah ditentukan sebelumnya dan memenuhi batasan. Dalam suatu populasi, panjang kromosom pada tiap individu bisa berbeda-beda. Namun setiap individu harus memiliki titik awal dan titik akhir yang sama. Jumlah populasi pada setiap algoritma dapat berbeda. Sesuai dengan [14] jumlah populasi pada AG adalah 100 dan jumlah populasi pada AM adalah 10. Pada penelitian ini jumlah populasi yang digunakan untuk setiap algoritma disesuaikan menurut konfigurasi terbaiknya karena akan tidak menguntungkan apabila hanya salah satu algoritma yang menggunakan konfigurasi terbaiknya.

4.6.2. Individu

Individu pada studi kasus ini adalah sebuah kromosom yang berupa deretan simpul-simpul. Setiap simpul yang terdapat pada sebuah kromosom merupakan simpul yang harus dilalui/dikunjungi dimulai dari simpul awal hingga menuju simpul akhir. Adapun contoh format penulisannya sesuai pada Gambar 4.2.

[simpul awal, simpul mana saja yang dikunjungi, simpul akhir] (waktu tempuh, skor)
Contoh:
[1, 4, 9, 20, 47] (59, 356)

Gambar 4.2 Format Penulisan Individu

4.6.3. Gen

Gen pada studi kasus ini adalah simpul yang dikunjungi dalam setiap kromosom/solusi. Jumlah gen dalam suatu individu adalah tidak tetap karena penetapan jumlah gen ditentukan secara acak. Maksimum jumlah gen dalam suatu individu adalah 47 sesuai dengan jumlah titik yang ada pada model jejaring. Seperti pada Gambar 4.1, dalam kromosom tersebut terdapat 5 gen yang selanjutnya disebut simpul yaitu simpul nomor 1, 4, 9, 20, dan 47.

4.7. Desain Algoritma Genetika

Pada bagian ini menjelaskan alur kerja Algoritma Genetika dalam pembuatan solusi dari model OP. Algoritma dibuat menggunakan bahasa pemrograman Java dengan menggunakan masukan yang dijelaskan pada bagian 4.3 dan menggunakan parameter yang dijelaskan pada bagian 4.5. Berikut tahapan penerapan Algoritma Genetika :

4.7.1. Inisialisasi Parameter

Pada tahap ini ditentukan nilai parameter dan komponen dari Algoritma Genetika. Sesuai dengan rekomendasi pada tahap sebelumnya, maka penentuan parameter dan komponennya sebagai berikut:

- Probabilitas Perkawinan Silang (Crossover) = 0.95
- Probabilitas Mutasi = 0.05
- Ukuran populasi = 100

4.7.2. Pembangkitan Populasi Awal

Pada tahap ini dibangkitkan populasi awal sebagai solusi layak awal dengan panjang kromosom yang bervariasi. Pembangkitan populasi awal dilakukan secara acak namun tetap harus memenuhi batasan-batasan yang telah ditetapkan seperti gen awal dan akhir dari setiap kromosom harus sama dan tidak boleh melebihi waktu tempuh maksimal. Sebagai gambaran pembangkitan solusi awal, digunakan *pseudocode* seperti pada Kode 4.1.

```
1: procedure Initial Solution
2:   Define tMax
3:   Define MaxPop
4:   Pop = 0
5:    repeat
6:      generate array listRl = random[1,N]
7:       for i=0 to listRl.length do
8:       generate Rn = random[1,N]
9:       insert Rn to listRl
10:      end for
11:    generate Rs = 1stsimpul + listRl +
Nthsimpul
12:    compute travel time of Rs, tRS
13:    if tRS <= tMax then Pop=Pop+1
14:  until Pop > MaxPop
```

Kode 4.1 *Pseudocode* Pembangkitan Solusi Awal

Dalam studi kasus ini, tMax telah ditentukan sebesar 80 menit. Untuk pembangkitan solusi awal telah ditentukan sesuai dengan jumlah populasi untuk Algoritma Genetika yaitu sebanyak 100 individu.

4.7.3. Evaluasi Fitness

Pada tahap ini dilakukan penghitungan nilai fitness dari simpul-simpul yang dilewati oleh setiap solusi. Nilai fitness pada studi kasus ini adalah jumlah skor. Sesuai dengan fungsi tujuan dari pemodelan OP yaitu memaksimalkan jumlah skor, maka individu dengan nilai fitness terbaik adalah individu dengan jumlah skor terbanyak. Fitness ini digunakan untuk mengukur kelayakan dari individu pada populasi awal dan populasi baru yang dibentuk dari hasil perkawinan silang dan mutasi. Untuk gambaran mencari solusi terbaik dari populasi awal, digunakan pseudocode seperti pada Kode 4.2.

```
1: procedure Select Best Solution
2:   current ← Solution(0)
3:   for i=1 to MaxPop do
4:     if f(Solution(i)) > f(current) then
5:       current ← Solution(i)
6:     end if
7:   end for
8:   best ← current
9: end procedure
```

Kode 4.2 *Pseudocode* Pencarian Solusi Terbaik

4.7.4. Seleksi Individu

Pada proses ini digunakan seleksi untuk menentukan induk pada proses selanjutnya yaitu perkawinan silang. Dalam tahap ini digunakan metode *roulette wheel selection* dimana akan diambil sebanyak 10 individu yang tidak sama pada populasi. Seleksi tersebut akan memilih individu dari populasi berdasarkan nilai *fitness*. Semakin tinggi nilai fitness maka semakin besar peluang individu tersebut terpilih. Langkah-langkah penerapan metode *roulette wheel selection* adalah sebagai berikut[26].

a. Hitung nilai *fitness* dari masing-masing individu (jumlah skor tiap *simpul* yang dikunjungi).
b. Hitung total *fitness* semua individu.
c. Dihitung probabilitas masing-masing individu (nilai *fitness* individu dibagi total *fitness*).
d. Berdasarkan probabilitas tersebut, dihitung jatah masing-masing individu dari angka 0 sampai 1.
e. Dibangkitkan bilangan random dari 0 sampai 1.
f. Dari bilangan random yang dihasilkan, ditentukan individu mana saja yang terpilih dari proses seleksi.

Untuk gambaran penggunaan *roulette wheel selection*, digunakan *pseudocode* seperti pada Kode 4.3.

```
1: procedure Roulette Wheel Selection
2:   generate r= random[0,1]
3:   totFitness = sum of each f(Solution)
4:   i = 0
5:   sum = f(Solution(i))/totFitness
6:   while sum < r then
7:     i=i+1
8:     sum = (sum + f(Solution(i)))/totFitness
9:   end while
10:  Select Solution(i)
11: end procedure
```

Kode 4.3 ***Pseudocode Roulette Wheel Selection***

4.7.5. Perkawinan Silang

Setelah proses seleksi dilakukan, individu yang terpilih selanjutnya disebut induk akan dilakukan proses perkawinan silang dengan induk lainnya sehingga menghasilkan solusi baru yang diharapkan lebih baik dari individu sebelumnya. Belum tentu setiap individu yang terpilih dari proses seleksi akan terjadi perkawinan silang dikarenakan adanya probabilitas perkawinan silang yang telah ditentukan. Untuk melakukan perkawinan silang dibutuhkan 2 induk yang berbeda. Pada studi kasus ini metode perkawinan silang yang digunakan adalah *single-point crossover*[27]. Metode ini dilakukan dengan menentukan *insertion point* pada kedua induk sebagai titik pemotongan kromosom yang kemudian potongan kromosom saling dipindah-silangkan ke masing-masing induk yang berbeda sehingga menghasilkan 2 keturunan baru. Contoh proses *single-point crossover* sebagai berikut.

Induk 1 : 1, 4, 7, 19, 15, 11, 9, 23, 47
Induk 2 : 1, 3, 6, 9, 11, 13, 19, 23, 27, 40, 47

Proto-Child 1 : 1, 4, 7, 19, 15, 13, 19, 23, 27, 40, 47
Keturunan 1 : 1, 4, 7, 19, 15, 13, 23, 27, 40, 47

Proto-Child 2 : 1, 3, 6, 9, 11, 11, 9, 23, 47

Keturunan 2 : 1, 3, 6, 9, 11, 23, 47

Penjelasan proses *single-point crossover* diatas sebagai berikut.

a. Memastikan induk yang memiliki lebih sedikit jumlah gen bertindak sebagai pembuat *insertion point*. (Dalam contoh induk 1 memiliki jumlah gen yang lebih sedikit)
b. Membuat *insertion point* untuk kedua induk yang merupakan bilangan acak antara indeks 1 sampai indeks 2 terakhir pada induk 1.
c. Pada induk 1 akan dibentuk 2 *sub list* berupa kepala kromosom dan buntut kromosom dimana kepala kromosom mengambil gen pada induk 1 dimulai dari indeks 0 (simpul awal) sampai indeks *insertion point* berada dan buntut kromosom mengambil gen pada induk 1 dimulai dari indeks setelah *insertion point* hingga indeks terakhir (simpul akhir).
d. Selanjutnya pada induk 2 juga akan dibentuk 2 *sub list* berupa kepala kromosom dan buntut kromosom dimana kepala kromosom mengambil gen pada induk 2 dimulai dari indeks 0 (simpul awal) sampai indeks *insertion point* berada dan buntut kromosom mengambil gen pada induk 2 dimulai dari indeks setelah *insertion point* hingga indeks terakhir (simpul akhir).
e. Kemudian kepala kromosom dari induk 1 (hasil langkah c) digabungkan dengan buntut kromosom dari induk 2 (hasil langkah d) untuk menghasilkan *proto child* 1.
f. Juga sebaliknya, kepala kromosom dari induk 2 (hasil langkah d) digabungkan dengan buntut kromosom dari induk 1 (hasil langkah c) untuk menghasilkan *proto child* 2.
g. Gen yang terduplikasi pada *proto child* akan dihapus untuk mendapatkan keturunan yang sesuai dengan batasan.

h. Setiap perkawinan silang menghasilkan 2 keturunan.

4.7.6. Mutasi

Pada tahap ini dilakukan perubahan nilai pada keturunan yang dihasilkan dari proses kawin silang. Solusi baru yang dihasilkan dari mutasi tidak selalu menjadi lebih baik atau bahkan tidak *feasible* dikarenakan melampaui batasan waktu tempuh. Pada Algoritma Genetika, probabilitas mutasi yang digunakan sebesar 0.05 [14]. Apabila keturunan berhasil mengalami mutasi, maka dilakukan perubahan nilai berdasarkan operator yang dipilih secara acak. Menurut Han, terdapat 14 operator pada Algoritma Genetika dan pada studi kasus ini dipilih 4 operator yaitu *add*, *add-swap*, *add-remove*, dan *remove*[28]. Berikut contoh penggunaan setiap operator.

a. Operator *Add*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

b. Operator *Add-Swap*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Tukar nilai gen tersebut dengan gen lain yang terdapat dalam keturunan secara acak, misalnya gen yang ditukar adalah gen **19**.

1, 2, 3, 7, **13**, 25, 23, 22, **19**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

c. Operator *Add-Remove*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Hilangkan 1 gen yang terdapat dalam keturunan secara acak, misalnya gen yang dihilangkan adalah gen 19.

1, 2, 3, 7, 25, 23, 22, **13**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

d. Operator *Remove*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Hilangkan 1 gen yang terdapat dalam keturunan secara acak, misalnya gen yang dihilangkan adalah gen 19.

1, 2, 3, 7, 25, 23, 22, 29, 35, 47

- Apabila solusi baru menghasilkan waktu tempuh yang lebih kecil dari solusi sebelumnya, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

4.7.7. Pembuatan Populasi Baru

Solusi pada populasi awal serta hasil keturunan dari proses mutasi akan digabung yang selanjutnya dilakukan evaluasi fitness yang sama seperti pada bagian 4.7.3 dan dipilih 100 solusi terbaik berdasarkan nilai fitness-nya untuk menjadi populasi baru. Setelah populasi baru terbentuk, maka akan dilakukan pengulangan pada bagian 4.7.4 dan seterusnya hingga kriteria pemberhentian tercapai.

4.7.8. Pemberhentian Algoritma Genetika

Pembuatan populasi baru akan mengalami perulangan hingga memenuhi kondisi yang telah ditentukan yaitu jumlah generasi, dalam studi kasus ini algoritma akan berhenti apabila telah mengalami perulangan sebanyak 1000 generasi. Setelah generasi ke-1000 maka ditentukan solusi yang paling optimal pada populasi tersebut.

4.8. Desain Algoritma Memetika

Pada bagian ini akan dijelaskan alur kerja Algoritma Memetika dalam pembuatan solusi dari model OP. Algoritma dibuat menggunakan bahasa pemrograman Java dengan menggunakan masukan yang dijelaskan pada bagian 4.3 dan menggunakan parameter yang dijelaskan pada bagian 4.5. Terdapat dua jenis Algoritma Memetika yang diterapkan, yaitu Algoritma Memetika Hill Climbing dan Algoritma Memetika Simulated Annealing. Kedua jenis algoritma ini mempunyai perbedaan pada metode pencarian lokal yang digunakan. Berikut tahapan penerapan Algoritma Memetika.

4.8.1. Inisialisasi Parameter

Pada tahap ini ditentukan nilai parameter dan komponen dari Algoritma Memetika. Sesuai dengan rekomendasi pada tahap sebelumnya, maka penentuan parameter dan komponennya sebagai berikut:

- Probabilitas Perkawinan Silang (Crossover) = 0.5

- Probabilitas Mutasi = 0.5
- Ukuran populasi = 10

4.8.2. Pembangkitan Solusi Awal

Pada tahap ini dibangkitkan populasi awal sebagai solusi layak awal dengan panjang kromosom yang bervariasi. Pembangkitan populasi awal dilakukan secara acak namun tetap harus memenuhi batasan-batasan yang telah ditetapkan seperti gen awal dan akhir dari setiap kromosom harus sama dan tidak boleh melebihi waktu tempuh maksimal. Dalam studi kasus ini, tMax telah ditentukan sebesar 80 menit. Untuk pembangkitan solusi awal telah ditentukan sesuai dengan jumlah populasi untuk Algoritma Memetika yaitu sebanyak 10 individu.

4.8.3. Evaluasi Nilai Fitness

Pada tahap ini dilakukan penghitungan nilai fitness dari simpul-simpul yang dilewati oleh setiap solusi. Nilai fitness pada studi kasus ini adalah jumlah skor. Sesuai dengan fungsi tujuan dari pemodelan OP yaitu memaksimalkan jumlah skor, maka individu dengan nilai fitness terbaik adalah individu dengan jumlah skor terbanyak. Fitness ini digunakan untuk mengukur kelayakan dari individu pada populasi awal dan populasi baru yang dibentuk dari hasil perkawinan silang dan mutasi.

4.8.4. Seleksi Individu

Pada proses ini digunakan seleksi untuk menentukan induk pada proses selanjutnya yaitu perkawinan silang. Dalam tahap ini digunakan metode *roulette wheel selection* dimana akan diambil sebanyak 10 individu yang tidak sama pada populasi. Seleksi tersebut akan memilih individu dari populasi berdasarkan nilai *fitness*. Semakin tinggi nilai fitness maka semakin besar peluang individu tersebut terpilih. Langkah-langkah penerapan metode *roulette wheel selection* adalah sebagai berikut[26].

a. Hitung nilai *fitness* dari masing-masing individu (jumlah skor tiap *simpul* yang dikunjungi).

b. Hitung total *fitness* semua individu.
c. Dihitung probabilitas masing-masing individu (nilai *fitness* individu dibagi total *fitness*).
d. Berdasarkan probabilitas tersebut, dihitung jatah masing-masing individu dari angka 0 sampai 1.
e. Dibangkitkan bilangan random dari 0 sampai 1.
f. Dari bilangan random yang dihasilkan, ditentukan individu mana saja yang terpilih dari proses seleksi.

4.8.5. Perkawinan Silang

Setelah proses seleksi dilakukan, individu yang terpilih selanjutnya disebut induk akan dilakukan proses perkawinan silang dengan induk lainnya sehingga menghasilkan solusi baru yang diharapkan lebih baik dari individu sebelumnya. Belum tentu setiap individu yang terpilih dari proses seleksi akan terjadi perkawinan silang dikarenakan adanya probabilitas perkawinan silang yang telah ditentukan. Untuk melakukan perkawinan silang dibutuhkan 2 induk yang berbeda. Pada studi kasus ini metode perkawinan silang yang digunakan adalah *single-point crossover*[27]. Metode ini dilakukan dengan menentukan *insertion point* pada kedua induk sebagai titik pemotongan kromosom yang kemudian potongan kromosom saling dipindah-silangkan ke masing-masing induk yang berbeda sehingga menghasilkan 2 keturunan baru. Contoh proses *single-point crossover* sebagai berikut.

Induk 1 : 1, 4, 7, 19, 15, 11, 9, 23, 47
Induk 2 : 1, 3, 6, 9, 11, 13, 19, 23, 27, 40, 47

Proto-Child 1 : 1, 4, 7, 19, 15, 13, 19, 23, 27, 40, 47
Keturunan 1 : 1, 4, 7, 19, 15, 13, 23, 27, 40, 47

Proto-Child 2 : 1, 3, 6, 9, 11, 11, 9, 23, 47
Keturunan 2 : 1, 3, 6, 9, 11, 23, 47

Penjelasan proses *single-point crossover* diatas sebagai berikut.

a. Memastikan induk yang memiliki lebih sedikit jumlah gen bertindak sebagai pembuat *insertion point*. (Dalam contoh induk 1 memiliki jumlah gen yang lebih sedikit)
b. Membuat *insertion point* untuk kedua induk yang merupakan bilangan acak antara indeks 1 sampai indeks 2 terakhir pada induk 1.
c. Pada induk 1 akan dibentuk 2 *sub list* berupa kepala kromosom dan buntut kromosom dimana kepala kromosom mengambil gen pada induk 1 dimulai dari indeks 0 (simpul awal) sampai indeks *insertion point* berada dan buntut kromosom mengambil gen pada induk 1 dimulai dari indeks setelah *insertion point* hingga indeks terakhir (simpul akhir).
d. Selanjutnya pada induk 2 juga akan dibentuk 2 *sub list* berupa kepala kromosom dan buntut kromosom dimana kepala kromosom mengambil gen pada induk 2 dimulai dari indeks 0 (simpul awal) sampai indeks *insertion point* berada dan buntut kromosom mengambil gen pada induk 2 dimulai dari indeks setelah *insertion point* hingga indeks terakhir (simpul akhir).
e. Kemudian kepala kromosom dari induk 1 (hasil langkah c) digabungkan dengan buntut kromosom dari induk 2 (hasil langkah d) untuk menghasilkan *proto child* 1.
f. Juga sebaliknya, kepala kromosom dari induk 2 (hasil langkah d) digabungkan dengan buntut kromosom dari induk 1 (hasil langkah c) untuk menghasilkan *proto child* 2.
g. Gen yang terduplikasi pada *proto child* akan dihapus untuk mendapatkan keturunan yang sesuai dengan batasan.
h. Setiap perkawinan silang menghasilkan 2 keturunan.

4.8.6. Mutasi

Pada tahap ini dilakukan perubahan nilai pada keturunan yang dihasilkan dari proses kawin silang. Solusi baru yang dihasilkan dari mutasi tidak selalu menjadi lebih baik atau bahkan tidak *feasible* dikarenakan melampaui batasan waktu tempuh. Pada Algoritma Memetika, probabilitas mutasi yang digunakan sebesar 0.05 [14]. Apabila keturunan berhasil mengalami mutasi, maka dilakukan perubahan nilai berdasarkan operator yang dipilih secara acak. Menurut Han, terdapat 14 operator pada Algoritma Genetika dan pada studi kasus ini dipilih 4 operator yaitu *add*, *add-swap*, *add-remove*, dan *remove*[28]. Berikut contoh penggunaan setiap operator.

a. Operator *Add*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

b. Operator *Add-Swap*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Tukar nilai gen tersebut dengan gen lain yang terdapat dalam keturunan secara acak, misalnya gen yang ditukar adalah gen **19**.

1, 2, 3, 7, **13**, 25, 23, 22, **19**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

c. Operator *Add-Remove*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Munculkan 1 gen yang tidak ada dalam keturunan secara acak, misalnya gen **13**.
- Masukkan gen tersebut ke dalam keturunan pada posisi yang acak.

1, 2, 3, 7, 19, 25, 23, 22, **13**, 29, 35, 47

- Hilangkan 1 gen yang terdapat dalam keturunan secara acak, misalnya gen yang dihilangkan adalah gen 19.

1, 2, 3, 7, 25, 23, 22, **13**, 29, 35, 47

- Apabila solusi baru menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

d. Operator *Remove*

1, 2, 3, 7, 19, 25, 23, 22, 29, 35, 47

- Hilangkan 1 gen yang terdapat dalam keturunan secara acak, misalnya gen yang dihilangkan adalah gen 19.

1, 2, 3, 7, 25, 23, 22, 29, 35, 47

- Apabila solusi baru menghasilkan waktu tempuh yang lebih kecil dari solusi sebelumnya, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Jika tidak, maka solusi dikembalikan seperti semula.

4.8.7. Pencarian Lokal (*Hill Climbing*)

Tahap ini diimplementasikan hanya untuk Algoritma Memetika Hill Climbing. Semua keturunan yang termutasi maupun tidak

termutasi akan dilakukan perbaikan melalui pencarian lokal. Solusi hasil mutasi akan dijadikan sebagai solusi awal dalam algoritma Hill Climbing. Low-level heuristic berupa add-remove akan diimplementasikan pada solusi awal dan hasil solusi baru kemudian disebut neighborhood. Apabila neighborhood menghasilkan nilai fitness (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal, maka solusi tersebut disimpan menggantikan solusi sebelumnya. Dilakukan pengulangan Low-level heuristic hingga neighborhood memenuhi kriteria pemberhentian. Kriteria pemberhentian yang digunakan adalah pengulangan sebanyak 100 kali iterasi. Untuk gambaran penggunaan hill climbing, digunakan pseudocode seperti pada Kode 4.4.

```
1: procedure LS Hill Climbing
2:   for i=0 to offspring.length do
3:     loop = 0
4:     maxLoop = 100
5:     current ← offspring(i)
6:     repeat
7:       new ← GenerateNeighborhood(current)
8:       if f(new) > f(current) then
9:         current ← new
10:      end if
11:      loop++
12:    until loop > maxLoop
13:  end for
14: end procedure
```

Kode 4.4 *Pseudocode* Pencarian Lokal *Hill Climbing*

4.8.8. Pencarian Lokal (*Simulated Annealing*)

Tahap ini diimplementasikan hanya untuk Algoritma Memetika *Simulated Annealing*. Semua keturunan yang termutasi maupun tidak termutasi akan dilakukan perbaikan melalui pencarian lokal. Solusi hasil mutasi akan dijadikan sebagai solusi awal dalam algoritma *Simulated Annealing*. *Low-level heuristic* berupa *add-remove* akan diimplementasikan pada solusi awal dan hasil solusi baru kemudian disebut *neighborhood*. Kriteria

penerimaan solusi baru terjadi apabila neighborhood menghasilkan nilai *fitness* (skor) yang lebih besar dan memenuhi batasan waktu tempuh maksimal dari nilai *fitness* (skor) pada solusi awal. Apabila nilai *fitness* dari neighborhood lebih buruk, maka akan masuk ke dalam fungsi pengontrol. Apabila nilai dari fungsi pengontrol (Persamaan Boltzman) memenuhi probabilitas maka solusi neighborhood tadi akan diterima menggantikan solusi awal[20][29]. Parameter dan komponen Persamaan Boltzman yang ditentukan adalah temperature awal, *cooling rate*, dan kriteria pemberhentian. Temperatur awal ditentukan sebesar 100 dan *cooling rate* sebesar 0.9. Untuk kriteria pemberhentian dapat menggunakan 2 cara yaitu sampai temperature mencapai temperature akhir atau banyaknya iterasi. Pada studi kasus ini yang digunakan adalah banyak iterasi sebanyak 30 iterasi. Penggambaran *pseudocode* mengenai algoritma *Simulated Annealing* dapat ditunjukkan pada Kode 4.5

```
1: procedure LS Simulated Annealing
2:     current ← offspring(i)
3:     t ← initial temperature
4:     loop = 0
5:     maxLoop = 30
5:     repeat
6:      for i = 1 to l do
7:        candidate ← Neighborhood(current)
8:          if f(candidate) >= f(current) then
9:             current ← candidate
10:          else if
11:           exp(f(current) – f(candidate):t) >
              random[0,1)then
12:     current ← candidate
13:   end if
14:    end for
15:    update t and loop++
16:   until loop > maxLoop
17: end procedure
```

Kode 4.5 *Pseudocode* Pencarian Lokal *Simulated Annealing*

4.8.9. Pembuatan Solusi Baru

Solusi pada populasi awal serta hasil keturunan dari proses mutasi akan digabung yang selanjutnya dilakukan evaluasi fitness yang sama seperti pada bagian 4.8.3 dan dipilih 10 solusi terbaik berdasarkan nilai fitness-nya untuk menjadi populasi baru. Setelah populasi baru terbentuk, maka akan dilakukan pengulangan pada bagian 4.8.4 dan seterusnya hingga kriteria pemberhentian tercapai.

4.8.10. Pemberhentian Algoritma Memetika

Pembuatan populasi baru akan mengalami perulangan hingga memenuhi kondisi yang telah ditentukan yaitu jumlah generasi, dalam studi kasus ini algoritma akan berhenti apabila telah mengalami perulangan sebanyak 1000 generasi. Setelah generasi ke-1000 maka ditentukan solusi yang paling optimal pada populasi tersebut.

"Halaman ini sengaja dikosongkan"

BAB V
IMPLEMENTASI

Pada bab ini menjelaskan mengenai proses implementasi Algoritma Genetika dan Algoritma Memetika untuk mendapatkan solusi yang optimal dari studi kasus tugas akhir. Algoritma diimplementasikan dalam bentuk bahasa pemrograman Java dengan menggunakan tools NetBeans 8.0.2.

5.1. Pembaruan Matriks Waktu Tempuh dengan Algoritma Dijkstra

Salah satu batasan pada permasalahan OP adalah setiap simpul harus terhubung dengan simpul lainnya, akan tetapi pada matriks awal banyak sel yang berisi nilai 0 dikarenakan simpul tersebut tidak terhubung secara langsung dengan simpul yang dituju. Untuk mengisi nilai 0 tersebut, maka diimplementasikan algoritma Dijkstra. Algoritma ini bekerja dengan cara mencari jarak terpendek pada simpul yang tidak berhubungan secara langsung dengan menambahkan jarak-jarak pada simpul yang berhubungan. Matriks hasil dari algoritma ini akan digunakan sebagai masukan untuk implementasi Algoritma Genetika dan Algoritma Memetika.

```
22      static int jumlNode = 47; //inisialisasi jumlah titik
23      static int src = 0;
24
25      //fungsi untuk menemukan node dengan nilai waktu tempuh minimum
26      //dari set node yang belum memiliki nilai terpendek
27      int minDistance(int dist[], Boolean sptSet[]) {
28          // Initialize min value
29          int min = Integer.MAX_VALUE, min_index = 0;
30
31          for (int v = 0; v < jumlNode; v++) {
32              if (sptSet[v] == false && dist[v] <= min) {
33                  min = dist[v];
34                  min_index = v;
35              }
36          }
37          return min_index;
38      }
```

Kode 5.1 Fungsi Pencarian Waktu Tempuh Minimum

Kode 5.1 merupakan sebuah fungsi untuk menemukan simpul dengan nilai waktu tempuh minimum dari sekumpulan simpul yang belum memiliki nilai waktu tempuh terpendek.

```
40      // Method untuk mencetak array jarak yang telah di susun
41      void printSolution(int dist[], int n) {
42          for (int i = 0; i < jumlNode; i++) {
43              System.out.print(dist[i] + ",");
44          }
45      }
```

Kode 5.2 Fungsi Cetak Hasil Waktu Tempuh Tercepat

Kode 5.2 merupakan fungsi untuk mencetak matriks yang dihasilkan oleh algoritma Dijkstra.

```
47      void dijkstra(int graph[][], int src) {
48          int dist[] = new int[jumlNode];
49
50          Boolean sptSet[] = new Boolean[jumlNode];
51
52          for (int i = 0; i < jumlNode; i++) {
53              dist[i] = Integer.MAX_VALUE;
54              sptSet[i] = false;
55          }
56
57          dist[src] = 0;
58
59          for (int count = 0; count < jumlNode - 1; count++) {
60
61              int u = minDistance(dist, sptSet);
62
63              sptSet[u] = true;
64
65              for (int v = 0; v < jumlNode; v++)
66
67              {
68                  if (!sptSet[v] && graph[u][v] != 0
69                          && dist[u] != Integer.MAX_VALUE
70                          && dist[u] + graph[u][v] < dist[v]) {
71                      dist[v] = dist[u] + graph[u][v];
72                  }
73              }
74          }
75          // print the constructed distance array
76          printSolution(dist, jumlNode);
77      }
```

Kode 5.3 Fungsi Penerapan Algoritma Dijkstra

Kode 5.3 merupakan fungsi untuk mencari waktu tempuh tercepat dari simpul sumber ke seluruh simpul lainnya hingga memanggil fungsi mencetak hasil waktu tempuh dari seluruh simpul.

```
81      public static void main(String[] args) throws
82              FileNotFoundException, IOException {
83
84          String thisLine;
85          BufferedReader nodeMatrix = new BufferedReader
86          (new FileReader("data/MatriksJarakAwal.csv"));
87
88          ArrayList<String[]> lines = new ArrayList<>();
89          while ((thisLine = nodeMatrix.readLine()) != null) {
90              lines.add(thisLine.split(","));
91          }
92
93          //Convert our list to a String array
94          String[][] array = new String[lines.size()][0];
95          lines.toArray(array);
96
97          //Convert String array to Integer array
98          for (int i = 0; i < array.length; i++) {
99              for (int j = 0; j < array.length; j++) {
100                 arrNode[i][j] = Integer.parseInt(array[i][j]);
101             }
102         }
```

Kode 5.4 Fungsi Import Matriks Waktu Tempuh

Kode 5.4 merupakan fungsi untuk membaca file, mengkonversi data pada file menjadi bentuk integer yang akan digunakan sebagai input dari Algoritma Dijkstra.

```
104             Djikstra t = new Djikstra();
105             for (int i = 0; i < jumlNode; i++) {
106                 t.dijkstra(arrNode, i);
107                 System.out.println();
108             }
```

Kode 5.5 Fungsi Memanggil Algoritma Dijkstra

Kode 5.5 merupakan fungsi untuk memanggil Kode 5.3 yang telah berisi input dari file sebelumnya. Kode 5.3 akan dilakukan perulangan untuk seluruh simpul yang terdapat pada file. Hasil dari algoritma ini akan disimpan secara manual ke dalam bentuk

.csv sebagai masukan untuk penerapan Algoritma Genetika dan Algoritma Memetika.

5.2. Implementasi Algoritma Genetika

Pada sub bab ini dijelaskan mengenai implementasi Algoritma Genetika yang telah dirancang pada bab sebelumnya. Setiap tahapan AG akan disertakan dengan potongan kode-kode, fungsi, serta penjelasan secukupnya.

5.2.1. Inisialisasi Parameter dan Data

Pada tahap ini dijelaskan mengenai masukan yang digunakan dalam algoritma. Masukan tersebut meliputi parameter dan data matriks. Penentuan parameter yang akan digunakan yaitu deklarasi simpul awal, simpul akhir, jumlah simpul, batasan waktu, jumlah populasi, jumlah individu yang diseleksi, jumlah generasi, probabilitas perkawinan silang, dan probabilitas mutasi. Penentuan parameter dapat dilihat pada Kode 5.6.

```
//0.1 SET PARAMETER DAN KOMPONEN----------
    static int nodeAwal = 1; //Set simpul
    static int nodeAkhir = 47; //Set simpu
    static int jumlahNode = 48; //Jumlah s
    static int tMax = 80; //Set waktu temp
    static int jumlahPop = 100; //Set juml
    static int jumlahSelect = 10; //Set ju
    static int jumlahGenerasi = 1000; //Se
    static double probCrossOver = 0.95; //
    static double probMutation = 0.05; //S
//---------------------------------------
```

Kode 5.6 Inisialisasi Parameter AG

Kode 5.6 menjelaskan bahwa pada studi kasus ini terdapat sebanyak 48 simpul (jumlahSimpul), dan sesuai pada penjelasan bab sebelumnya, dicontohkan simpul awal yang digunakan adalah simpul 1 (simpulAwal) dan simpul akhir yang digunakan adalah simpul 47 (simpulAkhir). Batasan waktu

maksimal yang telah ditentukan sebesar 80 menit (tMax). Untuk mencari solusi, maka ditentukan jumlah populasi sebanyak 100 individu (jumlahPop), dengan jumlah pengulangan sebanyak 1000 kali iterasi (jumlahGenerasi). Pada setiap pengulangan, ditentukan jumlah individu yang diseleksi sebanyak 10 individu untuk dilakukan perkawinan silang. Probabilitas kawin silang yang akan digunakan sebesar 0.95 (probCrossOver) dan probabilitas mutasi sebesar 0.05 (probMutation).

Selanjutnya data matriks yang digunakan sebagai masukan AG akan disimpan dalam bentuk *array* yang ditunjukkan pada Kode 5.7

```
static int[][] nilaiArc = new int[jumlahNode][jumlahNode];
static int[] nilaiSkor = new int[jumlahNode];
```

Kode 5.7 *Array* Penyimpan Data Waktu Tempuh dan Skor

Kode 5.7 menjelaskan bahwa terdapat 2 data yang akan disimpan dalam bentuk *array*. Data waktu tempuh akan disimpan dalam *array* 2 dimensi (nilaiArc) dan data skor akan disimpan dalam *array* 1 dimensi (nilaiSkor).

Setelah tempat penyimpanan berbentuk *array* dibuat, langkah selanjutnya adalah mengkonversi data waktu tempuh dan data skor pada file .csv dan disimpan dalam *array* penyimpan data yang telah dibuat. Konversi data waktu tempuh dan data skor secara berurutan dapat ditunjukkan pada Kode 5.8 dan 5.9.

```
75  //0.3 KONVERSI ARRAY 2D UNTUK NILAI JARAK DARI FILE .CSV----------------
76          String thisLine2D;
77          BufferedReader bufRdr2D
78              = new BufferedReader(
79                      new FileReader("data/MatriksJarakKeseluruhan.csv"));
80
82          ArrayList<String[]> lines = new ArrayList<String[]>();
83          while ((thisLine2D = bufRdr2D.readLine()) != null) {
84              lines.add(thisLine2D.split(","));
85          }
86
87          //Mengkonversi list ke dalam String array
88          String[][] array = new String[lines.size()][0];
89          lines.toArray(array);
90
91          //Mengkonversi String array ke Integer array
92          for (int i = 0; i < array.length; i++) {
93              for (int j = 0; j < array.length; j++) {
94                  nilaiArc[i][j] = Integer.parseInt(array[i][j]);
95              }
          }
```

Kode 5.8 Konversi Data Waktu Tempuh

Kode 5.8 mengkonversi data dengan cara membaca file yang telah ditentukan, merubah tipe datanya menjadi *String*, merubah *String* menjadi *Integer*, dan menyimpannya ke dalam *array* (nilaiArc).

```
97  //0.4 KONVERSI ARRAY 1D UNTUK SCORE TIAP NODE DARI FILE .CSV-------------
98          BufferedReader bufRdr1D
99                  = new BufferedReader(new FileReader("data/MatriksSkor.csv"));
100
101         String thisLine1D;
102         int o = 0;
103         //Membaca file .csv dan memasukkannya ke dalam array integer
104         while ((thisLine1D = bufRdr1D.readLine()) != null) {
105             nilaiSkor[o] = Integer.parseInt(thisLine1D);
106             o++;
107         }
```

Kode 5.9 Konversi Data Skor

Kode 5.9 mengkonversi data dengan cara membaca file yang telah ditentukan, dan dikarenakan data yang terbaca sudah dalam bentuk *Integer*, maka dilanjutkan dengan menyimpan data ke dalam *array* (nilaiSkor).

5.2.2. Pembangkitan Populasi Awal

Pada tahap ini, mulai dilakukan pembangkitan populasi awal sebanyak jumlah individu yang telah ditetapkan pada parameter (pada kasus ini 100 individu). Dibangkitkan sejumlah individu berupa solusi layak yang memenuhi batasan pada bagian 4.4.3.

Selain itu dalam mencari solusi (individu) dilakukan iterasi sampai mencapai jumlah populasi yang telah ditentukan sebelumnya.

```
109   //1. PEMBANGKITAN POPULASI AWAL--------------------------
110   int pop = 0; //Menetapkan inisiasi jumlah individu dalam po
111   int loop = 0;
112   int maxLoop = 10000;
113   do {
114       //1.1. Menghasilkan nilai random, Rn , diantara indeks
115       //digunakan untuk menentukan panjang solusi yang berva
116       int Rn = new Random().nextInt(jumlahNode - 2) + 1;
117
118       //1.2. Menghasilkan list, Rl , diantara [1,N] secara ra
119       ArrayList<Integer> listRl = new ArrayList();
120       ArrayList<Integer> listRl_rand //Array yang digunakan u
121               = new ArrayList();     //node diantara node awa
122
123       //1.3. Menggenerate isi listRl_rand diantara node awal
124       for (int i = 1; i < jumlahNode + 1; i++) {
125           if (i == nodeAwal || i == nodeAkhir) {
126               continue; //Memastikan tidak ada node awal dan
127           }             //yang muncul di tengah kromosom
128           listRl_rand.add(i);
129       }
130       Collections.shuffle(listRl_rand); //Mengacak node yang
131
132       //1.4. Mengisi listRl
133       listRl.add(nodeAwal); //Merepresentasikan node awal
          for (int i = 0; i < listRl_rand.size(); i++) {
135           listRl.add(listRl_rand.get(i)); //Representasi node
136       }                                   //ke N-1 yang telah
137       listRl.add(nodeAkhir); //Merepresentasikan node akhir
```

Kode 5.10 Penggalan Kode Pembangkitan Populasi Awal (1)

Kode 5.10 menunjukkan inisiasi populasi awal yaitu 0 individu. Dilakukan pengulangan sebanyak mungkin sehingga individu pada populasi awal bertambah hingga mencapai jumlah individu yang diinginkan. Pada setiap iterasi akan ditentukan panjang gen setiap individu menggunakan angka acak (Rn). Selanjutnya dibuat *array* listRl_rand yang berisi panjang solusi maksimal tanpa simpul awal dan akhir (pada kasus ini berisi 45 simpul) yang kemudian urutan posisinya diacak. Selanjutnya membuat *array* listRl sebagai penyimpan solusi dengan menambahkan simpul awal, listRl_rand, dan simpul akhir.

Selanjutnya pada Kode 5.11 dibuat *array* subListRs untuk menyimpan solusi dari listRl sesuai dengan panjang gen yang telah ditetapkan oleh Rn. Dicontohkan apabila Rn mempunyai nilai 7 maka subListRs berisi panjang solusi sebesar 9 (simpul awal + Rn + simpul akhir). Urutan indeks subListRs diawali oleh simpul awal, indeks ke-2 sampai Rn+1 dari listRl, ditutup dengan simpul akhir. Setelah *array* subListRs terisi, selanjutnya menghitung nilai waktu tempuh dari subListRs. Apabila batasan waktu tempuh subListRs terpenuhi dan solusi tersebut tidak ada pada *array* penyimpan populasi (arrPopulation), maka solusi disimpan pada *array* populasi dan dilakukan pengulangan.

```
139     //1.5. Menghasilkan sebuah sub list, Rs dengan ukuran
140     //yang bervariasi sesuai dengan Rn.
141     ArrayList<Integer> subListRs = new ArrayList();
142     subListRs.add(listRl.get(0));
143     for (int i = 1; i < (Rn + 1); i++) {
144         subListRs.add(listRl.get(i));
145     }
146     subListRs.add(listRl.get(listRl.size() - 1));
147
148     //1.6. Hanya memasukkan subList Rs (individu) yang uni
149     //dan tidak melebihi batasan waktu tMax ke dalam popul
        for (int i = 0; i < subListRs.size(); i++) {
151         if (hitungWaktuTempuh(subListRs) <= tMax
152                 && !arrPopulation.contains(subListRs)) {
153             arrPopulation.add(subListRs);
154             pop++;
155         }
156     }
157     loop++;
```

Kode 5.11 Penggalan Kode Pembangkitan Populasi Awal (2)

```
159     if (loop == maxLoop) {
160         System.out.println("Tidak ditemukan solusi yang "
161                 + "layak untuk tMax = " + tMax);
162         System.exit(0);
163     }
```

Kode 5.12 Pembatalan Pembangkitan Populasi Awal

Apabila setelah dilakukan pengulangan hingga batas maksimal pengulangan tercapai, namun jumlah individu pada populasi tidak mencapai ketentuan, maka proses Algoritma Genetika dihentikan seperti pada Kode 5.12.

Selanjutnya apabila jumlah individu pada populasi awal sudah terisi maksimal, maka dilakukan pengisian *array* waktu tempuh (arrPopulation_WaktuTempuh) dan *array fitness* (arrPopulation_Fitness) pada populasi awal. Lalu dilakukan pencarian terhadap individu pada populasi awal dengan nilai *fitness* terbaik. Individu terbaik pada populasi awal dapat dicetak melalui proses yang dapat digambarkan pada Kode 5.13.

```
167   //1.7. Mengisi array untuk waktu tempuh dan fitness setiap
168   //individu dari populasi awal.
      for (int i = 0; i < arrPopulation.size(); i++) {
170       arrPopulation_WaktuTempuh.add(
171               hitungWaktuTempuh(arrPopulation.get(i)));
172       arrPopulation_Fitness.add(
173               hitungFitness(arrPopulation.get(i)));
174   }
175
176   //1.8. Mencari individu terbaik dari proses pembangkitan populasi awal.
177   int indSolusiAwal = 0;
178   int optFitnessAwal = arrPopulation_Fitness.get(0);
179   for (int i = 1; i < arrPopulation.size(); i++) {
180       if (arrPopulation_Fitness.get(i) >= optFitnessAwal) {
181           indSolusiAwal = i;
182           optFitnessAwal = arrPopulation_Fitness.get(i);
183       }
184   }
185
186   //1.9. Mencetak hasil individu terbaik.
187   pw.println("Generasi 0 | Rute: "
188           + arrPopulation.get(indSolusiAwal) + " | Waktu Tempuh: "
189           + arrPopulation_WaktuTempuh.get(indSolusiAwal) + " | Fitness: "
190           + arrPopulation_Fitness.get(indSolusiAwal));
```

Kode 5.13 Pencarian dan Pencetakan Individu Populasi Awal Terbaik

5.2.3. Perkawinan Silang

Pada tahap ini individu hasil seleksi dijadikan sebagai induk. Selanjutnya dipilih 2 induk secara berurutan dari indeks untuk dilakukan proses perkawinan silang. Proses ini dapat dijalankan apabila memenuhi probabilitas perkawinan silang yang telah ditentukan. Apabila tidak memenuhi probabilitas perkawinan silang yang ada, maka perkawinan silang pada indeks tersebut

dibatalkan dan dilanjutkan ke indeks selanjutnya. Jenis perkawinan silang yang digunakan adalah *single-point crossover* yang menghasilkan 2 keturunan setiap perkawinan silang.

```
228                 //3.1. Looping untuk semua kemungkinan perkawinan silang.
229                 for (int i = 0; i < arrSelection.size(); i += 2) {
230
231                     double rndProbCO = new Random().nextDouble();
                        List<Integer> offSpring1 = new ArrayList<Integer>();
                        List<Integer> offSpring2 = new ArrayList<Integer>();
234
235                     //3.2. Memastikan perkawinan silang dilakukan satu kali
236                     //pada setiap individu hasil seleksi.
237                     int j = i + 1;
238
239                     if (rndProbCO > probCrossOver) {
240                         continue; //Crossover hanya dilakukan jika memenuhi
241                     }             //probabilitas crossover
```

Kode 5.14 *Looping* Seluruh Solusi Seleksi dan Pemenuhan Probabilitas Perkawinan Silang

Pada Kode 5.18 menjelaskan penentuan kedua induk yang berasal dari indeks *ArrayList* seleksi secara berurutan, deklarasi bilangan random antara 0 dan 1 (*double*) yang digunakan untuk menentukan apakah pada indeks tersebut boleh dilakukan perkawinan silang atau tidak, dan membuat 2 array (offSpring1 dan offSpring2) untuk menyimpan hasil perkawinan silang.

Setelah 2 induk ditetapkan, induk yang mempunyai panjang kromosom lebih kecil akan bertindak sebagai penentu titik pemisah. Titik pemisah berguna untuk memisahkan kromosom pada masing-masing induk menjadi 2 bagian.

```
246                 if (arrSelection.get(i).size()
247                         <= arrSelection.get(j).size()) {
248
249                     //Membuat insertion point untuk Induk 1.
250                     int insertionPoint = new Random().nextInt(
251                             arrSelection.get(i).size() - 1) + 1;
252                     //Mengambil gen sejumlah insertion point
253                     //pada Induk 1 dimulai dari indeks 0.
254                     List<Integer> head1 = arrSelection.get(i).subList(0,
255                             insertionPoint);
256                     //Mengambil gen sejumlah insertion point
257                     //pada Induk 2 dimulai dari indeks 0.
258                     List<Integer> head2 = arrSelection.get(j).subList(0,
259                             insertionPoint);
260                     //Mengambil gen setelah insertion point pada
261                     //Induk 1 sampai indeks sebelum node akhir.
262                     List<Integer> tail1 = arrSelection.get(i).subList(
263                             insertionPoint,
264                             arrSelection.get(i).size() - 1);
265                     //Mengambil gen setelah insertion point pada
266                     //Induk 2 sampai indeks sebelum node akhir.
267                     List<Integer> tail2 = arrSelection.get(j).subList(
268                             insertionPoint,
269                             arrSelection.get(j).size() - 1);
```

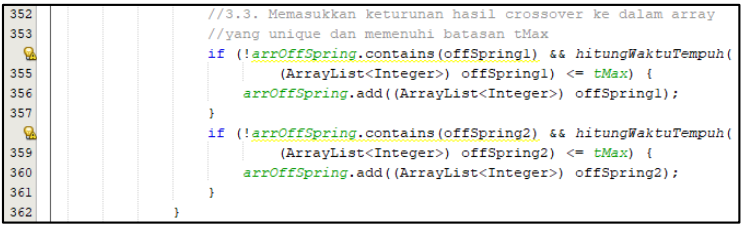
Kode 5.15 Pemisahan Gen pada Induk 1 dan Induk 2

Pada Kode 5.19 ditentukan terlebih dahulu induk yang mempunyai panjang kromosom lebih kecil dari induk lainnya. Induk dengan kromosom lebih kecil bertindak sebagai Induk 1 dan Induk dengan kromosom lebih besar bertindak sebagai Induk 2. Lalu ditentukan dimana letak titik pemisah (insertionPoint) pada kromosom Induk 1 secara acak. Setelah itu membagi kromosom pada Induk 1 menjadi 2 bagian yaitu head1 dengan panjang gen mulai indeks 0 hingga insertionPoint, dan tail1 dengan panjang gen mulai setelah insertionPoint hingga indeks terakhir. Hal yang sama diterapkan pada Induk 2 yaitu membagi kromosom menjadi 2 bagian yaitu head2 dengan panjang gen mulai indeks 0 hingga insertionPoint, dan tail2 dengan panjang gen mulai setelah insertionPoint hingga indeks terakhir.

```
271                 //Memasukkan semua gen dari head ke keturunan.
272                 offSpring1.addAll(head1);
273                 //Memasukkan gen dari tail ke dalam keturunan
274                 //dengan menyesuaikan ukuran Induk 1.
                    for(int k = 0; k < tail2.size(); k++){
276                     if (!offSpring1.contains(
277                                 tail2.get(k))) {
278                             offSpring1.add(tail2.get(k));
279                         }
280                 }
281                 //Memasukkan gen akhir
282                 offSpring1.add(nodeAkhir);
283
284                 //Memasukkan semua gen dari head ke keturunan.
285                 offSpring2.addAll(head2);
286                 //Memasukkan gen dari tail ke dalam keturunan
287                 //dengan menyesuaikan ukuran Induk 1.
                    for(int k = 0; k < tail1.size(); k++){
289                     if (!offSpring2.contains(
290                                 tail1.get(k))) {
291                             offSpring2.add(tail1.get(k));
292                         }
293                 }
294                 //Memasukkan gen akhir
295                 offSpring2.add(nodeAkhir);
```

Kode 5.16 Penyatuan Bagian Induk 1 dan Induk 2 Menjadi 2 Keturunan

Pada Kode 5.20 dilakukan penyatuan bagian kromosom sehingga terbentuk 2 individu baru yaitu keturunan 1 (offSpring1) dan keturunan 2 (offSpring2). Penyatuan keturunan 1 (offSpring1) dilakukan dengan cara menggabungkan bagian kromosom dari kepala induk 1 (head1) dengan bagian kromosom ekor induk 2 (tail2). Penyatuan keturunan 2 (offSpring2) dilakukan dengan cara menggabungkan bagian kromosom dari kepala induk 2 (head2) dengan bagian kromosom ekor induk 1 (tail1). Gen yang terduplikasi di setiap keturunan akibat penyatuan bagian akan dihilangkan mengikuti bagian kromosom kepala-nya.



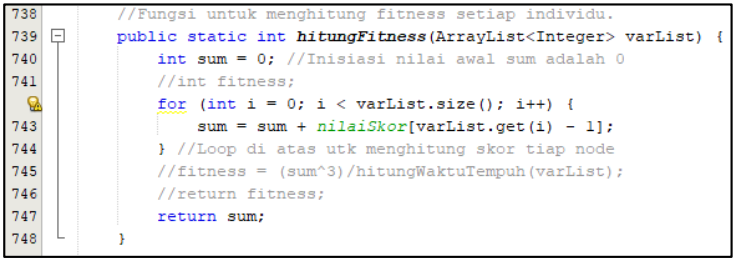
```
352                     //3.3. Memasukkan keturunan hasil crossover ke dalam array
353                     //yang unique dan memenuhi batasan tMax
                        if (!arrOffSpring.contains(offSpring1) && hitungWaktuTempuh(
355                         (ArrayList<Integer>) offSpring1) <= tMax) {
356                         arrOffSpring.add((ArrayList<Integer>) offSpring1);
357                     }
                        if (!arrOffSpring.contains(offSpring2) && hitungWaktuTempuh(
359                         (ArrayList<Integer>) offSpring2) <= tMax) {
360                         arrOffSpring.add((ArrayList<Integer>) offSpring2);
361                     }
362                 }
```

Kode 5.17 Menyimpan Hasil Perkawinan Silang ke dalam *Array* Keturunan

Setelah 2 keturunan baru terbentuk, selanjutnya menyimpan keturunan tersebut ke dalam *array* Keturunan (arrOffSpring). Setiap keturunan yang dihasilkan akan melalui pengecekan batas waktu tempuh, sehingga keturunan yang melebihi batas waktu tempuh maksimal tidak akan dimasukkan ke dalam *array* Keturunan (arrOffSpring). Proses ini dijabarkan dalam Kode 5.21.

5.2.4. Evaluasi Fitness

Pada tahap ini dilakukan perhitungan nilai *fitness* terbaik untuk mengevaluasi suatu populasi. Perhitungan nilai *fitness* dibuat dalam bentuk *method* dengan mengisikan simpul yang dilalui oleh individu dan menjumlahkan skor untuk setiap simpul yang dilalui. *Method* ini berfungsi agar nilai *fitness* dari setiap individu pada populasi dapat dibandingkan satu sama lain. Kode dari *method* ini dijabarkan dalam Kode 5.14.



```
738     //Fungsi untuk menghitung fitness setiap individu.
739     public static int hitungFitness(ArrayList<Integer> varList) {
740         int sum = 0; //Inisiasi nilai awal sum adalah 0
741         //int fitness;
            for (int i = 0; i < varList.size(); i++) {
743             sum = sum + nilaiSkor[varList.get(i) - 1];
744         } //Loop di atas utk menghitung skor tiap node
745         //fitness = (sum^3)/hitungWaktuTempuh(varList);
746         //return fitness;
747         return sum;
748     }
```

Kode 5.18 Fungsi Menghitung Nilai Fitness Individu

Berdasarkan Kode 5.14, masukan untuk *method* ini berupa *ArrayList* dari sebuah individu. *ArrayList* berisi simpul-simpul yang dilalui oleh individu. Nilai skor pada setiap simpul yang dilalui akan dijumlahkan dan disimpan dalam variabel sum. Ketika *method* ini dipanggil, maka akan mengeluarkan nilai *fitness* dari *ArrayList* yang digunakan.

5.2.5. Seleksi

Pada tahap ini dilakukan proses pemilihan sejumlah individu dalam populasi yang akan digunakan untuk proses perkawinan silang. Jumlah individu yang diseleksi sesuai dengan parameter yang telah ditentukan. Seleksi dilakukan dengan menggunakan *Roulette Wheel Selection* agar individu dapat dipilih berdasarkan probabilitas sesuai dengan nilai *fitness*-nya. Individu dengan nilai *fitness* tertinggi mempunyai peluang terpilih lebih besar daripada individu lainnya.

Untuk menentukan probabilitas setiap individu pada populasi, terlebih dahulu dibuat *method* untuk menjumlahkan seluruh nilai *fitness* individu pada populasi. Hasil dari *method* ini selanjutnya digunakan untuk menjalankan *Roulette Wheel Selection*. *Method* ini dijabarkan pada Kode 5.15.

```
750     //Fungsi untuk menghitung total fitness pada populasi.
751     public static int totalFitness() {
752         int sum = 0; //Inisiasi nilai awal sum adalah 0
            for (int i = 0; i < arrPopulation_Fitness.size(); i++) {
754             sum = sum + arrPopulation_Fitness.get(i);
755         } //Loop di atas utk menghitung fitness tiap individu pd populasi
756         return sum;
757     }
```

Kode 5.19 Fungsi Menghitung Total Fitness Populasi

```
760     public static int rouletteWheelSelection() {
761         double r = new Random().nextDouble();
762         int totFitness = totalFitness();
763         int k = 0;
764         double sum = (double) arrPopulation_Fitness.get(k) / totFitness;
765         while (sum < r) {
766             k++;
767             sum = sum + (double) arrPopulation_Fitness.get(k) / totFitness;
768         }
769         return k;
770     }
```

Kode 5.20 Fungsi *Roulette Wheel Selection*

Selanjutnya sesuai dengan Kode 5.16 *method Roulette Wheel Selection* dijalankan dengan cara membuat bilangan random antara 0 sampai 1 (*double*), lalu memanggil *method* totalFitness yang disimpan dalam variabel totFitness, mendeklarasikan variabel k untuk menentukan solusi yang akan terpilih, dan variabel sum yang digunakan untuk menyimpan nilai probabilitas dari individu sesuai indeks k. Probabilitas didapatkan dengan membagi nilai *fitness* individu tersebut dengan totFitness. Apabila probabilitas individu lebih besar dibandingkan bilangan random maka individu dengan indeks k akan terpilih, namun jika tidak, maka pencarian akan berlanjut ke indeks berikutnya.

```
202                 //2.1. Mengambil individu dari populasi awal dengan jumlah
203                 //yang telah ditentukan sebelumnya
204                 do {
205                     int id = rouletteWheelSelection();
206                     if (!arrSelection.contains(arrPopulation.get(id))) {
207                         arrSelection.add(arrPopulation.get(id));
208                     }
209                 } while (arrSelection.size() < jumlahSelect);
```

Kode 5.21 Proses Seleksi Menggunakan *Roulette Wheel Selection*

Berdasarkan Kode 5.17, proses seleksi dilakukan dengan memanggil *method Roulette Wheel Selection* untuk menentukan indeks pada *ArrayList* populasi yang selanjutnya disimpan dalam *ArrayList* seleksi. Dilakukan pengulangan pemanggilan *method Roulette Wheel Selection* hingga *ArrayList* seleksi terisi sesuai jumlah seleksi yang telah ditetapkan pada parameter awal.

5.2.6. Mutasi

Keturunan hasil dari perkawinan silang akan dimutasi sesuai dengan probabilitas masing-masing keturunan. Apabila nilai probabilitas saat mutasi lebih besar dari dari probabilitas mutasi yang telah ditentukan, maka proses mutasi keturunan tersebut tidak akan dilakukan. Setiap keturunan yang berhasil memenuhi probabilitas akan dilakukan mutasi dengan menerapkan salah satu operator mutasi dari empat operator mutasi yang ada.

Operator tersebut adalah operator *add*, *add-remove*, *add-swap*, dan *remove*.

```
380   //4. MELAKUKAN MUTASI-----------------------------------------
381           for (int i = 0; i < arrOffSpring.size(); i++) {
382               double rndProbMutation = new Random().nextDouble();
383               if (rndProbMutation > probMutation) {
384                   continue;
385               }
386               int loopMutation = 0;
387               int maxLoopMutation = 1;
388
389               do{
390                   int randomOperator = new Random().nextInt(3) + 1;
391                   switch (randomOperator) {
```

Kode 5.22 *Looping* Seluruh Keturunan dan Pemenuhan Probabilitas Mutasi

Pada Kode 5.22 setiap keturunan dilakukan pembentukan bilangan random antara 0 dan 1 (*double*). Keturunan dengan bilangan random lebih kecil dari probabilitas mutasi akan mengalami 1 dari 4 operasi mutasi yang ada secara acak.

```
393     case 1:
394     //4.1. Operator Add Random.
395     int loopAdd = 0;
396     int maxLoopAdd = 1;
397
398     do {
399         int rndInsert = new Random().nextInt(
400                 arrOffSpring.get(i).size() - 2) + 1;
401         int rndNilaiInsert
402                 = new Random().nextInt(jumlahNode) + 1;
403         if (arrOffSpring.get(i).contains(rndNilaiInsert)
404                 || rndNilaiInsert == nodeAwal
405                 || rndNilaiInsert == nodeAkhir) {
406             continue; //Memastikan node yang dimasukkan bukan
407         }             //merupakan node awal dan akhir
408         arrOffSpring.get(i).add(rndInsert, rndNilaiInsert);
409         //Menyimpan waktu tempuh dan fitness dari hasil proses add
410         int currentWaktuTempuh
411                 = hitungWaktuTempuh(arrOffSpring.get(i));
412         int currentFitness
413                 = hitungFitness(arrOffSpring.get(i));
414         //Jika proses add menghasilkan fitness
415         //yang lebih baik atau sama dengan sebelumnya, maka
416         //hasil mutasi akan disimpan menggantikan yang lama.
417         if (currentWaktuTempuh <= tMax
418                 && currentFitness >= arrOffSpring_Fitness.get(i)) {
419             arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
420             arrOffSpring_Fitness.set(i, currentFitness);
421         } else {
422            //Jika tidak lebih baik, maka dikembalikan seperti semula
423             arrOffSpring.get(i).remove(rndInsert);
424         }
425         loopAdd++;
426
427     } while (loopAdd < maxLoopAdd);
428     loopMutation++;
429     break;
```

Kode 5.23 Operator Mutasi *Add*

Pada Kode 5.23 operasi mutasi yang dilakukan yaitu dengan menambahkan 1 gen yang berisi simpul dengan nomor acak ke dalam kromosom keturunan. Posisi gen tersebut diselipkan secara acak diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan nilai *fitness* antara individu keturunan dengan individu yang telah termutasi. Apabila nilai *fitness* individu termutasi lebih baik dari individu keturunan dan masih memenuhi batas waktu tempuh maksimal, maka individu keturunan digantikan dengan individu termutasi. Apabila nilai *fitness* individu termutasi lebih

buruk dari individu keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

```
431     case 2:
432     //4.2. Operator Add-Remove Random.
433     int loopAddRemove = 0;
434     int maxLoopAddRemove = 5;
435
436     do {
437         int rndInsert = new Random().nextInt(
438                 arrOffSpring.get(i).size() - 2) + 1;
439         int rndNilaiInsert
440                 = new Random().nextInt(jumlahNode) + 1;
441         if (arrOffSpring.get(i).contains(rndNilaiInsert)
442                 || rndNilaiInsert == nodeAwal
443                 || rndNilaiInsert == nodeAkhir) {
444             continue; //Memastikan node yang dimasukkan bukan
445         }             //merupakan node awal dan akhir
446         arrOffSpring.get(i).add(rndInsert, rndNilaiInsert);
447
448         //Menyimpan waktu tempuh dan fitness dari hasil proses add
449         int currentWaktuTempuh
450                 = hitungWaktuTempuh(arrOffSpring.get(i));
451         int currentFitness
452                 = hitungFitness(arrOffSpring.get(i));
453         //Jika proses add menghasilkan waktu tempuh dan fitness
454         //yang lebih baik atau sama dengan sebelumnya, maka
455         //hasil mutasi akan disimpan menggantikan yang lama.
456         if (currentWaktuTempuh <= arrOffSpring_WaktuTempuh.get(i)
457                 && currentFitness >= arrOffSpring_Fitness.get(i)) {
458             arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
459             arrOffSpring_Fitness.set(i, currentFitness);
460             //loopMutation = 0; //looping dikembalikan ke awal
461         } else {
462             arrOffSpring.get(i).remove(rndInsert);
463         } //Jika tidak lebih baik, maka dikembalikan seperti semula
464         loopAddRemove++;
465
466     } while (loopAddRemove < maxLoopAddRemove);
```

Kode 5.24 Operator Mutasi *Add-Remove* (1)

Pada Kode 5.24 operasi mutasi yang dilakukan yaitu dengan menambahkan 1 gen yang berisi simpul dengan nomor acak ke dalam kromosom keturunan. Posisi gen tersebut diselipkan secara acak diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan nilai *fitness* antara individu keturunan dengan individu yang telah termutasi. Apabila nilai *fitness* individu termutasi lebih baik dari individu keturunan dan masih memenuhi batas waktu

tempuh maksimal, maka individu keturunan digantikan dengan individu termutasi. Apabila nilai *fitness* individu termutasi lebih buruk dari individu keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

```
468     do {
469         int rndDelete = new Random().nextInt(
470                 arrOffSpring.get(i).size() - 2) + 1;
471         //Proses omit dilakukan jika ukuran keturunan lebih dari 3
472         if (arrOffSpring.get(i).size() > 3) {
473             //Temp menyimpan nilai dari gen/node yang dihapus
474             int temp = arrOffSpring.get(i).get(rndDelete);
475             arrOffSpring.get(i).remove(rndDelete);
476             //Menyimpan waktu tempuh & fitness hasil proses omit
477             int currentWaktuTempuh
478                     = hitungWaktuTempuh(arrOffSpring.get(i));
479             int currentFitness
480                     = hitungFitness(arrOffSpring.get(i));
481             if (currentWaktuTempuh
482                     < arrOffSpring_WaktuTempuh.get(i)) {
483                 arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
484                 arrOffSpring_Fitness.set(i, currentFitness);
485             } else {
486                 arrOffSpring.get(i).add(rndDelete, temp);
487             } //Jika tdk lebih baik, maka dikembalikan sprt semula
488
489         } else {
490             break;
491         }
492
493         loopAddRemove++;
494
495     } while (loopAddRemove < maxLoopAddRemove);
496     loopMutation++;
497     break;
```

Kode 5.25 Operator Mutasi *Add-Remove* (2)

Dilanjutkan dengan operasi mutasi pada Kode 5.25 yaitu dengan menghilangkan 1 gen secara acak yang terdapat pada kromosom individu keturunan. Gen yang dihilangkan tersebut berada diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan waktu tempuh antara individu keturunan dengan individu yang telah termutasi. Apabila waktu tempuh individu termutasi lebih kecil dari individu keturunan, maka individu keturunan digantikan dengan individu termutasi. Apabila waktu tempuh individu termutasi lebih besar atau sama dari individu

keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

```
499     case 3:
500     //4.3. Operator Add-Swap Random.
501     int loopAdd2 = 0;
502     int maxLoopAdd2 = 1;
503
504     int loopSwap = 0;
505     int maxLoopSwap = 1;
506
507     do {
508         int rndInsert = new Random().nextInt(
509                 arrOffSpring.get(i).size() - 2) + 1;
510         int rndNilaiInsert
511                 = new Random().nextInt(jumlahNode) + 1;
512         if (arrOffSpring.get(i).contains(rndNilaiInsert)
513                 || rndNilaiInsert == nodeAwal
514                 || rndNilaiInsert == nodeAkhir) {
515             continue; //Memastikan node yang dimasukkan bukan
516         }             //merupakan node awal dan akhir
517         arrOffSpring.get(i).add(rndInsert, rndNilaiInsert);
518         int currentWaktuTempuh
519             = hitungWaktuTempuh(arrOffSpring.get(i));
520         int currentFitness
521             = hitungFitness(arrOffSpring.get(i));
522         //Jika proses add menghasilkan fitness
523         //yang lebih baik atau sama dengan sebelumnya, maka
524         //hasil mutasi akan disimpan menggantikan yang lama.
525         if (currentWaktuTempuh <= tMax
526                 && currentFitness >= arrOffSpring_Fitness.get(i)) {
527             arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
528             arrOffSpring_Fitness.set(i, currentFitness);
529         } else {
530             //Jika tidak lebih baik, maka dikembalikan seperti semula
531             arrOffSpring.get(i).remove(rndInsert);
532         }
533         loopAdd2++;
534     } while (loopAdd2 < maxLoopAdd2);
```

Kode 5.26 Operator Mutasi *Add-Swap* (1)

Pada Kode 5.26 operasi mutasi yang dilakukan yaitu dengan menambahkan 1 gen yang berisi simpul dengan nomor acak ke dalam kromosom keturunan. Posisi gen tersebut diselipkan secara acak diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan nilai *fitness* antara individu keturunan dengan individu yang telah termutasi. Apabila nilai *fitness* individu termutasi lebih baik dari individu keturunan dan masih memenuhi batas waktu

tempuh maksimal, maka individu keturunan digantikan dengan individu termutasi. Apabila nilai *fitness* individu termutasi lebih buruk dari individu keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

```
535     do {
536         if (arrOffSpring.get(i).size() > 3) {
537             int tukar1 = new Random().nextInt(
538                 arrOffSpring.get(i).size() - 2) + 1;
539             int tukar2 = new Random().nextInt(
540                 arrOffSpring.get(i).size() - 2) + 1;
541             Collections.swap(arrOffSpring.get(i), tukar1, tukar2);
542             int currentWaktuTempuh
543                 = hitungWaktuTempuh(arrOffSpring.get(i));
544             int currentFitness
545                 = hitungFitness(arrOffSpring.get(i));
546             //Jika proses swap menghasilkan waktu tempuh
547             //yang lebih baik atau sama dengan sebelumnya, maka
548             //hasil mutasi akan disimpan menggantikan yang lama.
549             if (currentWaktuTempuh <= arrOffSpring_WaktuTempuh.get(i)) {
550                 arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
551                 arrOffSpring_Fitness.set(i, currentFitness);
552                 //loopMutation = 0; //looping dikembalikan ke awal
553             } else {
554                 //Jika tidak lebih baik, maka dikembalikan seperti semula
555                 Collections.swap(arrOffSpring.get(i), tukar2, tukar1);
556                 arrOffSpring.get(i).remove(tukar1);
557             }
558         } else {
559             break;
560         }
561         loopSwap++;
562 
563     } while (loopSwap < maxLoopSwap);
564     loopMutation++;
565     break;
```

Kode 5.27 Operator Mutasi *Add-Swap* (2)

Dilanjutkan dengan operasi mutasi pada Kode 5.27 yaitu dengan menukar nilai gen hasil operasi mutasi Kode 5.26 dengan nilai gen pada indeks lain secara acak. Gen yang dapat ditukar berada diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan nilai *fitness* antara individu keturunan dengan individu yang telah termutasi. Apabila nilai *fitness* individu termutasi lebih baik dari individu keturunan dan masih memenuhi batas waktu tempuh maksimal, maka individu keturunan digantikan dengan individu termutasi. Apabila nilai

fitness individu termutasi lebih buruk dari individu keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

```
567     case 4:
568     //4.4 Operator Remove Random
569         int loopRemove = 0;
570         int maxLoopRemove = 5;
571
572     do {
573         int rndDelete = new Random().nextInt(
574                 arrOffSpring.get(i).size() - 2) + 1;
575         //Proses omit dilakukan jika ukuran keturunan lebih dari 3
576         if (arrOffSpring.get(i).size() > 3) {
577             //Temp menyimpan nilai dari gen/node yang dihapus
578             int temp = arrOffSpring.get(i).get(rndDelete);
579             arrOffSpring.get(i).remove(rndDelete);
580             //Menyimpan waktu tempuh & fitness hasil proses omit
581             int currentWaktuTempuh
582                     = hitungWaktuTempuh(arrOffSpring.get(i));
583             int currentFitness
584                     = hitungFitness(arrOffSpring.get(i));
585             if (currentWaktuTempuh
586                     < arrOffSpring_WaktuTempuh.get(i)) {
587                 arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
588                 arrOffSpring_Fitness.set(i, currentFitness);
589                 //loopMutation = 0; //looping dikembalikan ke awal
590             } else {
591                 arrOffSpring.get(i).add(rndDelete, temp);
592             } //Jika tdk lebih baik, maka dikembalikan sprt semula
593         } else {
594                 break;
595             }
596             loopRemove++;
597         } while (loopRemove < maxLoopRemove);
598         loopMutation++;
599     break;
```

Kode 5.28 Operator Mutasi *Remove*

Pada Kode 5.28 operasi mutasi yang dilakukan yaitu dengan menghilangkan 1 gen secara acak yang terdapat pada kromosom individu keturunan. Gen yang dihilangkan tersebut berada diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Setelah itu dilakukan perbandingan waktu tempuh antara individu keturunan dengan individu yang telah termutasi. Apabila waktu tempuh individu termutasi lebih kecil dari individu keturunan, maka individu keturunan digantikan dengan individu termutasi. Apabila waktu tempuh

individu termutasi lebih besar atau sama dari individu keturunan maka proses mutasi pada individu keturunan tersebut tidak dilakukan.

5.2.7. Pembuatan Populasi Baru

Pada tahap ini populasi baru dimunculkan dengan cara menggabungkan individu yang terdapat pada populasi awal dan individu keturunan hasil mutasi ke dalam 1 *array* penyimpan. Setelah itu dilakukan proses penyeleksian individu untuk menentukan sebagian individu yang bertahan pada populasi baru berdasarkan jumlah populasi yang telah ditentukan. Sebagian individu yang dibuang merupakan individu dengan nilai *fitness* terburuk dari populasi yang ada.

```
612     //5. MEMBUAT POPULASI BARU--------
613     //Memastikan array utk simpan indi
614     arrSimpanan.clear();
615     arrSimpanan_WaktuTempuh.clear();
616     arrSimpanan_Fitness.clear();
```

Kode 5.29 Penghapusan Isi *Array* Penyimpan

Sebelum populasi baru dibuat, terlebih dahulu dilakukan penghapusan isi dari *array* penyimpan (arrSimpanan) seperti pada Kode 5.29. *Array* penyimpan ini berfungsi untuk menyimpan individu populasi awal dengan individu hasil mutasi.

```
618     //Memasukkan populasi ke dalam array simpanan.
        for (int i = 0; i < arrPopulation.size(); i++) {
620         if (!arrSimpanan.contains(arrPopulation.get(i))) {
621             arrSimpanan.add(arrPopulation.get(i));
622         }
623     }
624
625     //Memasukkan keturunan ke dalam array simpanan
626     for (int i = 0; i < arrOffSpring.size(); i++) {
627         if (arrOffSpring_WaktuTempuh.get(i) <= tMax
628                 && !arrSimpanan.contains(arrOffSpring.get(i))) {
629             arrSimpanan.add(arrOffSpring.get(i));
630         }
631     }
```

Kode 5.30 Penggabungan Individu Populasi Awal dengan Individu Hasil Mutasi

Pada Kode 5.30 dilakukan penyimpanan individu dari populasi awal dan individu hasil mutasi ke dalam *array* penyimpan (arrSimpanan). Dilakukan juga pengecekan setiap individu agar individu yang tersimpan dalam *array* bersifat *unique* atau kromosom tidak sama.

```
633     //Mengisi array untuk waktu tempuh dan fitness setiap
634     //individu dari array simpanan.
        for (int i = 0; i < arrSimpanan.size(); i++) {
636         arrSimpanan_WaktuTempuh.add(
637                 hitungWaktuTempuh(arrSimpanan.get(i)));
638         arrSimpanan_Fitness.add(
639                 hitungFitness(arrSimpanan.get(i)));
640     }
```

Kode 5.31 Penyimpanan Waktu Tempuh dan *Fitness* dari *Array* Penyimpan

Pada Kode 5.31 dilakukan penyimpanan waktu tempuh dan nilai *fitness* dari seluruh individu dalam *array* penyimpan.

```
642     //Mengosongkan isi dari array yang menyimpan populasi
643     arrPopulation.clear();
644     arrPopulation_WaktuTempuh.clear();
645     arrPopulation_Fitness.clear();
646
647     //Mengosongkan isi dari array yang menyimpan hasil seleksi
648     arrSelection.clear();
649     arrSelection_WaktuTempuh.clear();
650     arrSelection_Fitness.clear();
651
652     //Mengosongkan isi dari array yang menyimpan keturunan
653     arrOffSpring.clear();
654     arrOffSpring_WaktuTempuh.clear();
655     arrOffSpring_Fitness.clear();
```

Kode 5.32 Penghapusan Isi *Array* Populasi, Seleksi, dan Keturunan

Selanjutnya menghapus isi *array* populasi (arrPopulation), hasil seleksi (arrSelection), dan keturunan (arrOffSpring) seperti pada Kode 5.32. Proses ini dilakukan agar *array* dapat digunakan kembali pada pengulangan selanjutnya.

```
657     //Memilih individu terbaik dari populasi sebelumnya dan
658     //sejumlah populasi sebelumnya.
659     do {
660         int indSolusi = 0;
661         int optFitness = arrSimpanan_Fitness.get(0);
662         for (int i = 1; i < arrSimpanan.size(); i++) {
663             if (arrSimpanan_Fitness.get(i) >= optFitness) {
664                 indSolusi = i;
665                 optFitness = arrSimpanan_Fitness.get(i);
666             }
667         }
668         arrPopulation.add(arrSimpanan.get(indSolusi));
669         arrPopulation_WaktuTempuh.add(
670                 arrSimpanan_WaktuTempuh.get(indSolusi));
671         arrPopulation_Fitness.add(
672                 arrSimpanan_Fitness.get(indSolusi));
673         arrSimpanan.remove(indSolusi);
674         arrSimpanan_WaktuTempuh.remove(indSolusi);
675         arrSimpanan_Fitness.remove(indSolusi);
676     } while (arrPopulation.size() < jumlahPop);
```

Kode 5.33 Pemilihan Individu dari *Array* Penyimpan untuk Populasi Baru

Berdasarkan Kode 5.33, dilakukan sejumlah pengulangan untuk mendapatkan individu pada *array* penyimpan (arrSimpanan) yang kemudian individu tersebut disimpan ke dalam *array* populasi (arrPopulasi). Pada setiap pengulangan, individu dipilih berdasarkan nilai *fitness* terbaik dalam *array* penyimpan, lalu disimpan dalam *array* populasi, kemudian individu tersebut dihapus pada *array* penyimpan. Pengulangan berhenti hingga isi dari *array* populasi mencapai batas jumlah populasi yang telah ditentukan.

```
694     //Mencari individu terbaik dari populasi baru yang terbentuk
695     int indSolusi = 0;
696     int optFitness = arrPopulation_Fitness.get(0);
697     int optWaktuTempuh = arrPopulation_WaktuTempuh.get(0);
698     int jmlNode = arrPopulation.get(0).size();
699     for (int i = 1; i < arrPopulation.size(); i++) {
700         if (arrPopulation_Fitness.get(i) >= optFitness
701         /*    && arrPopulation_WaktuTempuh.get(i) <= optWaktuTempuh */
702             indSolusi = i;
703             optFitness = arrPopulation_Fitness.get(i);
704             optWaktuTempuh = arrPopulation_WaktuTempuh.get(i);
705         }
706     }
707
708     //Mencetak hasil individu terbaik dari setiap generasi
709     pw.println("Generasi " + (generasi + 1) + " | Rute: "
710             + arrPopulation.get(indSolusi) + " | Waktu Tempuh: "
711             + arrPopulation_WaktuTempuh.get(indSolusi) + " | Fitness: "
712             + arrPopulation_Fitness.get(indSolusi));
```

Kode 5.34 Pencarian Solusi Terbaik pada Populasi Baru

Selanjutnya pada Kode 5.34 dilakukan iterasi untuk membandingkan satu per satu individu yang ada dalam array populasi (arrPopulation) dimana individu dengan nilai fitness tertinggi akan terpilih sebagai solusi terbaik dan solusi tersebut dapat dicetak ke dalam file keluaran.

5.2.8. Pemberhentian Algoritma Genetika

Setelah semua tahapan sebelumnya dilakukan, maka variabel generasi akan bertambah dan dilakukan pengulangan berdasarkan tahapan pada bagian 5.2.4. Pengulangan terjadi hingga variabel generasi memenuhi batasan jumlah generasi yang telah ditentukan. Proses ini dijabarkan dalam Kode 5.35.

```
714         generasi++;
715
716     } while (generasi < jumlahGenerasi);
717
718     pw.close();
```

Kode 5.35 Kriteria Pemberhentian Algoritma

5.3. Implementasi Algoritma Memetika *Hill Climbing*

Pada sub bab dijelaskan mengenai implementasi Algoritma Memetika Hill Climbing (AMHC) yang telah dirancang pada bab sebelumnya. Tahapan-tahapan pada AMHC hampir sama dengan AG, sehingga yang akan dijelaskan di sub bab ini adalah tahapan yang membedakan AMHC dengan AG. Tahapan yang berbeda adalah pada tahap inisialiasi parameter dan tahapan pencarian lokal. Tahapan pencarian lokal dilakukan setelah melalui tahapan mutasi (5.2.6) dan sebelum tahapan pembuatan populasi baru (5.2.7). Tahapan tersebut akan disertakan dengan potongan kode-kode, fungsi, serta penjelasan secukupnya.

5.3.1. Inisialisasi Parameter

Perbedaan tahap ini dengan tahap 5.2.1 adalah deklarasi variabel parameter yang digunakan. Penentuan parameter yang akan digunakan yaitu deklarasi simpul awal, simpul akhir, jumlah simpul, batasan waktu, jumlah populasi, jumlah individu yang diseleksi, jumlah generasi, probabilitas perkawinan silang, dan probabilitas mutasi. Penentuan parameter dapat dilihat pada Kode 5.36.

```
static int nodeAwal = 1;
static int nodeAkhir = 47;
static int jumlahNode = 48;
static int tMax = 80; //Menetapkan ni
static int jumlahPop = 10; //Menetapk
static int jumlahSelect = 10; //Menet
static int jumlahGenerasi = 1000; //M
static double probCrossOver = 0.5;
static double probMutation = 0.5;
```

Kode 5.36 Inisialisasi Parameter AMHC

Kode 5.36 menjelaskan bahwa pada studi kasus ini terdapat sebanyak 48 simpul (jumlahSimpul), dan sesuai pada penjelasan bab sebelumnya, simpul awal yang digunakan adalah simpul 1 (simpulAwal) dan simpul akhir yang digunakan adalah simpul 47 (simpulAkhir). Batasan waktu maksimal yang telah ditentukan sebesar 80 menit (tMax). Untuk mencari solusi, maka ditentukan jumlah populasi sebanyak 10 individu (jumlahPop), dengan jumlah pengulangan sebanyak 1000 kali iterasi (jumlahGenerasi). Pada setiap pengulangan, ditentukan jumlah individu yang diseleksi sebanyak 10 individu untuk dilakukan perkawinan silang. Probabilitas kawin silang yang akan digunakan sebesar 0.5 (probCrossOver) dan probabilitas mutasi sebesar 0.5 (probMutation).

5.3.2. Pencarian Lokal *Hill Climbing*

Setelah tahap 5.3.1 dilakukan, dilanjutkan dengan tahapan sesuai pada AG (5.2.2 hingga 5.2.6). Individu keturunan hasil mutasi akan melalui tahapan perbaikan lokal. Perbaikan lokal dilakukan dengan menerapkan sejumlah pengulangan pada setiap individu keturunan. Pada setiap pengulangan, dilakukan operasi *add-remove* pada individu keturunan, dilanjutkan dengan membandingkan hasil operasi dengan individu keturunan. Apabila individu hasil operasi mempunyai nilai *fitness* lebih baik dari individu keturunan, maka individu tersebut diganti. Lalu dilakukan pengulangan terhadap individu

baru dengan operasi yang sama hingga mencapai batas jumlah pengulangan pencarian lokal yang ada.

```
629     //5. LOCAL SEARCH (HILL CLIMBING)
630     for (int i = 0; i < arrOffSpring.size(); i++) {
631
632         int loopLocalSearch = 0;
633         int maxLoopLocalSearch = 1000;
634
635         do {
636
637             int rndInsert = new Random().nextInt(
638                     arrOffSpring.get(i).size() - 2) + 1;
639             int rndNilaiInsert
640                     = new Random().nextInt(jumlahNode) + 1;
641             if (arrOffSpring.get(i).contains(rndNilaiInsert)
642                     || rndNilaiInsert == nodeAwal
643                     || rndNilaiInsert == nodeAkhir) {
644                 continue; //Memastikan node yang dimasukkan bu
645             }             //merupakan node awal dan akhir
646             arrOffSpring.get(i).add(rndInsert, rndNilaiInsert);
647             int rndDelete = new Random().nextInt(
648                     arrOffSpring.get(i).size() - 2) + 1;
649             //Proses omit dilakukan jika ukuran keturunan lebi
650             if (arrOffSpring.get(i).size() > 3) {
651                 //Temp menyimpan nilai dari gen/node yang diha
652                 int temp = arrOffSpring.get(i).get(rndDelete);
653                 arrOffSpring.get(i).remove(rndDelete);
```

Kode 5.37 Penggalan Kode Pencarian Lokal *Hill Climbing* (1)

Pada Kode 5.37 perbaikan lokal dilakukan dengan menambahkan 1 gen yang berisi simpul dengan nomor acak ke dalam kromosom keturunan. Posisi gen tersebut diselipkan secara acak diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Lalu menghilangkan 1 gen secara acak yang terdapat pada kromosom individu keturunan. Gen yang dihilangkan tersebut berada diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir.

Setelah itu diterapkan Kode 5.38 yaitu melakukan perbandingan nilai *fitness* antara individu keturunan dengan individu hasil perbaikan lokal. Apabila nilai *fitness* individu hasil perbaikan lokal lebih baik dari individu keturunan dan masih memenuhi batas waktu tempuh maksimal, maka individu

keturunan digantikan dengan individu hasil perbaikan lokal. Apabila nilai *fitness* individu hasil perbaikan lokal lebih buruk dari individu keturunan maka dilakukan pengulangan perbaikan lokal.

```
655             int currentWaktuTempuh
656                     = hitungWaktuTempuh(arrOffSpring.get(i));
657             int currentFitness
658                     = hitungFitness(arrOffSpring.get(i));
659             //Jika proses add menghasilkan waktu tempuh dan fitness
660             //yang lebih baik atau sama dengan sebelumnya, maka
661             //hasil mutasi akan disimpan menggantikan yang lama.
662             if (currentWaktuTempuh <= tMax
663                     && currentFitness >= arrOffSpring_Fitness.get(i)) {
664                 arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
665                 arrOffSpring_Fitness.set(i, currentFitness);
666                 //loopMutation = 0; //looping dikembalikan ke awal
667             } else {
668                 arrOffSpring.get(i).add(rndDelete, temp);
669                 arrOffSpring.get(i).remove(rndInsert);
670             } //Jika tidak lebih baik, maka dikembalikan seperti semula
671         } else {
672             arrOffSpring.get(i).remove(rndInsert);
673         }
674         loopLocalSearch++;
675
676     } while (loopLocalSearch < maxLoopLocalSearch);
```

Kode 5.38 Penggalan Kode Pencarian Lokal *Hill Climbing* (2)

5.4. Implementasi Algoritma Memetika *Simulated Annealing*

Pada sub bab ini dijelaskan mengenai implementasi Algoritma Memetika Simulated Annealing (AMSA) yang telah dirancang pada bab sebelumnya. Tahapan-tahapan pada AMSA hampir sama dengan AG, sehingga yang akan dijelaskan di sub bab ini adalah tahapan yang membedakan AMSA dengan AG. Tahapan yang berbeda adalah pada tahap inisialiasi parameter dan tahapan pencarian lokal. Tahapan pencarian lokal dilakukan setelah melalui tahapan mutasi (5.2.6) dan sebelum tahapan pembuatan populasi baru (5.2.7). Tahapan tersebut akan disertakan dengan potongan kode-kode, fungsi, serta penjelasan secukupnya.

5.4.1. Inisialisasi Parameter

Perbedaan tahap ini dengan tahap 5.2.1 adalah deklarasi variabel parameter yang digunakan. Penentuan parameter yang akan digunakan yaitu deklarasi simpul awal, simpul akhir, jumlah simpul, batasan waktu, jumlah populasi, jumlah individu yang diseleksi, jumlah generasi, probabilitas perkawinan silang, probabilitas mutasi, dan koefisien penurunan suhu. Penentuan parameter dapat dilihat pada Kode 5.39.

```
static int nodeAwal = 1;
static int nodeAkhir = 47;
static int jumlahNode = 48;
static int tMax = 100; //Menetapkan n
static int jumlahPop = 10; //Menetapk
static int jumlahSelect = 10; //Menet
static int jumlahGenerasi = 1000; //M
static double probCrossOver = 0.5;
static double probMutation = 0.4;
static double coolingRate = 0.9;
```

Kode 5.39 Inisialisasi Parameter AMSA

Kode 5.39 menjelaskan bahwa pada studi kasus ini terdapat sebanyak 48 simpul (jumlahSimpul), dan sesuai pada penjelasan bab sebelumnya, simpul awal yang digunakan adalah simpul 1 (simpulAwal) dan simpul akhir yang digunakan adalah simpul 47 (simpulAkhir). Batasan waktu maksimal yang telah ditentukan sebesar 80 menit (tMax). Untuk mencari solusi, maka ditentukan jumlah populasi sebanyak 10 individu (jumlahPop), dengan jumlah pengulangan sebanyak 1000 kali iterasi (jumlahGenerasi). Pada setiap pengulangan, ditentukan jumlah individu yang diseleksi sebanyak 10 individu untuk dilakukan perkawinan silang. Probabilitas kawin silang yang akan digunakan sebesar 0.5 (probCrossOver) dan probabilitas mutasi sebesar 0.5 (probMutation). Koefisien penurunan suhu

pada setiap iterasi untuk *Simulated Annealing* ditentukan sebesar 0.9 (coolingRate) dari temperatur suhu sebelumnya.

5.4.2. Pencarian Lokal *Simulated Annealing*

Setelah tahap 5.4.1 dilakukan, dilanjutkan dengan tahapan sesuai pada AG (5.2.2 hingga 5.2.6). Individu keturunan hasil mutasi akan melalui tahapan perbaikan lokal. Perbaikan lokal dilakukan dengan menerapkan sejumlah pengulangan pada setiap individu keturunan. Pada setiap pengulangan, dilakukan operasi *add-remove* pada individu keturunan, dilanjutkan dengan membandingkan hasil operasi dengan individu keturunan. Apabila individu hasil operasi mempunyai nilai *fitness* lebih baik dari individu keturunan, maka individu tersebut diganti. Namun apabila sebaliknya, individu hasil operasi berkesempatan dijadikan individu baru apabila nilai probabilitas yang diciptakan oleh individu tersebut memenuhi probabilitas dari persamaan Boltzman. Lalu dilakukan pengulangan terhadap individu baru dengan operasi yang sama hingga mencapai batas jumlah pengulangan pencarian lokal yang ada.

```
628     //5. LOCAL SEARCH (SIMULATED ANNEALING)
629     for (int i = 0; i < arrOffSpring.size(); i++) {
630         double temperature = 100;
631         int loopSA = 0;
632         int maxLoopSA = 20;
633 
634             do {
635 
636                 int rndInsert = new Random().nextInt(
637                         arrOffSpring.get(i).size() - 2) + 1;
638                 int rndNilaiInsert
639                         = new Random().nextInt(jumlahNode) + 1;
640                 if (arrOffSpring.get(i).contains(rndNilaiInsert)
641                         || rndNilaiInsert == nodeAwal
642                         || rndNilaiInsert == nodeAkhir) {
643                     continue; //Memastikan node yang dimasukkan buka
644                 }               //merupakan node awal dan akhir
645                 arrOffSpring.get(i).add(rndInsert, rndNilaiInsert);
646 
647                 int rndDelete = new Random().nextInt(
648                         arrOffSpring.get(i).size() - 2) + 1;
649                 //Proses omit dilakukan jika ukuran keturunan lebih
650                 if (arrOffSpring.get(i).size() > 3) {
651                     //Temp menyimpan nilai dari gen/node yang dihapu
652                     int temp = arrOffSpring.get(i).get(rndDelete);
653                     arrOffSpring.get(i).remove(rndDelete);
```

Kode 5.40 Penggalan Kode Pencarian Lokal *Simulated Annealing* (1)

Pada Kode 5.40 dideklarasikan temperatur awal sebesar 100 dan perbaikan lokal dilakukan dengan menambahkan 1 gen yang berisi simpul dengan nomor acak ke dalam kromosom keturunan. Posisi gen tersebut diselipkan secara acak diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir. Lalu menghilangkan 1 gen secara acak yang terdapat pada kromosom individu keturunan. Gen yang dihilangkan tersebut berada diantara indeks setelah simpul awal hingga indeks sebelum simpul akhir.

Setelah itu diterapkan Kode 5.41 yaitu melakukan perbandingan nilai *fitness* antara individu keturunan dengan individu hasil perbaikan lokal. Apabila nilai *fitness* individu hasil perbaikan lokal lebih baik dari individu keturunan dan masih memenuhi batas waktu tempuh maksimal, maka individu keturunan digantikan dengan individu hasil perbaikan lokal. Apabila nilai *fitness* individu hasil perbaikan lokal lebih buruk

dari individu keturunan maka dilakukan pengecekan nilai probabilitas individu. Probabilitas individu (rndProbSimulated) berisi bilangan random antara 0 dan 1 (*double*). Lalu rndProbSimulated dibandingkan dengan nilai probabilitas (probExponen) berdasarkan persamaan Boltzman yaitu $P = e^{-\frac{c}{t}}$, dimana *P* adalah Probabilitas Boltzman, *e* adalah bilangan eksponensial, *c* adalah perbedaan nilai *fitness* kedua individu, dan *t* adalah parameter suhu saat ini. Apabila nilai rndProbSimulated lebih besar dari probExponen, maka individu keturunan digantikan dengan individu hasil perbaikan lokal.

```
654         //Menyimpan waktu tempuh & fitness hasil proses omit
655         int currentWaktuTempuh
656                 = hitungWaktuTempuh(arrOffSpring.get(i));
657         int currentFitness
658                 = hitungFitness(arrOffSpring.get(i));
659         if (currentWaktuTempuh <= arrOffSpring_WaktuTempuh.get(i)
660             && currentFitness >= arrOffSpring_Fitness.get(i)) {
661         arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
662         arrOffSpring_Fitness.set(i, currentFitness);
663         } else if (currentWaktuTempuh <= tMax
664             && currentFitness < arrOffSpring_Fitness.get(i)){
665         double rndProbSimulated = new Random().nextDouble();
666         double probExponen = Math.pow(Math.E, (-(Math.abs(
667         arrOffSpring_Fitness.get(i)-currentFitness)/temperature)));
668         //Solusi neighborhood diterima apabila nilai random yang dig
669         //lebih kecil daripada nilai probability
670             if (rndProbSimulated > probExponen) {
671                 arrOffSpring_WaktuTempuh.set(i, currentWaktuTempuh);
672                 arrOffSpring_Fitness.set(i, currentFitness);
673             }
674             else {
675                 arrOffSpring.get(i).add(rndDelete, temp);
676                 arrOffSpring.get(i).remove(rndInsert);
677             }
678         } else {
679             arrOffSpring.get(i).add(rndDelete, temp);
680             arrOffSpring.get(i).remove(rndInsert);
681         } //Jika tdk lebih baik, maka dikembalikan sprt semula
682 
683     } else {
684         arrOffSpring.get(i).remove(rndInsert);
```

Kode 5.41 Penggalan Kode Pencarian Lokal *Simulated Annealing* (2)

Selanjutnya pada Kode 5.42 dilakukan penambahan variabel pengulangan SA dan pembaruan temperatur dengan mengkalikan temperatur saat ini dengan koefisien penurunan

suhu (coolingRate). Pengulangan terus dilakukan hingga memenuhi batasan maksimal pengulangan yang telah ditetapkan sebelumnya.

```
686             loopSA++;
687             temperature=coolingRate*temperature;
688     } while (loopSA < maxLoopSA);
```

Kode 5.42 Kriteria Pemberhentian Pencarian Lokal *Simulated Annealing*

"Halaman ini sengaja dikosongkan"

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan hasil dan pembahasan yang didapatkan dari tugas akhir ini.

6.1. Lingkungan Uji Coba

Lingkungan uji coba merupakan kriteria perangkat pengujian meliputi perangkat keras dan perangkat lunak yang digunakan dalam menyelesaikan tugas akhir ini. Spesifikasi dari perangkat keras yang digunakan ditunjukkan pada Tabel 6.1.

Tabel 6.1 Spesifikasi Perangkat Keras

PERANGKAT KERAS	SPESIFIKASI
Jenis	Laptop
Processor	Intel Core i5 7200 2.50GHz
RAM	4096MB
Hard Disk Drive	500GB

Selain itu, terdapat perangkat lunak yang digunakan dalam menyelesaikan tugas akhir ini yang ditunjukkan pada Tabel 6.2.

Tabel 6.2 Perangkat Lunak Yang Digunakan

PERANGKAT LUNAK	FUNGSI
Windows 10	Sistem Operasi
Paint	Membuat model jejaring
NetBeans IDE 8.0.2	Membuat dan menjalankan algoritma
Java Development Kit 1.8	Bahasa pemrograman
Microsoft Office Excel 2013	Merekap data
Microsoft Office Word 2013	Membuat laporan

6.2. Hasil Penggambaran Network Model

Dalam model jejaring yang dibuat, simpul merepresentasikan tempat pemberhentian bis yang digambarkan dalam bentuk lingkaran, untuk objek wisata digambarkan dengan lingkaran

berwarna ungu, dan untuk pertemuan antar rute digambarkan dengan lingkaran berwarna abu-abu. Selanjutnya busur/jalur bis yang menghubungkan antar simpul digambarkan dalam bentuk garis berwarna biru. Jalur dua arah digambarkan dengan garis biru tanpa anak panah, sedangkan jalur satu arah digambarkan dengan garis biru dengan anak panah. Secara keseluruhan model jejaring dapat dilihat pada LAMPIRAN E.

6.3. Hasil Pembaruan Waktu Tempuh Menggunakan Algoritma Dijkstra

Algoritma Dijkstra dibuat melalui aplikasi Netbeans 8.0.2, lalu program dijalankan dengan menggunakan data matriks yang telah dibuat pada sub bab 4.3. Program ini akan menghasilkan sebuah matriks baru yang berisikan setiap waktu tempuh antar simpul yang kemudian disimpan ke dalam Excel dalam format .csv. Tabel 6.3 menunjukkan penggalan hasil matriks Algoritma Dijkstra berukuran 10x10. Untuk tabel matriks Algoritma Dijkstra selengkapnya terdapat pada LAMPIRAN D.

Tabel 6.3 Penggalan Hasil Algoritma Dijkstra

simpul	1	2	3	4	5	6	7	8	9	10
1	0	6	9	11	13	26	13	14	16	18
2	11	0	3	5	7	20	7	8	10	12
3	14	3	0	2	4	17	4	5	7	9
4	21	10	7	0	2	15	2	3	5	7
5	22	11	8	1	0	13	3	4	6	8
6	37	26	23	16	15	0	18	19	19	17
7	24	13	10	3	2	15	0	1	3	5
8	25	14	11	4	3	16	1	0	2	4
9	27	16	13	6	5	18	3	2	0	2
10	29	18	15	8	7	16	5	4	2	0

6.4. Hasil Pencarian Solusi dengan Algoritma Genetika

Pada sub bab ini, program dijalankan sesuai dengan ketentuan pada sub bab 5.2 yaitu mencari solusi menggunakan AG dari permasalahan pencarian rute optimum dari simpul 1 (Terminal Purabaya) menuju simpul 47 (Pelabuhan Tanjung Perak) dengan batasan waktu tempuh maksimal adalah 80 menit.

6.4.1. Validasi Solusi Layak Awal

Hal yang dilakukan sebelum mendapatkan solusi optimal yaitu membangkitkan solusi layak awal. Solusi yang dibangkitkan harus memenuhi batasan-batasan yang telah ditentukan pada model OP. Pencarian solusi layak awal akan dilakukan pengulangan hingga mencapai jumlah populasi yang telah ditentukan yaitu 100 individu. Tabel 6.4 merupakan hasil dari pembangkitan populasi awal Algoritma Genetika.

Tabel 6.4 Hasil Pembangkitan Populasi Awal AG

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 31, 33, 43, 47] (77, 28)
2	[1, 32, 47] (51, 21)
3	[1, 24, 20, 47] (79, 116)
4	[1, 6, 46, 47] (75, 27)
5	[1, 9, 23, 47] (54, 54)
6	[1, 37, 47] (59, 35)
7	[1, 12, 15, 47] (61, 94)
8	[1, 27, 44, 47] (65, 0)
9	[1, 7, 47] (51, 29)
10	[1, 36, 18, 47] (69, 61)
11	[1, 40, 47] (51, 0)
12	[1, 2, 8, 11, 19, 20, 39, 44, 25, 47] (78, 260)
13	[1, 18, 47] (66, 51)
14	[1, 12, 47] (58, 61)
15	[1, 20, 4, 47] (72, 111)

Individu	[Solusi] (Waktu Tempuh, Skor)
16	[1, 46, 47] (52, 0)
17	[1, 46, 41, 27, 47] (62, 50)
18	[1, 3, 23, 47] (54, 35)
19	[1, 9, 20, 47] (51, 121)
20	[1, 4, 36, 39, 47] (51, 54)
21	[1, 30, 47] (53, 10)
22	[1, 10, 47] (51, 0)
23	[1, 26, 24, 42, 44, 47] (72, 132)
24	[1, 43, 47] (56, 28)
25	[1, 36, 47] (51, 10)
26	[1, 22, 17, 36, 31, 47] (75, 155)
27	[1, 44, 18, 47] (67, 51)
28	[1, 3, 47] (51, 35)
29	[1, 37, 27, 29, 47] (69, 82)
30	[1, 45, 14, 47] (70, 15)
31	[1, 35, 28, 47] (62, 51)
32	[1, 42, 40, 47] (56, 0)
33	[1, 21, 47] (54, 38)
34	[1, 3, 8, 4, 41, 30, 37, 31, 36, 47] (72, 208)
35	[1, 8, 47] (51, 24)
36	[1, 21, 42, 47] (56, 38)
37	[1, 3, 20, 47] (51, 102)
38	[1, 9, 28, 47] (52, 72)
39	[1, 29, 34, 47] (54, 47)
40	[1, 5, 47] (54, 0)
41	[1, 30, 46, 24, 41, 47] (71, 109)
42	[1, 31, 25, 26, 47] (59, 125)
43	[1, 14, 42, 23, 47] (65, 15)
44	[1, 20, 46, 30, 47] (54, 77)
45	[1, 19, 47] (51, 52)

Individu	[Solusi] (Waktu Tempuh, Skor)
46	[1, 13, 45, 30, 47] (71, 97)
47	[1, 40, 34, 47] (68, 0)
48	[1, 34, 38, 47] (54, 0)
49	[1, 17, 47] (64, 93)
50	[1, 46, 34, 23, 47] (72, 0)
51	[1, 9, 13, 47] (58, 141)
52	[1, 21, 37, 47] (62, 73)
53	[1, 30, 33, 18, 47] (79, 61)
54	[1, 13, 41, 47] (58, 137)
55	[1, 3, 38, 37, 23, 47] (73, 70)
56	[1, 4, 47] (51, 44)
57	[1, 29, 23, 40, 47] (64, 47)
58	[1, 13, 36, 39, 32, 47] (73, 118)
59	[1, 44, 34, 47] (66, 0)
60	[1, 46, 21, 34, 47] (80, 38)
61	[1, 27, 45, 33, 23, 25, 28, 47] (74, 60)
62	[1, 19, 33, 47] (51, 52)
63	[1, 44, 16, 24, 32, 47] (76, 136)
64	[1, 19, 6, 47] (80, 79)
65	[1, 11, 23, 47] (58, 39)
66	[1, 8, 40, 31, 47] (69, 24)
67	[1, 27, 35, 47] (54, 33)
68	[1, 14, 47] (60, 15)
69	[1, 23, 42, 18, 47] (69, 51)
70	[1, 35, 47] (53, 33)
71	[1, 26, 47] (52, 83)
72	[1, 14, 24, 47] (65, 64)
73	[1, 5, 26, 17, 47] (68, 176)
74	[1, 24, 34, 47] (63, 49)

Individu	[Solusi] (Waktu Tempuh, Skor)
75	[1, 35, 33, 47] (67, 33)
76	[1, 19, 44, 46, 39, 47] (67, 52)
77	[1, 3, 35, 47] (53, 68)
78	[1, 43, 14, 47] (63, 43)
79	[1, 11, 9, 15, 17, 41, 38, 47] (75, 269)
80	[1, 10, 35, 39, 46, 32, 47] (78, 54)
81	[1, 23, 47] (54, 0)
82	[1, 2, 42, 13, 20, 47] (79, 190)
83	[1, 11, 47] (55, 39)
84	[1, 17, 43, 28, 47] (72, 139)
85	[1, 4, 25, 47] (54, 86)
86	[1, 34, 17, 47] (69, 93)
87	[1, 21, 25, 47] (56, 80)
88	[1, 15, 47] (61, 33)
89	[1, 16, 14, 47] (68, 81)
90	[1, 10, 6, 31, 47] (78, 27)
91	[1, 38, 47] (51, 0)
92	[1, 17, 23, 47] (72, 93)
93	[1, 16, 44, 47] (61, 66)
94	[1, 33, 26, 38, 47] (53, 83)
95	[1, 21, 7, 47] (71, 67)
96	[1, 12, 13, 46, 24, 41, 31, 47] (72, 247)
97	[1, 46, 15, 47] (71, 33)
98	[1, 15, 44, 37, 17, 47] (79, 161)
99	[1, 44, 46, 25, 47] (69, 42)
100	[1, 17, 23, 46, 47] (72, 93)

Solusi yang dihasilkan pada Tabel 6.4 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

a) Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
b) Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
c) Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
d) Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Dapat disimpulkan bahwa solusi awal yang dibangkitkan untuk dijadikan populasi awal merupakan solusi layak dan valid.

6.4.2. Validasi Solusi Akhir

Populasi awal yang terbentuk selanjutnya mengalami proses seleksi, perkawinan silang, dan mutasi yang dilakukan secara berurutan dan mengalami pengulangan proses tersebut hingga memenuhi kriteria pemberhentian yaitu 1000 kali pengulangan. Pada setiap pengulangan akan dihasilkan solusi akhir sebanyak jumlah populasi yaitu 100 individu. Tabel 6.5 merupakan solusi akhir pada iterasi ke-1000 yang berisikan 100 individu dengan menggunakan Algoritma Genetika.

Tabel 6.5 Hasil Populasi Akhir AG

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 9, 20, 12, 13, 16, 17, 18, 26, 27, 41, 47] (78, 612)
2	[1, 9, 20, 12, 13, 16, 17, 18, 26, 41, 47] (78, 612)
3	[1, 20, 19, 12, 13, 16, 18, 17, 26, 41, 47] (78, 610)
4	[1, 20, 19, 12, 13, 16, 17, 18, 26, 41, 47] (78, 610)
5	[1, 19, 20, 12, 13, 16, 17, 18, 26, 41, 47] (78, 610)
6	[1, 20, 19, 12, 13, 16, 17, 18, 26, 27, 41, 47] (78, 610)
7	[1, 19, 20, 12, 13, 16, 17, 18, 26, 27, 41, 47] (78, 610)
8	[1, 19, 20, 12, 13, 16, 18, 17, 26, 41, 47] (78, 610)

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
9	[1, 9, 20, 19, 13, 16, 17, 18, 26, 41, 47] (78, 603)
10	[1, 9, 20, 19, 13, 16, 17, 18, 26, 27, 41, 47] (78, 603)
11	[1, 9, 20, 12, 13, 16, 17, 25, 26, 41, 47] (76, 603)
12	[1, 19, 20, 12, 13, 16, 17, 25, 26, 41, 47] (76, 601)
13	[1, 20, 19, 12, 13, 16, 17, 25, 26, 41, 47] (76, 601)
14	[1, 20, 12, 13, 16, 17, 18, 25, 26, 27, 41, 47] (78, 600)
15	[1, 20, 12, 13, 16, 17, 18, 25, 26, 41, 47] (78, 600)
16	[1, 9, 19, 12, 13, 16, 17, 18, 26, 27, 41, 47] (74, 597)
17	[1, 9, 19, 12, 13, 16, 17, 18, 26, 41, 47] (74, 597)
18	[1, 9, 19, 12, 13, 16, 18, 17, 26, 41, 47] (74, 597)
19	[1, 2, 20, 12, 13, 16, 17, 18, 26, 41, 47] (78, 594)
20	[1, 9, 20, 13, 16, 17, 18, 25, 26, 27, 41, 47] (77, 593)
21	[1, 9, 20, 13, 16, 17, 18, 25, 26, 41, 47] (77, 593)
22	[1, 20, 12, 13, 15, 16, 17, 18, 26, 27, 41, 47] (78, 591)
23	[1, 19, 20, 13, 16, 17, 18, 25, 26, 41, 47] (77, 591)
24	[1, 20, 12, 13, 15, 16, 17, 18, 26, 41, 47] (78, 591)
25	[1, 20, 19, 13, 16, 17, 18, 25, 26, 41, 47] (78, 591)
26	[1, 19, 20, 13, 16, 17, 18, 25, 26, 27, 41, 47] (77, 591)
27	[1, 9, 20, 12, 13, 17, 18, 25, 26, 41, 47] (78, 588)
28	[1, 9, 20, 12, 13, 25, 17, 18, 26, 41, 47] (80, 588)
29	[1, 9, 20, 12, 13, 17, 18, 25, 26, 27, 41, 47] (78, 588)
30	[1, 7, 20, 12, 13, 16, 17, 18, 26, 41, 47] (78, 587)
31	[1, 9, 12, 13, 16, 25, 17, 18, 26, 41, 47] (80, 587)
32	[1, 9, 12, 13, 16, 25, 17, 18, 26, 27, 41, 47] (80, 587)
33	[1, 9, 12, 13, 16, 17, 18, 25, 26, 41, 47] (72, 587)
34	[1, 20, 19, 12, 13, 25, 17, 18, 26, 27, 41, 47] (80, 586)
35	[1, 19, 20, 12, 13, 25, 17, 18, 26, 27, 41, 47] (80, 586)
36	[1, 19, 20, 12, 13, 17, 18, 25, 26, 41, 47] (78, 586)
37	[1, 20, 19, 12, 13, 25, 17, 18, 26, 41, 47] (80, 586)

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
38	[1, 19, 20, 12, 13, 25, 17, 18, 26, 41, 47] (80, 586)
39	[1, 19, 20, 12, 13, 17, 18, 25, 26, 27, 41, 47] (78, 586)
40	[1, 20, 19, 12, 13, 17, 18, 25, 26, 41, 47] (78, 586)
41	[1, 9, 20, 13, 15, 16, 17, 18, 26, 41, 47] (77, 584)
42	[1, 20, 12, 13, 15, 16, 17, 25, 26, 41, 47] (76, 582)
43	[1, 20, 19, 13, 15, 16, 17, 18, 26, 27, 41, 47] (78, 582)
44	[1, 20, 19, 13, 15, 16, 17, 18, 26, 41, 47] (78, 582)
45	[1, 19, 20, 13, 15, 16, 17, 18, 26, 41, 47] (77, 582)
46	[1, 9, 20, 19, 13, 25, 17, 18, 26, 41, 47] (80, 579)
47	[1, 9, 20, 19, 13, 25, 17, 18, 26, 27, 41, 47] (80, 579)
48	[1, 9, 20, 19, 13, 17, 18, 25, 26, 41, 47] (78, 579)
49	[1, 9, 19, 13, 16, 17, 18, 25, 26, 41, 47] (74, 578)
50	[1, 9, 12, 13, 15, 16, 17, 18, 26, 41, 47] (72, 578)
51	[1, 9, 12, 13, 15, 16, 17, 18, 26, 27, 41, 47] (72, 578)
52	[1, 9, 20, 19, 12, 16, 17, 18, 26, 27, 41, 47] (78, 577)
53	[1, 9, 20, 19, 12, 16, 17, 18, 26, 41, 47] (78, 577)
54	[1, 9, 19, 12, 13, 17, 18, 25, 26, 41, 47] (74, 573)
55	[1, 20, 19, 13, 15, 16, 17, 25, 26, 41, 47] (76, 573)
56	[1, 7, 19, 12, 13, 16, 17, 18, 26, 27, 41, 47] (74, 572)
57	[1, 2, 20, 12, 13, 17, 18, 25, 26, 41, 47] (78, 570)
58	[1, 2, 20, 12, 13, 25, 17, 18, 26, 27, 41, 47] (80, 570)
59	[1, 2, 20, 12, 13, 25, 17, 18, 26, 41, 47] (80, 570)
60	[1, 2, 20, 12, 13, 17, 18, 25, 26, 27, 41, 47] (78, 570)
61	[1, 9, 19, 13, 15, 16, 18, 44, 17, 26, 41, 47] (80, 569)
62	[1, 9, 19, 13, 15, 16, 17, 18, 26, 41, 47] (74, 569)
63	[1, 9, 12, 13, 16, 17, 35, 25, 26, 41, 47] (76, 569)
64	[1, 9, 20, 19, 12, 16, 17, 25, 26, 41, 47] (76, 568)
65	[1, 19, 20, 13, 16, 17, 18, 26, 27, 28, 41, 47] (77, 567)
66	[1, 2, 20, 13, 15, 16, 17, 18, 26, 41, 47] (77, 566)

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
67	[1, 19, 20, 12, 16, 17, 18, 25, 26, 41, 47] (78, 565)
68	[1, 9, 19, 12, 13, 16, 17, 28, 26, 41, 47] (80, 564)
69	[1, 9, 20, 12, 13, 16, 17, 18, 26, 38, 47] (78, 562)
70	[1, 9, 20, 12, 13, 16, 17, 18, 26, 47] (78, 562)
71	[1, 9, 20, 12, 13, 16, 18, 25, 26, 27, 41, 47] (78, 561)
72	[1, 2, 9, 20, 12, 16, 17, 18, 26, 41, 47] (78, 561)
73	[1, 9, 20, 12, 13, 16, 17, 26, 27, 41, 47] (76, 561)
74	[1, 9, 20, 12, 13, 16, 17, 26, 41, 47] (76, 561)
75	[1, 9, 20, 12, 13, 16, 17, 26, 41, 38, 47] (76, 561)
76	[1, 2, 9, 20, 12, 16, 17, 18, 26, 27, 41, 47] (78, 561)
77	[1, 20, 9, 12, 13, 16, 17, 26, 41, 47] (80, 561)
78	[1, 20, 9, 12, 13, 16, 17, 26, 41, 38, 47] (80, 561)
79	[1, 20, 9, 12, 13, 16, 17, 26, 27, 41, 47] (80, 561)
80	[1, 19, 20, 12, 13, 16, 17, 18, 26, 27, 47] (78, 560)
81	[1, 19, 20, 12, 13, 16, 18, 17, 26, 27, 47] (78, 560)
82	[1, 19, 20, 12, 13, 16, 17, 18, 26, 47] (78, 560)
83	[1, 20, 19, 12, 13, 16, 17, 18, 26, 38, 47] (78, 560)
84	[1, 19, 20, 12, 13, 16, 17, 18, 26, 38, 47] (78, 560)
85	[1, 20, 19, 12, 13, 16, 17, 18, 26, 47] (78, 560)
86	[1, 9, 19, 13, 15, 16, 17, 25, 26, 41, 47] (72, 560)
87	[1, 20, 19, 12, 13, 16, 18, 17, 26, 38, 47] (78, 560)
88	[1, 19, 20, 12, 13, 16, 18, 25, 26, 27, 41, 47] (78, 559)
89	[1, 20, 19, 12, 13, 16, 18, 25, 26, 41, 47] (78, 559)
90	[1, 20, 19, 12, 13, 16, 17, 26, 41, 38, 47] (76, 559)
91	[1, 19, 20, 12, 13, 16, 17, 26, 41, 38, 47] (76, 559)
92	[1, 19, 20, 12, 13, 16, 17, 26, 41, 47] (76, 559)
93	[1, 20, 19, 12, 13, 16, 17, 26, 41, 47] (76, 559)
94	[1, 20, 19, 12, 13, 16, 17, 26, 27, 41, 47] (76, 559)
95	[1, 19, 20, 12, 13, 16, 17, 26, 27, 41, 47] (76, 559)

Individu	[Solusi] (Waktu Tempuh, Skor)
96	[1, 20, 19, 12, 13, 16, 18, 25, 26, 27, 41, 47] (78, 559)
97	[1, 19, 20, 12, 13, 16, 18, 25, 26, 41, 47] (78, 559)
98	[1, 20, 12, 13, 16, 44, 17, 18, 26, 41, 47] (78, 558)
99	[1, 9, 20, 12, 15, 16, 17, 18, 26, 41, 47] (78, 558)
100	[1, 20, 12, 13, 16, 17, 18, 26, 41, 38, 47] (78, 558)

Solusi yang dihasilkan pada Tabel 6.5 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
3. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Seluruh solusi akhir pada Tabel 6.5 masih layak dan valid karena memenuhi batasan OP. Dari seluruh solusi akhir tersebut diambil solusi terbaik dengan melihat jumlah skor yang paling tinggi. Didapatkan solusi dengan nilai *fitness* tertinggi adalah solusi pada baris pertama dengan nilai *fitness* sebesar 612 dan waktu tempuh sebesar 78 menit. Solusi tersebut dianggap sebagai rekomendasi rute yang paling optimum dengan menggunakan Algoritma Genetika. Tabel lintasan solusi tersebut terdapat pada Tabel 6.6.

Tabel 6.6 Solusi Optimal AG

No	Titik	Lokasi
1	1	Terminal Bungurasih
2	9	Royal Plaza
3	20	Kebun Binatang Surabaya
		Patung Suro dan Boyo
4	12	Marvel City
5	13	Monumen Kapal Selam
		Plaza Surabaya
6	16	Hi-Tech Mall
7	17	Pasar Atom Surabaya
		ITC Surabaya Mega Grosir
8	18	Wisata Religi Ampel
9	26	BG Junction
		Pasar Blauran Baru
10	27	Pertigaan Bubutan Raden Saleh
11	41	Pertigaan Bubutan Kebun Rojo
12	47	Pelabuhan Tanjung Perak

6.4.3. Validasi Solusi Paling Optimal

Setelah mendapatkan solusi optimal pada sub bab 6.4.2, tahap selanjutnya yaitu memvalidasi solusi tersebut dengan solusi hasil menjalankan algoritma yang sama berkali-kali. Tujuan dari tahap ini adalah mengecek apakah solusi yang didapatkan pada tahap sebelumnya merupakan solusi paling optimal. Validasi dilakukan dengan cara membandingkan hasil solusi pada sub bab 6.4.2 (hasil *running* program ke-1) dengan hasil solusi *running* program dari ke-2 hingga ke-10. Tabel 6.7 merupakan perbandingan solusi running program ke-1 hingga ke-10.

Tabel 6.7 Perbandingan Solusi Optimal AG

Running Program ke	Solusi Optimal
1	[1, 9, 20, 12, 13, 16, 17, 18, 26, 27, 41, 47] | Waktu Tempuh: 78 | Fitness: 612
2	[1, 9, 19, 20, 29, 17, 16, 15, 13, 43, 25, 26, 47] | Waktu Tempuh: 79 | Fitness: 745
3	[1, 2, 19, 20, 9, 12, 13, 15, 16, 17, 26, 47] | Waktu Tempuh: 80 | Fitness: 725
4	[1, 4, 9, 20, 12, 13, 16, 17, 44, 26, 29, 28, 47] | Waktu Tempuh: 78 | Fitness: 713
5	[1, 4, 9, 20, 19, 12, 13, 16, 26, 25, 38, 47] | Waktu Tempuh: 78 | Fitness: 649
6	[1, 7, 9, 20, 13, 16, 17, 18, 25, 26, 29, 35, 36, 47] | Waktu Tempuh: 80 | Fitness: 755
7	[1, 9, 20, 22, 13, 16, 18, 17, 25, 26, 29, 32, 47] | Waktu Tempuh: 78 | Fitness: 756
8	[1, 9, 19, 12, 13, 26, 17, 18, 25, 29, 32, 36, 47] | Waktu Tempuh: 79 | Fitness: 694
9	[1, 4, 20, 19, 12, 13, 15, 16, 17, 26, 32, 40, 47] | Waktu Tempuh: 77 | Fitness: 700
10	[1, 9, 19, 20, 12, 13, 16, 17, 18, 25, 26, 47] | Waktu Tempuh: 78 | Fitness: 749

Tabel 6.7 menunjukkan bahwa dari hasil *running* program sebanyak 10 kali, solusi optimal yang didapatkan berbeda-beda. Meskipun solusi optimal yang terdapat pada sub bab 6.4.2 dinyatakan valid terhadap batasan OP, solusi optimal tersebut bukan merupakan solusi paling optimal AG. Hal tersebut terbukti dengan adanya solusi yang lebih optimal dari hasil running program yaitu solusi hasil running program ke-7 dengan nilai fitness sebesar 756. Hasil tersebut nantinya akan dijadikan perbandingan pada sub bab 6.10 Uji Coba 4. Hasil tersebut belum tentu merupakan solusi paling optimal dari AG dikarenakan ruang pencarian dari AG masih sangat luas dan kemungkinan masih banyak solusi yang lebih baik.

6.5. Hasil Pencarian Solusi dengan Algoritma Memetika *Hill Climbing*

Pada sub bab ini, program dijalankan sesuai dengan ketentuan pada sub bab 5.3 yaitu mencari solusi menggunakan AMHC dari permasalahan pencarian rute optimum dari simpul 1 (Terminal Purabaya) menuju simpul 47 (Pelabuhan Tanjung Perak) dengan batasan waktu tempuh maksimal adalah 80 menit.

6.5.1. Validasi Solusi Layak Awal

Hal yang dilakukan sebelum mendapatkan solusi optimal yaitu membangkitkan solusi layak awal. Solusi yang dibangkitkan harus memenuhi batasan-batasan yang telah ditentukan pada model OP. Pencarian solusi layak awal akan dilakukan pengulangan hingga mencapai jumlah populasi yang telah ditentukan yaitu 10 individu. Tabel 6.8 merupakan hasil dari pembangkitan populasi awal AMHC.

Tabel 6.8 Hasil Pembangkitan Populasi Awal AMHC

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 9, 3, 47] (71, 89)
2	[1, 42, 17, 47] (65, 93)
3	[1, 16, 47] (61, 66)
4	[1, 21, 47] (54, 38)
5	[1, 45, 13, 46, 31, 47] (75, 87)
6	[1, 7, 4, 41, 47] (56, 123)
7	[1, 43, 35, 47] (58, 61)
8	[1, 42, 36, 23, 47] (78, 10)
9	[1, 8, 47] (51, 24)
10	[1, 22, 44, 42, 47] (71, 52)

Solusi yang dihasilkan pada Tabel 6.8 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

3. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
4. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
5. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
6. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Dapat disimpulkan bahwa solusi awal yang dibangkitkan untuk dijadikan populasi awal merupakan solusi layak dan valid.

6.5.2. Validasi Solusi Akhir

Populasi awal yang terbentuk selanjutnya mengalami proses seleksi, perkawinan silang, mutasi, dan perbaikan lokal yang dilakukan secara berurutan dan mengalami pengulangan proses tersebut hingga memenuhi kriteria pemberhentian yaitu 1000 kali pengulangan. Pada setiap pengulangan akan dihasilkan solusi akhir sebanyak jumlah populasi yaitu 10 individu. Tabel 6.9 merupakan solusi akhir pada iterasi ke-1000 yang berisikan 10 individu dengan menggunakan AMHC.

Tabel 6.9 Hasil Populasi Akhir AMHC

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 2, 3, 4, 7, 8, 9, 20, 19, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] (78, 911)
2	[1, 2, 3, 4, 8, 7, 9, 19, 20, 11, 12, 13, 14, 15, 16, 44, 17, 18, 25, 26, 47] (80, 911)
3	[1, 2, 3, 4, 7, 8, 9, 19, 10, 20, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] (80, 911)
4	[1, 2, 3, 4, 7, 8, 9, 19, 20, 11, 12, 13, 14, 15, 16, 44, 17, 18, 25, 26, 47] (78, 911)
5	[1, 2, 3, 4, 7, 8, 9, 19, 20, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] (78, 911)

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
6	[1, 2, 3, 4, 7, 8, 9, 20, 19, 10, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] (78, 911)
7	[1, 2, 3, 4, 7, 8, 9, 20, 19, 11, 12, 13, 14, 15, 16, 18, 17, 25, 26, 47] (78, 911)
8	[1, 2, 3, 4, 7, 8, 9, 20, 19, 11, 12, 13, 14, 15, 16, 44, 17, 18, 25, 26, 47] (78, 911)
9	[1, 2, 3, 4, 8, 7, 9, 20, 19, 11, 12, 13, 14, 15, 16, 18, 17, 25, 26, 47] (80, 911)
10	[1, 2, 3, 4, 8, 7, 9, 19, 20, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] (80, 911)

Solusi yang dihasilkan pada Tabel 6.9 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
3. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Seluruh solusi akhir pada Tabel 6.9 masih layak dan valid karena memenuhi batasan OP. Dari seluruh solusi akhir tersebut diambil solusi terbaik dengan melihat jumlah skor yang paling tinggi. Didapatkan solusi dengan nilai *fitness* tertinggi adalah solusi pada baris pertama dengan nilai *fitness* sebesar 911 dan waktu tempuh sebesar 78 menit. Solusi tersebut dianggap sebagai rekomendasi rute yang paling optimum dengan menggunakan AMHC. Tabel lintasan solusi tersebut terdapat pada Tabel 6.10.

Tabel 6.10 Solusi Optimal AMHC

No	Titik	Lokasi
1	1	Terminal Bungurasih
2	2	Citi of Tomorrow Mall
3	3	Masjid Al-Akbar
4	4	Taman Pelangi
5	7	DBL Arena
6	8	Maspion Square
7	9	Royal Plaza
8	20	Kebun Binatang Surabaya
		Patung Suro dan Boyo
9	19	DTC Wonokromo
10	11	Stasiun Wonokromo
11	12	Marvel City
12	13	Monumen Kapal Selam
		Plaza Surabaya
13	14	Stasiun Gubeng
14	15	Grand City Mall
15	16	Hi-Tech Mall
16	17	Pasar Atom Surabaya
		ITC Surabaya Mega Grosir
17	18	Wisata Religi Ampel
18	25	Museum Surabaya (Gedung Siola)
19	26	BG Junction
		Pasar Blauran Baru
20	47	Pelabuhan Tanjung Perak

6.5.3. Validasi Solusi Paling Optimal

Setelah mendapatkan solusi optimal pada sub bab 6.5.2, tahap selanjutnya yaitu memvalidasi solusi tersebut dengan solusi hasil menjalankan algoritma yang sama berkali-kali. Tujuan dari tahap ini adalah mengecek apakah solusi yang didapatkan

pada tahap sebelumnya merupakan solusi paling optimal. Validasi dilakukan dengan cara membandingkan hasil solusi pada sub bab 6.5.2 (hasil running program ke-1) dengan hasil solusi running program dari ke-2 hingga ke-10. Tabel 6.11 merupakan perbandingan solusi running program ke-1 hingga ke-10.

Tabel 6.11 Perbandingan Solusi Optimal AMHC

Running Program ke	Solusi Optimal
1	[1, 2, 3, 4, 7, 8, 9, 20, 19, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 47] | Waktu Tempuh: 78 | Fitness: 911
2	[1, 2, 3, 4, 7, 8, 9, 19, 20, 11, 12, 13, 14, 15, 16, 44, 17, 18, 25, 26, 47] | Waktu Tempuh: 78 | Fitness: 1004
3	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 17, 44, 16, 15, 14, 13, 43, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 971
4	[1, 2, 3, 4, 7, 8, 9, 19, 10, 20, 21, 22, 43, 14, 13, 15, 16, 17, 18, 25, 26, 47] | Waktu Tempuh: 79 | Fitness: 1022
5	[1, 2, 3, 4, 5, 7, 8, 9, 19, 20, 21, 22, 43, 14, 13, 15, 16, 18, 17, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 1022
6	[1, 2, 3, 4, 7, 8, 9, 10, 20, 19, 11, 12, 13, 14, 15, 16, 17, 18, 44, 25, 26, 47] | Waktu Tempuh: 78 | Fitness: 1004
7	[1, 2, 3, 4, 8, 7, 9, 20, 19, 10, 11, 12, 13, 14, 15, 16, 18, 17, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 1004
8	[1, 2, 3, 4, 8, 7, 9, 19, 20, 11, 12, 13, 14, 15, 16, 44, 17, 18, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 1004
9	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 11, 12, 13, 43, 25, 26, 29, 32, 31, 35, 18, 17, 47] | Waktu Tempuh: 80 | Fitness: 1019
10	[1, 2, 3, 4, 7, 8, 9, 10, 20, 19, 11, 12, 13, 43, 25, 26, 29, 32, 35, 18, 17, 47] | Waktu Tempuh: 80 | Fitness: 1019

Tabel 6.11 menunjukkan bahwa dari hasil running program sebanyak 10 kali, solusi optimal yang didapatkan berbeda-beda. Meskipun solusi optimal yang terdapat pada sub bab 6.4.2 dinyatakan valid terhadap batasan OP, solusi optimal tersebut bukan merupakan solusi paling optimal AMHC. Hal tersebut terbukti dengan adanya solusi yang lebih optimal dari hasil running program yaitu solusi hasil running program ke-4 dan ke-5 dengan nilai fitness sebesar 1022. Hasil tersebut nantinya akan dijadikan perbandingan pada sub bab 6.10 Uji Coba 4. Hasil tersebut belum tentu merupakan solusi paling optimal dari AMHC dikarenakan adanya kemungkinan ditemukan solusi yang lebih optimal apabila dilakukan *running* program lebih banyak lagi.

6.6. Hasil Pencarian Solusi dengan Algoritma Memetika *Simulated Annealing*

Pada sub bab ini, program dijalankan sesuai dengan ketentuan pada sub bab 5.4 yaitu mencari solusi menggunakan AMSA dari permasalahan pencarian rute optimum dari simpul 1 (Terminal Purabaya) menuju simpul 47 (Pelabuhan Tanjung Perak) dengan batasan waktu tempuh maksimal adalah 80 menit.

6.6.1. Validasi Solusi Layak Awal

Hal yang dilakukan sebelum mendapatkan solusi optimal yaitu membangkitkan solusi layak awal. Solusi yang dibangkitkan harus memenuhi batasan-batasan yang telah ditentukan pada model OP. Pencarian solusi layak awal akan dilakukan pengulangan hingga mencapai jumlah populasi yang telah ditentukan yaitu 10 individu. Tabel 6.12 merupakan hasil dari pembangkitan populasi awal AMSA.

Tabel 6.12 Hasil Pembangkitan Populasi Awal AMSA

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 7, 40, 27, 47] (76, 29)
2	[1, 19, 47] (51, 52)

Individu	[Solusi] (Waktu Tempuh, Skor)
3	[1, 26, 30, 47] (54, 93)
4	[1, 19, 41, 47] (51, 102)
5	[1, 10, 8, 47] (59, 24)
6	[1, 45, 22, 29, 25, 47] (70, 141)
7	[1, 29, 47] (51, 47)
8	[1, 24, 13, 39, 47] (70, 136)
9	[1, 12, 29, 47] (59, 108)
10	[1, 9, 47] (51, 54)

Solusi yang dihasilkan pada Tabel 6.12 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
3. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Dapat disimpulkan bahwa solusi awal yang dibangkitkan untuk dijadikan populasi awal merupakan solusi layak dan valid.

6.6.2. Validasi Solusi Akhir

Populasi awal yang terbentuk selanjutnya mengalami proses seleksi, perkawinan silang, mutasi, dan perbaikan lokal yang dilakukan secara berurutan dan mengalami pengulangan proses tersebut hingga memenuhi kriteria pemberhentian yaitu 1000 kali pengulangan. Pada setiap pengulangan akan dihasilkan

solusi akhir sebanyak jumlah populasi yaitu 10 individu. Tabel 6.13 merupakan solusi akhir pada iterasi ke-1000 yang berisikan 10 individu dengan menggunakan AMSA.

Tabel 6.13 Hasil Populasi Akhir AMSA

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 44, 25, 26, 29, 32, 35, 41, 47] (80, 1080)
2	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 25, 26, 27, 29, 32, 35, 41, 47] (80, 1080)
3	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 44, 25, 26, 29, 32, 35, 41, 47] (80, 1080)
4	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 25, 26, 29, 32, 31, 35, 41, 47] (80, 1080)
5	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 25, 26, 29, 32, 35, 41, 47] (80, 1080)
6	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 25, 26, 27, 29, 32, 35, 41, 47] (80, 1080)
7	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 14, 15, 16, 44, 17, 18, 25, 26, 29, 32, 35, 41, 47] (80, 1080)
8	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 25, 26, 29, 32, 35, 41, 47] (80, 1080)
9	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 43, 13, 15, 16, 17, 18, 25, 26, 27, 29, 32, 35, 41, 47] (80, 1065)
10	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 13, 15, 16, 17, 18, 25, 26, 27, 29, 32, 35, 41, 47] (80, 1065)

Solusi yang dihasilkan pada Tabel 6.13 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 1 dan berakhir simpul ke 47.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.

3. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Seluruh solusi akhir pada Tabel 6.13 masih layak dan valid karena memenuhi batasan OP. Dari seluruh solusi akhir tersebut diambil solusi terbaik dengan melihat jumlah skor yang paling tinggi. Didapatkan solusi dengan nilai *fitness* tertinggi adalah solusi pada baris pertama dengan nilai *fitness* sebesar 1080 dan waktu tempuh sebesar 80 menit. Solusi tersebut dianggap sebagai rekomendasi rute yang paling optimum dengan menggunakan AMSA. Tabel lintasan solusi tersebut terdapat pada Tabel 6.14.

Tabel 6.14 Solusi Optimal AMSA

No	Titik	Lokasi
1	1	Terminal Bungurasih
2	2	Citi of Tomorrow Mall
3	3	Masjid Al-Akbar
4	4	Taman Pelangi
5	7	DBL Arena
6	8	Maspion Square
7	9	Royal Plaza
8	10	Pertigaan Wonokromo
9	19	DTC Wonokromo
10	20	Kebun Binatang Surabaya
		Patung Suro dan Boyo
11	21	Taman Bungkul
12	22	Tunjungan Plaza
13	43	Balai Kota Surabaya
14	13	Monumen Kapal Selam
		Plaza Surabaya
15	14	Stasiun Gubeng

No	Titik	Lokasi
16	15	Grand City Mall
17	16	Hi-Tech Mall
18	17	Pasar Atom Surabaya
		ITC Surabaya Mega Grosir
19	18	Wisata Religi Ampel
20	44	Perempatan Kalianyar Pengampon
21	25	Museum Surabaya (Gedung Siola)
22	26	BG Junction
		Pasar Blauran Baru
23	29	Stasiun Pasar Turi
24	32	Pasar Turi Baru
25	35	Stasiun Surabaya Kota (Semut)
26	41	Pertigaan Bubutan Kebun Rojo
27	47	Pelabuhan Tanjung Perak

6.6.3. Validasi Solusi Paling Optimal

Setelah mendapatkan solusi optimal pada sub bab 6.6.2, tahap selanjutnya yaitu memvalidasi solusi tersebut dengan solusi hasil menjalankan algoritma yang sama berkali-kali. Tujuan dari tahap ini adalah mengecek apakah solusi yang didapatkan pada tahap sebelumnya merupakan solusi paling optimal. Validasi dilakukan dengan cara membandingkan hasil solusi pada sub bab 6.6.2 (hasil running program ke-1) dengan hasil solusi running program dari ke-2 hingga ke-10. Tabel 6.15 merupakan perbandingan solusi running program ke-1 hingga ke-10.

Tabel 6.15 Perbandingan Solusi Optimal AMSA

Running Program ke	Solusi Optimal
1	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 44, 25, 26, 29, 32, 35, 41, 47] | Waktu Tempuh: 80 | Fitness: 1080

Running Program ke	Solusi Optimal
2	[1, 2, 3, 4, 7, 8, 9, 10, 11, 19, 20, 21, 22, 43, 13, 14, 15, 16, 17, 18, 47] | Waktu Tempuh: 78 | Fitness: 936
3	[1, 2, 3, 4, 8, 7, 9, 19, 20, 21, 22, 42, 43, 13, 14, 15, 16, 17, 18, 25, 26, 47] | Waktu Tempuh: 79 | Fitness: 1022
4	[1, 2, 3, 4, 8, 7, 9, 11, 10, 19, 20, 21, 22, 14, 15, 16, 13, 43, 25, 26, 29, 47] | Waktu Tempuh: 78 | Fitness: 964
5	[1, 2, 3, 4, 7, 8, 9, 19, 21, 20, 11, 12, 14, 13, 43, 25, 24, 23, 22, 26, 29, 32, 47] | Waktu Tempuh: 78 | Fitness: 996
6	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 43, 26, 29, 32, 31, 35, 18, 17, 25, 24, 47] | Waktu Tempuh: 80 | Fitness: 971
7	[1, 2, 3, 4, 7, 8, 9, 20, 21, 19, 11, 12, 14, 13, 15, 16, 43, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 926
8	[1, 2, 3, 4, 7, 8, 9, 19, 20, 11, 12, 13, 14, 15, 16, 17, 18, 25, 26, 29, 47] | Waktu Tempuh: 79 | Fitness: 1051
9	[1, 2, 3, 4, 7, 8, 9, 10, 19, 20, 21, 22, 17, 44, 16, 15, 14, 13, 43, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 971
10	[1, 2, 3, 4, 8, 7, 9, 19, 21, 20, 11, 12, 15, 16, 14, 13, 43, 25, 26, 47] | Waktu Tempuh: 80 | Fitness: 926

Tabel 6.15 menunjukkan bahwa dari hasil running program sebanyak 10 kali, solusi optimal yang didapatkan berbeda-beda. Solusi optimal yang terdapat pada sub bab 6.4.2 dinyatakan valid terhadap batasan OP dan solusi tersebut dapat dikatakan paling optimal dari AMSA. Hal tersebut terbukti dari hasil running program ke-1 merupakan solusi hasil running program dengan nilai fitness tertinggi sebesar 1080. Hasil tersebut nantinya akan dijadikan perbandingan pada sub bab 6.10 Uji Coba 4. Hasil tersebut belum tentu merupakan solusi paling optimal dari AMSA dikarenakan adanya kemungkinan ditemukan solusi yang lebih optimal apabila dilakukan running program lebih banyak lagi.

6.7. Hasil Uji Coba 1: Mencari Parameter Terbaik Algoritma Genetika

Pada sub bab ini dilakukan uji coba merubah parameter-parameter yang dapat mempengaruhi hasil solusi dari implementasi AG meliputi probabilitas perkawinan silang, probabilitas mutasi, dan jumlah populasi. Dengan merubah parameter AG diharapkan mampu menemukan kombinasi parameter terbaik yang menghasilkan nilai *fitness* tertinggi. Adapun kombinasi parameter yang akan digunakan sebagai berikut.

a. Parameter probabilitas perkawinan silang yaitu 0.85, 0.9, dan 0.95.
b. Parameter probabilitas mutasi yaitu 0.05, 0.1, dan 0.15.
c. Parameter jumlah populasi sebanyak 30, 40, 50, 60, hingga 100 individu.

Dalam uji coba ini, jumlah generasi dan waktu tempuh maksimal dibuat sama seperti kasus sebelumnya yaitu 1000 iterasi dan 80 menit. Selanjutnya dibuat beberapa skenario uji coba untuk menentukan parameter terbaik AG sebagai berikut.

a. Skenario 1 mencari kombinasi parameter terbaik dengan merubah nilai probabilitas perkawinan silang dan probabilitas mutasi dengan jumlah populasi tetap pada setiap kombinasi yaitu 100 individu.
b. Skenario 2 menggunakan kombinasi parameter terbaik dari hasil skenario 1 dengan jumlah populasi yang berbeda.
c. Skenario 3 menggunakan kombinasi parameter terbaik ke 2 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
d. Skenario 4 menggunakan kombinasi parameter terbaik ke 3 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
e. Menguji konsistensi kombinasi parameter terbaik hasil skenario 2 dengan hasil skenario 3 dan 4.

6.7.1. Skenario 1

Pada skenario ini program Algoritma Genetika dijalankan berdasarkan parameter probabilitas perkawinan silang (Prob CO) 0.85, 0.9, 0.95, probabilitas mutasi (Prob Mut) 0.05, 0.1, 0.15, dan jumlah populasi (Jum Pop) 100. Secara keseluruhan hasil dari uji coba 1 skenario 1 ada pada Tabel 6.16.

Tabel 6.16 Hasil Uji Coba 1 Skenario 1

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AG1	0.95	0.05	100	672
AG2	0.95	0.1	100	604
AG3	0.95	0.15	100	800
AG4	0.9	0.05	100	627
AG5	0.9	0.1	100	649
AG6	0.9	0.15	100	779
AG7	0.85	0.05	100	661
AG8	0.85	0.1	100	685
AG9	0.85	0.15	100	666

Dari hasil uji coba 1 skenario 1 didapatkan bahwa parameter terbaik untuk AG yaitu parameter kode AG3 dengan probabilitas perkawinan silang 0.95, probabilitas mutasi 0.15, dan nilai *fitness* sebesar 800.

6.7.2. Skenario 2

Pada skenario ini program Algoritma Genetika dijalankan berdasarkan parameter AG3 probabilitas perkawinan silang (Prob CO) 0.95, probabilitas mutasi (Prob Mut) 0.15, dan jumlah populasi (Jum Pop) 30, 40, 50, hingga 100. Secara keseluruhan hasil dari uji coba 1 skenario 2 ada pada Tabel 6.17.

Tabel 6.17 Hasil Uji Coba 1 Skenario 2

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.95	0.15	30	961
Pop2	0.95	0.15	40	790
Pop3	0.95	0.15	50	844
Pop4	0.95	0.15	60	824
Pop5	0.95	0.15	70	848
Pop6	0.95	0.15	80	822
Pop7	0.95	0.15	90	879
Pop8	0.95	0.15	100	800

Dari hasil uji coba 1 skenario 2 didapatkan bahwa parameter terbaik untuk AG3 yaitu parameter dengan jumlah populasi 30 individu dengan probabilitas perkawinan silang 0.95, probabilitas mutasi 0.15, dan nilai *fitness* sebesar 961.

6.7.3. Skenario 3

Pada skenario ini program Algoritma Genetika dijalankan berdasarkan parameter AG6 (terbaik ke-2) probabilitas perkawinan silang (Prob CO) 0.9, probabilitas mutasi (Prob Mut) 0.15, dan jumlah populasi (Jum Pop) 30, 40, 50, hingga 100. Secara keseluruhan hasil dari uji coba 1 skenario 3 ada pada Tabel 6.18.

Tabel 6.18 Hasil Uji Coba 1 Skenario 3

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.9	0.15	30	933
Pop2	0.9	0.15	40	855
Pop3	0.9	0.15	50	793
Pop4	0.9	0.15	60	837
Pop5	0.9	0.15	70	781

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop6	0.9	0.15	80	799
Pop7	0.9	0.15	90	739
Pop8	0.9	0.15	100	779

Dari hasil uji coba 1 skenario 3 didapatkan bahwa parameter terbaik AG6 yaitu parameter dengan jumlah populasi 30 individu dengan probabilitas perkawinan silang 0.9, probabilitas mutasi 0.15, dan nilai *fitness* sebesar 933.

6.7.4. Skenario 4

Pada skenario ini program Algoritma Genetika dijalankan berdasarkan parameter AG8 (terbaik ke-3) probabilitas perkawinan silang (Prob CO) 0.85, probabilitas mutasi (Prob Mut) 0.1, dan jumlah populasi (Jum Pop) 30, 40, 50, hingga 100. Secara keseluruhan hasil dari uji coba 1 skenario 4 ada pada Tabel 6.19.

Tabel 6.19 Hasil Uji Coba 1 Skenario 4

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.85	0.1	30	825
Pop2	0.85	0.1	40	804
Pop3	0.85	0.1	50	855
Pop4	0.85	0.1	60	758
Pop5	0.85	0.1	70	753
Pop6	0.85	0.1	80	641
Pop7	0.85	0.1	90	720
Pop8	0.85	0.1	100	685

Dari hasil uji coba 1 skenario 4 didapatkan bahwa parameter terbaik AG8 yaitu parameter dengan jumlah populasi 50

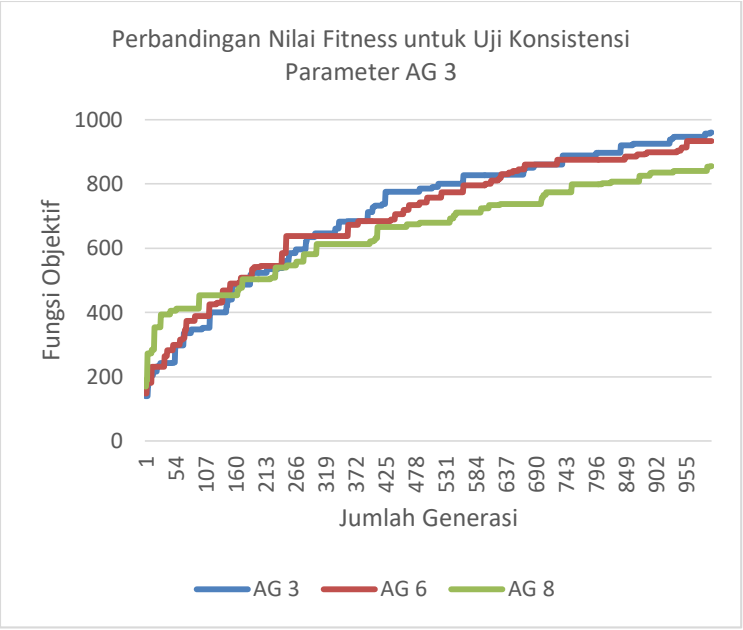
individu dengan probabilitas perkawinan silang 0.85, probabilitas mutasi 0.1, dan nilai *fitness* sebesar 855.

6.7.5. Uji Konsistensi

Skenario ini menguji parameter terbaik hasil skenario 2 dengan membandingkannya dengan hasil skenario 3 dan 4. Tujuan dari pengujian ini adalah mengetahui apakah perubahan pada jumlah populasi pada setiap kombinasi parameter mempengaruhi pemilihan kombinasi parameter terbaik pada skenario 1. Apabila hasil perbandingan nilai fitness menunjukkan bahwa parameter skenario 2 lebih baik dari parameter skenario 3 dan 4 maka dapat dikatakan bahwa pemilihan kombinasi parameter pada skenario 1 konsisten. Tabel 6.20 menunjukkan hasil perbandingan parameter terbaik dari skenario 2, 3, dan 4 berdasarkan nilai *fitness* pada generasi terakhir. Sedangkan Gambar 6.1 menunjukkan hasil perbandingan nilai *fitness* pada setiap generasi dalam bentuk grafik.

Tabel 6.20 Hasil Uji Coba 1 Perbandingan Parameter Terbaik

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AG3	0.95	0.15	30	961
AG6	0.9	0.15	30	933
AG8	0.85	0.1	50	855



Gambar 6.1 Grafik Performa AG3, AG6, dan AG8

Dari hasil Tabel 6.20 didapatkan bahwa parameter AG3 merupakan kombinasi parameter yang terbaik berdasarkan nilai *fitness* pada akhir generasi sebesar 961. Sedangkan pada Gambar 6.1 grafik AG3 merupakan grafik dengan trend terbaik. Dapat dikatakan bahwa dengan mengubah jumlah populasi, tidak mempengaruhi pemilihan parameter terbaik pada skenario 1. Sehingga dapat dikatakan bahwa parameter AG3 dengan Prob CO 0.95 dan Prob Mutasi 0.15 konsisten.

6.8. Hasil Uji Coba 2: Mencari Parameter Terbaik Algoritma Memetika *Hill Climbing*

Pada sub bab ini dilakukan uji coba merubah parameter-parameter yang dapat mempengaruhi hasil solusi dari implementasi AMHC meliputi probabilitas perkawinan silang, probabilitas mutasi, dan jumlah populasi. Dengan merubah parameter AMHC diharapkan mampu menemukan kombinasi

parameter terbaik yang menghasilkan nilai *fitness* tertinggi. Adapun kombinasi parameter yang akan digunakan sebagai berikut.

a. Parameter probabilitas perkawinan silang yaitu 0.4, 0.5, dan 0.6.
b. Parameter probabilitas mutasi yaitu 0.4, 0.5, dan 0.6.
c. Parameter jumlah populasi sebanyak 10, 20, 30, 40, hingga 80 individu.

Dalam uji coba ini, jumlah generasi dan waktu tempuh maksimal dibuat sama seperti kasus sebelumnya yaitu 1000 iterasi dan 80 menit. Selanjutnya dibuat beberapa skenario uji coba untuk menentukan parameter terbaik AMHC sebagai berikut.

a. Skenario 1 mencari kombinasi parameter terbaik dengan merubah nilai probabilitas perkawinan silang dan probabilitas mutasi dengan jumlah populasi tetap pada setiap kombinasi yaitu 10 individu.
b. Skenario 2 menggunakan kombinasi parameter terbaik dari hasil skenario 1 dengan jumlah populasi yang berbeda.
c. Skenario 3 menggunakan kombinasi parameter terbaik ke 2 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
d. Skenario 4 menggunakan kombinasi parameter terbaik ke 3 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
e. Menguji konsistensi kombinasi parameter terbaik hasil skenario 2 dengan hasil skenario 3 dan 4.

6.8.1. Skenario 1

Pada skenario ini program AMHC dijalankan berdasarkan parameter probabilitas perkawinan silang (Prob CO) 0.4, 0.5, 0.6, probabilitas mutasi (Prob Mut) 0.4, 0.5, 0.6, dan jumlah populasi (Jum Pop) 10. Secara keseluruhan hasil dari uji coba 2 skenario 1 ada pada Tabel 6.21.

Tabel 6.21 Hasil Uji Coba 2 Skenario 1

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AMHC1	0.4	0.4	10	976
AMHC2	0.4	0.5	10	911
AMHC3	0.4	0.6	10	911
AMHC4	0.5	0.4	10	1029
AMHC5	0.5	0.5	10	929
AMHC6	0.5	0.6	10	976
AMHC7	0.6	0.4	10	929
AMHC8	0.6	0.5	10	894
AMHC9	0.6	0.6	10	976

Dari hasil uji coba 2 skenario 1 didapatkan bahwa parameter terbaik untuk AMHC yaitu parameter kode AMHC4 dengan probabilitas perkawinan silang 0.5, probabilitas mutasi 0.4, dan nilai *fitness* sebesar 1029.

6.8.2. Skenario 2

Pada skenario ini program AMHC dijalankan berdasarkan parameter AMHC4 probabilitas perkawinan silang (Prob CO) 0.5, probabilitas mutasi (Prob Mut) 0.4, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 1 skenario 2 ada pada Tabel 6.22. Untuk grafik uji coba 2 skenario 2 dapat dilihat pada LAMPIRAN F.

Tabel 6.22 Hasil Uji Coba 2 Skenario 2

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.5	0.4	10	1029
Pop2	0.5	0.4	20	911
Pop3	0.5	0.4	30	1029
Pop4	0.5	0.4	40	952

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop5	0.5	0.4	50	890
Pop6	0.5	0.4	60	814
Pop7	0.5	0.4	70	843
Pop8	0.5	0.4	80	792

Dari hasil uji coba 2 skenario 2 didapatkan bahwa parameter terbaik untuk AMHC4 berdasarkan nilai fitness generasi terakhir dan grafik trend yaitu parameter dengan jumlah populasi 10 individu dengan probabilitas perkawinan silang 0.5, probabilitas mutasi 0.4, dan nilai *fitness* sebesar 1029.

6.8.3. Skenario 3

Pada skenario ini program AMHC dijalankan berdasarkan parameter AMHC9 (terbaik ke-2) probabilitas perkawinan silang (Prob CO) 0.6, probabilitas mutasi (Prob Mut) 0.6, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 2 skenario 3 ada pada Tabel 6.23.

Tabel 6.23 Hasil Uji Coba 2 Skenario 3

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.6	0.6	10	976
Pop2	0.6	0.6	20	911
Pop3	0.6	0.6	30	976
Pop4	0.6	0.6	40	1029
Pop5	0.6	0.6	50	911
Pop6	0.6	0.6	60	990
Pop7	0.6	0.6	70	969
Pop8	0.6	0.6	80	990

Dari hasil uji coba 2 skenario 3 didapatkan bahwa parameter terbaik AMHC9 yaitu parameter dengan jumlah populasi 40 individu dengan probabilitas perkawinan silang 0.6, probabilitas mutasi 0.6, dan nilai *fitness* sebesar 1029.

6.8.4. Skenario 4

Pada skenario ini program AMHC dijalankan berdasarkan parameter AMHC6 (terbaik ke-3) probabilitas perkawinan silang (Prob CO) 0.5, probabilitas mutasi (Prob Mut) 0.6, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 2 skenario 4 ada pada Tabel 6.24.

Tabel 6.24 Hasil Uji Coba 2 Skenario 4

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.5	0.6	10	976
Pop2	0.5	0.6	20	976
Pop3	0.5	0.6	30	1029
Pop4	0.5	0.6	40	1014
Pop5	0.5	0.6	50	1014
Pop6	0.5	0.6	60	987
Pop7	0.5	0.6	70	909
Pop8	0.5	0.6	80	925

Dari hasil uji coba 2 skenario 4 didapatkan bahwa parameter terbaik AMHC6 yaitu parameter dengan jumlah populasi 30 individu dengan probabilitas perkawinan silang 0.5, probabilitas mutasi 0.6, dan nilai *fitness* sebesar 1029.

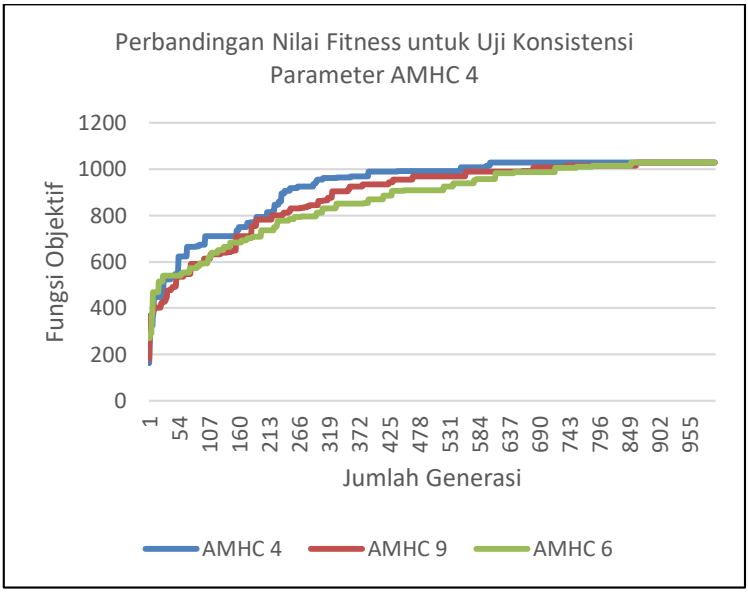
6.8.5. Uji Konsistensi

Skenario ini menguji parameter terbaik hasil skenario 2 dengan membandingkannya dengan hasil skenario 3 dan 4. Tujuan dari pengujian ini adalah mengetahui apakah perubahan pada jumlah populasi pada setiap kombinasi parameter mempengaruhi pemilihan kombinasi parameter terbaik pada

skenario 1. Apabila hasil perbandingan nilai fitness menunjukkan bahwa parameter skenario 2 lebih baik dari parameter skenario 3 dan 4 maka dapat dikatakan bahwa pemilihan kombinasi parameter pada skenario 1 konsisten. Tabel 6.25 menunjukkan hasil perbandingan parameter terbaik dari skenario 2, 3, dan 4 berdasarkan nilai *fitness* pada generasi terakhir. Sedangkan Gambar 6.2 menunjukkan hasil perbandingan nilai *fitness* pada setiap generasi dalam bentuk grafik.

Tabel 6.25 Hasil Uji Coba 2 Perbandingan Parameter Terbaik

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AMHC4	0.5	0.4	10	1029
AMHC9	0.6	0.6	40	1029
AMHC6	0.5	0.6	30	1029



Gambar 6.2 Grafik Performa AMHC4, AMHC9, dan AMHC6

Dari hasil Tabel 6.25 didapatkan bahwa ketiga parameter AMHC merupakan kombinasi parameter yang terbaik berdasarkan nilai *fitness* pada akhir generasi sebesar 1029. Namun pada Gambar 6.2 grafik AMHC4 merupakan grafik dengan trend terbaik. Dapat dikatakan bahwa dengan mengubah jumlah populasi, tidak mempengaruhi pemilihan parameter terbaik pada skenario 1. Sehingga dapat dikatakan bahwa parameter AMHC4 dengan Prob CO 0.5 dan Prob Mutasi 0.4 konsisten.

6.9. Hasil Uji Coba 3: Mencari Parameter Terbaik Algoritma Memetika *Simulated Annealing*

Pada sub bab ini dilakukan uji coba merubah parameter-parameter yang dapat mempengaruhi hasil solusi dari implementasi AMSA meliputi probabilitas perkawinan silang, probabilitas mutasi, dan jumlah populasi. Dengan merubah parameter AMSA diharapkan mampu menemukan kombinasi parameter terbaik yang menghasilkan nilai *fitness* tertinggi. Adapun kombinasi parameter yang akan digunakan sebagai berikut.

a. Parameter probabilitas perkawinan silang yaitu 0.4, 0.5, dan 0.6.
b. Parameter probabilitas mutasi yaitu 0.4, 0.5, dan 0.6.
c. Parameter jumlah populasi sebanyak 10, 20, 30, 40, hingga 80 individu.

Dalam uji coba ini, jumlah generasi dan waktu tempuh maksimal dibuat sama seperti kasus sebelumnya yaitu 1000 iterasi dan 80 menit. Selanjutnya dibuat beberapa skenario uji coba untuk menentukan parameter terbaik AMHC sebagai berikut.

a. Skenario 1 mencari kombinasi parameter terbaik dengan merubah nilai probabilitas perkawinan silang dan probabilitas mutasi dengan jumlah populasi tetap pada setiap kombinasi yaitu 10 individu.

b. Skenario 2 menggunakan kombinasi parameter terbaik dari hasil skenario 1 dengan jumlah populasi yang berbeda.
c. Skenario 3 menggunakan kombinasi parameter terbaik ke 2 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
d. Skenario 4 menggunakan kombinasi parameter terbaik ke 3 dari hasil skenario 1 dengan jumlah populasi yang berbeda.
e. Menguji konsistensi kombinasi parameter terbaik hasil skenario 2 dengan hasil skenario 3 dan 4.

6.9.1. Skenario 1

Pada skenario ini program AMSA dijalankan berdasarkan parameter probabilitas perkawinan silang (Prob CO) 0.4, 0.5, 0.6, probabilitas mutasi (Prob Mut) 0.4, 0.5, 0.6, dan jumlah populasi (Jum Pop) 10. Secara keseluruhan hasil dari uji coba 3 skenario 1 ada pada Tabel 6.26.

Tabel 6.26 Hasil Uji Coba 3 Skenario 1

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AMSA1	0.4	0.4	10	880
AMSA2	0.4	0.5	10	1029
AMSA3	0.4	0.6	10	1035
AMSA4	0.5	0.4	10	1080
AMSA5	0.5	0.5	10	921
AMSA6	0.5	0.6	10	874
AMSA7	0.6	0.4	10	976
AMSA8	0.6	0.5	10	891
AMSA9	0.6	0.6	10	951

Dari hasil uji coba 3 skenario 1 didapatkan bahwa parameter terbaik untuk AMSA yaitu parameter kode AMSA4 dengan probabilitas perkawinan silang 0.5, probabilitas mutasi 0.4, dan nilai *fitness* sebesar 1080.

6.9.2. Skenario 2

Pada skenario ini program AMSA dijalankan berdasarkan parameter AMSA4 probabilitas perkawinan silang (Prob CO) 0.5, probabilitas mutasi (Prob Mut) 0.4, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 3 skenario 2 ada pada Tabel 6.27.

Tabel 6.27 Hasil Uji Coba 3 Skenario 2

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.5	0.4	10	1080
Pop2	0.5	0.4	20	971
Pop3	0.5	0.4	30	938
Pop4	0.5	0.4	40	889
Pop5	0.5	0.4	50	764
Pop6	0.5	0.4	60	862
Pop7	0.5	0.4	70	869
Pop8	0.5	0.4	80	793

Dari hasil uji coba 3 skenario 2 didapatkan bahwa parameter terbaik untuk AMSA4 berdasarkan nilai fitness generasi terakhir yaitu parameter dengan jumlah populasi 10 individu dengan probabilitas perkawinan silang 0.5, probabilitas mutasi 0.4, dan nilai *fitness* sebesar 1080.

6.9.3. Skenario 3

Pada skenario ini program AMSA dijalankan berdasarkan parameter AMSA3 (terbaik ke-2) probabilitas perkawinan silang (Prob CO) 0.4, probabilitas mutasi (Prob Mut) 0.6, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 3 skenario 3 ada pada Tabel 6.28.

Tabel 6.28 Hasil Uji Coba 3 Skenario 3

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.4	0.6	10	1035
Pop2	0.4	0.6	20	941
Pop3	0.4	0.6	30	980
Pop4	0.4	0.6	40	853
Pop5	0.4	0.6	50	791
Pop6	0.4	0.6	60	811
Pop7	0.4	0.6	70	807
Pop8	0.4	0.6	80	769

Dari hasil uji coba 3 skenario 3 didapatkan bahwa parameter terbaik AMSA3 yaitu parameter dengan jumlah populasi 10 individu dengan probabilitas perkawinan silang 0.4, probabilitas mutasi 0.6, dan nilai *fitness* sebesar 1035.

6.9.4. Skenario 4

Pada skenario ini program AMSA dijalankan berdasarkan parameter AMSA2 (terbaik ke-3) probabilitas perkawinan silang (Prob CO) 0.4, probabilitas mutasi (Prob Mut) 0.5, dan jumlah populasi (Jum Pop) 10, 20, 30, hingga 80. Secara keseluruhan hasil dari uji coba 2 skenario 4 ada pada Tabel 6.29.

Tabel 6.29 Hasil Uji Coba 3 Skenario 4

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop1	0.4	0.5	10	1029
Pop2	0.4	0.5	20	912
Pop3	0.4	0.5	30	979
Pop4	0.4	0.5	40	904
Pop5	0.4	0.5	50	822
Pop6	0.4	0.5	60	800

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
Pop7	0.4	0.5	70	785
Pop8	0.4	0.5	80	721

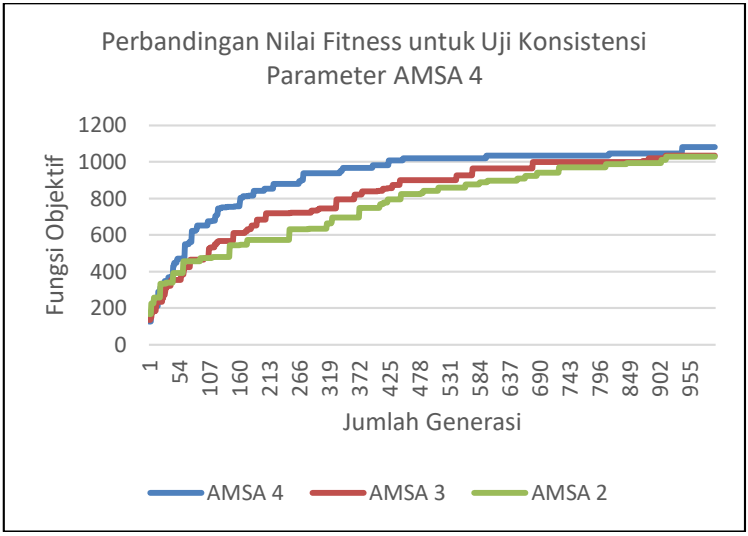
Dari hasil uji coba 3 skenario 4 didapatkan bahwa parameter terbaik AMSA2 yaitu parameter dengan jumlah populasi 10 individu dengan probabilitas perkawinan silang 0.4, probabilitas mutasi 0.5, dan nilai *fitness* sebesar 1029.

6.9.5. Uji Konsistensi

Skenario ini menguji parameter terbaik hasil skenario 2 dengan membandingkannya dengan hasil skenario 3 dan 4. Tujuan dari pengujian ini adalah mengetahui apakah perubahan pada jumlah populasi pada setiap kombinasi parameter mempengaruhi pemilihan kombinasi parameter terbaik pada skenario 1. Apabila hasil perbandingan nilai fitness menunjukkan bahwa parameter skenario 2 lebih baik dari parameter skenario 3 dan 4 maka dapat dikatakan bahwa pemilihan kombinasi parameter pada skenario 1 konsisten. Tabel 6.30 menunjukkan hasil perbandingan parameter terbaik dari skenario 2, 3, dan 4 berdasarkan nilai *fitness* pada generasi terakhir. Sedangkan Gambar 6.3 menunjukkan hasil perbandingan nilai *fitness* pada setiap generasi dalam bentuk grafik.

Tabel 6.30 Hasil Uji Coba 3 Perbandingan Parameter Terbaik

Kode	Prob CO	Prob Mut	Jum Pop	Fitness
AMSA4	0.5	0.4	10	1080
AMSA3	0.4	0.6	10	1035
AMSA2	0.4	0.5	10	1029



Gambar 6.3 Grafik Performa AMSA4, AMSA3, dan AMSA2

Dari hasil Tabel 6.27 didapatkan bahwa parameter AMSA4 merupakan kombinasi parameter yang terbaik berdasarkan nilai *fitness* pada akhir generasi sebesar 1080. Sedangkan pada Gambar 6.3 grafik AMSA4 merupakan grafik dengan trend terbaik. Dapat dikatakan bahwa dengan mengubah jumlah populasi, tidak mempengaruhi pemilihan parameter terbaik pada skenario 1. Sehingga dapat dikatakan bahwa parameter AMSA4 dengan Prob CO 0.5 dan Prob Mutasi 0.4 konsisten.

6.10. Hasil Uji Coba 4: Membandingkan Hasil Algoritma Genetika, Algoritma Memetika *Hill Climbing*, dan Algoritma Memetika *Simulated Annealing*

Pada sub bab ini dilakukan uji coba yaitu membandingkan hasil algoritma yang menggunakan parameter sesuai pada referensi dengan hasil algoritma parameter uji coba terbaik. Hasil perbandingan AG, AMHC, dan AMSA dapat dilihat pada Tabel 6.31 dan Gambar 6.4.

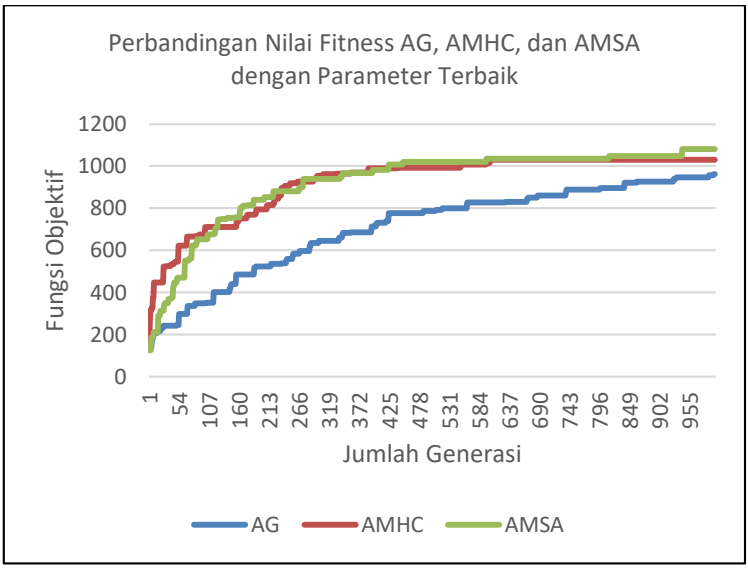
Tabel 6.31 Hasil Uji Coba 4 Perbandingan Parameter AG, AMHC, dan AMSA

	Parameter Referensi				Parameter Uji Coba			
Kode	**Prob CO**	**Prob Mutasi**	**Jum Pop**	**Fitness**	**Prob CO**	**Prob Mutasi**	**Jum Pop**	**Fitness**
AG	0.95	0.05	100	**756**	0.95	0.15	30	**961**
AMHC	0.5	0.5	10	**1022**	0.5	0.4	10	**1029**
AMSA	0.5	0.5	10	**1080**	0.5	0.4	10	**1080**

Dari hasil Tabel 6.28, menunjukkan adanya perbedaan nilai fitness dari penggunaan parameter sesuai dengan referensi yang digunakan dengan parameter uji coba dari data yang ada. Untuk parameter AG yang dijalankan sesuai dengan referensi menghasilkan nilai fitness sebesar 612, sedangkan dengan menggunakan parameter uji coba pada sub bab 6.7 menghasilkan nilai fitness yang lebih tinggi yaitu 961. Untuk parameter AMHC yang dijalankan sesuai dengan referensi menghasilkan nilai fitness sebesar 911, sedangkan dengan menggunakan parameter uji coba pada sub bab 6.8 menghasilkan nilai fitness yang lebih tinggi yaitu 1029. Untuk parameter AMSA yang dijalankan sesuai dengan referensi menghasilkan nilai fitness sebesar 1080, sedangkan dengan menggunakan parameter uji coba pada sub bab 6.9 menghasilkan nilai fitness yang sama yaitu 1080.

Hasil Tabel 6.28 menyatakan bahwa dengan menggunakan konfigurasi parameter terbaik sesuai pada referensi belum tentu menghasilkan solusi yang paling optimal, terbukti dengan pencarian parameter terbaik untuk setiap algoritma menggunakan data uji coba trayek bis kota Surabaya menghasilkan solusi yang lebih optimal daripada konfigurasi parameter terbaik dari referensi. Hal tersebut dapat terjadi karena konfigurasi parameter terbaik sangat relatif terhadap model optimasi dan data uji coba yang digunakan. Namun khusus pada AMSA, solusi yang dihasilkan menggunakan konfigurasi parameter referensi ternyata sama dengan solusi

yang dihasilkan menggunakan parameter terbaik dari data uji coba. Sehingga dapat dinyatakan bahwa penggunaan AMSA lebih stabil untuk pembuatan model optimasi yang berbeda.



Gambar 6.4 Grafik Performa AG, AMHC, dan AMSA

Dari hasil Gambar 6.4 dapat disimpulkan bahwa dengan menggunakan data uji coba jalur bis kota Surabaya, Algoritma Memetika *Simulated Annealing* (AMSA) memberikan solusi dengan nilai fitness yang lebih tinggi daripada Algoritma Genetika (AG) dan Algoritma Memetika *Hill Climbing* (AMHC), sehingga juga dapat disimpulkan bahwa performa Algoritma Memetika lebih baik daripada Algoritma Genetika.

6.11. Hasil Uji Coba 5: Mengubah Simpul Awal dan Akhir

Pada sub bab ini dilakukan uji coba dengan mencari rute optimum menggunakan titik/simpul berbeda dari simpul 9 (Royal Plaza) ke simpul 37 (Jembatan Merah Plaza) dengan parameter jumlah populasi, probabilitas perkawinan silang, dan probabilitas mutasi terbaik dari Algoritma Memetika *Simulated*

Annealing, yaitu masing-masing 10 individu, 0.5, dan 0.4. Dikarenakan jarak dari simpul awal ke akhir lebih dekat, maka batasan waktu tempuh maksimal dikurangi menjadi 40 menit. Tujuan dari uji coba ini adalah memastikan bahwa program yang dibuat dapat digunakan untuk studi kasus dengan simpul awal dan akhir yang berbeda dimana masih dapat menghasilkan solusi yang layak dan memenuhi batasan model OP.

6.11.1. Validasi Solusi Layak Awal

Hal yang dilakukan sebelum mendapatkan solusi optimal yaitu membangkitkan solusi layak awal. Solusi yang dibangkitkan harus memenuhi batasan-batasan yang telah ditentukan pada model OP. Pencarian solusi layak awal akan dilakukan pengulangan hingga mencapai jumlah populasi yang telah ditentukan yaitu 10 individu. Tabel 6.32 merupakan hasil dari pembangkitan populasi awal AMSA.

Tabel 6.32 Hasil Pembangkitan Populasi Awal Uji Coba 5

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[9, 27, 37] (27, 89)
2	[9, 33, 37] (26, 89)
3	[9, 29, 37] (26, 136)
4	[9, 12, 46, 37] (38, 150)
5	[9, 41, 37] (26, 139)
6	[9, 20, 25, 17, 37] (40, 291)
7	[9, 10, 37] (26, 89)
8	[9, 26, 45, 37] (38, 172)
9	[9, 45, 37] (26, 89)
10	[9, 21, 37] (29, 127)

Solusi yang dihasilkan pada Tabel 6.32 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 9 dan berakhir simpul ke 37.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
3. Batasan 3 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 40 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Dapat disimpulkan bahwa solusi awal yang dibangkitkan untuk dijadikan populasi awal merupakan solusi layak dan valid.

6.11.2. Validasi Solusi Akhir

Populasi awal yang terbentuk selanjutnya mengalami proses seleksi, perkawinan silang, mutasi, dan perbaikan lokal yang dilakukan secara berurutan dan mengalami pengulangan proses tersebut hingga memenuhi kriteria pemberhentian yaitu 1000 kali pengulangan. Pada setiap pengulangan akan dihasilkan solusi akhir sebanyak jumlah populasi yaitu 10 individu. Tabel 6.33 merupakan solusi akhir pada iterasi ke-1000 yang berisikan 10 individu dengan menggunakan AMSA.

Tabel 6.33 Hasil Populasi Akhir Uji Coba 5

Individu	[Solusi] (Waktu Tempuh, Skor)
1	[9, 19, 20, 21, 22, 14, 13, 42, 43, 25, 26, 27, 29, 32, 31, 41, 37] (39, 671)
2	[9, 19, 20, 21, 22, 14, 13, 43, 25, 26, 29, 32, 31, 41, 37] (39, 671)
3	[9, 19, 20, 21, 22, 42, 14, 13, 43, 25, 26, 27, 29, 32, 41, 37] (39, 671)
4	[9, 19, 20, 21, 22, 42, 14, 13, 43, 25, 26, 29, 32, 31, 41, 37] (39, 671)
5	[9, 10, 19, 20, 21, 22, 14, 13, 43, 25, 26, 27, 29, 32, 41, 37] (39, 671)

Indi vidu	[Solusi] (Waktu Tempuh, Skor)
6	[9, 19, 20, 21, 22, 14, 13, 43, 25, 26, 27, 29, 32, 41, 37] (39, 671)
7	[9, 19, 20, 21, 22, 14, 13, 42, 43, 25, 26, 27, 29, 32, 41, 37] (39, 671)
8	[9, 19, 20, 21, 22, 42, 14, 13, 43, 25, 26, 27, 29, 32, 31, 41, 37] (39, 671)
9	[9, 10, 19, 20, 21, 22, 14, 13, 43, 25, 26, 29, 32, 31, 41, 37] (39, 671)
10	[9, 19, 20, 21, 22, 14, 13, 42, 43, 25, 26, 29, 32, 31, 41, 37] (39, 671)

Solusi yang dihasilkan pada Tabel 6.30 dapat dinyatakan layak dan valid apabila tidak melanggar batasan-batasan yang telah ditentukan sebelumnya, yaitu:

1. Batasan 1 terpenuhi. Seluruh solusi awal memiliki simpul awal dan simpul akhir yang tetap yaitu dimulai dengan simpul ke 9 dan berakhir simpul ke 37.
2. Batasan 2 terpenuhi. Semua solusi awal mempunyai simpul saling terhubung (hasil Algoritma Dijkstra) dan setiap simpul hanya dapat dikunjungi maksimum satu kali.
3. Batasan 4 terpenuhi. Tidak ada solusi awal yang melebihi batas waktu tempuh maksimal yaitu 80 menit.
4. Batasan 4 dan 5 terpenuhi. Seluruh solusi awal memenuhi batasan 1, 2, dan 3 sehingga solusi layak awal tidak mengalami *subtours*.

Seluruh solusi akhir pada Tabel 6.30 masih layak dan valid karena memenuhi batasan OP. Dari seluruh solusi akhir tersebut diambil solusi terbaik dengan melihat jumlah skor yang paling tinggi. Didapatkan solusi dengan nilai *fitness* tertinggi adalah solusi pada baris pertama dengan nilai *fitness* sebesar 671 dan waktu tempuh sebesar 39 menit. Solusi tersebut dianggap sebagai rekomendasi rute yang paling optimum dengan menggunakan AMSA. Tabel lintasan solusi tersebut terdapat pada Tabel 6.34.

Tabel 6.34 Solusi Optimal Uji Coba 5

No	Titik	Lokasi
1	9	Royal Plaza
2	19	DTC Wonokromo
3	20	Kebun Binatang Surabaya
4	21	Taman Bungkul
5	22	Tunjungan Plaza
6	14	Stasiun Gubeng
7	13	Monumen Kapal Selam
8	42	Perempatan Pemuda
9	43	Balai Kota Surabaya
10	25	Museum Surabaya (Gedung Siola)
11	26	BG Junction
		Pasar Blauran
12	27	Pertigaan Bubutan Raden Saleh
13	29	Stasiun Pasar Turi
14	32	Pasar Turi Baru
15	31	Perempatan Tembaan Bubutan
16	41	Pertigaan Bubutan Kebun Rojo
17	37	Jembatan Merah Plaza

Dari uji coba ini dapat dilihat bahwa program yang dibuat dapat digunakan pada simpul awal dan akhir yang berbeda dimana masih dapat menghasilkan solusi yang layak dan valid yaitu memenuhi batasan model OP.

6.12. Hasil Uji Coba 6: Mengubah Nilai Tmax

Pada uji coba kali ini mencari rute optimum sesuai dengan percontohan rute dengan kode trayek P1 yaitu dari simpul 1 (Terminal Purabaya) menuju simpul 47 (Pelabuhan Tanjung Perak) dengan parameter AMSA terbaik, yaitu jumlah populasi 10 individu, probabilitas kawin silang 0.5, dan probabilitas

mutasi 0.4. Khususnya pada uji coba ini, batasan waktu (Tmax) dirubah dimana Tmax dibuat bervariasi mulai dari 10 sampai 100 menit dengan rentang waktu setiap 10 menit. Tujuan dari uji coba ini adalah memastikan bahwa program yang dibuat dapat digunakan untuk menghasilkan solusi yang layak dan memenuhi batasan model OP dengan ukuran Tmax yang bervariasi/beragam. Berikut ditampilkan solusi terbaik untuk setiap batasan waktu tempuh (Tmax) dalam Tabel 6.35.

Tabel 6.35 Solusi Terbaik untuk Setiap Tmax

Tmax	Skor	Waktu Tempuh	Solusi
10	0	0	Tidak ditemukan solusi yang layak
20	0	0	Tidak ditemukan solusi yang layak
30	0	0	Tidak ditemukan solusi yang layak
40	0	0	Tidak ditemukan solusi yang layak
50	0	0	Tidak ditemukan solusi yang layak
60	623	60	[1, 2, 3, 4, 7, 8, 9, 11, 10, 19, 20, 21, 22, 42, 43, 25, 26, 47] (60, 623)
70	903	70	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 42, 43, 14, 13, 15, 16, 25, 26, 29, 32, 41, 47] (70, 903)
80	1080	80	[1, 2, 3, 4, 7, 8, 9, 19, 20, 21, 22, 42, 43, 14, 13, 15, 16, 17, 18, 25, 26, 29, 32, 31, 35, 41, 47] (80, 1080)
90	1119	89	[1, 2, 3, 4, 7, 8, 9, 11, 19, 20, 21, 22, 26, 29, 32, 41, 35, 18, 17, 44, 16, 15, 14, 13, 43, 25, 47] (89, 1119)
100	1236	100	[1, 2, 3, 4, 7, 8, 9, 19, 20, 11, 12, 13, 14, 15, 16, 24, 22, 43, 25, 26, 29, 32, 41, 35, 36, 37, 18, 17, 47] (100, 1236)

Dapat dilihat pada Tabel 6.32 bahwa solusi terbaik pada Tmax 10 hingga 50 menit tidak memiliki solusi yang layak. Dari uji coba ini juga dapat dilihat bahwa semakin besar tMax maka semakin banyak simpul yang dapat dikunjungi sehingga nilai *fitness* juga semakin besar.

BAB VII
KESIMPULAN DAN SARAN

Bab ini menjelaskan beberapa kesimpulan yang dapat diperoleh dari dari tugas akhir. Selain itu, dalam bab ini juga diuraian beberapa perbaikan yang dapat dapat dilakukan untuk menyempuranakan hasil dari Tugas Akhir.

7.1. Kesimpulan

Daei keseluruan pengerjaan Tugas Akhir dapat ditarik beberapa kesimpulan sebagai berikut:

a) Tugas Akhir ini telah berhasil mengimpelementasikan solusi terhadap *reorienteering problem* menggunakan algoritma memetika untuk memperoleh rute perjalanan yang optimal dalam menempuh tempat-tempat objek wisata di kota Surabaya menggunakan transportasi umum bis kota.

b) Hasil uji coba menunjukkan bahwa Algoritma Memetika yang melibatkan algoritma pencarian lokal Simulated Annealing meberikan hasil solusi akhir yang lebih baik dobandingkan dengan Algoritma Genetika dasar dan Algoritma Memetika yang melibatkan pencarian lokal berbasis algoritma Hill Climbing.

c) Dalam implememtai ketiga algoritma yang dibandingkan, pemilihan parameter probabilitas perkawinan silang, probabilitas mutasi, dan jumlah populasi sangat mempengaruhi solusi yang dihasilkan.

7.2. Saran

Saran yang diberikan berdasarkan hasil penelitian yang dilakukan untuk penelitian selanjutnya antara lain:

a) Operasi perkawinan silang yang digunakan dalam Tugas Akhir ini hanya melibatkan metode *single-point*

crossover. Untuk mendapatkan solusi yang lebih baik dapat dicoba dan dibandingkan dengan beberapa metode perkawinan silang lainnya, seperti *multi-point crossover*, *uniform crossover*, atau *injection crossover*.

b) Operator mutasi yang digunakan dalam tugas akhir ini terbatas pada pengguna operator *add*, *add-swap*, *add-remove*, dan *jenis* operator lainnya, seperti *move* dan *replace* yang untuk menghasilkan solusi akhir yang lebih acak.

c) Waktu tempuh antar simpul yang digunakan pada tugas akhir ini bersifat statis tanpa memperhitungkan pengaruh tingkat kemacetan pada jam-jam tertentu. Selain itu, Tugas Akhir ini mengasumsikan bahwa tidak terdapat waktu tunggu pada saat bis kota berhenti di simpul yang dilalui.Waktu tunggu dan waktu tempuh yang lebih dinamis berdasarkan data real-time di lapangan dapat diujicobakan untuk melihat pengaruhnya.

DAFTAR PUSTAKA

[1] Aloisius de Rozari and Yudi Hari Wibowo, "Faktor-faktor Yang Menyebabkan Kemacetan Lalu Lintas di Jalan Utama Kota Surabaya," *J. Penelit. Adm. Publik*, vol. 1, no. 1, pp. 1–5, 2015.

[2] A. Aris, "Analisis Dampak Sosial Ekonomi Jalan Akibat Kemacetan Lalu Lintas (Studi Kasus Area Universitas Brawijaya Malang)," vol. 1, pp. 1–14, 2012.

[3] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering Problem: A Survey of Recent Variants, Solution Approaches and Applications," *Eur. J. Oper. Res.*, vol. 255, no. 2, pp. 315–332, 2016.

[4] M. A. Albar, "Algoritma Genetik Tabu Search Dan Memetika Pada Permasalahan Penjadwalan Kuliah," *Semin. Nas. Teknol. Inf. dan Multimed.*, no. 2008, pp. 45–50, 2013.

[5] C. R. Reeves and T. Yamada, "Genetic Algorithms, Path Relinking and The Flowshop Sequencing Problem.," *Evol. Comput.*, vol. 6, no. 1, pp. 45–60, 1998.

[6] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms," *Caltech Concurr. Comput. Program, C3P Rep.*, vol. 826, p. 1989, 1989.

[7] E. K. Burke, J. R. Newall, and R. E. Weare, "A Memetic Algorithm for University Exam Timetabling," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1153, pp. 241–250, 1996.

[8] U. Derigs, "Optimazion and Operations Research," *Encyclopedia of Life Support Systems (EOLSS)*, vol. I. .

[9] R. Bronson, *Theory and Problem of Operation Research*. Singapore: McGraw- Hill Book Co., 1983.

[10] L. . Mays and Y.-K. Tung, *Hydrosystems Engineering & Management*. Singapore: McGraw- Hill, 1992.

[11] B. W. Taylor, *Introduction to Management Science Ed. 11*. Pearson, 2013.

[12] T. Abiy, H. Pang, and C. Williams, “Dijkstra’s Shortest Path Algorithm.” [Online]. Available: https://brilliant.org/wiki/dijkstras-short-path-finder/.

[13] B. L. Golden, L. Levy, and R. Vohra, “The Orienteering Problem,” *Nav. Res. Logist.*, vol. 34, no. 3, pp. 307–318, 1987.

[14] P. Garg, “A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm,” *Int. J. Netw. Secur. Its Appl.*, vol. 1, no. 1, pp. 34–42, 2009.

[15] A. Schatten, “Genetic Algorithms.” [Online]. Available: http://web.cs.ucdavis.edu/~vemuri/classes/ecs271/Genetic Algorithms Short Tutorial.htm.

[16] L. Hakim, B. S. R, and N. Qodariyah, “Penerapan Algoritma Memetika pada Penentuan Komposisi Pakan Ayam Petelur,” no. 0331.

[17] P. Moscato and C. Cotta, *Handbook of Metaheuristics*, vol. 57. 2003.

[18] S. Kusumawati, *Pengantar Kecerdasan Buatan: Teknik Pencarian Heuristik*. Graha Ilmu, 2003.

[19] E. V. Dangkua, V. Gunawan, and K. Adi, “Penerapan Metode Hill Climbing Pada Sistem Informasi Geografis Untuk Mencari Lintasan Terpendek,” *J. Sist. Inf. Bisnis*, vol. 5, no. 1, pp. 19–25, 2016.

[20] H. A. S. Shiddiq, Sugiono, and C. F. M. Tantrika, “Implementasi Algoritma Simulated Annealing pada Penjadwalan Produksi untuk Meminimasi Makespan (Studi Kasus di PT . Gatra Mapan , Karang Ploso , Malang),” *J. REKAYASA DAN Manaj. Sist. Ind. VOL. 3 NO. 1 Tek. Ind. Univ. BRAWIJAYA*, vol. 3, no. 1, pp. 43–52.

[21] E. Soubeiga, “Development and Application of Hyperheuristics to Personnel Scheduling,” Nottingham, 2003.

[22] I. W. A. K. Yoga, “Advanced Traveler Information Systems: Optimasi Rencana Perjalanan dengan Model Orienteering Problem dan Genetic Algorithm (Studi Kasus: Trayek Angkot Surabaya),” Institut Teknologi

Sepuluh Nopember, 2017.

[23] Y. Lu, U. Benlic, and Q. Wu, "A memetic algorithm for the Orienteering Problem with Mandatory Visits and Exclusionary Constraints," *Eur. J. Oper. Res.*, 2018.

[24] P. Merz and B. Freisleben, "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem," *Proc. 1999 Congr. Evol. Comput. CEC 1999*, vol. 3, no. Fb 12, pp. 2063–2070, 1999.

[25] Pemerintah Kota Surabaya, "Transportasi," 2015. [Online]. Available: https://www.surabaya.go.id/id/page/0/8263/transportasi. [Accessed: 08-Feb-2018].

[26] "Bab 7 Algoritma Genetika." [Online]. Available: http://entin.lecturer.pens.ac.id/Kecerdasan Buatan/Buku/Bab 7 Algoritma Genetika.pdf.

[27] R. Poli and W. B. Langdon, "Genetic Programming with One-Point Crossover and Point Mutation," pp. 1–11, 1997.

[28] L. Han and G. Kendall, "Guided Operators for a Hyper-Heuristic Genetic Algorithm," 2003.

[29] S. Abdullah, E. K. Burke, and B. McCollum, "A Hybrid Evolutionary Approach to the University Course Timetabling Problem," *IEEE Congr. Evol. Comput.*, pp. 1764–1768, 2007.

"Halaman ini sengaja dikosongkan"

BIODATA PENULIS

Penulis bernama lengkap Fauzi Rakhmad Firdausi, dilahirkan di Probolinggo pada tanggal 12 Oktober 1995. Penulis merupakan anak terakhir dari tiga bersaudara. Beberapa pendidikan formal yang telah ditempuh penulis yaitu SDN Sukabumi III Kota Probolinggo pada tahun 2002-2008, SMPN 5 Kota Probolinggo pada tahun 2008-2011, dan SMAN 1 Kota Probolinggo pada tahun 2011-2014. Pada tahun 2014, penulis melanjutkan pendidikan ke jenjang perguruan tinggi di Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember melalui jalur SNMPTN dan terdaftar sebagai mahasiswa aktif dengan NRP 05211440000035.

Selama menjadi mahasiswa, penulis aktif dalam berbagai kegiatan kepanitiaan. Penulis pernah menjadi panitia Gerakan Integralistik (GERIGI) ITS pada tahun 2015. Penulis juga menjadi panitia dalam Information System Expo (ISE) pada tahun 2015 sebagai sie acara dan diberikan kesempatan untuk menjadi koordinator acara pada tahun 2016 . Penulis pernah menjalani kerja praktik di PT PLN Persero Area Pengatur Distribusi Jawa Timur di bagian Master pada bulan Juni sampai Juli tahun 2017.

Pada tahun 2018, penulis mengambil laboratorium Rekayasa Data dan Inteligensi Bisnis (RDIB) di Departemen Sistem Informasi, ITS, dengan mengambil topik tugas akhir mengenai optimasi. Untuk keperluan penelitian, penulis dapat dihubungi melalui e-mail: fauzirakhmadf@gmail.com

"Halaman ini sengaja dikosongkan"

LAMPIRAN A:
DESKRIPSI LOKASI DAN SKOR

Lampiran ini merupakan deskripsi lokasi dan nilai skor yang dijadikan sebagai sebuah titik/simpul. Merujuk pada Tabel A.1.

Tabel A.1 Daftar Lokasi dan Skor

Titik	Lokasi	Skor	Skor Total
1	Terminal Bungurasih	43	43
2	City of Tomorrow Mall	36	36
3	Masjid Al-Akbar	35	35
4	Taman Pelangi	44	44
5	Pertigaan Ahmad Yani Jemur Andayani	0	0
6	Taman Flora	27	27
7	DBL Arena	29	29
8	Maspion Square	24	24
9	Royal Plaza	54	54
10	Pertigaan Wonokromo	0	0
11	Stasiun Wonokromo	39	39
12	Marvel City	61	61
13	Monumen Kapal Selam	46	87
	Plaza Surabaya	41	
14	Stasiun Gubeng	15	15
15	Grand City Mall	33	33
16	Hi-Tech Mall	66	66
17	Pasar Atom Surabaya	38	93
	ITC Surabaya Mega Grosir	55	
18	Wisata Religi Ampel	51	51
19	DTC Wonokromo	52	52
20	Kebun Binatang Surabaya	23	67

A-2

Titik	Lokasi	Skor	Skor Total
	Patung Suro dan Boyo	44	
21	Taman Bungkul	38	38
22	Tunjungan Plaza	52	52
23	Monumen Pers Perjuangan Surabaya	0	0
24	Hotel Majapahit (Yamato)	49	49
25	Museum Surabaya (Gedung Siola)	42	42
26	BG Junction	56	83
	Pasar Blauran Baru	27	
27	Pertigaan Bubutan Raden Saleh	0	0
28	Museum Dr. Soetomo GNI	18	18
29	Stasiun Pasar Turi	47	47
30	Tugu Pahlawan	10	10
31	Perempatan Tembaan Bubutan	0	0
32	Pasar Turi Baru	21	21
33	Perempatan Kranggan Tembok Dukuh	0	0
34	Perempatan Demak Dupak	0	0
35	Stasiun Surabaya Kota (Semut)	33	33
36	Museum Kesehatan Dr. Adhyatma	10	10
37	Jembatan Merah Plaza	35	35
38	Pertigaan Rajawali Indrapura	0	0
39	Pertigaan Perak Rajawali	0	0
40	Monumen Pelayaran	0	0
41	Pertigaan Bubutan Kebun Rojo	0	0
42	Perempatan Pemuda	0	0
43	Balai Kota Surabaya	28	28

Titik	Lokasi	Skor	Skor Total
44	Perempatan Kalianyar Pengampon	0	0
45	Pertigaan Pasar Kembang Arjuno	0	0
46	Perempatan Embong Malang Blauran	0	0
47	Pelabuhan Tanjung Perak	50	50
48	Terminal Tambak Osowilangun	46	46

"Halaman ini sengaja dikosongkan"

LAMPIRAN B:
WAKTU TEMPUH ANTAR TITIK

Lampiran ini merupakan daftar seluruh waktu tempuh antar simpul yang saling terhubung langsung. Merujuk pada Tabel B.1.

Tabel B.1 Daftar Nilai Waktu Tempuh

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
1	2	6
2	1	11
2	3	3
3	2	3
3	4	2
4	3	7
4	5	2
4	7	2
5	4	1
5	6	13
6	5	15
6	12	10
7	5	2
7	8	1
8	7	1
8	9	2
9	8	2
9	10	2
10	9	2
10	11	2
10	19	1
11	10	2

B-2

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
11	12	4
12	6	10
12	11	5
12	13	7
13	12	9
13	14	1
13	42	3
14	13	1
14	15	1
15	14	1
15	16	3
16	15	3
16	44	4
17	18	1
17	44	3
18	17	1
18	44	4
19	10	1
19	20	2
20	19	2
20	21	2
20	45	8
21	20	2
21	22	8
22	23	1
22	42	1
23	22	1
23	46	3
24	23	1

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
25	24	1
25	26	1
25	44	3
26	27	1
27	28	1
27	29	1
28	31	1
29	27	1
29	32	1
29	33	2
30	25	3
30	31	1
31	30	1
31	32	2
31	41	1
32	31	1
32	34	3
33	25	4
33	26	3
33	29	2
33	34	4
33	45	3
34	32	4
34	33	4
34	39	6
34	48	16
35	17	2
35	30	2

B-4

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
35	41	1
35	44	6
36	38	2
37	18	2
37	35	2
38	37	2
38	39	1
39	34	5
39	38	1
39	40	3
40	39	3
40	47	7
41	31	1
41	35	1
41	36	1
42	21	8
42	43	1
43	13	3
43	14	2
43	25	4
43	42	4
43	44	5
44	16	4
44	17	3
44	25	5
44	35	6
44	43	6
45	20	8
45	33	3

Simpul Awal	Simpul Akhir	Waktu tempuh (menit)
45	46	4
46	25	2
46	26	1
46	33	4
46	45	4
47	40	7
48	34	17

“Halaman ini sengaja dikosongkan”

LAMPIRAN C:
VARIABEL KEPUTUSAN OP

Lampiran ini merupakan daftar keterangan variabel yang digunakan untuk pemodelan *Orienteering Problem*. Merujuk pada Tabel C.1.

Tabel C.1 Daftar Variabel Keputusan

Variabel	Keterangan
x1.2	Kunjungan dari simpul 1 ke simpul 2
x2.1	Kunjungan dari simpul 2 ke simpul 1
x2.3	Kunjungan dari simpul 2 ke simpul 3
x3.2	Kunjungan dari simpul 3 ke simpul 2
x3.4	Kunjungan dari simpul 3 ke simpul 4
x4.3	Kunjungan dari simpul 4 ke simpul 3
x4.5	Kunjungan dari simpul 4 ke simpul 5
x4.7	Kunjungan dari simpul 4 ke simpul 3
x5.4	Kunjungan dari simpul 5 ke simpul 4
x5.6	Kunjungan dari simpul 5 ke simpul 6
x6.5	Kunjungan dari simpul 6 ke simpul 5
x6.12	Kunjungan dari simpul 6 ke simpul 12
x7.5	Kunjungan dari simpul 7 ke simpul 5
x7.8	Kunjungan dari simpul 7 ke simpul 8
x8.7	Kunjungan dari simpul 8 ke simpul 7
x8.9	Kunjungan dari simpul 8 ke simpul 9
x9.8	Kunjungan dari simpul 9 ke simpul 8
x9.10	Kunjungan dari simpul 9 ke simpul 10
x10.9	Kunjungan dari simpul 10 ke simpul 9
x10.11	Kunjungan dari simpul 10 ke simpul 11
x10.19	Kunjungan dari simpul 10 ke simpul 19
x11.10	Kunjungan dari simpul 11 ke simpul 10

C-2

Variabel	Keterangan
x11.12	Kunjungan dari simpul 11 ke simpul 12
x12.6	Kunjungan dari simpul 12 ke simpul 6
x12.11	Kunjungan dari simpul 12 ke simpul 11
x12.13	Kunjungan dari simpul 12 ke simpul 13
x13.12	Kunjungan dari simpul 13 ke simpul 12
x13.14	Kunjungan dari simpul 13 ke simpul 14
x13.42	Kunjungan dari simpul 13 ke simpul 42
x14.13	Kunjungan dari simpul 14 ke simpul 13
x14.15	Kunjungan dari simpul 14 ke simpul 15
x15.14	Kunjungan dari simpul 15 ke simpul 14
x15.16	Kunjungan dari simpul 15 ke simpul 16
x16.15	Kunjungan dari simpul 16 ke simpul 15
x16.44	Kunjungan dari simpul 16 ke simpul44
x17.18	Kunjungan dari simpul 17 ke simpul 18
x17.44	Kunjungan dari simpul 17 ke simpul 44
x18.17	Kunjungan dari simpul 18 ke simpul 17
x18.44	Kunjungan dari simpul 18 ke simpul 44
x19.10	Kunjungan dari simpul 19 ke simpul 10
x19.20	Kunjungan dari simpul 19 ke simpul 20
x20.19	Kunjungan dari simpul 20 ke simpul 19
x20.21	Kunjungan dari simpul 20 ke simpul 21
x20.45	Kunjungan dari simpul 20 ke simpul 45
x21.20	Kunjungan dari simpul 21 ke simpul 20
x21.22	Kunjungan dari simpul 21 ke simpul 22
x22.23	Kunjungan dari simpul 22 ke simpul 23
x22.42	Kunjungan dari simpul 22 ke simpul 42
x23.22	Kunjungan dari simpul 23 ke simpul 22
x23.46	Kunjungan dari simpul 23 ke simpul 46
x24.23	Kunjungan dari simpul 24 ke simpul 23

Variabel	Keterangan
x25.24	Kunjungan dari simpul 25 ke simpul 24
x25.26	Kunjungan dari simpul 25 ke simpul 26
x25.44	Kunjungan dari simpul 25 ke simpul 44
x26.27	Kunjungan dari simpul 26 ke simpul 27
x27.28	Kunjungan dari simpul 27 ke simpul 28
x27.29	Kunjungan dari simpul 27 ke simpul 29
x28.31	Kunjungan dari simpul 28 ke simpul 31
x29.27	Kunjungan dari simpul 29 ke simpul 27
x29.32	Kunjungan dari simpul 29 ke simpul 32
x29.33	Kunjungan dari simpul 29 ke simpul 33
x30.25	Kunjungan dari simpul 30 ke simpul 25
x30.31	Kunjungan dari simpul 30 ke simpul 31
x31.30	Kunjungan dari simpul 31 ke simpul 30
x31.32	Kunjungan dari simpul 31 ke simpul 32
x31.41	Kunjungan dari simpul 31 ke simpul 41
x32.31	Kunjungan dari simpul 32 ke simpul 31
x32.34	Kunjungan dari simpul 32 ke simpul 34
x33.25	Kunjungan dari simpul 33 ke simpul 25
x33.26	Kunjungan dari simpul 33 ke simpul 26
x33.29	Kunjungan dari simpul 33 ke simpul 29
x33.34	Kunjungan dari simpul 33 ke simpul 34
x33.45	Kunjungan dari simpul 33 ke simpul 45
x34.32	Kunjungan dari simpul 34 ke simpul 32
x34.33	Kunjungan dari simpul 34 ke simpul 33
x34.39	Kunjungan dari simpul 34 ke simpul 39
x34.48	Kunjungan dari simpul 34 ke simpul 48
x35.17	Kunjungan dari simpul 35 ke simpul 17
x35.30	Kunjungan dari simpul 35 ke simpul 30
x35.41	Kunjungan dari simpul 35 ke simpul 41

C-4

Variabel	Keterangan
x35.44	Kunjungan dari simpul 35 ke simpul 44
x36.38	Kunjungan dari simpul 36 ke simpul 38
x37.18	Kunjungan dari simpul 37 ke simpul 18
x37.35	Kunjungan dari simpul 37 ke simpul 35
x38.37	Kunjungan dari simpul 38 ke simpul 37
x38.39	Kunjungan dari simpul 38 ke simpul 39
x39.34	Kunjungan dari simpul 39 ke simpul 34
x39.38	Kunjungan dari simpul 39 ke simpul 38
x39.40	Kunjungan dari simpul 39 ke simpul 40
x40.39	Kunjungan dari simpul 40 ke simpul 39
x40.47	Kunjungan dari simpul 40 ke simpul 47
x41.31	Kunjungan dari simpul 41 ke simpul 31
x41.35	Kunjungan dari simpul 41 ke simpul 35
x41.36	Kunjungan dari simpul 41 ke simpul 36
x42.21	Kunjungan dari simpul 42 ke simpul 21
x42.43	Kunjungan dari simpul 42 ke simpul 43
x43.13	Kunjungan dari simpul 43 ke simpul 13
x43.14	Kunjungan dari simpul 43 ke simpul 14
x43.25	Kunjungan dari simpul 43 ke simpul 25
x43.42	Kunjungan dari simpul 43 ke simpul 42
x43.44	Kunjungan dari simpul 43 ke simpul 44
x44.16	Kunjungan dari simpul 44 ke simpul 16
x44.17	Kunjungan dari simpul 44 ke simpul 17
x44.25	Kunjungan dari simpul 44 ke simpul 25
x44.35	Kunjungan dari simpul 44 ke simpul 35
x44.43	Kunjungan dari simpul 44 ke simpul 43
x45.20	Kunjungan dari simpul 45 ke simpul 20
x45.33	Kunjungan dari simpul 45 ke simpul 33
x45.46	Kunjungan dari simpul 45 ke simpul 46

Variabel	Keterangan
x46.25	Kunjungan dari simpul 46 ke simpul 25
x46.26	Kunjungan dari simpul 46 ke simpul 26
x46.33	Kunjungan dari simpul 46 ke simpul 33
x46.45	Kunjungan dari simpul 46 ke simpul 45
x47.40	Kunjungan dari simpul 47 ke simpul 40
x48.34	Kunjungan dari simpul 48 ke simpul 34

“Halaman ini sengaja dikosongkan”

LAMPIRAN D: MATRIKS JARAK KESELURUHAN

Berikut merupakan tabel matriks jarak keseluruhan yang dihasilkan dengan cara menjalankan Algoritma Dijkstra. Merujuk pada Tabel D.1.

Tabel D.1 Matriks Jarak Keseluruhan

Simpul	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	6	9	11	13	26	13	14	16	18	20	24	31	32	33	36	40	41	19	21	23	31	32	0
2	11	0	3	5	7	20	7	8	10	12	14	18	25	26	27	30	34	35	13	15	17	25	26	11
3	14	3	0	2	4	17	4	5	7	9	11	15	22	23	24	27	31	32	10	12	14	22	23	14
4	21	10	7	0	2	15	2	3	5	7	9	13	20	21	22	25	29	30	8	10	12	20	21	21
5	22	11	8	1	0	13	3	4	6	8	10	14	21	22	23	26	30	31	9	11	13	21	22	22
6	37	26	23	16	15	0	18	19	19	17	15	10	17	18	19	22	29	30	18	20	22	28	27	37
7	24	13	10	3	2	15	0	1	3	5	7	11	18	19	20	23	27	28	6	8	10	18	19	24
8	25	14	11	4	3	16	1	0	2	4	6	10	17	18	19	22	26	27	5	7	9	17	18	25
9	27	16	13	6	5	18	3	2	0	2	4	8	15	16	17	20	24	25	3	5	7	15	16	27
10	29	18	15	8	7	16	5	4	2	0	2	6	13	14	15	18	22	23	1	3	5	13	14	29
11	31	20	17	10	9	14	7	6	4	2	0	4	11	12	13	16	23	24	3	5	7	15	16	31
12	36	25	22	15	14	10	12	11	9	7	5	0	7	8	9	12	19	20	8	10	12	18	17	36
13	45	34	31	24	23	19	21	20	18	16	14	9	0	1	2	5	12	13	15	13	11	11	10	45
14	46	35	32	25	24	20	22	21	19	17	15	10	1	0	1	4	11	12	16	14	12	12	11	46
15	47	36	33	26	25	21	23	22	20	18	16	11	2	1	0	3	10	11	17	15	13	13	12	47
16	50	39	36	29	28	24	26	25	23	21	19	14	5	4	3	0	7	8	20	18	16	12	11	50
17	54	43	40	33	32	31	30	29	27	25	26	21	12	11	10	7	0	1	24	22	20	11	10	54
18	55	44	41	34	33	32	31	30	28	26	27	22	13	12	11	8	1	0	25	23	21	12	11	55
19	30	19	16	9	8	17	6	5	3	1	3	7	14	15	16	19	21	22	0	2	4	12	13	30
20	32	21	18	11	10	19	8	7	5	3	5	9	15	14	15	18	19	20	2	0	2	10	11	32
21	34	23	20	13	12	21	10	9	7	5	7	11	13	12	13	16	18	19	4	2	0	8	9	34
22	43	32	29	22	21	24	19	18	16	14	16	14	5	4	5	8	10	11	13	11	9	0	1	43
23	44	33	30	23	22	25	20	19	17	15	17	15	6	5	6	9	11	12	14	12	10	1	0	44
24	45	34	31	24	23	26	21	20	18	16	18	16	7	6	7	10	12	13	15	13	11	2	1	45
25	46	35	32	25	24	27	22	21	19	17	19	17	8	7	8	7	6	7	16	14	12	3	2	46
26	47	36	33	26	25	34	23	22	20	18	20	24	15	14	15	14	7	8	17	15	17	10	9	47
27	46	35	32	25	24	33	22	21	19	17	19	23	14	13	14	13	6	7	16	14	16	9	8	46
28	51	40	37	30	29	32	27	26	24	22	24	22	13	12	13	12	5	6	21	19	17	8	7	51
29	45	34	31	24	23	32	21	20	18	16	18	22	14	13	14	13	6	7	15	13	15	9	8	45
30	49	38	35	28	27	30	25	24	22	20	22	20	11	10	11	10	5	6	19	17	15	6	5	49
31	50	39	36	29	28	31	26	25	23	21	23	21	12	11	12	11	4	5	20	18	16	7	6	50
32	50	39	36	29	28	32	26	25	23	21	23	22	13	12	13	12	5	6	20	18	17	8	7	50
33	43	32	29	22	21	30	19	18	16	14	16	20	12	11	12	11	8	9	13	11	13	7	6	43
34	47	36	33	26	25	34	23	22	20	18	20	24	16	15	16	15	9	10	17	15	17	11	10	47
35	51	40	37	30	29	32	27	26	24	22	24	22	13	12	12	9	2	3	21	19	17	8	7	51
36	55	44	41	34	33	38	31	30	28	26	28	28	19	18	17	14	7	6	25	23	23	14	13	55
37	53	42	39	32	31	34	29	28	26	24	26	24	15	14	13	10	3	2	23	21	19	10	9	53
38	53	42	39	32	31	36	29	28	26	24	26	26	17	16	15	12	5	4	23	21	21	12	11	53
39	52	41	38	31	30	37	28	27	25	23	25	27	18	17	16	13	6	5	22	20	22	13	12	52
40	55	44	41	34	33	40	31	30	28	26	28	30	21	20	19	16	9	8	25	23	25	16	15	55
41	51	40	37	30	29	32	27	26	24	22	24	22	13	12	13	10	3	4	21	19	17	8	7	51
42	42	31	28	21	20	23	18	17	15	13	15	13	4	3	4	7	9	10	12	10	8	8	7	42
43	46	35	32	25	24	22	22	21	19	17	17	12	3	2	3	6	8	9	16	14	12	7	6	46
44	51	40	37	30	29	28	27	26	24	22	23	18	9	8	7	4	3	4	21	19	17	8	7	51
45	40	29	26	19	18	27	16	15	13	11	13	17	14	13	14	13	11	12	10	8	10	9	8	40
46	44	33	30	23	22	29	20	19	17	15	17	19	10	9	10	9	8	9	14	12	14	5	4	44
47	62	51	48	41	40	47	38	37	35	33	35	37	28	27	26	23	16	15	32	30	32	23	22	62
48	63	52	49	42	41	50	39	38	36	34	36	40	32	31	32	31	25	26	33	31	33	27	26	25

“Halaman ini sengaja dikosongkan”

Simpul	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
1	35	34	35	36	34	37	36	35	32	36	38	38	42	40	41	44	37	32	33	38	29	33	51	53
2	29	28	29	30	28	31	30	29	26	30	32	32	36	34	35	38	31	26	27	32	23	27	45	47
3	26	25	26	27	25	28	27	26	23	27	29	29	33	31	32	35	28	23	24	29	20	24	42	44
4	24	23	24	25	23	26	25	24	21	25	27	27	31	29	30	33	26	21	22	27	18	22	40	42
5	25	24	25	26	24	27	26	25	22	26	28	28	32	30	31	34	27	22	23	28	19	23	41	43
6	25	26	27	28	28	30	29	29	30	32	31	31	35	33	34	37	30	20	21	26	28	30	44	49
7	22	21	22	23	21	24	23	22	19	23	25	25	29	27	28	31	24	19	20	25	16	20	38	40
8	21	20	21	22	20	23	22	21	18	22	24	24	28	26	27	30	23	18	19	24	15	19	37	39
9	19	18	19	20	18	21	20	19	16	20	22	22	26	24	25	28	21	16	17	22	13	17	35	37
10	17	16	17	18	16	19	18	17	14	18	20	20	24	22	23	26	19	14	15	20	11	15	33	35
11	19	18	19	20	18	21	20	19	16	20	22	22	26	24	25	28	21	14	15	20	13	17	35	37
12	15	16	17	18	18	20	19	19	20	22	21	21	25	23	24	27	20	10	11	16	18	20	34	39
13	8	9	10	11	11	13	12	12	13	15	14	14	18	16	17	20	13	3	4	9	16	13	27	32
14	9	10	11	12	12	14	13	13	14	16	14	15	19	17	18	21	14	4	5	8	17	14	28	33
15	10	11	12	13	13	15	14	14	15	17	13	15	19	17	18	21	14	5	6	7	18	15	28	34
16	9	10	11	12	12	12	12	13	14	16	10	12	16	14	15	18	11	8	9	4	17	14	25	33
17	8	9	10	11	11	11	11	12	13	15	9	11	15	13	14	17	10	12	9	3	16	13	24	32
18	9	10	11	12	12	12	12	13	14	16	10	12	16	14	15	18	11	13	10	4	17	14	25	33
19	16	15	16	17	15	18	17	16	13	17	19	19	23	21	22	25	18	13	14	19	10	14	32	34
20	14	13	14	15	13	16	15	14	11	15	17	17	21	19	20	23	16	11	12	17	8	12	30	32
21	14	13	14	15	15	17	16	16	13	17	18	18	22	20	21	24	17	9	10	15	10	12	31	34
22	6	5	6	7	7	9	8	8	8	11	10	10	14	12	13	16	9	1	2	7	8	4	23	28
23	5	4	5	6	6	8	7	7	7	10	9	9	13	11	12	15	8	2	3	8	7	3	22	27
24	6	5	6	7	7	9	8	8	8	11	10	10	14	12	13	16	9	3	4	9	8	4	23	28
25	0	1	2	3	3	5	4	4	5	7	6	6	10	8	9	12	5	4	5	3	8	5	19	24
26	7	0	1	2	2	4	3	3	4	6	5	5	9	7	8	11	4	11	12	10	7	11	18	23
27	6	6	0	1	1	3	2	2	3	5	4	4	8	6	7	10	3	10	11	9	6	10	17	22
28	5	6	7	0	8	2	1	3	10	6	3	3	7	5	6	9	2	9	10	8	13	10	16	23
29	6	5	1	2	0	3	2	1	2	4	4	4	8	6	7	10	3	10	11	9	5	9	17	21
30	3	4	5	6	6	0	1	3	8	6	3	3	7	5	6	9	2	7	8	6	11	8	16	23
31	4	5	6	7	7	1	0	2	9	5	2	2	6	4	5	8	1	8	9	7	12	9	15	22
32	5	6	7	8	8	2	1	0	7	3	3	3	7	5	6	9	2	9	10	8	10	10	16	20
33	4	3	3	4	2	5	4	3	0	4	6	6	10	8	9	12	5	8	9	7	3	7	19	21
34	8	7	7	8	6	6	5	4	4	0	7	7	9	7	6	9	6	12	13	11	7	11	16	17
35	5	6	7	8	8	2	2	4	10	7	0	2	6	4	5	8	1	9	10	5	13	10	15	24
36	11	12	13	14	14	8	8	10	12	8	6	0	4	2	3	6	7	15	16	10	15	16	13	25
37	7	8	9	10	10	4	4	6	12	9	2	4	0	6	7	10	3	11	12	6	15	12	17	26
38	9	10	11	12	12	6	6	8	10	6	4	6	2	0	1	4	5	13	14	8	13	14	11	23
39	10	11	12	13	11	7	7	9	9	5	5	7	3	1	0	3	6	14	15	9	12	15	10	22
40	13	14	15	16	14	10	10	12	12	8	8	10	6	4	3	0	9	17	18	12	15	18	7	25
41	5	6	7	8	8	2	1	3	10	6	1	1	5	3	4	7	0	9	10	6	13	10	14	23
42	5	6	7	8	8	10	9	9	10	12	11	11	15	13	14	17	10	0	1	6	13	10	24	29
43	4	5	6	7	7	9	8	8	9	11	10	10	14	12	13	16	9	4	0	5	12	9	23	28
44	5	6	7	8	8	8	8	9	10	12	6	8	12	10	11	14	7	9	6	0	13	10	21	29
45	6	5	6	7	5	8	7	6	3	7	9	9	13	11	12	15	8	10	11	9	0	4	22	24
46	2	1	2	3	3	5	4	4	4	7	6	6	10	8	9	12	5	6	7	5	4	0	19	24
47	20	21	22	23	21	17	17	19	19	15	15	17	13	11	10	7	16	24	25	19	22	25	0	32
48	24	23	23	24	22	22	21	20	20	16	23	23	25	23	22	25	22	28	29	27	23	27	32	0

“Halaman ini sengaja dikosongkan”

LAMPIRAN E: MODEL JEJARING

Lampiran ini merupakan gambar model jejaring yang digunakan merujuk pada Gambar E.1. Untuk lebih detilnya, gambar dapat diunduh di http://bit.ly/modjejaring2019

Gambar E.1 Model Jejaring