TUGAS AKHIR – KS 141501

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

ADVANCED TRAVELER INFORMATION SYSTEM: ITINERARY OPTIMISATION USING ORIENTEERING PROBLEM MODEL AND ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (CASE STUDY: PUBLIC TRANSPORTATION IN SURABAYA)

JOCKEY SATRIA WIJAYA
NRP 5213 100 177

Dosen Pembimbing :
Wiwik Anggraeni, S.Si, M.Kom
Ahmad Mukhlason, S.Kom, M.Sc, Ph.D

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KS 141501

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

JOCKEY SATRIA WIJAYA
NRP 5213 100 177

Dosen Pembimbing :
Wiwik Anggraeni, S.Si, M.Kom
Ahmad Mukhlason, S.Kom, M.Sc, Ph.D

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

FINAL PROJECT – KS 141501

ADVANCED TRAVELER INFORMATION SYSTEM: ITINERARY OPTIMISATION USING ORIENTEERING PROBLEM MODEL AND ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (CASE STUDY: PUBLIC TRANSPORTATION IN SURABAYA)

JOCKEY SATRIA WIJAYA
NRP 5213 100 177

SUPERVISOR:
Wiwik Anggraeni, S.Si, M.Kom
Ahmad Mukhlason, S.Kom, M.Sc, Ph.D

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

LEMBAR PENGESAHAN

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

TUGAS AKHIR

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

Oleh:

JOCKEY SATRIA WIJAYA
NRP. 5213 100 177

Surabaya, Juli 2017

KEPALA DEPARTEMEN SISTEM INFORMASI

Dr. Ir. Aris Tjahyanto, M.Kom
NIP.19650310 199102 1 001

LEMBAR PERSETUJUAN

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

TUGAS AKHIR

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

JOCKEY SATRIA WIJAYA
NRP. 5213 100 177

Disetujui Tim Penguji :Tanggal Ujian : 06 Juli 2017
Periode Wisuda : September 2017

Wiwik Anggraeni, S.Si, M.Kom	**(Pembimbing I)**
Ahmad Mukhlason, S.Kom, M.Sc, Ph.D	**(Pembimbing II)**
Edwin Riksakomara, S.Kom, M.T.	**(Penguji I)**
Radityo Prasetianto W., S.Kom, M.Kom	**(Penguji II)**

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

Nama Mahasiswa	**: Jockey Satria Wijaya**
NRP	**: 5213 100 177**
Departemen	**: Sistem Informasi FTIf-ITS**
Pembimbing 1	**: Wiwik Anggraeni, S.Si, M.Kom**
Pembimbing 2	**: Ahmad Mukhlason, S.Kom, M.Sc, Ph.D**

ABSTRAK

Kemacetan lalu lintas merupakan permasalahan yang menjadi perhatian pemerintah di kota – kota besar seperti Surabaya. Berbagai upaya telah dilakukan pemerintah Kota Surabaya untuk mengurangi intensitas kemacetan yang terjadi, mulai dari berbagai kebijakan seperti jalur khusus sepeda motor, plat ganjil – genap, dan sebagainya. Namun hal tersebut masih dirasa belum mampu mengurangi alur lalu lintas padat di kota besar. Faktor penentu terjadinya kemacetan adalah volume kendaraan pribadi yang terlalu tinggi. Karena itu pemerintah Kota Surabaya telah menyediakan transportasi umum seperti angkot yang saat ini merupakan transportasi umum utama di Kota Surabaya.Untuk mendukung program pemerintah Kota Surabaya, sekaligus mengembangkan Surabaya Intelligent Transport System, penelitian ini mengusulkan pembuatan model matematika untuk mengetahui jalur alternatif optimal, saat masyarakat ingin bepergian menggunakan angkot.

Pembuatan model matematika dilakukan dengan metode Orienteering Problem dan dengan kode trayek angkot JK, F, JMK, I, RT, GS yang didapat dari website Dinas Perhubungan Kota Surbaya. Data jarak dan waktu tempuh tiap tempat yang dikunjungi angkot di dapat dari Google Maps. Model yang telah

dibentuk dengan Orienteering Problem, akan dicari solusi model optimal dengan menggunakan Iterative Local Search.

Dalam pencarian solusi model, perlu dibentuk sebuah data set untuk membantu pencarian solusi dengan cepat dan tepat. Data set pada penelitian ini berbentuk matriks yang berisikan waktu tempuh terpendek pada seluruh node. Algoritma Dijkstra digunakan untuk membantu konstruksi data set.

Penelitian ini menghasilkan network model dari ke 6 rute lengkap dengan lokasi tiap node, waktu tempuh, dan juga jarak antar node. Selain itu penelitian ini juga menghasilkan model matematika, algoritma penyelesaian model dengan menggunakan program java, serta data set waktu tempuh untuk ke 6 rute yang dapat digunakan untuk pengembangan selanjutnya.

Hasil dari pencarian solusi model OP ini menunjukkan Orienteering Problem dapat digunakan untuk memodelkan trayek angkot Surabaya. Penggunaan Roulette Wheel Selection dalam seleksi solusi pada penelitian ini menunjukkan semakin banyak solusi yang dipilih, maka kemungkinan algoritma menemukan solusi terbaik semakin tinggi. Algoritma Iterative Local Search yang digunakan untuk mencari solusi OP didasarkan pada operasi pencarian lokal yaitu swap, insertion, dan deletion. Penggabungan ketiga metode pencarian lokal tersebut menghasilkan solusi yang lebih baik apabila dibandingkan dengan hanya menggunakan salah satu metode pencarian lokal.

Secara umum algoritma Iterative Local Search dapat menyelesaikan permasalahan OP dengan cepat dan efisien. Dengan adanya penelitian ini diharapkan dapat mengembangkan Surabaya Intelligent Transport System sekaligus membantu pemerintah Kota Surabaya dalam rencana revitalisasi angkot di Surabaya.

Kata Kunci: ***Orienteering Problem, Iterative Local Search, Transportasi Umum, Optimasi, Pemodelan***

ADVANCED TRAVELER INFORMATION SYSTEM: ITINERARY OPTIMISATION USING ORIENTEERING PROBLEM MODEL AND ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (CASE STUDY: PUBLIC TRANSPORTATION IN SURABAYA)

Student Name	**: Jockey Satria Wijaya**
NRP	**: 5213 100 177**
Department	**: Sistem Informasi FTIf-ITS**
Supervisor 1	**: Wiwik Anggraeni, S.Si, M.Kom**
Supervisor 2	**: Ahmad Mukhlason, S.Kom, M.Sc, Ph.D**

ABSTRACT

Traffic congestion is a problem that becomes government concern in a big city like Surabaya. Many things have been done by the government to decrease the intensity of traffic congestion, ranging from various policy like special lane for motorcycle, odd – even license plate, etc. However, it is still considered not be able to decrease the heavy traffic in a big city. The determinants of traffic congestion is the volume of private vehicle is too high. Because of it, Surabaya's government has provided city transport like angkot. In the mean time, angkot is one of the major public transportation in Surabaya. To support government's program, while developing Surabaya Intelligent Transport System, this research propose a mathematical modeling to find out the optimal alternative path, when people want to go somewhere using angkot.

The making of mathematical models is done with Orienteering Problem method. We use angkot's route with code JK, F, JMK, O, RT, and GS obtained from Dinas Perhubungan Kota Surabaya's website. While the distance and travelling time data obtained from Google Maps. The created model, will be searched for optimal model solutions using Iterative Local Search.

In the search for model solutions, we need a data set to help us find the solution with high accuracy but also fast. The data set will be in the form of matrix that contain fastest travelling time

on all nodes. Dijkstra's Algorithm is used to help create such data set.

This research produce a network model from the 6 angkot's route complete with the location of every nodes, travelling time, and distance between node. Furthermore, this research also produce mathematical model, algorithm to solve model using Java, and data set that contain travel time on all nodes that can be used in next development.

The search result for an OP solution shows that Orienteering Problem can be use to model angkot's route in Surabaya. The use of Roulette Wheel Selection in this research indicates that the more solutions are selected, then the algorithm is more likely to find the most optimal solution. The Iterative Local Search algorithm used to find OP solution is based on several local search operations namely swap, insertion, and deletion. The best solution found when the three methods integrated rather than used individually.

Generally speaking, Iterative Local Search algorithm can solve OP problem fast and effectively. The existence of this research is expected to further develop Surabaya Intelligent Transport System while helping the government in their plan to revitalize angkot in Surabaya.

Keywords: ***Orienteering Problem, Iterative Local Search, Public Transport, Optimation, Modeling***

KATA PENGANTAR

Puji syukur pada Tuhan Yang Maha Esa selama ini sehingga penulis mendapatkan kelancaran dalam menyelesaikan tugas akhir dengan judul :

ADVANCED TRAVELER INFORMATION SYSTEM: OPTIMASI RENCANA PERJALANAN DENGAN ORIENTEERING PROBLEM MODEL DAN ITERATIVE LOCAL SEARCH WITH HILL CLIMBING (STUDI KASUS: TRAYEK ANGKOT KOTA SURABAYA)

yang merupakan salah satu syarat kelulusan pada Jurusan Sistem Informasi, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Terima kasih atas pihak-pihak yang telah mendukung, memberikan saran, motivasi, semangat, dan bantuan baik berupa materiil maupun moril demi tercapainya tujuan pembuatan tugas akhir ini. Secara khusus penulis akan menyampaikan ucapan terima kasih yang sedalam-dalamnya kepada:

1) Kedua orang tua dan kakak penulis, Bapak I Made Jaya Wijaya, Ibu Luh Gde Murwadhi, dan Jodie Pratama Wijaya yang telah memberikan motivasi, semangat, dan doa sehingga penulis mampu menyelesaikan pendidikan S1 ini dengan baik.
2) Ibu Wiwik Anggraeni, S.Si, M.Kom. serta Pak Ahmad Mukhlason, S.Kom, M.Sc, Ph.D selaku dosen pembimbing yang memberikan ilmu, petunjuk, dan motivasi untuk kelancaran Tugas Akhir ini.
3) Bapak Bekti Cahyo Hidayanto, S.Si, M.Kom. sebagai dosen wali penulis selama menempuh pendidikan di Jurusan Sistem Informasi yang telah membantu penulis selama perkuliahan.
4) Bapak Edwin Riksa Komara, S.Kom., M.T. serta Bapak Radityo Prasetianto W., S.Kom., M.Kom. selaku dosen

penguji yang telah memberikan kritik, saran, dan masukan yang dapat menyempurnakan Tugas Akhir ini.

5) Seluruh dosen pengajar beserta staf dan karyawan di Jurusan Sistem Informasi, FTIF ITS Surabaya yang telah memberikan ilmu dan bantuan kepada penulis selama ini.

6) Tim optimasi trayek angkot Surabaya Aka Yoga, dan Dhamar Bagas yang selalu bersama dalam suka, duka dan saling membantu dalam pengerjaan tugas akhir ini.

7) Sahabat – sahabat dekat penulis Gayatri Grace, Steven Kurniawan, Dessy Puspita, Kusnanta, Caesar Fajriansah, Pandu Satrio, Denny Angga, Rizky Riyandhi, Chitra Utami, Oriehanna, Garini Anindya, Hisyam Naufal, Fitri Larasati, Egan, Valliant, Maulana, Rizza Firmansyah, Agung, Tetha, Niko, dan Asvin yang senantiasa menjadi sumber semangat penulis dan selalu berbagi canda dan tawa dengan penulis selama pengerjaan tugas akhir ini.

8) Teman – teman dan senior JSI BELTRANIS, SOLARIS, BASILISK, dan OSIRIS atas seluruh bantuan yang diberikan ketikan penulis berkuliah di Sistem Informasi ITS.

9) Rekan – rekan penulis di organisasi HMSI ITS, dan TPKB ITS, dan juga rekan kepanitian Information Systems Expo 2015 yang telah banyak memberikan pelajaran dan pengalaman serta membantu penulis selama menjabat.

10) Serta semua pihak yang telah membantu dalam pengerjaan Tugas Akhir ini yang belum mampu penulis sebutkan diatas.

Terima kasih atas segala bantuan, dukungan, serta doanya. Semoga Tuhan Yang Maha Esa senantiasa melimpahkan anugerah serta membalas kebaikan yang telah diberikan kepada penulis.

Surabaya, Juni 2017

Penulis

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

KODE

Halaman ini sengaja dikosongkan

DAFTAR TABEL

BAB I
PENDAHULUAN

Pada bab ini akan dibahas mengenai identifikasi permasalahan penelitian meliptui latar belakang masalah, perumusan masalah, batasan masaah, tujuan tugas akhir, manfaat tugas akhir, dan relevansi terhadap pengerjaan tugas akhir ini. Berdasarkan uraian dalam bab ini, diharapkan dapat memberikan gambaran mengenai permasalahan yang dibahas serta dapat memahami tujuan dan manfaat tugas akhir ini.

1.1. Latar Belakang Masalah

Kemacetan lalu lintas menjadi problematika ibu kota dan kota – kota besar di beberapa negara di dunia. Salah satunya adalah Jakarta dan Surabaya. Dimana menurut Index Stop-Start Castrol-Magnatec yang dipublikasikan oleh perusahaan oli asal Inggris Castrol, Jakarta menduduki peringkat pertama kemacetan terburuk diikuti Istanbul dan Mexico City. Surabaya sendiri menempati posisi ke 4 sebagai kota dengan kemacetan terburuk berdasarkan index dari Castrol tersebut [1]. Terdapat banyak faktor terjadinya kemacetan di kota – kota besar. Faktor – faktor tersebut antara lain jumlah penduduk yang padat, lalu lintas yang tidak tertata, kualitas jalan yang tidak baik sehingga memunculkan banyak lubang, dan lain – lain. Untuk jumlah penduduk sendiri, Jakarta menempati peringkat pertama di Indonesia dengan jumlah penduduk sebanyak 9.988.495 jiwa, kemudian diikuti Surabaya dengan jumlah penduduk sebanyak 2.863.059 jiwa [2] [3].

Permasalahan kemacetan lalu lintas menjadi permasalahan yang menjadi perhatian bagi kota Jakarta dan Surabaya. Selain kemacetan menyebabkan waktu seseorang terbuang di jalan, kemacetan juga memunculkan polusi karbon yang sangat besar. Daerah perkotaan yang jalanannya dipenuhi kendaraan bermotor menghasilkan banyaknya gas buangan knalpot termasuk Greenhouse Gases (GHGs) atau yang biasa dikenal Gas Rumah Kaca (GRK), dan juga polusi yang dapat menurunkan kualitas udara [4]. Bahkan tidak menutup

kemungkinan adanya kematian yang disebabkan kemacetan panjang. Menurut laporan kepala petugas medis di Brebes, terdapat 18 orang meninggal dunia di jalan yang memiliki lalu lintas padat. 12 orang diantaranya meninggal karena kelelahan, 5 orang meninggal karena kecelakaan di persimpangan jalan, dan 1 orang meninggal karena penyebab yang belum diketahui [5]. Salah satu korban kemacetan adalah anak berusia 14 tahun yang meninggal diduga karena keracunan gas karbon dioksida setelah mobil yang ia tumpangi terjebak macet selama lebih dari 6 jam [5].

Pemerintah kota telah melakukan beberapa upaya untuk mengurangi kemacetan yang terjadi di kotanya masing – masing. Upaya yang dilakukan seperti pembangunan tol dalam kota, penerapan peraturan plat nomor ganjil dan genap, pelarangan jalur sepeda motor, penambahan ruas jalan, dan lain – lain [6]. Namun hal tersebut masih dirasa kurang membantu mengurangi kemacetan yang terjadi di kota – kota besar. Pemerintah harus dapat menciptakan kebijakan transportasi perkotaan yang komprehensif, dan juga mengoperasikan sarana angkutan umum yang tepat kapasitas [7].

Dalam mengantisipasi dan mengurangi kemacetan, pemerintah Kota Surabaya telah melakukan penambahan ruas jalan, pengaturan lalu lintas yang lebih tertib, dan peningkatan sarana transportasi umum seperti pembangunan *Mass Rapid Transit* (MRT), dan adanya rencana revitalisasi angkot [8]. Peningkatan sarana transportasi umum merupakan upaya yang tepat untuk mengurangi kemacetan karena salah satu faktor utama kemacetan adalah banyaknya volume kendaraan pribadi, maka dari itu salah satu upaya yang digencarkan pemerintah Kota Surabaya untuk mengurangi kemacetan adalah dengan mengadakan dan terus meningkatkan moda transportasi umum.

Namun dengan adanya moda transportasi umum saja, tidak dapat menarik perhatian masyarakat untuk beralih menggunakan transportasi umum dibandingkan dengan kendaraan pribadi. Dalam menarik minat masyarakat,

pemerintah perlu menyediakan transportasi umum yang nyaman, cepat, murah, tepat waktu dan terjadwal. Kota Surabaya sendiri telah memiliki *Surabaya Intelligent Transport System* (SITS) yang bertujuan untuk meningkatkan kelancaran dan keselamatan lalu lintas [9].

Penerapan ITS di Surabaya masih pada tahap pemantauan kondisi kemacetan lalu lintas secara real-time dengan menggunakan CCTV yang telah terpasang di jalan – jalan Kota Surabaya. Penelitian ini bertujuan memberikan masukan berupa model yang dapat digunakan untuk mengembangkan SITS ke depannya. Berdasarkan tujuan tersebut, maka pemodelan optimasi dilakukan dengan menggunakan metode *Orienteering Problem*.

Orienteering Problem (OP) merupakan gabungan dari pemilihan node dan menentukan jalur optimal dari tiap node yang telah dipilih. Sehingga, OP dapat dilihat sebagai kombinasi antara *Knapsack Problem* (KP) dengan *Travelling Salesman Problem* (TSP). Dalam TSP, kita mencari jalur optimum untuk mengunjungi semua node, sedangkan dalam OP kita tidak perlu mengunjungi semua node yang ada sebelum mencapai tujuan [10]. Penelitian yang dilakukan Vansteenwegen et al. (2011) menyatakan aplikasi praktikal yang kompleks dapat dimodelkan sebagai *Orienteering Problem* atau varian dari *Orienteering Problem*. Contohnya seperti pembuatan model untuk aplikasi wisatawan (*tourism application*), dimana para wisatawan tidak mungkin dapat mengunjungi semua tempat wisata yang disukai karena keterbatasan waktu [10]. Dalam kasus transportasi ini dirasa cocok menggunakan OP karena karakteristiknya yang mencari jalur optimum berdasarkan jalur yang telah ada tanpa perlu mengunjungi seluruh node.

Sejak tahun 1980, beberapa solusi model OP telah dikembangkan dan diaplikasikan, terutama pada *routing* dan *tourism* [11]. Salah satu metode optimasi yang dapat digunakan untuk mencari solusi model OP adalah *Iterative Local Search.*

Dengan menggunakan *Iterative Local Search*, pencarian solusi dapat dilakukan dengan cepat dan efisien [12].

Namun tidak banyak referensi atau penelitian mengenai OP dan pengembangan solusi dalam bidang optimasi transportasi umum. Dengan adanya penelitian ini diharapkan mampu membantu pemerintah kota Surabaya mengembangkan SITS dengan masukkan berupa model, dan juga menjadi refrensi tambahan terkait OP bidang optimasi transportasi umum.

1.2. Perumusan Masalah

Berdasarkan latar belakang tersebut, adapun rumusan masalah dalam tugas akhir ini antara lain:

1. Seperti apa bentuk model rekomendasi jalur optimum dengan peluang jumlah angkot terbesar?
2. Bagaimana model *Orienteering Problem* dapat menyelesaikan permasalahan penentuan jalur alternatif paling optimum pada trayek angkot Surabaya?
3. Bagaimana menyelesaikan model yang telah dibuat dengan melakukan *Iterative Local Search with Hill Climbing*?

1.3. Batasan Masalah

Dari perumusan masalah yang telah dipaparkan sebelumnya, maka yang menjadi batasan dalam tugas akhir ini adalah sebagai berikut:

1. Objek dari penelitian dalam Tugas Akhir ini adalah transportasi umum yaitu angkot/mikrolet yang berada di Kota Surabaya.
2. Luaran dari penelitian ini ditujukan kepada masyarakat dan pengunjung Kota Surabaya, khususnya yang ingin berkeliling mengunjungi banyak tempat di Kota Surabaya.
3. Data menggunakan data trayek mikrolet Surabaya pada jalur Joyoboyo – Kenjeran (JK), Kenjeran –

Kalimas Barat (JMK), Endrosono – Joyoboyo (F), Kalimas Barat – Keputih (O), Rungkut – Pasar Turi (RT), Gunung Anyar – Sidorame (GS).

4. Waktu tempuh tiap node pada masing – masing trayek di dapat dari informasi waktu pada Google Maps, dimana node merepresentasikan jalan yang dikunjungi angkot.
5. Dalam penelitian ini tidak memperhitungkan sisi psikologis dan finansial dari penumpang.

1.4. Tujuan Penelitian

Adapun tujuan dari pengerjaan tugas akhir ini adalah:

1. Membuat model yang dapat digunakan dalam optimasi rute perjalanan dengan angkot di Kota Surabaya menggunakan *Orienteering Problem*.
2. Menghasilkan solusi dari model yang telah dibuat dengan menggunakan algoritma i*terative local search*
3. Menghasilkan data set yang dapat digunakan untuk pengembangan selanjutnya

1.5. Manfaat Penelitian

Manfaat yang diberikan dengan adanya tugas akhir ini dibagi menjadi 2 bagian, manfaat untuk keilmuan, dan manfaat untuk pemerintah Kota Surabaya. Manfaat untuk pemerintah Kota Surabaya antara lain:

1. Memberikan masukkan berupa model untuk pengembangan Surabaya Intelligent Transport System (SITS)
2. Luaran dari penelitian ini dapat dijadikan referensi untuk diterapkan pada kota – kota lain di Indonesia

Sedangkan manfaat keilmuan dari penelitian ini adalah:

1. Sebagai referensi pengerjaan penelitian terkait optimasi transportasi menggunakan metode *Orienteering Problem*

2. Data set yang dihasilkan dapat digunakan untuk penelitian selanjutnya terkait optimasi trayek yang sama

1.6. Relevansi

Tugas akhir ini diharapkan dapat digunakan sebagai alat untuk pengembangan aplikasi rekomendasi jalur angkutan umum khususnya angkot pada kota Surabaya serta diharapkan dapat menjadi sumber pustaka untuk penelitian terkait optimasi jalur transportasi dengan teknik Orienteering Problem. Tugas akhir ini berkaitan dengan banyaknya pendatang dari luar kota Surabaya yang ingin menggunakan transportasi umum namun masih belum memiliki pegetahuan tentang jalur transportasi umum yang ada di Surabaya. Dengan adanya tugas akhir ini diharapkan transportasi umum di Surabaya semakin efektif dan efisien, masyarakat dan pendatang lebih mudah bepergian menggunakan transportasi umum angkot sehingga masalah kemacetan di kota besar dapat dikurangi. Tugas Akhir ini terkait dengan Sistem Pendukung Keputusan, Riset Operasi, Sistem Cerdas, dan Riset Operasi Lanjut.

BAB II
TINJAUAN PUSTAKA

Bab ini akan menjelaskan mengenai penelitian sebelumnya dan dasar teori yang dijadikan acuan atau landasan dalam pengerjaan tugas akhir ini. Landasan teori akan memberikan gambaran secara umum dari landasan penjabaran tugas akhir ini.

2.1. Studi Sebelumnya

Penelitian yang dijadikan acuan dalam pengerjaan tugas akhir ini terdapat pada:

Tabel 2-1 Studi Literatur 1

Judul Paper	The Orienteering Problem: A Survey [10]
Penulis; Tahun	Pieter Vansteenwegen, Woutar Souffriau, Dirk Van Oudheusden; 2011
Deskripsi Umum Penelitian	Paper ini khusus membahas mengenai *Orienteering Problem*, mulai dari literature terkait, perluasan dan variasinya, dan solusi OP dan aplikasinya. Pada paper ini dibahas varian OP seperti *Orienteering Problem with Time Windows, Team Orienteering Problem*, dan *Team Orienteering Problem with Time Windows*. Setiap metode membahas mengenai definisi, formula matematika. pengaplikasian praktikal tiap metode, *benchmark instance*, dan pendekatan solusinya. Penelitian ini menyimpulkan banyak kasus sulit yang dapat dimodelkan sebagai atau varian dari *orienteering problem*
Keterkaitan Penelitian	Pada paper ini membahas mengenai OP, formula matematikanya, serta penggunaannya. Tugas akhir ini

	merupakan penelitian optimasi dengan menggunakan OP untuk pemodelan matematika.

Tabel 2-2 Studi Literatur 2

Judul Paper	Iterated Local Search for the Team Orienteering Problem with Time Windows [12]
Penulis; Tahun	Pieter Vansteenwegen, Woutar Souffriau, Greet Vanded Berghe, Dirk Van Oudheusden; 2009
Deskripsi Umum Penelitian	Penelitian ini membahas tentang penggunaan *Orienteering Problem*, dalam membantu turis merencanakan dan perjalanannya. Permasalahan tersebut dimodelkan dalam bentuk *Team Orienteering Problem with Time Windows* (TOPTW). Pada TOPTW, setiap lokasi memiliki skor, *service time* dan *time window*. Tujuan TOPTW adalah memaksimalkan skor. Kontribusi utama dari penelitian ini adalah pencarian lokal meta-heuristic yang cepat dan simple untuk menyelesaikan TOPTW. Pencarian lokal pada paper ini menggabungkan *insertion method* dan *shake step method*. Hasil penelitian menunjukkan *iterated local search* dapat menemukan solusi TOPTW dengan cepat dan efisien.
Keterkaitan Penelitian	Penelitian ini dilakukan dengan menggunakan metode optimasi *Iterative Local Search* untuk mendapatkan solusi dari OP yang dapat diterapkan dalam kasus optimasi transportasi angkutan umum di tugas akhir ini.

Tabel 2-3 Studi Literatur 3

Judul Paper	An Iterated Local Search Algorithm for Solving the Orienteering Problem with Time Windows [13]
Penulis; Tahun	Aldy Gunawan, Hoong Chuin Lau, dan Kun Lu; 2015
Deskripsi Umum Penelitian	Pada penelitian ini, dibahas mengenai *Orienteering Problem with Time Windows* (OPTW). OTWP diselesaikan dengan menggunakan *Iterated Local Search* dengan operasi pencarian lokal yang dilakukan adalah *swap*, *insertion*, dan *replace*. Paper ini juga mengimplementasikan mekanisme *acceptance criterion* dan *perturbation* untuk menyeimbangkan diversifikasi dan instensitas pencarian. Hasil dari penelitian ini menyatakan algoritma *Iterated Local Search* efektif dalam menyelesaikan permasalahan OPTW jika dibandingkan dengan *state-of-the-art algorithm* yang dibahas pada paper.
Keterkaitan Penelitian	Paper ini membahas mengenai penyelesaian OP dengan menggunakan algoritma *Iterated Local Search* yang menunjukkan algoritma cocok digunakan untuk penyelesaian solusi model OP. Tugas akhir ini juga merupakan optimasi model OP dengan menggunakan algoritma *Iterated Local Search*.

2.2. Dasar Teori

2.2.1. Kemacetan Lalu Lintas

Kemacetan lalu lintas merupakan permasalahan besar transportasi perkotaan [14] [15]. Namun, tidak ada definisi yang diterima secara universal mengenai kemacetan lalu lintas

itu sendiri. Penelitian yang dilakukan Aftabuzzaman [16] merangkum beberapa definisi mengenai kemacetan lalu lintas yang dibagi menjadi tiga kategori: Definisi terkait kapasitas permintaan, definisi terkait waktu tempuh, dan definisi terkait biaya:

1. Terkait permintaan, kemacetan dapat dikatakan sebagai suatu kondisi dimana kendaraan yang menggunakan jalan di waktu tertentu melampaui kemampuan jalan raya menampung kendaraan tersebut [17].
2. Terkait waktu tempuh, kemacetan dapat terjadi apabila adanya ketidakseimbangan antara traffic flow dan kapasitas jalan yang menyebabkan alur lalu lintas melambat sehingga waktu tempuh meningkat [18].
3. Terkait biaya, kemacetan mengacu pada bertambahnya biaya yang dihasilkan dari gangguan tiap pengguna jalan [16].

Berdasarkan poin – poin di atas, dapat disimpulkan bahwa yang dimaksud dengan kemacetan lalu lintas adalah kondisi dimana volume kendaraan melebihi kapasitas jalan yang disediakan sehingga menyebabkan bertambahnya waktu tempuh dan biaya.

2.2.2. Optimasi

Optimasi merupakan salah satu ilmu matematika yang berfokus mendapatkan nilai minimum atau maksimum secara sistematis dari suatu fungsi, peluang, maupun pencarian nilai lainnya dalam berbagai kasus [19]. Optimasi adalah tugas yang dasar dan sering diaplikasikan untuk aktivitas teknik. Namun, dalam banyak kasus, tugas – tugas dilakukan dengan *trial and error*. Untuk menghindari aktivitas yang membosankan tersebut, peneliti mengambil pendekatan sistematis yang seefisien mungkin dan juga menyediakan jaminan bahwa solusi yang lebih baik tidak dapat ditemukan lagi [20].

Menurut Business Dictionary, optimasi merupakan cara menemukan suatu alternatif yang memiliki biaya paling efektif atau pencapaian performa paling tinggi dengan adanya batasan – batasan. Dengan optimasi kita dapat memaksimalkan faktor yang diinginkan dan meminimalkan faktor yang tidak diinginkan [21]

2.2.3. *Orienteering Problem*

Orienteering Problem (OP) adalah varian dari *Travelling Salesman Problem* (TSP) dengan menggunakan score. Score dalam OP adalah keuntungan atau kepuasan yang didapat dengan mengunjungi sebuah node (kota, tempat, lokasi) [22]. Terbentuknya OP terinspirasi dari sebuah *outdoor game* yang biasa dimainkan di daerah pegunungan atau hutan, dimana terdapat kumpulan *checkpoint* (landmark). Setiap *checkpoint* dihubungkan dengan profit atau skor dan hanya dapat dikunjungi satu kali. Dengan waktu yang telah ditentukan, kontestan memulai dari checkpoint awal, mengunjungi beberapa *checkpoint*, dan selesai di tujuan akhir. Karena tidak memungkinkan untuk mengunjungi seluruh *checkpoint*, tujuan dari OP adalah mengumpulkan profit atau skor sebanyak – banyaknya dengan mengunjungi *checkpoint* yang disukai [23].

OP dapat didefinisikan sebagai berikut [24]. Misal ada sebuah set nodes $N = \{1, \dots, |N|\}$ dimana tiap node $i, j \in N$ memiliki *non-negative score* S_i. Node awal adalah 1 dan node akhir adalah $|N|$. Tujuan dari OP adalah menentukan jalur optimal, terbatas dengan waktu yang diberikan dinotasi dengan T_{max} yang mengunjungi beberapa node dan memaksimalkan *total score* yang dikumpulkan tiap kunjungan. Diasumsikan tiap kunjungan node menambah *score* dan tiap node hanya dapat dikunjungi paling banyak satu kali. Waktu tempuh dari node i ke node j derepresentasikan dengan notasi t_{ij}.

OP dapat diformulasikan sebagai model *integer programming* dengan variable keputusan $X_{ij} = 1$ apabila kunjungan ke node *i* di ikuti dengan kunjungan ke node *j*, jika tidak maka $X_{ij} = 0$. Variabel u_i akan digunakan untuk batasan *subtour elimination* dan memungkinkan untuk menentukan posisi node yang telah dikunjungi. Fungsi tujuan dari OP dapat dirumuskan sebagai berikut [24]:

$$Maximize \sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} S_i X_{ij} \quad (0)$$

Fungsi tujuan di atas adalah untuk memaksimalkan *total score* yang dikumpulkan. Selain fungsi tujuan, OP juga memiliki batasan – batasan berikut:

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

Batasan (1) memastikan jalur dimulai dari node 1 dan berhenti di node $|N|$

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

Batasan (2) memastikan setiap jalur terhubung dan menjamin tiap node hanya dikunjungi paling banyak satu kali.

$$\sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} X_{ij} \leq T_{max} \quad (3)$$

Batasan (3) membatasi total waktu tempuh sesuai dengan Tmax.

$$2 \leq u_i \leq |N|; \forall i = 2, \dots, |N| \quad (4)$$

$$u_i - u_j + 1 \leq (|N| - 1)\left(1 - X_{ij}\right); \forall i = 2, \dots, |N| \quad (5)$$

Kombinasi dari batasan (4) dan (5) mencegah terjadinya subtour, yaitu kondisi dimana lintasan dimulai dan diakhiri pada titik yang sama.

2.2.4. *Iterative Local Search*

Iterative local search atau yang lebih dikenal dengan *iterated local search*, merupakan teknik metaheuristic dan *global optimization* [25]. Algoritma ini adalah perpanjangan dari Multi-Restart Search dan dipertimbangkan sebagai orang tua dari banyaknya pencarian dua fase seperti Greedy Randomized Adaptive Search Procedure (GRASP) dan Variable Neighborhood Search (VNS).

Tujuan dari *Iterative Local Search* adalah untuk meningkatkan pencarian stokastik dengan melakukan *sampling* yang lebih luas di lingkungan kandidat solusi dan menggunakan teknik pencarian lokal untuk menyaring solusi menjadi optimal [25]. Dalam *iterative local search* terdapat beberapa metode pencarian yang akan diaplikasikan. Metode tersebut antara lain:

a. *Deletion Method*: Penghapusan titik atau *node* dalam rute, yang dimulai dari *node* ke-2. Penghapusan dilakukan kecuali *node* awal dan *node* akhir
b. *Swap Method*: Tukar sepasang *node* dalam rute, kecuali *node* awal dan *node* akhir
c. *Insertion Method*: Tambahkan *node* yang belum dikunjungi, jika ada, ke posisi diantara *node* awal dan *node* akhir

Kode 2-1 merupakan *pseude code* umum untuk *iterative local search* [26]. *Iterative local search* akan terus melakukan perubahan data hingga tidak ada lagi solusi lebih yang dapat ditemukan dan jumlah iterasi telah mencapai batas maksimal. Pencarian lokal dapat dilakukan dengan berbagai macam metode seperti *swap, insert, delete, replace, shake step*, dan lain – lain. Pada penelitian ini, metode yang digunakan saat pencarian lokal adalah *swap*, *insert* dan *delete*.

```
Procedure Iterative Local Search
s₀ = Generate Initial Solution
s* = LocalSearch(s₀)
      Repeat
      s' = Perturbation (s*, history)
      s*' = LocalSearch (s')
      s* = AcceptanceCriterion (s*, s*',
      history)
      until termination condition met
end
```

Kode 2-1 Pseudo Code Iterative Local Search

Terdapat banyak teknik pencarian heuristic pada ILS, salah satunya adalah *hill climbing search*. Strategi *hill climbing* selalu mengevaluasi tiap hasil solusi baru dan mengambil yang terbaik. Pencarian berhenti apabila algoritma tidak dapat menemukan solusi yang lebih baik dibandingkan solusi saat ini. Pencarian tersebut dinamakan *hill climbing* karena diibaratkan sebuah pendaki gunung yang bersemangat namun tidak tahur arah, selalu mengambil jalur dengan tanjakan paling curam yang dapat dilalui sampai pendaki berada di puncak [27].

2.2.5. Dijkstra's Algorithm

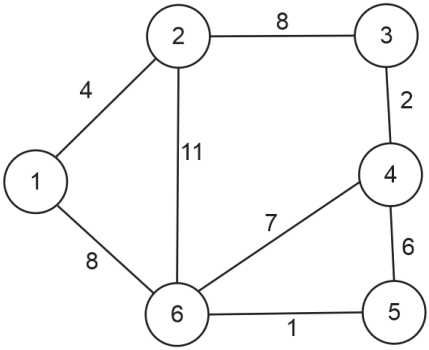
Ditemukan oleh ilmuwan komputer asal Belanda, bernama Edsger Dijkstra pada tahun 1956 dan dipublikasikan di tahun 1959 [28]. Algoritma Dijsktra merupakan pencarian graph yang dapat menyelesaikan permasalahan *shortest path* dari satu sumber untuk graph yang tidak memiliki nilai negatif pada jalurnya. Algoritma ini menghitung jarak/waktu/biaya terpendek dari sumber ke setiap *node* yang ada dalam graph. Dijkstra hanya dapat digunakan pada *directed-weighted graph* dan setiap *node* tidak boleh memiliki nilai negatif, jika terdapat *node* yang memiliki nilai negaitf, maka jalur terpendek yang sesungguhnya tidak dapat ditemukan.

Algortima ini bekerja dengan menetapkan nilai jarak awal dan mencoba memperbaruinya di setiap iterasi [29].

Berikut langkah – langkah yang dilakukan algoritma Dijkstra untuk mencari jarak terpendek [28]:

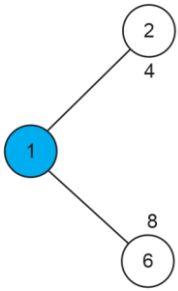
1. Pertama seluruh *node* akan diberikan nilai sementara. *Node* sumber diberikan nilai 0. Sedangkan *node* sisanya diberikan nilai *infinity* atau tak terhingga untuk mengindikasi bahwa *node* tersebut belum diproses.
2. Kemudian tandai semua *node* sebagai *unvisited node*. Atur nilai *node* awal atau *node* sumber dengan nilai sebenarnya. Buat sebuah set yang berisikan *node* yang masuk di dalam *shortest path tree* atau *node* dengan jalur terpendek. Pada awalnya, set tersebut merupakan set kosong dan diberi nama misalnya sptSet.
3. Pada *node* awal, hitung jarak *node* yang bertetangga dengan *node* sumber. Walaupun *node* yang dekat telah diperiksa, *node* tersebut masih belum masuk ke dalam sptSet, dan masih berada pada kumpulan *unvisited set*.
4. Setelah selesai menghitung seluruh jarak pada *node* awal, tandai *node* tersebut sebagai *visited node* dan *node* tersebut masuk ke dalam sptSet. *Node* yang telah dikunjungi tidak akan diperiksa kembali
5. Apabila *node* tujuan telah ditandai sebagai *visited node* atau nilai terkecil dari jarak sementara dalam kumpulan *unvisited set* bernilai tak terhingga, maka algoritma Dijsktra telah selesai.
6. Pilih *unvisited node* yang memiliki nilai jarak terkecil, dan atur *node* tersebut sebagai *node* awal dan ulangi dari langkah 3.

Untuk lebih mudahnya, Gambar 2-1 merupakan contoh algoritma Dijkstra pada sebuah graph dengan 6 *node*.



Gambar 2-1 Simple Network Model

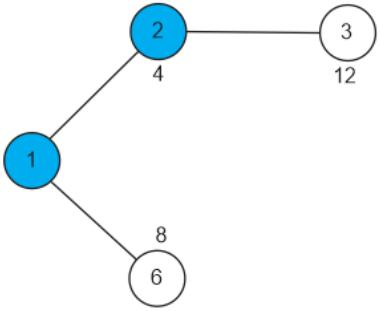
Jarak dari *node* 1 ke seluruh *node* adalah {0, INF, INF, INF, INF, INF} dimana INF menandakan tak terhingga. sptSet masih merupakan set kosong. Pilih *node* dengan nilai paling kecil, maka *node* 1 terpilih. Masukkan *node* tersebut ke dalam sptSet sehingga spetSet berisikan {1}. Kemudian perbarui jarak *node* terdekat dengan *node* 1. Yaitu *node* 2 dan 6. Jarak untuk 2 dan 6 diperbarui menjadi 4 dan 8. Gambar 2-2 merupakan penggambarannya:



Gambar 2-2 Langkah 1 algoritma Dijkstra

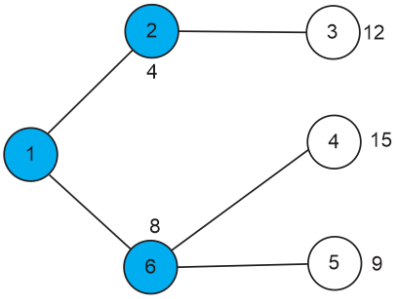
Node yang dimasukkan ke dalam sptSet ditandai dengan warna biru. Langkah selanjutnya adalah memilih *node* dengan nilai paling kecil dan belum masuk ke dalam sptSet. *Node* 2 dipilih dan sptSet menjadi {1,2}. Ubah nilai jarak pada

node yang berdekatan dengan *node* 2, Jarak ke *node* 3 menjadi 12.



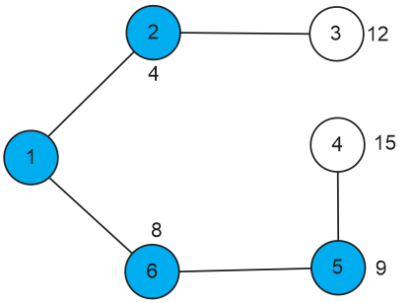
Gambar 2-3 Langkah 2 Algoritma Dijkstra

Pilih *node* dengan jarak paling kecil dan belum masuk ke dalam sptSet. *Node* 6 dipilih dan sptSet menjadi {1,2,6}. Ubah nilai jarak pada *node* yang berdekatan dengan *node* 6. *Node* 4 menjadi 12 dan *node* 5 menjadi 9.



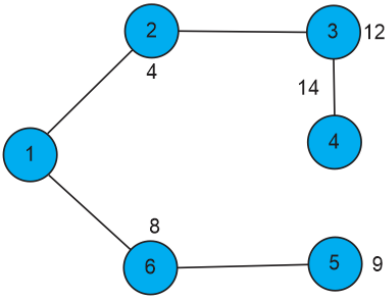
Gambar 2-4 Langkah 3 Algoritma Dijkstra

Pilih *node* dengan jarak paling kecil dan belum masuk ke dalam sptSet. *Node* 5 dipilih dan sptSet menjadi {1,2,6,5}. Nilai jarak dari 5 ke 8 di perbarui.



Gambar 2-5 Langkah 4 Algoritma Dijkstra

Ulangi proses tersebut sehingga sptSet berisikan seluruh *node* dan membentuk sebuah *Shortest Path Tree* seperti ditunjukkan pada Gambar 2-6:



Gambar 2-6 Hasil Akhir Algoritma Dijkstra

Gambar 2-6 merupakan hasil perhitungan algoritma Dijkstra dengan *node* 1 sebagai *node* sumber. Dengan menggunakan algoritma ini, kita dapat mengetahui berapa jarak dari *node* 1 ke seluruh *node* yang ada di dalam diagram.

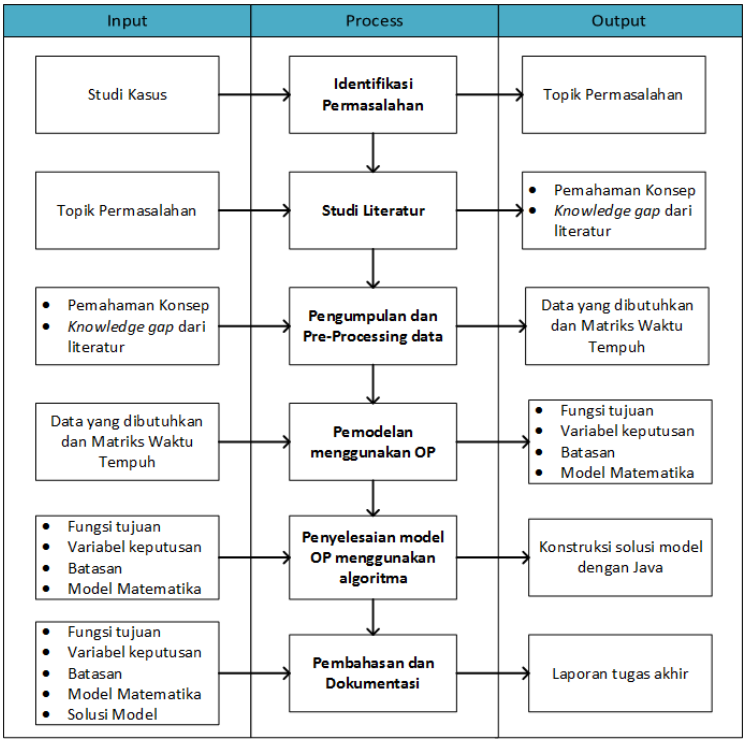
BAB III
METODOLOGI PENELITIAN

Pada bab metode penelitian akan dijelaskan mengenai tahapan – tahapan apa saja yang dilakukan dalam pengerjaan tugas akhir ini beserta deskripsi dan penjelasan tiap tahapan tersebut. Lalu disertakan jadwal pengerjaan tiap tahapanan.

3.1. Tahapan Pelaksanaan Tugas Akhir

Pada sub bab ini akan menjelaskan mengenai metodologi dalam pelaksanaan tugas akhir. Metodologi ini dapat dilihat pada Tabel 3-1:

Tabel 3-1 Metodologi Penelitian



3.1.1. Identifikasi Permasalahan

Pada tahap ini dilakukan identifikasi permasalahan dengan melakukan analisa kondisi lalu lintas saat ini di Indonesia khususnya di Kota Surabaya. Proses identifikasi permasalahan dilakukan dengan cara mencari informasi terkait kondisi lalu lintas dan kemacetan di Indonesia khususnya di surabaya. Permasalahan yang ditemukan terkait dengan banyaknya jumlah kendaraan pribadi yang ada di Kota Surabaya dan kota besar lainnya sehingga menyebabkan kemacetan. Maka dari itu pemerintah kota membangun transportasi umum seperti angkot. Pada penelitian ini permasalahan hanya dibatasi pada 6 trayek angkot di Surabaya, yaitu trayek dengan kode JK, F, JMK, O, RT, GS.

3.1.2. Studi Literatur

Studi literatur dilakukan dengan mengumpulkan berbagai referensi seperti penelitian, buku, dan dokumen terkait. Studi literatur didasarkan pada topik yang telah didapatkan sebelumnya. Pada tugas akhir ini diusulkan topik optimasi penentuan jalur terpendek dengan menggunakan angkutan umum khususnya angkot dengan menggunakan model optimasi tertentu. Setelah itu dilakukan kembali studi literatur untuk memahami teori dasar yang berkaitan dengan permasalahan yang ingin diselesaikan.

Berdasarkan proses ini, dapat diketahui permasalahan, tujuan, dan batasan dari topik yang ada. Setelah mendapatkan permasalahan, tujuan, dan batasan dari topik, langkah berikutnya adalah mencari metode yang komprehensif untuk menyelesaikan permasalahan tugas akhir. Untuk mencari metode ini, dilakukan analisa terkait metode – metode yang digunakan pada permasalahan serupa. Kemudian dilakukan gap analysis untuk mencari tahu kelebihan dan kekurangan tiap metode yang telah dilakukan di penelitian sebelum – sebelumnya. Seperti yang telah dijelaskan pada bab sebelumnya, penelitian pada tugas akhir ini menggunakan

metode orienteering problem dengan pencarian solusi menggunakan metode optimasi *Iterative Local Search*.

3.1.3. Pengumpulan dan Pre-Processing Data

Setelah menemukan metode yang sesuai, tahapan selanjutnya adalah mengumpulkan data yang akan digunakan. Data merupakan pendukung utama dalam pengerjaan tugas akhir ini. Maka dari itu diperlukan data yang sesuai dengan topik dan batasan permasalahan yang diambil. Pada tugas akhir ini, data didapat dari Dinas Perhubungan Surabaya dan Google Maps diantaranya:

- Data jalur trayek angkutan umum (angkot)
- Data jumlah kendaraan pada masing – masing trayek
- Data waktu tempuh dari tiap pemberhentian trayek

Pre-processing dilakukan dengan memasukkan data waktu tempuh yang telah di catat ke dalam matriks waktu tempuh.

3.1.4. Pemodelan Menggunakan OP

Pada tahap ini, permasalahan dianalisis untuk menentukan fungsi tujuan dan batasan masalah. Fungsi tujuan dan batasan masalah ditentukan dengan menerjemahkan permasalahan ke dalam model matematika. Dari data yang diperoleh ditentukan skor, node, branch dan lain – lain agar bisa membuat model OP. Berikut merupakan penjabarannya

a. Score: Jumlah angkot yang melewati node
b. Node: Titik yang dilewati sebuah angkot
c. Arc: Waktu tempuh tiap node
d. T_{max}: Waktu tempuh maksimal yang diinginkan

Dari penjabaran notasi di atas, fungsi tujuan dan batasan dari model ini adalah:

1) Variabel Keputusan:

a. Kunjungan dari satu tempat ke tempat pemberhentian angkot yang lainnya

2) Fungsi Tujuan:
   a. Memaksimalkan skor yang dikumpulkan dengan mengunjungi tiap node
3) Batasan:
   a. Setiap node hanya dapat dikunjungi satu kali
   b. Tidak perlu mengunjungi seluruh node
   c. Semua node terhubung satu dengan lainnya
   d. Waktu tempuh dibatasi oleh T_{max}

3.1.5. Penyelesaian Model OP Menggunakan Algoritma

Tahap ini merupakan tahap konstruksi solusi dari model yang telah dibuat pada tahap sebelumnya. Metode optimasi *Iterative Local Search* diaplikasikan dengan menggunakan bahasa pemrograman *Java*. Gambaran secara keseluruhan mengenai proses ini dapat dilihat pada Tabel 3-2. Tahapan ini dibagi menjadi beberapa sub-tahapan, antara lain:

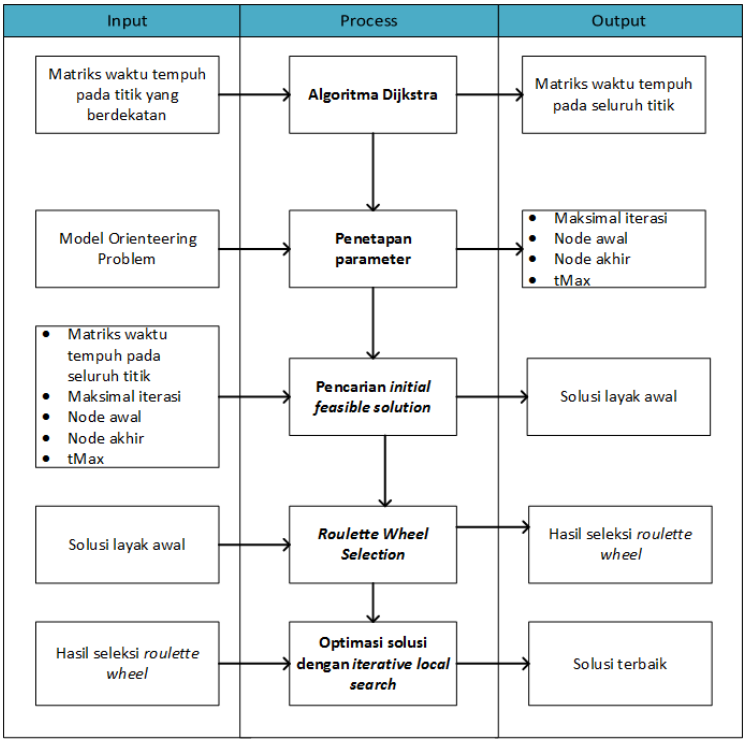
3.1.5.1. Pencarian Rute Terpendek

Sebelum melakukan pencarian solusi, lamanya perjalanan dari 1 *node* ke *node* lainnya perlu diketahui. Pada tahap pre-processing, sudah dibentuk sebuah matriks yang berisikan lamanya perjalanan pada tiap *node* yang berdekatan, namun matriks tersebut belum mencakup seluruh titik (*node*) yang ada dalam *network model*.

Pada tahap ini, dibentuk sebuah data set yang berisikan waktu tempuh untuk keseluruhan *node*. Data set yang dihasilkan pada penelitian ini berbentuk matriks. Pembentukan matriks waktu tempuh diselesaikan dengan bantuan algoritma Dijkstra. Algoritma Dijkstra digunakan karena pada matriks

sebelumnya, masih terdapat kemungkinan banyaknya jalur alternatif untuk 2 titik, sehingga algoritma Dijkstra dapat membantu menemukan jalur tercepat pada setiap titik yang ada di *network model*.

Tabel 3-2 Alur Penyelesaian Solusi Model OP



3.1.5.2. Penetapan Parameter Pencarian Solusi

Pada tahap ini, ditentukan parameter – parameter untuk pencarian solusi awal, seperti jumlah *node*, jumlah iterasi, dan maksimal waktu tempuh (tMax). Selain itu, pada tahap ini juga dilakukan deklarasi untuk *node* awal dan *node* akhir.

3.1.5.3. Pencarian *Initial Feasible Solution*

Setelah data set telah siap digunakan dan seluruh parameter telah ditetapkan, tahapan ini akan mencari solusi layak awal secara random. Solusi dapat dikatakan layak apabila tidak melanggar batasan – batasan model *Orienteering Problem*. Pencarian solusi layak awal akan terus dilakukan sampai maksimal iterasi tercapai.

3.1.5.4. Seleksi Solusi dengan *Roulette Wheel Selection*

Setelah didapatkan solusi layak awal di tahap sebelumnya. Pada tahap ini, hasil dari pencarian solusi layak awal akan diseleksi menggunakan *roulette wheel selection*. *Roulette Wheel Selection* digunakan karena sifatnya yang tidak mengambil nilai secara random, dan tidak hanya mengambil nilai terbaik saja. Solusi yang memiliki skor lebih tinggi, memiliki kemungkinan lebih tinggi juga untuk lolos seleksi *roulette wheel*, namun tidak menutup kemungkinan solusi dengan skor kecil juga terpilih.

Pertama ditentukan terlebih dahulu seberapa banyak solusi yang akan diambil. Hasil dari *roulette wheel selection* kemudian akan dioptimasi pada tahapan berikutnya. Proses *roulette wheel selection* akan dijelaskan lebih lanjut pada Bab 5.

3.1.5.5. Pengaplikasian algoritma optimasi dan pengambilan Solusi Terbaik.

Berdasarkan hasil *roulette wheel selection*, Seluruh hasil seleksi akan dilakukan optimasi menggunakan *iterative local search*. Pencarian solusi optimal dilakukan dengan menggunakan 3 metode antara lain *swap method*, *insertion method*, dan *deletion method*. Pencarian lokal dilakukan secara berulang. Setiap iterasi hanya mengaplikasikan satu metode, apabila solusi yang baru memiliki waktu tempuh yang lebih pendek atau skor yang lebih tinggi, maka perhitungan iterasi akan diulang menjadi 0. Probabilitas metode optimasi yang dipilih pada setiap iterasi adalah sama.

Setelah pencarian lokal selesai dilakukan, akan diambil solusi terbaik dari seluruh solusi yang dioptimasi. Solusi terbaik pada penelitian ini adalah solusi dengan skor tertinggi dan waktu terendah.

3.1.6. Pembahasan dan Dokumentasi

Semua proses di atas akan disusun dalam laporan tugas akhir sebagai bentuk dokumentasi atas terlaksananya tugas akhir ini. Laporan tersebut mencakup:

a. Bab I Pendahuluan

Pada bab ini dijelaskan mengenai latar belakang permasalahan, tujuan, batasan, manfaat, serta relevansi tugas akhir.

b. Bab II Tinjauan Pustaka

Pada bab ini dijelaskan mengenai teori – teori yang menunjang permasalahan yang dibahas pada tugas akhir.

c. Bab III Metodologi

Pada bab ini dijelaskan mengenai tahapan – tahapan apa saja yang akan dilakukan dalam pengerjaan tugas akhir.

d. Bab IV Perancangan

Pada bab ini berisi rancangan penelitian, rancangan bagaimana penelitian dilakukan, obyek penelitian, dan sebagainya.

e. Bab V Implementasi

Pada bab ini berisi proses pelaksanaan penelitian, bagaimana penelitian dilakukan penerapan strategi pelaksanaan, hambatan, rintangan dalam pelaksanaan, dan sebagainya.

f. Bab VI Analisis dan Pembahasan

Pada bab ini berisikan tentang pembahasan dari penyelesaian permasalahan yang dibahas pada tugas akhir.

g. Bab VII Kesimpulan dan Saran

Bab ini berisikan kesimpulan dari apa yang telah dikerjakan dalam tugas akhir ini, serta saran yang ditujukan untuk kelengkapan penyempuranaan penelitian terkait.

BAB IV
PERANCANGAN

Pada bab ini akan dijelaskan bagaimana rancangan dari penelitian tugas akhir yang meliputi subyek dan obyek dari penelitian, pemilihan subyek dan obyek penelitian dan bagaimana penelitian akan dilakukan.

4.1. Pengumpulan dan Deskripsi Data

Pengumpulan data dilakukan dengan mengambil data dari website resmi Dinas Perhubungan Kota Surabaya [31]. Data yang diambil disesuaikan dengan kebutuhan pengolahan data yaitu berupa rute trayek serta jumlah angkot yang ada pada masing – masing trayek. Selain itu, dilakukan pencatatan jarak dan waktu tempuh tiap rute trayek, dimana data jarak dan waktu tempuh didapat dari Google Maps. Trayek yang digunakan pada penelitian ini antara lain:

Tabel 4-1 Trayek yang Digunakan

Kode Trayek	Asal Tujuan (OD)	Jumlah
F	Endrosono – Joyoboyo PP	143
GS	Sidorame – Gunung Anyar PP	55
JK	Joyoboyo – Kenjeran PP	31
JMK	Kenjeran – Jembatan Merah Plaza – Petekan PP	54
O	Kalimas Barat – Keputih PP	133
RT	Pasar Turi – Rungkut Harapan PP	80

Tiap jalur trayek berisikan jalan yang dari berangkat hingga kembali. Berikut merupakan tabel jalur yang dilalui pada masing – masing trayek:

Tabel 4-2 Informasi Jalur Tiap Trayek

F	**Berangkat:** Terminal Joyoboyo - Jl. Raya Wonokromo - Jl. Jagir Wonokromo – Jembatan Jagir

	- Jl. Ngagel - Jl. Sulawesi - Jl. Raya Gubeng - Jl. Sumatera - Jl. Stasiun Gubeng - Jl. Anggrek - Jl. Kusama Bangsa - Jl. Kapasari - Jl. Simokerto - Jl. Sidotopo Lor - Jl. Sidorame - Jl. Karang Tembok - Jl. Wonosari - Jl. Wonosari Lor - Jl. Endrosono (Pangkalan Akhir)
	Kembali: Pangkalan Endrosono - Jl. Endrosono - Jl. Wonosari Lor - Jl. Wonosari - Jl. Karang Tembok - Jl. Sidorame - Jl. Simokerto - Jl. Kapasari - Jl. Kusuma Bangsa - Jl. Anggrek - Jl. Stasiun Gubeng - Jl. Sumatera - Jl. Raya Gubeng - Jl. Biliton - Jl. Sulawesi - Jl. Ngagel - Jl. Bung Tomo - Jl. Opojiwa - Jl. Ratna - Jl. Ngagel - Jembatan Jagir - Jl. Stasiun Wonokromo - Jl. A. Yani - U Turn RSI - Jl. Raya Wonokromo - Terminal Joyoboyo (Pangkalan Akhir).
GS	**Berangkat:** Pangkalan Sidorame – Jl. Sidorame – Jl. Pergiwati – Jl. Kunti – Jl. Sidotopo Kidul – Jl. Pargoto – Jl. Jl. Sombo – Jl. Sidodadi IV – Jl. Kampung Seng – Jl. Kapasan – Jl. Gembong – Jl. Pecindilan – Jl. Undaan Wetan – Jl. Kalisari I – Jl. Kalisari III – Jl. Jakgung Suprapto – Jl. Ambengan – Jl. Wijaya Kusuma – Jl. Stasiun Gubeng – Jl. Sumatra – Jl. Biliton – Jl. Sulawesi – Jl. Kalibokor I – Jl. Pucang Anom – Jl. Pucang Sewu – Jl. Ngagel Jaya – Jl. Ngagel Jaya Selatan – Jl. Ngagel Madya – Jl. Manyar – Jl. Raya Nginden – Jl. Panjang Jiwo SMPP – Jl. Prapen Indah I – Jl. Prapen Indah V – Jl. Tenggilis Utara X – Jl. Tenggilis Lama II – Jl. Tenggilis Lama IV – Jl. Tenggilis Lama III – Jl. Raya Tenggilis Mejoyo – Jl. Raya Kalirungkut – Jl. Kali Rungkut – Jl. Raya Rungkut – Jl. Rungkut Asri Utara I – Jl. Rungkut Asri – Jl. Rungkut Madya – Pangkalan Gunung Anyar (Pangkalan Akhir).
	Kembali: Pangkalan Gununganyar – Jl. Rungkut Madya – Jl. Rungkut Asri Barat VII – Jl. Rungkut

	Asri – Jl. Rungkut Asri Utara – Jl. Rungkut Mejoyo Selatan – Jl. Tenggilis Mejoyo – Jl. Raya Tenggilis – Jl. Tenggilis Kauman – Jl. Tenggilis Lor II – Jl. Tenggilis Utara V – Jl. Prapen Indah I – Jl. Prapen Indah – Jl. Raya Prapen – Jl. Panjang Jiwo SMPP – Jl. Barata Jaya XVII – Jl. Barata Jaya XIX – Jl. Barata Jaya – Jl. Bratang Wetan – Jl. Ngagel Jaya Selatan – Jl. Ngagel Jaya Barat – Jl. Ngagel Timur IV – Jl. Ngagel Timur – Jl. Kalibokor – Jl. Ngagel – Jl. Sulawesi – Jl. Raya Gubeng – Jl. Sumatera – Jl. Stasiun Gubeng – Jl. Kusuma Bangsa – Jl. Wijaya Kusuma – Jl. Ambengan – Jl. Jakgung Suprapto – Jl. Kalisari III – Jl. Kalianyar – Jl. Undaan Wetan – Jl. Undaan Kulon – Jl. Pengampon – Jl. Bunguran – Jl. Waspada (jl. Bongkaran – kembang jepun) – Jl. Kapasan – Jl. Sidodadi IV – Jl. Aswotomo – Jl. Sidorame – Pangkalan Sidorame (Pangkalan Akhir)
JK	**Berangkat:** Terminal Joyoboyo – Jl. Raya Darmo – Jl. Mojopahit – Jl. Dr. Wahidin – Jl. Dr. Sutomo – Jl. Sriwijaya – Jl. Pandegiling – Jl. Sulawesi – Jl. Raya Gubeng – Jl. Biliton – Jl. Kalimantan – Jl. Nias – Jl. Banda – Jl. Gubeng Masjid – Jl. Gembong – Jl. Tapak Siring – Jl. Indrakila – Jl. Kalasan – Jl. Jolotundo – Jl. Jagiran – Jl. Gersikan – Jl. Ploso Baru – Jl. Kalijudan – Jl. Kenjeran – Jl. Wiratno – Jl. Pangkalan Kenjeran (Pangkalan Akhir).
	Kembali: Pangkalan Kenjeran – Jl. Wiratno – Jl. Kenjeran – Jl. Kalijudan – Jl. Ploso Baru – Jl. Gersikan – Jl. Jagiran – Jl. Sawentar – Jl. Pacar Keling – Jl. Penataran – Jl. Tapak Siring – Jl. Prof. Dr. Mustopo – Jl. Karang Menjangan – Jl. Airlangga – Jl. Darmawangsa – Jl. Prof Dr. Mustopo – Jl. Gubeng Masjid – Jl. Banda – Jl. Nias – Jl. Kalimantan – Jl. Biliton – Jl. Sumbawa – Jl. Raya Gubeng – Jl. Karimun Jawa – Jl. Embong Sono Kembang – Jl. Panglima Sudirman – Jl. Urip Sumoharjo – Jl. Raya

	Darmo – Jl. Bengawan – Jl. Diponegoro – Jl. Ciliwung – Jl. Adityawarman – Jl. Hayam Wuruk – Jl. Joyoboyo – Terminal Joyoboyo (Pangkalan Akhir)
JMK	**Berangkat:** Pangkalan Kenjeran - Jl. Kenjeran Pantai Lama - Jl. Kyai Tambak Deres - Jl. Pogot - Jl. Pogot Lama - Jl. Platuk - Jl. Sidotopo Wetan Gg. I Luar - Jl. Tenggumung Baru Selatan - Jl. Sidotopo Wetan - Jl. Kenjeran - Jl. Simokerto - Jl. Simolawang baru - Jl. Sidodadi - Jl. Sidodadi IV - jl. Sidodadi V - Jl. Aswotomo - Jl. Pegirian - Jl. K.H. Mansyur - Jl. Panggung - Jl. Kembang Jepun - Jl. Karet - Jl. Jembatan Merah - Jl. Veteran - Jl. Indrapura - Jl. Krembangan Barat - Jl. Krembangan Timur - Jl. Rajawali - Jl. Kasuari - Jl. Kalimas Barat - Pangkalan Petekan (Pangkalan Akhir)
	Kembali: Pangkalan Petekan - Jl. Kalimas Baru - Jl. Jakarta - Jl. Kalimas Barat - Jl. Kelasi - Jl. Kasuari - Jembatan Merah Plaza - Jl. Jembatan Merah - Jl. Kembang Jepun - Jl. Kalimati Kulon - Jl. Kalimati Wetan - Jl. Dukuh - Jl. Nyampluangan - Jl. Danakarya - Jl. Pegirian - Jl. Aswotomo - Jl. Sidodadi - Jl. Kampung Seng - Jl. Kapasan - Jl. Gembong - Jl. Pecindilan - Jl. Kalianyar - Jl. Ngaglik - Jl. Tambak Adi - Jl. Donorejo Wetan - Jl. Tambak Rejo - Jl. Tambak Laban - Jl. Kenjeran - Jl. Sidotopo Wetan I Gg. Luar - Jl. Jl. Platuk - Jl. Pogot Lama - Jl. Pogot - Jl. Kyai Tambak Deres - Jl. Kenjeran Pantai Lama - Pangkalan Kenjeran (Pangkalan Akhir)
O	**Berangkat:** Pangkalan Jayengrono – Jl. Jembatan Merah – Jl. Veteran – Jl. Pahlawan – Jl. Pasar Besar Wetan – Jl. Peneleh – Jl. Makam Peneleh – Jl. Rp Sunaryo Gondo Kusumo – Jl. Mas Soedjoto – Jl. Undaan Kulon – Jl. Pengampon – Jl. Pecindilan – Jl. Kalianyar – Jl. Ngaglik – Jl. Kapas Krampung – Jl. Karang Asem – Jl. Bronggalan – Jl. Tambak Boyo – Jl. Prof Dr Moestopo – Jl. Raya Dharmahusada Indah

	– Jl. Kertajaya Indah – U Turn – Jl. Raya Kertajaya Indah – Jl. Kertajaya Indah Tengah – Jl. Manyar Kertoajo – Jl. Kertajaya Indah Timur – U Turn – Jl. Manyar Kertoadi – Jl. Gebang Putih – Jl. Arif Rahman Hakim – Jl. Kh Ahmad Dahlan – Pangkalan Keputih (Pangkalan Akhir).
	Kembali: Pangkalan Kh Ahmad Dalan – Jl. Arif Rachman Hakim – Jl. Gebang Putih – Jl. Manyar Kertoadi – Jl. Kertajaya Indah Tengah – Jl. Kertajaya Indah – Jl. Dharmahusada Indah – Jl. Prof Dr Moestopo – Jl. Karang Menjagan – Jl. Airlangga – Jl. Dharmawangsa – Jl. Prof Dr Moestopo – Jl. Kedung Sroko – Jl. Pacar Keling – Jl. Kalasan – Jl. Jolotundo – Jl. Tambang Boyo – Jl. Bronggalan – Jl. Karang Asem – Jl. Kapas Krampung – Jl. Tambak Sari – Jl. Ambengan – Jl. Kusuma Bangsa – Jl. Kalianyar – Jl. Undaan Wetan – Jl. Undaan Kulon – Jl. Jagalan – Jl. Pasar Besar Wetan – Jl. Tembaan – Jl. Bubutan – Jl. Indrapura – Jl. Krembangan Barat – Jl. Krembangaan Timur – Jl. Rajawali – Jl. Kasuari – Jl. Garuda – Jl. Taman Jayangrono (Pangkalan Akhir).
RT	**Berangkat:** Pangkalan Rungkut Harapan – Jl. Rungkut Harapan – Jl. Rungkut Alang-alang – Jl. Rungkut Asri Utar VI – Jl. Kalirungkut – Jl. Kedung Asem – Jl. Kedung Baruk – Jl. Panjang Jiwo – Jl. Barata Jaya 17 – Jl. Barata Jaya 19 – Jl. Bratang Binangun – Jl. Ngagel Jaya Selatan – Jl. Manyar – Jl. Manyar – Jl. Ngagel Jaya Utara – Jl. Ngagel Madya – Jl. Pucang Jajar – Jl. Pucang Anom Timur – Jl. Pucang Anom – Jl. Juwingan – Jl. Karimata – Jl. Flores – Jl. Lombok – Jl. Ngagel – Jl. Sulawesi – Jl. Raya Gubeng – Jl. Karimata – Jl. Embong Sono Kembang – Jl. Embong Cerme – Jl. Embong Cerme – Jl. Embong Kemiri – Jl. Panglima Sudirman – Jl. Basuki Rahmat – Jl. Embong Malang – Jl. Blauran –

	Jl. Bubutan – Jl. Indrapura – Jl. Pasar Turi – Pangkalan Pasar Turi (Pangkalan Akhir)
	Kembali: Pangkalan Pasar Turi – Jl. Pasar Turi – Jl. Semarang – Jl. Raden Saleh – Jl. Bubutan – Jl. Penghela – Jl. Pahlawan – Jl. Gemblongan – Jl. Tunjungan – Jl. Genteng Besar – Jl. Genteng Kali – Jl. Ngemplak – Jl. Walikota Mustadjab – Jl. Jaksa Agung Suprapto – Jl. Wilkota Mustadjab – Jl. Gubeng Pojok -Jl. Sumatra – Jl. Raya Gubeng – Jl. Biliton – Jl. Sulawesi – Jl. Kertajaya – Jl. Sulawesi – Jl. Flores – Jl. Lombok – Jl. Ngagel – Jl. Kalibokor 1 – Jl. Pucang Sewu – Jl. Pucang Anom – Jl. Pucang Jajar – Jl. Ngagel Madya – Jl. Ngagel Jaya Selatan – Jl. Manyar – Jl. Raya Nginden – U Turn – Jl. Raya Nginden – Terminal Bratang – Jl. Bratang Jaya – Jl. Barata Jaya XIX – Jl. Barata Jaya XVII – Jl. Raya Nginden – U Turn – Jl. Raya Nginden – Jl. Panjang Jiwo – Jl. Kedung Baruk – Jl. Kedung Asem – Jl. Penjaraingan Sari – Jl. Rungkut Harapan – Pangkalan Rungkut Harapan (Pangkalan Akhir)

4.2. Perancangan Network Model

Perancangan *Network Model* diawali dengan menggambakan rute trayek yang digunakan, setiap rute trayek digambarkan di atas peta digital menggunakan Corel Draw sebagai gambaran awal jalur tiap trayek sekaligus untuk mengetahui pada jalur mana saja tiap trayek bersinggungan.

Masing – masing jalur trayek direpresentasikan dengan warna berbeda, Tabel 4-3 merupakan rincian trayek dengan warna yang direpresentasikan. Hasil dari penggambaran jalur tersebut dapat dilihat pada Gambar 4-1

Tabel 4-3 Pewarnaan Jalur Trayek

Kode trayek	Warna
F	Hijau
JK	Biru
JMK	Hitam
GS	Merah
O	Ungu
RT	Kuning

Gambar 4-1 Penggambaran Jalur ke 6 Trayek

4.2.1. Penentuan Node dan Skor

Setelah tiap jalur trayek angkot digambarkan pada peta digital, maka kemudian ditentukan letak *node* pada jalur trayek. Yang berperan sebagai *node* adalah tiap – tiap jalan yang dilewati angkot. Berdasarkan Panduan Teknis Perekayasaan Tempat Perhentian Kendaraan Umum oleh Departemen Perhubungan Darat [32], jarak tempat henti yang cocok berdasarkan kondisi lokasi (misal pusat kota, padat, pemukiman) adalah 200 – 1000 meter. Pada penelitian ini, diasumsikan tidak ada perbedaan kondisi pada setiap jalur trayek.

Maka dari itu, penentuan lokasi tiap *node* pada penelitian ini dibuat mengikuti panduan dari Departemen Pehubungan Darat. Jarak antar *node* tidak lebih dari 1000 meter dan tidak kurang dari 200 meter. Tiap *node* juga memiliki nilai skor yang digunakan untuk pemodelan *Orienteering Problem*. Nilai skor pada *node* dihitung dari jumlah angkot yang melewati *node* tersebut.

Tabel 4-4 merupakan potongan tabel lokasi dimana tiap *node* berada beserta skornya. Untuk keseluruhan lokasi *node* dan skor tiap *node* dapat dilihat pada LAMPIRAN C.

4.2.2. Penentuan *arc*

Setelah ditentukannya lokasi – lokasi tiap *node*, seluruh *node* tersebut dihubungkan dengan sebuah garis yang memiliki jarak dan waktu tempuh. Garis penghubung tiap *node* tersebut disebut dengan *arc*. Jarak dan waktu tempuh saat berangkat dan pulang dari 1 *node* ke *node* yang lainnya dapat berbeda tergantung dengan kondisi sebenarnya di lapangan.

Seluruh nilai jarak dan waktu tempuh tiap *arc* didapat dengan menggunakan aplikasi *Google Maps*. Dengan mengacu pada Panduan Teknis Perekayasaan Tempat Perhentian Kendaraan Umum [32], seluruh nilai *arc* pada penelitian ini tidak ada yang kurang dari 200 meter dan lebih dari 1000 meter.

Tabel 4-4 Lokasi dan Skor Tiap *Node*

Node	Lokasi	Skor
1	Terminal Joyoboyo	174
2	Jalan Raya Wonokromo dekat jembatan	143
3	Jembatan Jagir	143
4	Jalan Raya Ngagel dekat SPBU	143
5	Jalan Raya Ngagel dekat Carefour Kalimas	143
6	Jalan Raya Ngagel dekat LBB Einstein	143
7	Jalan Raya Ngagel dekat Jalan Sulawesi	278
8	Jalan Sulawesi arah Raya Gubeng	309
9	Jalan Raya Gubeng	229
10	Jalan Raya Gubeng dekat Taman Lansia	278
11	Jalan Stasiun Gubeng	198
12	Jalan Kusuma Bangsa dekat Jalan melati	198
13	Perempatan Jalan Kusuma Bangsa dan Jalan Ambengan	331
14	Perempatan Jalan Kusuma Bangsa dan Jalan Kalianyar	330
15	Jalan Kapasari	143

Tabel 4-5 merupakan tabel nilai jarak dan waktu tempuh masing – masing *arc* dalam *network model*. Untuk keseluruhan data jarak dan waktu tempuh tiap *node* dapat dilihat pada LAMPIRAN D.

4.2.3. Asumsi Dalam Pembuatan *Network Model*

Karena terdapat beberapa ketidaksesuaian data dengan kondisi sebenarnya di lapangan, terdapat asumsi – asumsi yang digunakan dalam pembuatan *network model*. Asumsi yang digunakan pada penelitian ini dapat dilihat pada Tabel 4-6.

Tabel 4-5 Informasi Jarak dan Waktu Tempuh Tiap Node

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
1	2	900	2
1	27	700	2
2	1	140	1
2	3	450	1
3	4	450	1
3	25	650	2
4	3	450	1
4	5	650	1
5	4	650	1
5	6	900	2
6	7	550	1
6	24	650	1
7	6	550	1
7	8	500	2
7	107	550	2
8	9	500	1
8	54	600	1
9	10	400	1
9	34	870	2
10	11	600	2
10	22	500	1

Tabel 4-6 Asumsi Dalam Pembuatan Network Model

1	**Asumsi:** Jalan Anggrek pada trayek F dihilangkan
	Alasan: Data jalur menunjukkan angkot bergerak dari Jl. Stasiun Gubeng menuju Jl. Anggrek kemudian Jl. Kusuma Bangsa. Sedangkan di lapangan Jl. Kusuma

	Bangsa berada setelah Jl. Stasiun Gubeng. Pada Google Maps tidak terdapat Jl. Anggrek di sekitar area tersebut.
	Trayek: F
2	**Asumsi:** Jalan Mojopahit pada trayek JK diganti menjadi Jalan Majapahit
	Alasan: Dari Jalan Raya Darmo menuju Jalan Mojopahit kemudian Dr. Wahidin tidak sesuai, dan lebih cocok melalui Jalan Majapahit
	Trayek: JK
3	**Asumsi:** Pada jalur berangkat trayek JK, Jalan Biliton dihilangkan
	Alasan: Alur trayek menunjukkan dari Jl. Raya Gubeng menuju Jl. Biliton menuju Jl. Kalimantan, berdasarkan kondisi lapangan sendiri, dari Jl. Biliton tidak dapat menuju Jl. Kalimantan karena Jl. Biliton merupakan jalan satu arah
	Trayek: JK
4	**Asumsi:** Jalan Aswotomo dihilangkan
	Alasan: Tidak terdapat Jl. Aswotomo pada Gmaps
	Trayek: JMK dan GS
5	**Asumsi:** Pada trayek JMK jalur kembali tidak melewati Jl. Kelasi
	Alasan: Jalur menunjukkan angkot bergerak dari Kalimas Barat – Jl. Kelasi – Jl. Kasuari – JMP, dalam kenyataannya dari Jl. Kelasi, angkot tidak dapat langsung menuju Jl. Kasuari
	Trayek: JMK
6	**Asumsi:** Jl. Pucang Anom dihilangkan pada jalur berangkat trayek GS
	Alasan: Dari Kalibokor I dapat langsung menuju Jl. Pucang Sewu, apabila melewati Jl. Pucang Anom, maka angkot perlu melakukan U Turn di Jl. Pucang Anom
	Trayek: GS

7	**Asumsi:** Jalur trayek GS yang melewati Jl. Prapen Indah I dan Jl. Prapen Indah V diasumsikan hanya melewati Jl. Prapen Indah
	Alasan: Berdasarkan Gmasp, dari Jl. Panjang Jiwo tidak dapat menuju Jl. Prapen Indah I dan V, melainkan melewati Jl. Prapen Indah kemudian dapat menuju Jl. Tenggilis Utara X
	Trayek: GS
8	**Asumsi:** Pada jalur kembali trayek GS, angkot melewati Jl. Rungkut Lor Gg. III dan X agar dapat Jl. Rungkut Mejoyo Selatan dar Jl. Rungkut Asri Utara
	Alasan: dari Jl. Rungkut Asri Utara, angkot tidak dapat langsung menuju Jl. Rungkut Mejoyo Selatan
	Trayek: GS
9	**Asumsi:** Pada jalur kembali trayek GS, setelah melewati Jl. Waspada, angkot melewati Jl. Bongkaran kemudian Jl. Kembang Jepun untuk menuju Jl. Kapasan
	Alasan: Pada data jalur, angkot bergerak dari Jl. Bunguran – Jl. Waspada – Jl. Kapasan, dalam kenyataannya dari Jl. Waspada tidak dapat langsung menuju Jl. kapasan
	Trayek: GS
10	**Asumsi:** Jl. Embong Cerme pada jalur berangkat trayek RT diganti menjadi Jl. Kayon
	Alasan: Pada data, angkot bergerak dari Jl. Embong Sono Kembang – Jl. Embong Cerme – Jl. Embong Kemiri, pada kenyataannya kendaraan tidak dapat menuju Jl. Embong Cerme melalui Jl. Sono Kembang
	Trayek: RT
11	**Asumsi:** Jl. Penjaraingan Sari pada jalur kembali trayek RT dihilangkan
	Alasan: Pada data, angkot bergerak dari Jl. Kedung Asem – Jl. Penjaraingan Sari – Jl. Rungkut Harapan. Berdasarkan Gmaps, lokasi Jl. Penjaraingan Sari

	sangat jauh dengan Jl. Kedung Asem dan Jl. Rungkut Harapan
	Trayek: RT
12	**Asumsi:** Node disini merupakan representasi tempat pemberhentian tiap angkot. Tempat pemberhentian terletak pada jalur kanan dan jalur kiri trayek untuk jalan besar maupun kecil
	Alasan: Untuk simplifikasi dalam pengerjaan model
	Trayek: F,GS,JK,JMK,O,RT

4.3. Pemodelan *Orienteering Problem*

Dari *network model* di atas yang telah dibuat sebelumnya, dibuatlah model matematika yang menggunakan bentuk model *Orienteering Problem* (OP). Model OP dapat direpresentasikan ke dalam *integer programming* yang berisikan fungsi tujuan, variable keputusan, dan batasan. Berikut merupakan penjelasan rinci mengenai apa saja variable keputusannya, dan bagaimana fungsi tujuan serta batasan dari model.

4.3.1. Variabel Keputusan

Penelitian ini berfokus pada menentukan rute optimum dari satu titik ke titik lainnya. Maka dari itu, variable keputusan yang digunakan adalah kunjungan dari satu titik ke titik lainnya (*node to node*) yang direpresentasikan dengan x_{ij}. Dimana x_{ij} akan bernilai 1 apabila kunjungan dimulai dari *node i* dan berakhir di *node j*. Jika tidak, maka x_{ij} bernilai 0. Tabel 4-7 merupakan sebagian variable keputusan untuk model *Orienteering Problem*. Untuk keseluruhan vairabel keputusan dapat dilihat pada LAMPIRAN E.

4.3.2. Fungsi Tujuan

Fungsi tujuan dalam penelitian ini mengikuti fungsi tujuan OP pada umumnya yaitu mencari rute terpendek dengan jumlah skor maksimum.

Tabel 4-7 Variabel Keputusan Model OP

Variabel	Deskripsi
x1.2	Kunjungan dari node 1 ke node 2
x1.27	Kunjungan dari node 1 ke node 27
x2.1	Kunjungan dari node 2 ke node 1
x2.3	Kunjungan dari node 2 ke node 3
x3.4	Kunjungan dari node 3 ke node 4
x3.25	Kunjungan dari node 3 ke node 25
x4.3	Kunjungan dari node 4 ke node 3
x4.5	Kunjungan dari node 4 ke node 5
x5.4	Kunjungan dari node 5 ke node 4
x5.6	Kunjungan dari node 5 ke node 6
x6.7	Kunjungan dari node 6 ke node 7
x6.24	Kunjungan dari node 6 ke node 24
x7.6	Kunjungan dari node 7 ke node 6
x7.8	Kunjungan dari node 7 ke node 8
x7.107	Kunjungan dari node 7 ke node 107

Dalam studi kasus ini, *node* awal dan akhir telah ditentukan sebelumnya yaitu dari *node 1* (Terminal Joyoboyo) menuju *node 91* (Jembatan Merah Plaza). Seperti yang telah dijabarkan sebelumnya, terdapat 199 *node* pada *network* model. Seluruh *node* akan dimasukkan untuk mencari rute dengan skor tertinggi. Fungsi Tujuan dalam OP adalah sebagai berikut:

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

Dimana:
S_i: Skor pada *node* i
X_{ij}: Kunjungan dari *node i* ke *node j*
i: *node* 1, 2, 3, …, 198
j: *node* 2, 3, 4, …, 199

Sehingga penulisan fungsi tujuan pada studi kasus ini adalah:

$$Maximize \sum_{i=1}^{198} \sum_{j=2}^{199} S_i X_{ij}$$

Dimana skor untuk tiap *node* telah didefinisikan sebelumnya.

4.3.3. Batasan

Batasan yang digunakan dalam pemodelan *Orienteering Problem* adalah sebagai berikut:

- **Batasan 1:** Rute harus dimulai dari *node* 1 dan berakhir di *node* 91.
  Batasan ini bertujuan untuk memastikan rute yang dipilih dimulai dari lokasi awal dan berhenti di tempat yang dituju. Sehingga variable keputusan yang terkait dengan kunjungan dari *node* 1 dan kunjungan ke *node* 91 harus bernilai 1. Batasan 1 dijabarkan sebagai berikut:

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

  Untuk *node* awal penjabarannya adalah:

  $$\sum_{j=2}^{91} X_{1j} = 1$$

  Sedangkan *node* akhir penjabarannya adalah:

  $$\sum_{j=1}^{90} X_{i91} = 1$$

- **Batasan 2:** Setiap jalur terhubung dan setiap *node* dapat dikunjungi maksimal satu kali.
  Batasan ini bertujuan untuk memastikan setiap jalur yang dilalui pada *network model* terhubung dan menjamin setiap *node* hanya dapat dikunjungi satu kali.

Terhubungnya *node* ditunjukkan dengan adanya percabangan dan pertemuan pada beberapa *node*. Bentuk matematika dari batasan ini adalah sebagai berikut:

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

Untuk pertemuan *node* batasan dijabarkan seperti:

$$\sum_{i=1}^{90} X_{ik} \leq 1; \forall k = 2, \dots, 90$$

Untuk percabangan *node* batasan dijabarkan seperti:

$$\sum_{j=2}^{91} X_{kj} \leq 1; \forall k = 2, \dots, 90$$

- **Batasan 3:** Total waktu tempuh tidak boleh melebihi waktu yang telah ditentukan
  Pada studi kasus ini, waktu maksimum pencarian rute (Tmax) pada *network* model telah ditentukan sebelumnya yaitu 60 menit. Maka dari itu batasan dapat dijabarkan sebagai berikut:

$$\sum_{i=1}^{198} \sum_{j=2}^{199} t_{ij} X_{ij} \leq 60$$

- **Batasan 4**: Tidak terjadinya subtour, yaitu kondisi dimana lintasan dimulai dan diakhiri pada titik yang sama. Secara tidak langsung, batasan ini telah tergambarkan pada batasan – batasan yang lain.

4.4. Pencarian Solusi Model

Pada tahap ini, dilakukan pencarian solusi model dengan *random initial solution* dan *Iterative Local Search with Hill Climbing*. *Random initial solution* digunakan untuk mencari solusi layak awal, kemudian dari solusi yang layak tersebut, dipilih beberapa solusi yang akan di optimalkan.

Berikut dijabarkan mengenai input, proses, dan output dalam penelitian ini.

4.4.1. Perancangan Input

Yang berperan sebagai input dalam pencarian solusi adalah parameter berupa tMax (waktu maksimal), dan maksimal iterasi. Selain itu terdapat beberapa variable lain sebagai input seperti jumlah *node*, *node* awal, *node* akhir, waktu tempuh tiap *node*, dan skor tiap *node*.

4.4.2. Perancangan Matriks Waktu Tempuh

Sebelum dilakukan pencarian solusi, data waktu tempuh yang telah dijabarkan di atas perlu di olah agar dapat dibaca sebagai *network model* dalam java. Data waktu tempuh yang disajikan dalam bentuk tabel seperti Tabel 4-5 akan dirubah menjadi bentuk matriks dan disimpan dalam file csv. Pembuatan matriks waktu tempuh, dibuat dengan menginputkan secara manual waktu tempuh ke lokasi matriks dan dibantu dengan algoritma Dijkstra. Skor tiap *node* juga dibuatkan file csv dengan 1 nilai per barisnya.

4.4.3. Perancangan Solusi Layak Awal

Pada tahap ini akan dibentuk solusi layak awal dengan node yang bervariasi. Pembentukan solusi layak awal dilakukan berdasarkan pembentukan populasi pada metode optimasi seperti GA. Secara detail, pembentukan solusi layak awal dibuat berdasarkan pembentukan populasi oleh Tasgetiren (2001) [32]. Kode 4-1 merupakan *pseudocode* untuk pencarian solusi layak awal pada penelitian ini.

```
Define tMax
Define maxLoop
Set loop = 0
Do {
      Produce a random number, R_N, between
      [1,N]
      Produce a list, R_L, between [1,N]
      randomly
      Generate a sub list, R_S, by taking the
      first R_N part of the random list R_L
      Compute the total distance, d_RS, of
      the sub list R_S
      If d_RS <=DMAX, then loop=loop+1
      } while (loop<=maxloop)
```

Kode 4-1 Pseudo Code Pencarian Solusi Layak Awal

Proses pencarian solusi dimulai dengan menentukan tMax sebagai batasan total waktu tempuh maksimal dari solusi dan maxloop sebagai maksimal iterasi pencarian solusi. Kemudian temukan solusi layak awal dengan menggunakan algoritma *random*. Bentuk list RL yang berisikan *node* 1 – N kemudian bentuk sublist yang mengambil sebagian nilai dari list RL. Solusi layak awal adalah solusi yang tidak melebihi tMax.

4.4.4. *Roulette Wheel Selection* dan *Iterative Local Search*

Dari solusi awal yang telah dihasilkan, di ambil beberapa solusi yang akan dilakukan *local search*. Pemilihan solusi dilakukan dengan menerapkan algoritma *roulette wheel selection*. Algoritma r*oulette wheel* memberikan setiap solusi probabilitas yang berbeda – beda, dimana semakin tinggi skor yang dimiliki suatu solusi, probabilitas terpilihnya solusi tersebut akan semakin tinggi. Namun tidak menutup kemungkinan bahwa solusi yang memiliki skor kecil juga terpilih sebagai solusi yang akan dioptimalkan.

Solusi yang dipilih berdasarkan *roulette wheel selection* akan di optimalkan dengan melakukan *iterative local*

search pada masing - masing solusi. Metode pencarian lokal yang diterapkan dalam penelitian ini adalah *swap method*, *eliminate method*, dan *insertion method*. Dari hasil pencarian lokal, dipilihlah solusi terbaik yaitu solusi dengan nilai waktu tempuh terpendek dan skor tertinggi.

4.4.5. Perancangan Output

Output dari hasil pencarian solusi adalah jalur yang harus dilewati dimulai dari titik a ke titik b (misal sesuai dengan studi kasus di subbab 4.3 adalah dari Terminal Joyoboyo ke Jembatan Merah Plaza), serta total waktu tempuh dan total skor untuk solusi tersebut. Output dalam pencarian solusi ini berbentuk text dengan format sebagai berikut:

```
[a, b1, b2, … , bn, c](d, e)
```

Kurung siku […] menunjukkan *node* yang dikunjungi, sedangkan tanda kurung (…) menunjukkan total waktu tempuh dan skor pada solusi tersebut. Berikut merupakan notasinya:

- **a:** *node* awal
- **b1, b2, …, bn:** Rute yang dilewati menuju *node* akhir
- **c:** *node akhir*
- **d:** total waktu tempuh pada solusi terkait
- **e:** total skor pada solusi terkait

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Pada bab ini berisi tentang proses implementasi dalam pembuatan sistem. Proses implementasi dilakukan dengan menggunakan bahasa *Java*.

5.1. Pre-processing Data

Tahap pre-processing data dilakukan dengan mengolah data waktu tempuh antar node agar dapat dibaca dalam *java*. Pengolahan dimulai dengan membentuk matriks waktu tempuh antar *node* menggunakan Microsoft Excel, kemudian dilakukan pencarian waktu tempuh terpendek dari 1 *node* ke seluruh *node* dengan menggunakan algoritma Dijkstra.

5.1.1. Pembentukan Matriks Waktu Tempuh

Hasil rekapan data waktu tempuh tiap *node* yang diperoleh dari *Google Maps* masih tidak dapat dibaca sebagai *network model* oleh *java* karena formatnya yang masih berbentuk tabel pada Excel. Oleh karena itu, perlu dilakukan pengolahan data agar dapat digunakan untuk proses pencarian solusi.

Maka dari itu, dibentuklah matriks yang menunjukkan waktu tempuh dari satu *node* ke *node* yang lainnya. Matriks ini akan berbentuk tabel sebanyak 199 kolom dan 199 baris yang merepresentasikan posisi tiap *node* sesuai dengan *network model* ke enam jalur trayek angkot pada studi kasus ini. Setiap *cell* berisikan nilai waktu tempuh yang dibaca dari baris ke kolom. Agar lebih mudah dipahami Tabel 5-1 merupakan potongan dari matriks tersebut.

Tabel 5-1 merupakan potongan matriks waktu tempuh untuk *node* 1 hingga *node* 10. Dimana setiap nilai pada matriks tersebut menunjukkan lama waktu tempuh dalam satuan menit. Misal pada tabel, waktu tempuh dari *node* 1 ke *node* 2 adalah 2 menit, kemudian waktu tempuh dari *node* 2 ke *node* 1 adalah 1 menit dan seterusnya. Nilai 0 pada matriks menunjukkan waktu tempuh pada jalur tersebut masih belum diketahui. Nilai 0

tersebut akan digantikan dengan nilai waktu tempuh terpendek di tahapan berikutnya.

Tabel 5-1 Potongan Matriks Waktu Tempuh

	1	2	3	4	5	6	7	8	9	10
1	0	2	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0
5	0	0	0	1	0	2	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	1	0	2	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

Matriks Tabel 5-1 kemudian di simpan dalam format *Comma Separated Value* atau .csv untuk digunakan dalam pencarian jalur tercepat dengan algoritma Dijsktra.

5.1.2. Penerapan Algoritma Dijkstra

Dengan menggunakan algoritma *random* dalam pencarian solusi layak awal, matriks waktu tempuh seperti di atas masih tidak dapat digunakan sebagai input. Matriks tersebut masih belum berisikan informasi waktu tempuh dari satu *node* ke seluruh *node* lainnya. Dimana hal tersebut akan menyulitkan algoritma dalam pencarian solusi awal atau *initial feasible solution.*

Agar dapat menemukan *initial feasible solution* dengan lebih baik, seluruh waktu tempuh tercepat pada seluruh *node* perlu diketahui. Pencarian waktu tempuh tercepat dilakukan dengan menggunakan algoritma Dijkstra. Algoritma tersebut akan mencari jalur terpendek dari satu sumber, ke seluruh

tempat tujuan atau *node*. Dalam pengaplikasiannya pada *java* digunakan kode – kode berikut:

```
1. static int jumlNode = 199;
2. static int src = 0;
```

Kode 5-1 Deklarasi jumlah node dan node sumber

Kode 5-1 mendeklarasi variable jumlah *node* yang digunakan sebagai batasan dan variable source sebagai *node* sumber yang akan dicari waktu tempuhnya.

```
1. int minDistance(int dist[], Boolean
   sptSet[]) {
2. int min = Integer.MAX_VALUE, min_index =
   0;
3. for (int v = 0; v < jumlNode; v++) {
4. if (sptSet[v] == false && dist[v] <= min)
   {
5. min = dist[v];
6. min_index = v;
7. }
8. }
9. return min_index;
10.   }
```

Kode 5-2 Fungsi Pencarian Waktu Tempuh Minimum

Kode 5-2 berfungsi untuk menemukan *node* dengan nilai waktu tempuh minimum dari pada *node* yang belum masuk ke dalam set *node*, atau dalam kasus ini, *node* tersebut belum memiliki nilai waktu tempuh terpendek.

```
11.   void printSolution(int dist[], int n) {
12.   for (int i = 0; i < jumlNode; i++) {
13.   System.out.print(dist[i] + ",");
14.           }
15.       }
```

Kode 5-3 Fungsi Cetak Hasil Pencarian

Kode 5-3 berfungsi untuk mencetak seluruh solusi yang dihasilkan dari algoritma Dijkstra, dengan koma (,) sebagai pemisah.

```
1. void dijkstra(int graph[][], int src) {
2. int dist[] = new int[jumlNode];
3. Boolean sptSet[] = new Boolean[jumlNode];
4. for (int i = 0; i < jumlNode; i++) {
5. dist[i] = Integer.MAX_VALUE;
6. sptSet[i] = false;
7. }
8. dist[src] = 0;
9. for (int count = 0; count < jumlNode - 1;
   count++) {
10.   int u = minDistance(dist, sptSet);
11.   sptSet[u] = true;
12.   for (int v = 0; v < jumlNode; v++)
13.   {
14.   if (!sptSet[v] && graph[u][v] != 0
15.   && dist[u] != Integer.MAX_VALUE
16.   && dist[u] + graph[u][v] < dist[v]) {
17.   dist[v] = dist[u] + graph[u][v]; }
18.   }
19.   }
20.   printSolution(dist, jumlNode);
21.   }
```

Kode 5-4 Algoritma Dijkstra

Kode 5-4 merupakan algortima Dijkstra. Pertama program membuat array satu dimensi untuk menyimpan nilai waktu tempuh. Kemudian terdapat Boolean sptSet yang akan bernilai *true* jika jarak terpendek dari *node* sumber ke node tujuan sudah final. Array sptSet berfungsi untuk menyimpan *node* yang waktu tempuhnya telah dihitung dan final.

Pada baris 4 sampai 7 terdapat fungsi *loop* yang digunakan untuk mengawali seluruh isi array kecuali node sumber bernilai *infinite* dan sptSet bernilai *false*. Baris 8 menunjukkan jarak *node* sumber dengan *node* yang sama selalu bernilai 0. Kemudian baris 9 sampai 19 merupakan fungsi untuk menemukan jalur terpendek dari *node* sumber ke seluruh *node*. nilai jarak pada array dist[] akan di update apabila *node* terkait belum termasuk dalam array sptSet[], dan sudah merupakan jalur terpendek. Baris 21 memanggil method untuk mencetak hasil algoritma Dijkstra. Setelah algoritma Dijkstra selesai dikonfigurasi, berikutnya adalah memasukkan tabel matriks waktu tempuh ke dalam program java.

```
1. static int[][] arrNode = new
   int[jumlNode][jumlNode];
2. String thisLine;
3. BufferedReader nodeMatrix = new
   BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Documents\
   \NetBeansProjects\\ShortestPath\\Network
   Model Dijkstra.csv"));
4. ArrayList<String[]> lines = new
   ArrayList<>();
5. while ((thisLine =
   nodeMatrix.readLine()) != null) {
6. lines.add(thisLine.split(","));
7.         }
8. String[][] array = new
   String[lines.size()][0];
9. lines.toArray(array);
10.   for (int i = 0; i < array.length; i++)
   {
```

```
11.   for (int j = 0; j < array.length; j++)
   {
12.   arrNode[i][j] =
   Integer.parseInt(array[i][j]);
13.               }
14.           }
```

Kode 5-5 Import Matriks Waktu Tempuh

Kode 5-5 berfungsi untuk membaca file csv yang berisikan matriks waktu tempuh yang telah dijabarkan pada sub bab 5.1.1. Program mencari file .csv dengan nama Network Model Dijkstra.csv, kemudian membaca file tersebut per baris dan memasukkannya ke dalam string array dengan koma sebagai pemisah. Setelah itu dilakukan parsing dari string ke integer untuk seluruh nilai dan masukkan hasil parsing ke dalam array bernama arrNode. Proses tersebut ditunjukkan pada baris 10 sampai 12.

```
1. ShortestPath t = new ShortestPath();
2. for (int i = 0; i < jumlNode; i++) {
3. t.dijkstra(arrNode, i);
4. System.out.println();
5. }
```

Kode 5-6 Fungsi Memanggil Mehtod Dijkstra

Kode 5-6 berfungsi untuk memanggil method Dijkstra dengan matriks waktu tempuh dan node sumber sebagai input. Method Dijkstra dipanggil secara berulang dengan *node* sumber yang berbeda pada setiap perulangan sehingga menghasilkan matriks waktu tempuh baru. Matriks tersebut disimpan dengan nama networkDijkstra.csv.

5.2. Pencarian Solusi Model OP

Setelah matriks waktu tempuh siap digunakan, pencarian solusi dapat dilakukan. Pencarian solusi dimulai dengan pengambilan solusi layak awal secara *random*, kemudian solusi tersebut di seleksi menggunakan *roulette wheel selection*, setelah itu dilakukan optimasi dari hasil seleksi solusi dengan menggunakan *iterative local search with hill climbing*.

5.2.1. Deklarasi Variable dan Import Matriks

Agar pencarian solusi dapat dilakukan, *Java* perlu mengetahui struktur *network model* yang akan dicari solusinya. Maka dari itu, perlu di deklarasikan variable apa, dan data apa yang akan digunakan. Sebelum memasukkan data ke dalam java, deklarasikan juga sebuah array sebagai tempat penyimpanan data tersebut.

```
1. static int jumlNode = 199;
2. static int nodeAwal = 1;
3. static int nodeAkhir = 91;
4. static int[][] arrNode = new
   int[jumlNode][jumlNode];
5. static int[] scoreNode = new
   int[jumlNode];
```

Kode 5-7 Deklarasi Variabel dan Array

Kode 5-7 menunjukkan bahwa model memiliki 199 *node*, dan solusi yang akan dicari pada contoh ini adalah jalur optimum dari *node* 1 ke *node* 91. Array dengan nama arrNode digunakan untuk menyimpan nilai waktu tempuh pada keseluruhan *network model*. Sedangkan array dengan nama scoreNode digunakan untuk meyimpan nilai skor pada setiap *node*.

Selain ke dua array tersebut, perlu di deklarasikan arraylist yang digunakan untuk menyimpan *initial feasible solution* dan hasil seleksi *roulette wheel selection*

```
1. static ArrayList<ArrayList<Integer>>
   feasibleSol = new ArrayList<>();
2. static ArrayList<Integer>
   feasibleSolJarak = new ArrayList<>();
3. static ArrayList<Integer>
   feasibleSolScore = new ArrayList<>();
4. static ArrayList<ArrayList<Integer>>
   pickSolution = new ArrayList();
5. static ArrayList<Integer> pickSolJarak =
   new ArrayList<>();
6. static ArrayList<Integer> pickSolScore =
   new ArrayList<>();
```

Kode 5-8 Deklarasi ArrayList

Kode 5-8 merupakan deklarasi arraylist dengan nama feasibleSol, feasibleSolJarak, dan feasibleScore yang akan digunakan sebagai tempat untuk menyimpan *initial feasible solution*. Dimana feasibleSol akan menyimpan solusi jalur trayek, feasibleSolJarak menyimpan total waktu tempuh dari solusi, dan feasibleSolScore menyimpan total skor dari solusi.

Sedangkan arraylist pickSolution, pickSolJarak, dan pickSolScore digunakan untuk menyimpan hasil seleksi solusi dari *roulette wheel selection*. pickSolution akan menyimpan solusi jalur trayek, pickSolJarak menyimpan total waktu tempuh solusi, dan pickSolScore menyimpan total skor dari solusi.

Setelah terbentuknya array, dilakukan import matriks waktu tempuh agar dapat digunakan sebagai input dalam pencarian solusi.

```
1. String thisLine;
2. BufferedReader nodeMatrix = new
   BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Documents\
   \NetBeansProjects\\SolusiOP\\networkDijk
   stra.csv"));
3. ArrayList<String[]> lines = new
   ArrayList<>();
4. while ((thisLine =
   nodeMatrix.readLine()) != null) {
5. lines.add(thisLine.split(","));
6. }
7. String[][] array = new
   String[lines.size()][0];
8. lines.toArray(array);
9. for (int i = 0; i < array.length; i++) {
10.   for (int j = 0; j < array.length;
   j++) {
11.   arrNode[i][j] =
   Integer.parseInt(array[i][j]);
12.               }
13.           }
```

Kode 5-9 Import Matriks Waktu Tempuh 2

Kode 5-9 akan membaca file matriks waktu tempuh per *node* dan memasukkannya sebagai string ke dalam array bernama lines. Kemudian dari array lines tersebut, data dimasukkan ke dalam array 2 dimensi arrNode yang telah di deklarasikan sebelumnya. Karena nilai dari array lines masih ber tipe string, perlu dilakukan parsing menjadi integer yang ditunjukkan pada baris 11. Dengan begini, nilai waktu tempuh telah tersimpan dalam array arrNode.

Selain nilai waktu tempuh, nilai skor pada tiap node juga perlu di import untuk melengkapi model OP.

```
1. BufferedReader nodeScore = new
   BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Documents\
   \NetBeansProjects\\SolusiOP\\scoreNode.c
   sv"));
2. String barisIni;
3. int isi = 0; //representasi isi array
4. while ((barisIni = nodeScore.readLine())
   != null) {
5. scoreNode[isi] =
   Integer.parseInt(barisIni);
6. isi++;
7.          }
```

Kode 5-10 Import Skor Tiap Node

Kode 5-10 merupakan pengambilan nilai dari file scoreNode.csv. Kode tersebut serupa dengan kode untuk membaca nilai waktu tempuh di atas, namun karena file scoreNode.csv hanya memiliki satu nilai pada satu baris, kita dapat langsung memasukkan nilai ke dalam array 1 dimensi scoreNode.

5.2.2. *Initial Feasible Solution*

Setelah seluruh data input dapat dibaca oleh *java*, langkah berikutnya adalah menemukan solusi layak awal yang nantinya akan di optimalkan menggunakan *iterative local search with hill climbing*.

Pertama, deklarasikan dulu waktu maksimal, dan iterasi maksimal dari pencarian solusi layak awal.

```
1. int tMax = 60;
2. int maxLoop = 1000000;
3. int loop = 1;
```

Kode 5-11 Deklarasi Parameter tMax dan Maksimal Iterasi

Kode 5-11 menunjukkan pencarian solusi dilakukan dengan 1.000.000 kali iterasi dan lamanya perjalanan dari *node* awal ke *node* akhir adalah maksimal 60 menit. Setelah mendeklarasikan waktu maksimal dan iterasi maksimal, langkah berikutnya adalah mulai lakukan pencarian solusi. Pencarian solusi dilakukan berdasarkan batasan – batasan dari OP seperti:

- Solusi tidak boleh melebihi tMax atau waktu maksimal
- Rute harus dimulai dari *node* awal dan berakhir di *node* akhir
- Seluruh *node* harus terhubung

```
1. do {
2. int Rn = new Random().nextInt(jumlNode -
   2) + 1;
3. ArrayList<Integer> listRl = new
   ArrayList();
4. ArrayList<Integer> listRl1 = new
   ArrayList();
5. for (int i = 1; i < jumlNode + 1; i++) {
6. if (i == nodeAwal || i == nodeAkhir) {
7. continue;
8. }
9. listRl1.add(i);
10.   }
11.   Collections.shuffle(listRl1);
12.   listRl.add(nodeAwal);
13.   for (int i = 0 ; i < listRl1.size();
   i++) {
14.   listRl.add(listRl1.get(i));
15.   }
16.   listRl.add(nodeAkhir);
```

Kode 5-12 Membuat ArrayList yang Berisikan Rute Node Awal Sampai Node Akhir

Array listRl berguna untuk menyimpan seluruh *node* dalam bentuk arraylist. Sedangkan listRl1 digunakan untuk

menyimpan *node* yang akan di acak posisinya. Arraylist listRl1 tidak berisikan *node* awal dan *node* akhir, karena array tersebut merepresentasikan jalur – jalur yang dapat dilewati menuju *node* akhir.

Baris 6 ke 8 memastikan bahwa listRl1 tidak berisikan *node* awal dan *node* akhir. Baris 9 berfungsi memasukkan seluruh *node* (kecuali *node* awal dan *node* akhir) ke dalam arraylist listRl1. Setelah listRl1 berisikan keseluruhan *node*, listRl1 akan diacak posisinya dengan fungsi pada baris 11. Kemudian beris 12 sampai 16 merupakan fungsi untuk memasukkan *node* awal, listRl1, dan *node* akhir ke dalam listRl. Dengan begini listRl akan berisikan seluruh *node* dari 1 sampai 199 dan rute dimulai dari *node* awal dan berakhir di *node* akhir. Langkah berikutnya adalah membentuk sebuah arraylist yang memilih dan menyimpan rute mana saja untuk menuju *node* akhir

```
1. ArrayList<Integer> subListRs = new
   ArrayList();
2. subListRs.add(listRl.get(0));
3. for (int i = 1; i < (1 + Rn); i++) {
4. subListRs.add(listRl.get(i));
5. }
6. subListRs.add(listRl.get(listRl.size() -
   1));
```

Kode 5-13 Pengambilan Calon Solusi Layak

Kode 5-13 akan menyimpan sebagian *node* dari arryalist listRl. Array subListRs ini berperan sebagai tempat penyimpanan calon solusi yang layak. Baris 2 mengambil index ke 0 dari listRl yaitu *node* awal, dan baris ke 6 mengambil *node* terakhir dari listRl yang menunjukkan *node* akhir. Baris 3 sampai 4 merupakan fungsi menambahkan *node* secara random yang merepresentasikan *node* mana saja yang akan dilewati untuk menuju ke *node* akhir. Panjang arraylist subListRs akan berbeda – beda pada setiap iterasi. Tiap iterasi

akan menghasilkan satu calon solusi layak dan disimpan ke dalam subListRs. Sebelum kita dapat menemukan solusi layak awal, kita perlu sebuah method yang berfungsi untuk menghitung total waktu tempuh suatu array (solusi).

```
1. public static int
   kalkulasiJarak(ArrayList<Integer>
   hitungList) {
2. int sum = 0;
3. for (int i = 0; i < hitungList.size() -
   1; i++) {
4. sum = sum + arrNode[hitungList.get(i) -
   1][hitungList.get(i + 1) - 1];
5.          }
6. return sum;
7.      }
```

Kode 5-14 Method Untuk Menghitung Waktu Tempuh

Kode 5-14 merupakan method untuk menghitung total waktu tempuh pada suatu array. Deklarasi sum pada baris ke 2 berfungsi untuk menjumlah setiap waktu tempuh dari satu *node* ke *node* lainnya. Pengambilan nilai waktu tempuh ditunjukkan pada baris 3 sampai baris 5. Ketika method ini dipanggil, maka method akan mengeluarkan total waktu tempuh pada array terkait. Berikutnya adalah konfigurasi untuk solusi layak awal. Dimana solusi dinyatakan layak apabila waktu tempuh kurang dari waktu maksimal.

```
1. for (int i = 0; i < subListRs.size();
   i++) {
2. if (kalkulasiJarak(subListRs) < tMax &&
   !feasibleSol.contains(subListRs)) {
3. feasibleSol.add(subListRs);
4.                 }
5.             }
6. loop++;
7. } while (loop <= maxLoop);
```

Kode 5-15 Perhitungan Waktu Tempuh Feasible Solution

Kode 5-15 akan mencari solusi dari subListRs dengan nilai waktu tampuh kurang dari tMax dan solusi yang sudah di dapatkan tidak perlu disimpan kembali ke dalam arraylist feasibleSol. Fungsi di atas akan terus berjalan hingga maksimal *loop* tercapai. Dengan begini solusi yang memiliki waktu tempuh kurang dari tMax akan dianggap sebagai solusi layak awal atau *initial feasible solution*.

Agar dapat mengetahui berapa skor yang dimiliki pada tiap solusi, kita perlu membuat method untuk menghitung skor di masing – masing solusi layak tersebut

```
1. public static int
   kalkulasiScore(ArrayList<Integer>
   varListRl) {
2. int sum = 0;
3. for (int i = 0; i < varListRl.size();
   i++) {
4. sum = sum + scoreNode[varListRl.get(i) -
   1];
5. }
6. return sum;
7.     }
```

Kode 5-16 Method Untuk Menghitung Skor

Kode 5-16 merupakan method untuk menghitung skor pada masing – masing solusi yang dihasilkan pada suatu array. Kode tersebut serupa dengan perhitungan waktu tempuh namun dengan pengambilan nilai pada array 1 dimensi saja. Setelah method perhitungan waktu tempuh dan skor dibuat, kita dapat memasukkan hasil perhitungan ke dalam array feasibleSolJarak dan feasibleSolScore. Ke dua array tersebut berfungsi untuk menyimpan total waktu tempuh dan skor pada masing – masing solusi yang telah dinyatak layak.

```
1. for (int i = 0; i < feasibleSol.size();
   i++) {
2. feasibleSolJarak.add(kalkulasiJarak(feas
   ibleSol.get(i)));
3. feasibleSolScore.add(kalkulasiScore(feas
   ibleSol.get(i)));
4. }
```

Kode 5-17 Menyimpan Hasil Perhitungan Waktu Tempuh dan Skor

Kode 5-17 merupakan fungsi untuk menyimpan hasil perhitungan waktu tempuh dan score pada array feasibleSol kemudian memasukkan hasil perhitungan ke array waktu tempuh dan array score. Setelah terbentuknya array yang berisikan jalur yang layak, waktu tempuh, dan skor. Kita dapat memanggilnya dengan kode berikut

```
1. for (int i = 0; i < feasibleSol.size();
   i++) {
2. System.out.println(feasibleSol.get(i) +
   "(" + feasibleSolJarak.get(i) + ", " +
   feasibleSolScore.get(i) + ")");
3.          }
```

Kode 5-18 Cetak Initial Feasible Solution

5.2.3. *Roulette Wheel Selection*

Karena hasil dari *initial feasible solution* merupakan populasi dan jumlahnya sangat banyak, perlu dilakukan pemilihan solusi yang nantinya akan di optimasi dengan *iterative local search*. Seleksi dilakukan menggunakan *roulette wheel selection* karena sifatnya yang tidak melulu memilih solusi yang paling baik, dan juga tidak memilih secara acak. *Roulette wheel selection* akan memberikan probabilitas yang berbeda – beda pada setiap solusi, tergantung pada skor masing – masing. Semakin tinggi skor yang dimiliki suatu solusi, maka probabilitas solusi tersebut terpilih semakin besar. Suatu skor dapat dikatakan tinggi apabila skor tersebut berada di atas rata – rata skor solusi layak lainnya.

Maka dari itu, sebelum kita mulai membangun fungsi *roulette wheel selection*, terlebih dahulu kita harus membuat method untuk menghitung total skor dari solusi layak yang dihasilkan

```
1. public static int totalScore() {
2. int sum = 0;
3. for (int i = 0; i <
   feasibleSolScore.size(); i++) {
4. sum = sum + feasibleSolScore.get(i);
5.          }
6.          return sum;
7.      }
```

Kode 5-19 Method Untuk Menghitung Total Score dari Seluruh Initial Feasible Solution

Kode 5-19 akan menjumlahkan seluruh skor yang disimpan pada array feasibleSolScore. Method ini digunakan saat melakukan perhitungan *roulette wheel* di bawah ini

```
1. public static int roulleteWheelSelect()
   {
2. double rand = new Random().nextDouble();
3. int totScore = totalScore();
4. int k = 0;
5. double sum = (double)
   feasibleSolScore.get(k) / totScore;
6. while (sum < rand) {
7. k++;
8. sum = sum + (double)
   feasibleSolScore.get(k) / totScore;
9.          }
10.   return k;
11.       }
```

Kode 5-20 Method Roulette Wheel Selection

Kode 5-20 merupakan method dari *roulette wheel selection*. Pertama – tama hasilkan bilang random antara – sampai 1 (ditunjukkan pada baris 2). Kemudian hitung probability dengan membagi skor pada solusi ke- k dengan total skor dari Kode 5-19 (ditunjukkan pada baris 5) dimana k merupakan solusi pada index ke – 0 sampai index terakhir. Apabila hasil probabilitas lebih besar dari bilangan random yang di-generate maka solusi tersebut akan terpilih. Jika tidak, solusi tersebut dilewatkan dan pencarian berlanjut ke solusi pada index berikutnya. Setelah method selesai dibentuk, lakukan pemilihan solusi dengan memanggil method *roulleteWheelSelect()*.

```
1. int size = 30;
2. do {
3. pickSolution.add(feasibleSol.get(roullet
   eWheelSelect()));
4. } while (pickSolution.size() < size);
```

Kode 5-21 Fungsi Seleksi Solusi

Kode 5-21 merupakan fungsi untuk menyeleksi solusi dari array feasibleSol. Solusi yang lolos seleksi dimasukkan ke dalam array pickSolution sejumlah size. Berdasarkan kode di atas, dari keseluruhan solusi layak awal yang berhasil ditemukan, akan dipilih sebanyak 30 solusi untuk dioptimasi. Kemudian lakukan perhitungan waktu tempuh dan skor pada array pickSolution.

```
1. for (int i = 0; i < pickSolution.size();
   i++) {
2. pickSolJarak.add(kalkulasiJarak(pickSolu
   tion.get(i)));
3. pickSolScore.add(kalkulasiScore(pickSolu
   tion.get(i)));
4. System.out.println(pickSolution.get(i) +
   "(" + pickSolJarak.get(i) + ", " +
   pickSolScore.get(i) + ")");
5.          }
```

Kode 5-22 Menghitung Waktu Tempuh dan Skor dari Hasil Seleksi

Kode 5-22 berfungsi untuk menghitung waktu tempuh dan skor dari tiap solusi yang telah diseleksi. Baris 2 merupakan fungsi untuk menyimpan total waktu tempuh pada solusi ke-i. Baris 3 merupakan fungsi untuk menyimpan total skor pada solusi baris ke-i. Baris 4 akan mencetak solusi yang lolos seleksi beserta dengan lama waktu tempuh dan besarnya skor solusi tersebut.

5.2.4. *Iterative Local Search*

Dari hasil seleksi solusi yang telah dijabarkan sebelumnya, seluruh hasil seleksi tersebut akan di optimasi dengan cara pencarian lokal. Pencarian lokal yang dilakukan pada penelitian ini adalah *swap method*, *insertion method*, dan *deletion method*. Pencarian lokal ini dilakukan secara berulang hingga maksimal *loop* tercapai. Pada setiap iterasi, *node* atau rute yang ada pada tiap solusi akan di hilangkan, di tukar, atau di tambahkan sebuah rute/*node*. Apabila solusi hasil pencarian

lokal memiliki waktu tempuh yang lebih sedikit, atau memiliki skor yang lebih tinggi, maka solusi baru tersebut menggantikan solusi pada iterasi sebelumnya.

Pemilihan metode pencarian lokal dipilih secara random menggunakan *switch case*. Dimana *case* 1 adalah *swap method*, *case* 2 adalah *insertion method*, dan *case* 3 adalah *deletion method*. Pertama lakukan inisiasi looping dan tentukan maksimal *loop*.

```
1. for (int i = 0; i < pickSolution.size();
   i++) {
2. int counterSearch = 1;
3. int maxSearchLoop = 1000000;
```

Kode 5-23 Deklarasi Local Search dan Maksimal Iterasi

Baris 1 merupakan inisiasi looping agar dilakukan pencarian lokal untuk seluruh solusi dari hasil seleksi. Baris 2 merupakan deklarasi *counter* untuk pencarian lokal, dan baris 3 merupakan maksimal iterasi pada pencarian lokal. Kemudian dibuatlah *switch case* untuk tiap – tiap metode pencarian lokal.

```
1. do {
2. int randCase = new Random().nextInt(3) +
   1;
3. switch (randCase) {
4. case 1: //swap
5. int swap1 = new
   Random().nextInt(pickSolution.get(i).siz
   e() - 2) + 1;
6. int swap2 = new
   Random().nextInt(pickSolution.get(i).siz
   e() - 2) + 1;
7. Collections.swap(pickSolution.get(i),
   swap1, swap2);
8. int currentJarakSwap =
   kalkulasiJarak(pickSolution.get(i));
```

```
15.   int currentScoreSwap =
   kalkulasiScore(pickSolution.get(i));
16.   if (currentJarakSwap <
   pickSolJarak.get(i)) {
17.   pickSolJarak.set(i, currentJarakSwap);
18.   pickSolScore.set(i, currentScoreSwap);
19.   counterSearch = 1;
20.   } else {
21.   Collections.swap(pickSolution.get(i),
   swap2, swap1);
22.                   }
23.   break;
```

Kode 5-24 Local Search Swap Method

Kode 5-24 dimulai dengan inisiasi looping *do* pada baris 1. Kemudian dilakukan deklarasi nilai random randCase dengan random antara 1 sampai 3. Apabila *case* bernilai 1 maka pada iterasi ini *swap method* akan dilakukan, apabila *case* bernilai 2 maka pada iterasi ini *insertion method* akan dilakukan, sedangkan jika *case* bernilai 3 maka pada iterasi ini *deletion method* akan dilakukan.

Swap method dimulai pada baris 4 dan berakhir di baris 17. Pada *swap method* ini, deklarasikan dahulu 2 nilai random yang berkisar antara 1 hingga besarnya solusi (dikurangi dengan *node* awal dan *node* akhir). Nilai random ini berfungsi sebagai representasi index yang akan ditukarkan. Deklarasi nilai random ditunjukkan pada baris 5 dan 6 dengan nama variable untuk masing – masing nilai adalah swap1 dan swap2. Baris 7 merupakan fungsi *swap* itu sendiri, dimana setiap solusi di pickSolution akan dilakukan pertukaran pada index swap1 dan index swap2.

Setelah *swap* terjadi, lakukan perhitungan waktu tempuh dan skor. Perhitungan waktu tempuh dan skor ditunjukkan pada baris 8 dan 9. Apabila setelah dilakukan *swap* waktu tempuhnya lebih sedikit daripada waktu tempuh sebelumnya, *update* waktu tempuh dan skor solusi sebelumnya

menjadi solusi saat ini (solusi setelah dilakukan *swap*). *Update* solusi ditunjukkan pada baris 11 dan 12. Kemudian nilai counterSearch di set kembali menjadi 1, hal ini dilakukan agar pencarian solusi menjadi lebih optimal. Sedangkan jika waktu tempuh saat ini lebih lama dibanding waktu tempuh solusi sebelumnya, maka pertukaran dibatalkan dan solusi dikembalikan seperti pada iterasi sebelumnya. Fungsi tersebut ditunjukkan pada baris 14 sampai 16. *Swap method* berhenti di baris ke 17.

```
1. case 2: //insertion
2. int randIndex = new
   Random().nextInt(pickSolution.get(i).siz
   e() - 2) + 1;
3. int randNode = new
   Random().nextInt(jumlNode) + 1;
4. if (randNode == nodeAwal || randNode ==
   nodeAkhir ||
   pickSolution.get(i).contains(randNode))
   {
5. continue;
6. }
7. pickSolution.get(i).add(randIndex,
   randNode);
8. int currentJarakInsert =
   kalkulasiJarak(pickSolution.get(i));
9. int currentScoreInsert =
   kalkulasiScore(pickSolution.get(i));
10.  if (currentJarakInsert <=
     pickSolJarak.get(i) &&
     currentScoreInsert >=
     pickSolScore.get(i)) {
11.  pickSolJarak.set(i,
     currentJarakInsert);
12.  pickSolScore.set(i,
     currentScoreInsert);
13.   counterSearch = 1;
14.   } else {
```

```
15.   pickSolution.get(i).remove(randIndex)
   ;}
16.   break;
```

Kode 5-25 Local Search Insertion Method

Insertion method dimulai dengan mendeklarasikan variable random randIndex. Variable tersebut berfungsi untuk menunjukkan pada index ke berapa sebuah *node* akan dimasukkan. Kemudian mendeklarasikan variable random randNode. Variable randNode merupakan *node* yang akan dimasukkan ke dalam solusi. Deklarasi nilai random ditunjukkan pada baris 2 dan 3. Baris 4 sampai 6 berfungsi untuk memastikan nilai random yang dihasilkan randNode tidak sama dengan *node* awal dan juga *node* akhir, selain itu juga memastikan randNode tidak menghasilkan nilai yang sama dengan yang telah ada pada pickSolution.

Baris 7 merupakan fungsi dari *insert* itu sendiri. Nilai dan index yang masuk ke dalam solusi adalah random. Setelah dilakukan *insertion*, hitung waktu tempuh dan skor yang baru. Perhitungan waktu tempuh dan skor ditunjukkan pada baris 8 dan 9. Apabila setelah dilakukan *insertion* waktu tempuh menjadi lebih kecil atau sama dengan solusi seblumnya, dan skor menjadi lebih besar atau sama dengan skor sebelumnya, maka lakukan *update* nilai waktu tempuh dan skor solusi sebelumnya, menjadi waktu tempuh dan skor solusi saat ini (solusi setelah dilakukan *insertion*). Kemudian nilai counterSearch di set kembali menjadi 1, hal ini dilakukan agar pencarian solusi menjadi lebih optimal. Sedangkan apabila waktu tempuh menjadi lebih lama, dan skor menjadi lebih sedikit, maka *insertion* dibatalkan, dan *node* yang baru saja dimasukkan akan dihapus. Penghapusan *node* ditunjukkan pada baris 15. *Insertion method* berhenti pada baris 16.

```
1. case 3: //deletion
2. int delRandom = new
   Random().nextInt(pickSolution.get(i).siz
   e() - 2) + 1;
3. if (pickSolution.get(i).size() > 3 ){
4. int temp =
   pickSolution.get(i).get(delRandom);
5. pickSolution.get(i).remove(delRandom);
6. int currentJarakDel =
   kalkulasiJarak(pickSolution.get(i));
7. int currentScoreDel =
   kalkulasiScore(pickSolution.get(i));
8. if (currentJarakDel <=
   pickSolJarak.get(i) && currentScoreDel
   >= pickSolScore.get(i)) {
9. pickSolJarak.set(i, currentJarakDel);
10.   pickSolScore.set(i, currentScoreDel);
11.   counterSearch = 1;
12.   } else {
13.   pickSolution.get(i).add(delRandom,
   temp);
14.   }
15.   }
16.   break;
17.   }
18.   counterSearch++;
19.   } while (counterSearch <=
   maxSearchLoop);
20.          }
```

Kode 5-26 Local Search Deletion Method

Deletion method dimulai dengan mendeklarasikan variable random delRandom. delRandom berfungsi untuk menunjukkan index ke berapa yang akan dihapus. Setelah itu diberikan kondisi pada baris 3 bahwa solusi yang memiliki panjang kurang dari 3 *node* maka tidak akan dilakukan penghapusan *node* dan iterasi tersebut langsung berlanjut ke

iterasi berikutnya. Hal ini untuk mencegah terjadinya *out of bound exception*.

Masuk ke fungsi *deletion*, pada baris 4 kita mendeklarasikan variable temp yang berfungsi untuk menyimpan *value node* yang dihapus. Variable temp diperlukan saat kita ingin mengembalikan solusi ke kondisi sebelum dilakukan *deletion*. Baris 5 merupakan fungsi dari *deletion* itu sendiri, fungsi tersebut akan menghilangkan 1 *node* secara random kecuali *node* awal dan *node* akhir. Setelah 1 *node* di hapus, lakukan perhitungan waktu tempuh dan skor saat ini, ditunjukkan pada baris 6 dan 7.

Apabila setelah dilakukan *deletion*, total waktu tempuh dalam satu solusi menjadi lebih kecil atau sama dengan solusi sebelumnya, dan skor menjadi lebih besar atau sama dengan skor sebelumnya, maka *update* nilai waktu tempuh dan skor solusi sebelumnya menjadi solusi saat ini (solusi setelah dilakukan *deletion*). Kemudian nilai counterSearch di set kembali menjadi 1, hal ini dilakukan agar pencarian solusi menjadi lebih optimal. Sedangkan apabila solusi setelah dilakukan *deletion* memiliki total waktu tempuh yang lebih tinggi daripada sebelumnya, dan skor lebih kecil dari solusi sebelumnya, maka penghapusan *node* di batalkan. Pembatalan hapusnya *node* ditunjukkan pada baris 13. *Deletion method* berhenti di baris 16.

Setelah *swap, insertion,* atau *deletion method* dilakukan, nilai *counterSearch* akan bertambah. Iterasi akan terus berjalan selama nilai *counterSearch* lebih kecil sama dengan maksimal iterasi yang telah di deklarasikan sebelumnya.

5.2.5. Pengambilan Solusi Terbaik

Setelah *iterative local search* selesai dilakukan, maka seluruh solusi yang terpilih sudah dianggap solusi optimal. Dari banyaknya solusi optimal, akan diambil solusi terbaik dari permasalahan yang dikerjakan. Pengambilan solusi terbaik dilakukan dengan melakukan cek di seluruh hasil optimasi yang

dilakukan pada sub bab 5.2.4., kemudian ambil solusi yang memiliki skor paling tinggi.

```
1. int indexTerbaik = 0;
2. int scoreTerbaik = pickSolScore.get(0);
3. for (int i = 1; i < pickSolution.size();
   i++) {
4. if (pickSolScore.get(i) > scoreTerbaik)
   {
5. indexTerbaik = i;
6. scoreTerbaik = pickSolScore.get(i);
7.                }
8.           }
```

Kode 5-27 Pengambilan Solusi Terbaik

Kode 5-27 dimulai dengan mendaklarasikan variable bernama indexTerbaik. Variable tersebut berfungsi untuk menghitung pada index ke berapa sebuah solusi dinyatakan solusi terbaik. Kemudian variable scoreTerbaik berfungsi untuk menyimpan dan membandingkan solusi yang dicek program. Baris 3 sampai 8 merupakan fungsi pengecekkan dan penyimpanan solusi terbaik. Apabila solusi yang di cek memiliki skor lebih tinggi daripada scoreTerbaik, maka variable indexTerbaik dan scoreTerbaik di *update*. Dengan begini program akan dapat menemukan solusi yang memiliki skor tertinggi dari hasil optimasi.

Solusi terbaik dapat dicetak dengan menggunakan kode berikut

```
System.out.println("Solusi terbaik: " +
pickSolution.get(indexTerbaik) + "(" +
pickSolJarak.get(indexTerbaik) + ", " +
scoreTerbaik + ")");
```

Kode 5-28 Cetak Solusi Terbaik

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan proses pengujian dan analisis terhadap hasil pengujian yang diperoleh dari proses implementasi yang telah dibahas pada bab sebelumnya.

6.1. Lingkungan Uji Coba

Lingkungan uji coba merupakan kriteria perangkat pengujian yang digunakan dalam menguji model yang telah dibuat pada tugas akhir ini. Lingkungan uji coba terdiri dari perangkat keras dan perangkat lunak. Adapun perangkat keras yang digunakan ditunjukkan pada Tabel 6.1:

Tabel 6-1 : Lingkungan uji coba

Perangkat Keras	Spesifikasi
Jenis	*Laptop*
Processor	*Core i5*
RAM	4GB
Hard Disk Drive	500GB

Selain itu juga, terdapat lingkungan perangkat lunak yang digunakan dalam uji coba aplikasi. Tabel 6.2 adalah daftar perangkat lunak yang digunakan dalam uji coba.

Tabel 6-2 : Perangkat lunak yang digunakan

Perangkat Lunak	Fungsi
Windows 10	Sistem operasi
Corel Draw X7	Membuat gambaran jalur dan penentuan node
Netbeans IDE 8.1	Membuat dan mencari solusi model
Java Development Kit 1.8	Bahasa pemrograman yang digunakan
Micorsoft Excel 2013	Perekapan data

6.2. Network Model

Berdasarkan peta digital dan data jalur trayek yang disediakan website resmi Dinas Perhubungan Kota Surabaya, ditentukan lokasi – lokasi *node* pada setiap jalur, dapat dilihat pada LAMPIRAN H. Dari hasil penggambaran peta digital, dibentuk *network model* yang dapat dilihat pada LAMPIRAN I

Network model pada LAMPIRAN I merepresentasikan jalur yang dilalui angkot dengan trayek F, JK, JMK, GS, RT, dan O. *Node* dengan warna oranye menandakan *node* tersebut dilewati oleh dua atau lebih trayek angkot, sedangkan *node* berwarna biru menandakan *node* tersebut hanya dilalui satu jalur trayek angkot untuk studi kasus ini. Terdapat total sebanyak 199 *node* dalam *network model* di atas.

6.3. Hasil Algoritma Dijkstra

Setelah rancangan algoritma Dijkstra tidak terdapat kesalahan dan program dapat dijalankan. Program akan menghasilkan sebuah matriks yang berisikan nilai waktu tempuh untuk seluruh *node*. Tabel 6-3 merupakan potongan tabel matriks hasil keluaran algoritma Dijkstra:

Tabel 6-3 Matriks Hasil Generate Algoritma Dijkstra

	1	2	3	4	5	6	7	8	9	10
1	0	2	3	4	5	7	8	10	11	12
2	1	0	1	2	3	5	6	8	9	10
3	7	6	0	1	2	4	5	7	8	9
4	8	7	1	0	1	3	4	6	7	8
5	9	8	2	1	0	2	3	5	6	7
6	12	11	5	4	3	0	1	3	4	5
7	13	12	6	5	4	1	0	2	3	4
8	13	15	12	11	10	7	6	0	1	2
9	16	17	11	10	9	6	5	7	0	1
10	15	16	10	9	8	5	4	6	7	0

Berdasarkan hasil keluaram algoritma Dijkstra yang ditunjukkan pada Tabel 6-3, dapat dilihat bahwa algoritma Dijkstra dapat menemukan nilai waktu tempuh tercepat dari satu *node* sumber ke seluruh *node* yang ada pada *network model*.

6.4. Testing Pencarian Solusi *Node* 1 menuju *node* 91

Setelah konstruksi program selesai dilakukan, dan telah dipastikan tidak terdapat error, program akan dijalankan dan dilihat hasil pencarian solusinya. Pada sub bab ini akan dilakukan percobaan pencarian rute dari *node* 1 ke *node* 91. Dengan waktu maksimal adalah 60 menit. Percobaan ini merepresentasikan pencarian jalur dari terminal Joyoboyo menuju Jembatan Merah Plaza Surabaya.

6.4.1. Validasi Solusi Layak Awal

Sebelum solusi optimal dapat ditemukan, program akan mencari solusi layak awal. Pencarian solusi layak awal dilakukan sebanyak 1.000.000 kali iterasi. Pada pencarian ini, solusi dapat dikatakan layak apabila tidak melanggar batasan – batasan yang telah ditentukan. Antara lain:

1. Rute dimulai pada *node* 1 dan berhenti pada *node* 91
2. Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali
3. Waktu tempuh dari setiap solusi tidak boleh melebihi 60 menit
4. Tidak adanya subtour

Setelah program di jalankan, pertama – tama program akan mencari solusi yang tidak melanggar ke empat batasan di atas. Kemudian menyimpannya dalam sebuah *array*. Pada percobaan ini, program menemukan 1404 solusi layak awal. Dari ribuan solusi layak awal yang ditemukan, akan diambil sebagian solusi menggunakan *roulette wheel selection*. Seluruh solusi di simpan dalam format yang ditunjukkan Gambar 6-1:

```
[node awal, rute yang dilalui, node
    akhir](waktu tempuh, skor)
            Contoh:
1. [1, 6, 13, 14, 186, 91](47, 1245)
```

Gambar 6-1 Contoh Format Solusi

Gambar 6-1 merupakan format solusi dalam penelitian ini, kurung siku […] merupakan penggambaran rute yang dilewati dari *node* awal menuju *node* akhir. Sedangkan tanda kurung (…) menandakan total waktu tempuh dan skor untuk solusi tersebut. Pada percobaan ini, *roulette wheel selection* akan memilih 30 solusi yang nantinya akan diaplikasikan *local search*. Gambar 6-2 merupakan hasil seleksi *roulette wheel selection*:

```
   ---Hasil Roulette Wheel Selection---
1. [1, 39, 169, 91](45, 605)
2. [1, 21, 99, 91](53, 746)
3. [1, 167, 152, 91](45, 627)
4. [1, 39, 44, 91](58, 423)
5. [1, 185, 14, 91](56, 771)
6. [1, 6, 9, 91](31, 733)
7. [1, 182, 91](29, 441)
8. [1, 59, 99, 91](42, 634)
9. [1, 23, 148, 91](49, 692)
10.  [1, 43, 168, 91](56, 525)
11.  [1, 6, 13, 14, 186, 91](47, 1245)
12.  [1, 18, 13, 183, 91](59, 915)
13.  [1, 4, 91](29, 504)
14.  [1, 42, 152, 86, 91](53, 579)
15.  [1, 8, 12, 91](31, 868)
16.  [1, 13, 23, 91](47, 890)
17.  [1, 34, 42, 91](53, 423)
18.  [1, 27, 91](35, 392)
19.  [1, 84, 91](29, 548)
20.  [1, 25, 91](36, 504)
```

```
---Hasil Roulette Wheel Selection---
21. [1, 98, 149, 82, 91](51, 923)
22. [1, 9, 98, 89, 146, 91](56, 1019)
23. [1, 3, 11, 91](31, 702)
24. [1, 27, 151, 91](51, 525)
25. [1, 14, 152, 91](54, 824)
26. [1, 151, 83, 91](45, 681)
27. [1, 18, 13, 183, 91](59, 915)
28. [1, 71, 91](57, 415)
29. [1, 8, 61, 91](52, 701)
30. [1, 4, 197, 13, 91](41, 915)
```

Gambar 6-2 Hasil Roulette Wheel Selection Node 1 ke 91

Gambar 6-2 merupakan hasil pemilihan solusi dengan menggunakan *roulette wheel selection*. Dari hasil seleksi tersebut, dilakukan validasi apakah seluruh solusi tidak ada yang melanggar batasan OP. Tabel 6-4 merupakan validasi batasan untuk percobaan ini.

Dapat dilihat seluruh solusi yang dihasilkan dan dipilih tidak ada yang melanggar batasan *Orienteering problem*. Semua solusi dimulai pada *node* 1 dan berakhir pada *node* 91, tidak ada solusi yang memiliki waktu tempuh selama 60 menit, serta tidak adanya subtour dalam tiap solusi.

6.4.2. Optimasi Solusi Menggunakan *Iterative Local Search With Hill Climbing*

Setelah seluruh solusi layak awal di seleksi dan menghasilkan 30 solusi seperti di atas. Ke 30 solusi tersebut akan di optimasi dengan melakukan *swap, insertion,* dan *deletion* pada masing – masing solusi. Pencarian lokal dilakukan dengan menerapkan metode *swap, insertion* atau *deletion* pada setiap iterasi, Apabila dalam satu iterasi ditemukan solusi yang lebih baik dari sebelumnya, maka perhitungan iterasi dimulai lagi dari 0. Iterasi pada percobaan ini akan dijalankan selama 2.000.000 kali. Jadi Apabila dalam

2.000.000 iterasi, program tidak dapat menemukan solusi yang lebih baik dari sebelumnya, algoritma akan dihentikan. Tabel 6-5 merupakan hasil optimasi setelah dilakukan *iterative local search* pada ke 30 solusi di atas:

Tabel 6-4 Validasi Batasan Node 1 ke 91

No	Batasan	Keterangan
1	Rute dimulai pada *node* 1 dan berhenti pada *node* 91	Berdasarkan Gambar 6-2, seluruh solusi dimulai dari *node* 1 dan berakhir di *node* 91. Sehingga tidak ada solusi yang melanggar batasan 1.
2	Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali	Dapat dilihat pada **Error! Reference source not found.** seluruh rute telah terhubung dan pada Gambar 6-2 semua solusi tidak ada yang mengunjungi *node* yang sama, sehingga tidak ada solusi yang melanggar batasan 2
3	Waktu tempuh dari setiap solusi tidak boleh melebihi 60 menit	Berdasarkan Gambar 6-2, tiap solusi tidak memiliki total waktu tempuh lebih dari 60 menit, sehingga tidak ada solusi yang melanggar batasan 3.
4	Tidak adanya subtour	Berdasarkan Gambar 6-2, tidak ada solusi yang merupakan subtour, sehingga tidak ada solusi yang melanggar batasan 4.

Tabel 6-5 Hasil Optimasi Iterative Local Search Node 1 ke 91

Solusi ke-	Rute	Waktu	Skor
1	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 38, 39, 40, 151,	45	4023

Solusi ke-	Rute	Waktu	Skor
	150, 168, 13, 14, 99, 146, 169, 91]		
2	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 21, 20, 19, 18, 15, 99, 91]	53	4579
3	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 167, 152, 91]	45	2415
4	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 91]	58	2499
5	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 13, 12, 11, 22, 54, 182, 183, 184, 185, 91]	47	3967
6	[1, 2, 3, 4, 5, 6, 7, 8, 9, 91]	31	1892
7	[1, 2, 3, 4, 5, 6, 7, 8, 54, 182, 91]	29	1854
8	[1, 27, 28, 57, 58, 59, 60, 61, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 99, 91]	42	3686
9	[1, 2, 3, 4, 5, 6, 8, 9, 10, 22, 23, 7, 11, 12, 13, 14, 99, 146, 147, 148, 91]	49	4375
10	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 167, 41, 42, 43, 152, 40, 151, 150, 168, 91]	56	3071
11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14, 12, 10, 22, 54, 182, 183, 184, 185, 186, 91]	47	4047
12	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 17, 16, 15, 14, 13, 12, 22, 54, 182, 183, 91]	59	4433
13	[1, 2, 3, 4, 91]	29	790

Solusi ke-	Rute	Waktu	Skor
14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 38, 39, 42, 41, 152, 40, 151, 150, 168, 13, 14, 99, 146, 169, 87, 86, 91]	53	4326
15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 91]	31	2566
16	[1, 2, 3, 4, 5, 6, 22, 23, 7, 8, 9, 10, 11, 12, 13, 91]	39	3404
17	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 38, 39, 40, 41, 42, 91]	53	2437
18	[1, 27, 91]	35	392
19	[1, 2, 3, 4, 5, 6, 7, 8, 54, 182, 183, 184, 185, 186, 187, 169, 83, 84, 91]	29	2841
20	[1, 2, 3, 25, 91]	36	790
21	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 99, 98, 146, 147, 148, 149, 169, 83, 84, 85, 81, 82, 91]	51	5391
22	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 99, 98, 146, 169, 87, 89, 91]	42	4165
23	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 91]	31	2368
24	[1, 27, 28, 29, 30, 31, 32, 33, 8, 9, 34, 35, 36, 37, 38, 39, 40, 151, 91]	51	1732
25	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 167, 152, 40, 151, 150, 168, 13, 14, 91]	45	3639

Solusi ke-	Rute	Waktu	Skor
26	[1, 2, 3, 4, 5, 6, 7, 8, 9, 34, 35, 36, 37, 38, 39, 40, 151, 150, 168, 13, 14, 99, 146, 169, 83, 91]	45	4210
27	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 17, 16, 15, 14, 13, 12, 22, 54, 182, 183, 91]	57	4290
28	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 100, 101, 102, 74, 73, 72, 71, 91]	57	3738
29	[1, 27, 28, 57, 58, 59, 60, 61, 2, 3, 4, 5, 6, 7, 8, 91]	40	1880
30	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 22, 197, 181, 11, 12, 13, 91]	41	3366

Dapat dilihat dari hasil optimasi terdapat beberapa solusi yang mengalami penurunan waktu tempuh (ditunjukkan pada solusi yang ditandai warna hijau) dan beberapa solusi yang mengalami peningkatan skor. Dari ke-30 solusi yang tersedia akan di ambil solusi terbaik dan dijadikan solusi untuk percobaan pencarian rute dari *node* 1 ke *node* 91.

Solusi terbaik dipilih dengan cara mengambil solusi yang memiliki skor paling tinggi. Dari ke 30 solusi di atas, skor tertinggi adalah 5391. Solusi yang memiliki skor sebesar 5391 adalah solusi nomor 21, sehingga solusi tersebut merupakan solusi terbaik untuk pencarian rute dari *node* 1 ke *node* 91 pada percobaan ini.

Solusi 21 memiliki rute [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 99, 98, 146, 147, 148, 149, 169, 83, 84, 85, 81, 82, 91] maka rute yang dilewati dari Terminal Joyoboyo menuju Jembatan Merah Plaza adalah: Terminal Joyoboyo → Jalan Raya Wonokromo → Jembatan Jagir → Jalan Raya Ngagel → Jalan Sulawesi arah Gubeng → Jalan Raya Gubeng → Jalan Stasiun Gubeng → Jalan Kusuma Bangsa → Jalan Kalianyar → Jalan Pasar Besar Wetan → Jalan Bubutan → Jalan Indrapura

→ Jalan Krembangan Barat → Jalan Krembangan Timur → Jalan Jembatan Merah → Jalan Veteran → Jembatan Merah Plaza. Rute tersebut memiliki waktu tempuh selama 51 menit dengan skor 5391.

6.5. Testing Pencarian Solusi *Node* 165 Menuju *Node* 52

Setelah percobaan dengan 1 sebagai *node* awal dan 91 sebagai *node* akhir. Pada percobaan kedua ini akan dilakukan 2 percobaan dengan tMax yang berbeda. Percobaan dilakukan dengan mencari rute optimal dari *node* 165 menuju *node* 52 dengan waktu maksimal selama 30 menit dan 60 menit. Percobaan kali ini merupakan penggambaran rute dari Jalan Arief Rachman Hakim dekat Hang Tuah menuju Universitas Airlangga Kampus B yang berada di Jalan Airlangga.

6.5.1. Skenario 1: tMax 30 menit

Sebelum solusi optimal dapat ditemukan, program akan mencari solusi layak awal terlebih dahulu. Pencarian solusi layak awal ini dilakukan sebanyak 1.000.000 iterasi. Pada pencarian ini, solusi dapat dikatakan layak apabila tidak melanggar batasan – batasan yang telah ditentukan. Antara lain:

1. Rute dimulai pada *node* 165 dan berhenti pada *node* 52
2. Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali
3. Waktu tempuh dari setiap solusi tidak boleh melebihi 30 menit
4. Tidak adanya subtour

6.5.1.1. Validasi Solusi Layak Awal

Setelah program dijalankan, program akan mencari solusi yang tidak melanggar ke 4 batasan di atas dan menyimpannya ke dalam array. Pada percobaan ini program menemukan 394 solusi layak awal. Dari 394 solusi layak tersebut, akan diambil sebanyak 30 solusi untuk dioptimasi dengan pencarian lokal. Seleksi solusi dilakukan dengan

menggunakan *roulette wheel selection*. Gambar 6-3 merupakan solusi yang lolos dan akan dilakukan optimasi:

```
 ---Hasil Roulette Wheel Selection---
1. [165, 161, 157, 52](29, 563)
2. [165, 166, 159, 52](29, 563)
3. [165, 163, 158, 52](21, 563)
4. [165, 163, 50, 52](28, 594)
5. [165, 163, 51, 52](21, 594)
6. [165, 164, 158, 52](21, 563)
7. [165, 160, 52](21, 430)
8. [165, 158, 157, 52](21, 563)
9. [165, 50, 52](28, 461)
10. [165, 51, 52](21, 461)
11. [165, 51, 52](21, 461)
12. [165, 161, 157, 52](29, 563)
13. [165, 156, 52](21, 430)
14. [165, 154, 52](21, 430)
15. [165, 158, 52](21, 430)
16. [165, 163, 52](21, 430)
17. [165, 156, 154, 52](21, 563)
18. [165, 163, 50, 52](28, 594)
19. [165, 161, 157, 52](29, 563)
20. [165, 156, 157, 154, 52](23, 696)
21. [165, 163, 50, 52](28, 594)
22. [165, 156, 52](21, 430)
23. [165, 159, 158, 52](23, 563)
24. [165, 51, 52](21, 461)
25. [165, 155, 52](21, 430)
26. [165, 53, 52](28, 461)
27. [165, 160, 52](21, 430)
28. [165, 51, 53, 52](28, 625)
29. [165, 166, 160, 159, 52](29, 696)
30. [165, 163, 52](21, 430)
```

Gambar 6-3 Hasil Roulette Wheel Selection Node 165 ke 52 dengan tMax: 30 menit

Gambar 6-3 merupakan hasil pemilihan solusi dengan menggunakan *roulette wheel selection*. Dari hasil seleksi tersebut, dilakukan validasi apakah seluruh solusi tidak ada yang melanggar batasan OP. merupakan validasi batasan untuk percobaan ini.

Tabel 6-6 Validasi Batasan Node 165 ke 52 dengan tMax: 30 menit

No	Batasan	Keterangan
1	Rute dimulai pada *node* 165 dan berhenti pada *node* 52	Berdasarkan Gambar 6-3, seluruh solusi dimulai dari *node* 1 dan berakhir di *node* 91. Sehingga tidak ada solusi yang melanggar batasan 1.
2	Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali	Dapat dilihat pada **Error! Reference source not found.** seluruh rute telah terhubung dan pada Gambar 6-3Gambar 6-2 semua solusi tidak ada yang mengunjungi *node* yang sama, sehingga tidak ada solusi yang melanggar batasan 2
3	Waktu tempuh dari setiap solusi tidak boleh melebihi 30 menit	Berdasarkan Gambar 6-3, tiap solusi tidak memiliki total waktu tempuh lebih dari 30 menit, sehingga tidak ada solusi yang melanggar batasan 3.
4	Tidak adanya subtour	Berdasarkan Gambar 6-3, tidak ada solusi yang merupakan subtour, sehingga tidak ada solusi yang melanggar batasan 4.

Dapat dilihat pada Tabel 6-6 seluruh solusi yang dihasilkan dan dipilih tidak ada yang melanggar batasan *Orienteering problem*. Semua solusi dimulai pada *node* 165 dan berakhir pada *node* 52, tidak ada solusi yang memiliki waktu

tempuh selama 30 menit, serta tidak adanya subtour dalam tiap solusi.

6.5.1.2. Optimasi Solusi Menggunakan Iterative Local Search With Hill Climbing

Serupa dengan optimasi solusi pada sub bab sebelumnya, ke 30 solusi yang telah lolos seleksi di atas akan dioptimasi dengan melakukan *swap*, *insertion*, dan *deletion*. Ke 3 metode tersebut akan dilakukan secara berulang. Pencarian lokal dilakukan dengan menerapkan metode *swap, insertion* atau *deletion* pada setiap iterasi, Apabila dalam satu iterasi ditemukan solusi yang lebih baik dari sebelumnya, maka perhitungan iterasi dimulai lagi dari 0. Iterasi pada percobaan ini akan dijalankan selama 2.000.000 kali. Jadi Apabila dalam 2.000.000 iterasi, program tidak dapat menemukan solusi yang lebih baik dari sebelumnya, algoritma akan dihentikan. Berikut merupakan hasil optimasi setelah dilakukan *iterative local search* pada ke 30 solusi di atas:

Tabel 6-7 Hasil Optimasi Iterative Local Search Node 165 ke 52 dengan tMax: 30 menit

Solusi ke-	Rute	Waktu	Skor
1	[165, 164, 163, 161, 162, 160, 158, 159, 157, 52]	29	1361
2	[165, 166, 164, 163, 162, 160, 158, 159, 52]	29	1228
3	[165, 164, 163, 162, 160, 158, 52]	21	962
4	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 50, 52]	28	2119
5	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 52]	21	1791

Solusi ke-	Rute	Waktu	Skor
6	[165, 164, 163, 162, 160, 158, 52]	21	962
7	[165, 164, 163, 162, 160, 52]	21	829
8	[165, 164, 163, 162, 160, 158, 159, 157, 52]	21	1228
9	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 50, 52]	28	2119
10	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1791
11	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1627
12	[165, 164, 163, 161, 162, 160, 158, 159, 157, 52]	29	1361
13	[165, 164, 163, 162, 160, 158, 159, 157, 156, 52]	21	1361
14	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1627
15	[165, 164, 163, 162, 160, 158, 52]	21	962
16	[165, 164, 163, 52]	21	563
17	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1627
18	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 50, 52]	28	2119

Solusi ke-	Rute	Waktu	Skor
19	[165, 164, 163, 161, 162, 160, 158, 159, 157, 52]	29	1361
20	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1627
21	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 52]	28	1955
22	[165, 164, 163, 162, 160, 158, 159, 157, 156, 52]	21	1361
23	[165, 164, 163, 162, 160, 158, 159, 52]	21	1095
24	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 52]	21	1627
25	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 52]	21	1494
26	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 52]	28	1955
27	[165, 164, 163, 162, 160, 52]	21	829
28	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 52]	28	1955
29	[165, 166, 164, 163, 162, 160, 158, 159, 52]	29	1228
30	[165, 164, 163, 52]	21	563

Dapat dilihat dari hasil optimasi terdapat beberapa solusi yang mengalami penurunan waktu tempuh (ditunjukkan pada solusi yang ditandai warna hijau) dan semua solusi mengalami peningkatan skor. Dari ke-30 solusi yang tersedia

akan di ambil solusi terbaik dan dijadikan solusi untuk percobaan pencarian rute dari *node* 165 ke *node* 52.

Solusi terbaik dipilih dengan cara mengambil solusi yang memiliki *skor* tertinggi. Dari 30 solusi di atas, skor tertinggi adalah 2119. Solusi yang memiliki skor sebesar 4862 adalah solusi nomor 5 dan 23. Kedua solusi tersebut merupakan solusi yang sama, sehingga kedua solusi dapat dikatakan sebagai solusi terbaik untuk pencarian rute dari *node* 165 meunuju *node* 52.

Solusi 4, 9, dan 18 memiliki rute [165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 50, 52] maka rute yang dilewati dari Jalan Arief Rachman Hakim dekat Hang Tuah menuju Universitas Airlangga Kampus B adalah: Universitas Hang Tuah → Jalan Arief Rachman Hakim → Jalan Gebang Putih → Jalan Manyar Kerta Adi → Jalan Raya Kertajaya Indah → Jalan Manyar Kertoarjo → Jalan Dharmahusadha Indah → Jalan Prof. Dr. Moestopo → Jalan Dharmahusadha → Jalan Karang Menjangan → Jalan Dharmawangsa → Universitas Airlangga. Rute tersebut memiliki waktu tempuh selama 28 menit dengan skor 2119.

6.5.2. Skenario 2: tMax 60 menit

Pada pencarian ini, solusi dapat dikatakan layak apabila tidak melanggar batasan – batasan yang telah ditentukan. Antara lain:

1. Rute dimulai pada *node* 165 dan berhenti pada *node* 52
2. Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali
3. Waktu tempuh dari setiap solusi tidak boleh melebihi 60 menit
4. Tidak adanya subtour

6.5.2.1. Validasi Solusi Layak Awal

Setelah program dijalankan, program akan mencari solusi yang tidak melanggar ke 4 batasan di atas dan menyimpannya ke dalam array. Pada percobaan ini program

menemukan 394 solusi layak awal. Dari 394 solusi layak tersebut, akan diambil sebanyak 30 solusi untuk dioptimasi dengan pencarian lokal. Seleksi solusi dilakukan dengan menggunakan *roulette wheel selection*. Gambar 6-4 merupakan solusi yang lolos dan akan dilakukan optimasi:

```
 ---Hasil Roulette Wheel Selection---
1. [165, 4, 24, 52](58, 583)
2. [165, 7, 11, 52](56, 773)
3. [165, 38, 168, 52](59, 461)
4. [165, 53, 163, 52](55, 594)
5. [165, 8, 100, 52](59, 793)
6. [165, 41, 158, 52](56, 461)
7. [165, 34, 155, 52](41, 461)
8. [165, 167, 40, 52](38, 594)
9. [165, 14, 151, 52](59, 760)
10. [165, 155, 40, 52](38, 594)
11. [165, 160, 198, 52](57, 510)
12. [165, 51, 52](21, 461)
13. [165, 5, 24, 52](56, 583)
14. [165, 154, 167, 160, 52](54, 696)
15. [165, 166, 23, 52](52, 628)
16. [165, 56, 52](56, 328)
17. [165, 23, 6, 52](46, 638)
18. [165, 181, 167, 52](52, 510)
19. [165, 49, 42, 52](49, 359)
20. [165, 5, 24, 52](56, 583)
21. [165, 50, 37, 52](38, 625)
22. [165, 51, 37, 155, 52](42, 758)
23. [165, 158, 156, 100, 52](59, 750)
24. [165, 154, 181, 52](46, 510)
25. [165, 162, 151, 52](42, 563)
26. [165, 155, 8, 52](44, 739)
27. [165, 3, 50, 52](55, 604)
28. [165, 160, 55, 52](56, 461)
29. [165, 162, 23, 37, 52](49, 792)
30. [165, 162, 153, 52](33, 563)
```

Gambar 6-4 Hasil Roulette Wheel Selection Node 165 ke 52 dengan tMax: 60 menit

Gambar 6-4 merupakan hasil pemilihan solusi dengan menggunakan *roulette wheel selection*. Dari hasil seleksi tersebut, dilakukan validasi apakah seluruh solusi tidak ada yang melanggar batasan OP. merupakan validasi batasan untuk percobaan ini.

Tabel 6-8 Validasi Batasan Node 165 ke 52 dengan tMax: 60 menit

No	Batasan	Keterangan
1	Rute dimulai pada *node* 165 dan berhenti pada *node* 52	Berdasarkan Gambar 6-3, seluruh solusi dimulai dari *node* 1 dan berakhir di *node* 91. Sehingga tidak ada solusi yang melanggar batasan 1.
2	Seluruh rute terhubung dan setiap *node* hanya dapat dikunjungi paling banyak satu kali	Dapat dilihat pada **Error! Reference source not found.** seluruh rute telah terhubung dan pada Gambar 6-3Gambar 6-2 semua solusi tidak ada yang mengunjungi *node* yang sama, sehingga tidak ada solusi yang melanggar batasan 2
3	Waktu tempuh dari setiap solusi tidak boleh melebihi 60 menit	Berdasarkan Gambar 6-3, tiap solusi tidak memiliki total waktu tempuh lebih dari 60 menit, sehingga tidak ada solusi yang melanggar batasan 3.
4	Tidak adanya subtour	Berdasarkan Gambar 6-3, tidak ada solusi yang merupakan subtour, sehingga tidak ada solusi yang melanggar batasan 4.

Dapat dilihat pada Tabel 6-8 seluruh solusi yang dihasilkan dan dipilih tidak ada yang melanggar batasan *Orienteering problem*. Semua solusi dimulai pada *node* 165 dan berakhir pada *node* 52, tidak ada solusi yang memiliki waktu tempuh selama 60 menit, serta tidak adanya subtour dalam tiap solusi.

6.5.2.2. Optimasi Solusi Menggunakan Iterative Local Search

Serupa dengan optimasi solusi pada sub bab sebelumnya, ke 30 solusi yang telah lolos seleksi di atas akan dioptimasi dengan melakukan *swap*, *insertion*, dan *deletion*. Ke 3 metode tersebut akan dilakukan secara berulang. Pencarian lokal dilakukan dengan menerapkan metode *swap, insertion* atau *deletion* pada setiap iterasi, Apabila dalam satu iterasi ditemukan solusi yang lebih baik dari sebelumnya, maka perhitungan iterasi dimulai lagi dari 0. Iterasi pada percobaan ini akan dijalankan selama 2.000.000 kali. Jadi Apabila dalam 2.000.000 iterasi, program tidak dapat menemukan solusi yang lebih baik dari sebelumnya, algoritma akan dihentikan. Berikut merupakan hasil optimasi setelah dilakukan *iterative local search* pada ke 30 solusi di atas:

Tabel 6-9 Hasil Optimasi Iterative Local Search Node 165 ke 52 dengan tMax: 60 menit

Solusi ke-	Rute	Waktu	Skor
1	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 6, 24, 5, 4, 52]	53	3374
2	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 8, 9, 10, 11, 52]	56	3816
3	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 151, 150, 168, 52]	59	2806
4	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 52]	28	1955

Solusi ke-	Rute	Waktu	Skor
5	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 8, 9, 10, 11, 12, 13, 14, 100, 52]	59	4862
6	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 41, 52]	40	2438
7	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 52]	34	2017
8	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 167, 152, 40, 52]	38	2611
9	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 168, 13, 14, 100, 150, 151, 52]	59	3654
10	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 52]	38	2407
11	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 107, 198, 52]	57	3017
12	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 52]	21	1791

Solusi ke-	Rute	Waktu	Skor
13	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 6, 24, 5, 52]	51	3231
14	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 167, 52]	36	2314
15	[165, 166, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 52]	52	2657
16	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 54, 55, 56, 52]	56	2499
17	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 6, 52]	46	2945
18	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 34, 22, 197, 181, 8, 9, 35, 36, 37, 167, 52]	52	3352
19	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 41, 42, 152, 49, 52]	52	2633
20	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 6, 24, 5, 52]	51	3231

Solusi ke-	Rute	Waktu	Skor
21	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 37, 38, 49, 36, 50, 52]	35	2407
22	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 53, 35, 37, 38, 49, 36, 50, 51, 52]	35	2407
23	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 8, 9, 10, 11, 12, 13, 14, 100, 52]	59	4862
24	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 197, 181, 52]	46	2486
25	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 36, 37, 38, 39, 40, 151, 52]	42	2540
26	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 8, 52]	44	3111
27	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 24, 4, 3, 5, 6, 7, 8, 9, 36, 50, 52]	55	4250
28	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 54, 55, 52]	56	2468

Solusi ke-	Rute	Waktu	Skor
29	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 34, 22, 23, 7, 8, 9, 35, 36, 37, 52]	49	3535
30	[165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 50, 153, 52]	33	2252

Dapat dilihat dari hasil optimasi terdapat beberapa solusi yang mengalami penurunan waktu tempuh (ditunjukkan pada solusi yang ditandai warna hijau) dan semua solusi mengalami peningkatan skor. Dari ke-30 solusi yang tersedia akan di ambil solusi terbaik dan dijadikan solusi untuk percobaan pencarian rute dari *node* 165 ke *node* 52.

Solusi terbaik dipilih dengan cara mengambil solusi yang memiliki *skor* tertinggi. Dari 30 solusi di atas, skor tertinggi adalah 4862. Solusi yang memiliki skor sebesar 4862 adalah solusi nomor 5 dan 23. Kedua solusi tersebut merupakan solusi yang sama, sehingga kedua solusi dapat dikatakan sebagai solusi terbaik untuk pencarian rute dari *node* 165 meunuju *node* 52.

Solusi 5 dan 23 memiliki rute [165, 164, 163, 162, 160, 158, 159, 157, 156, 155, 154, 51, 53, 35, 34, 22, 23, 7, 8, 9, 10, 11, 12, 13, 14, 100, 52] maka rute yang dilewati dari Jalan Arief Rachman Hakim dekat Hang Tuah menuju Universitas Airlangga Kampus B adalah: Universitas Hang Tuah → Jalan Arief Rachman Hakim → Jalan Gebang Putih → Jalan Manyar Kerta Adi → Jalan Raya Kertajaya Indah → Jalan Manyar Kertoarjo → Jalan Dharmahusadha Indah → Jalan Prof. Dr. Moestopo → Jalan Dharmahusadha → Jalan Karang Menjangan → Jalan Dharmawangsa → Jalan Gubeng Masjid → Jalan Nias → Jalan Biliton → Jalan Sulawesi arah Ngagel → Jalan Raya Ngagel → Jalan Sulawesi arah Gubeng → Jalan Raya Gubeng → Jalan Stasiun Gubeng → Jalan Kusuma Bangsa → Jalan Ngaglik → Universitas Airlangga. Rute

tersebut memiliki waktu tempuh selama 59 menit dengan skor 4862.

6.6. Perbandingan Besar Hasil Seleksi Solusi

Pada percobaan kali ini, akan dilakukan perubahan besarnya solusi yang lolos *roulette wheel selection* (RWS). Program akan dijalankan sebanyak lima kali untuk tiap size RWS. Dari lima kali percobaan pada masing – masing size tersebut, di ambil solusi dengan skor tertinggi sebagai solusi optimal. Hasil seleksi RWS dioptimasi dengan *iterative local search*. Pada percobaan pertama, akan dipilih 10 solusi. Percobaan kedua, akan dipilih 30 solusi, dan percobaan ke 3, akan dipilih 50 solusi. Seluruh percobaan pada sub bab ini menggunakan *node* 1 sebagai *node* awal dan *node 91* sebagai *node* akhir dengan tMax adalah 60 menit. Setiap program dijalankan, program akan melakukan pencarian solusi awal sebanyak 1.000.000 kali iterasi, dan pencarian lokal sebanyak 2.000.000 iterasi.

Tabel 6-10 merupakan tabel perbandingan solusi optimal setiap percobaan dengan size RWS (*roulette wheel selection*) yang berbeda – beda.

Dapat dilihat pada Tabel 6-10, semakin besar size dari *roulette wheel selection*, maka semakin besar juga skor yang dihasilkan. Hal tersebut dapat terjadi karena *iterative local search* merupakan metode stokastik yang memiliki hasil pencarian berbeda di setiap iterasinya, maka apabila size dari *roulette wheel selection* semakin besar, kemungkinan didapatkan solusi yang paling optimal semakin tinggi. Dengan begitu, pada percobaan ini dapat dikatakan semakin besar size *roulette wheel selection*, maka solusi yang dihasilkan dapat semakin baik.

6.7. Perbandingan Tiap Metode *Iterative Local Search*

Penelitian ini menggunakan *iterative local search* untuk mengoptimalkan tiap solusi yang dipilih *roulette wheel selection*.

Tabel 6-10 Perbandingan Size RWS

Size RWS	Rute	Waktu	Skor
10	[1, 2, 3, 4, 5, 6, 7, 8, 54, 182, 183, 184, 185, 186, 187, 190, 191, 192, 193, 194, 195, 196, 106, 10, 11, 12, 13, 14, 99, 146, 169, 83, 84, 91]	56	5246
30	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 199, 103, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 91]	58	5421
50	[1, 2, 3, 4, 5, 6, 7, 8, 54, 182, 183, 184, 185, 186, 187, 190, 191, 192, 193, 194, 195, 196, 106, 10, 11, 12, 13, 14, 99, 146, 169, 83, 84, 85, 91]	56	5433

Metode optimasi yang dilakukan antara lain *swap method*, *insertion method*, dan *deletion method*. Pada percobaan ini, program akan dijalankan sebanyak 4 kali. Dimana setiap *running*, program hanya akan menggunakan satu metode dari 3 metode optimasi yang telah disebutkan. Pada *running* terakhir, program akan dijalankan dengan menggunakan ketiga metode tersebut bergantian. Ke 4 hasil percobaan akan dibandingkan dalam 1 tabel perbandingan.

Percobaan ini menggunakan *node* 1 sebagai *node* awal dan *node* 91 sebagai *node* akhir dengan tMax selama 40 menit. Jumlah solusi yang akan lolos *roulette wheel selection* adalah sebanyak 30 solusi. Pencarian solusi layak awal dilakukan sebanyak 1.000.000 iterasi, dan pencarian lokal dilakukan sebanyak 2.000.000 iterasi. Tabel 6-11 merupakan tabel perbandingan ke 4 percobaan tersebut.

Berdasarkan Tabel 6-11, dapat dilihat *deletion method* memiliki skor terkecil, diikuti dengan *swap method* sehingga kedua metode ini kurang tepat digunakan secara individu untuk pencarian solusi pada penelitian ini.

Tabel 6-11 Perbandingan Metode Local Search

Metode	Rute	Waktu	Skor
Swap	[1, 13, 14, 16, 91]	39	1219
Insertion	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 99, 97, 98, 146, 169, 87, 91]	37	4220
Deletion	[1, 22, 184, 169, 91]	33	963
All in one	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 99, 146, 169, 83, 91]	35	4145

Sedangkan *insertion method* memiliki nilai tertinggi dengan waktu tempuh yang lebih tinggi juga dibanding dengan *all in one*. Penggabungan ketiga metode juga memiliki skor yang cukup tinggi dengan total waktu tempuh yang lebih rendah. Sehingga untuk pencarian solusi optimal dari *node* 1 ke *node* 91, cocok menggunakan *insertion method*, atau gabungan dari ketiga metode *iterative local search*.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dibahas mengenai kesimpulan dari semua proses yang telah dilakukan dan saran yang dapat diberikan untuk pengembangan yang lebih baik.

7.1. Kesimpulan

Berdasarkan proses-proses yang telah dilakukan dalam penelitian tugas akhir ini, maka ada beberapa kesimpulan yang dapat diambil, di antaranya adalah:

1. Metode *Orienteering Problem* dapat digunakan untuk memodelkan jalur trayek angkot Kota Surabaya
2. Algoritma Dijkstra dapat digunakan untuk membantu pencarian waktu tempuh tercepat dari satu *node* sumber ke seluruh *node*. Serta algoritma tersebut dapat membantu pencarian solusi layak awal pada penelitian ini.
3. Hasil dari penerapan algoritma Dijkstra yang telah dilakukan menghasilkan data set yang dapat digunakan untuk pengembangan selanjutnya
4. Metode optimasi *iterative local search* dapat menghasilkan solusi dari model yang telah dibuat dengan baik, cepat dan optimal.
5. Jumlah pemilihan hasil seleksi dengan *roulette wheel selection* sangat berpengaruh dalam pencarian solusi optimal. Semakin besar jumlah pengambilan solusi maka kemungkinan solusi terbaik yang didapatkan juga akan semakin besar.
6. Penggunaan *swap, insertion,* dan *deletion method* pada *iterative local search* juga sangat berpengaruh dalam pencarian solusi optimal. Penggabungan ketiga metode pencarian lokal tersebut menghasilkan solusi yang lebih baik apabila dibandingkan dengan hanya menggunakan salah satu metode pencarian lokal.

7.2. Saran

Dari pengerjaan tugas akhir ini, terdapat hal-hal yang dapat diperbaiki agar dapat menghasilkan keluaran yang lebih baik lagi, antara lain:

1. Pada penelitian ini, optimasi hanya berada pada lingkup 6 trayek yaitu trayek F, JMK, RT, JK, O, dan GS. Pada penelitian berikutnya, dapat ditambahkan jalur trayek yang lain untuk memperluas cakupan optimasi angkot di Kota Surabaya.
2. Penelitian ini menggunakan *random algorithm* untuk mencari solusi layak awal. Sangat disarankan untuk menggunakan algoritma yang lebih baik seperti *Ant Colony Algorithm, Memetic Algorithm,* dan algoritma opimasi lainnya yang dapat digunakan untuk pencarian solusi sehingga solusi yang dihasilkan dapat lebih optimal.
3. Optimasi solusi pada penelitian ini hanya menggunakan metode pencarian lokal *swap, insertion* dan *deletion*. Pada pengembangan selanjutnya dapat ditambahkan metode pencarian lokal lain seperti *shake step, backward insertion, forward insertion* dan metode pencarian lokal lainnya.
4. Skor model *Orienteering Problem* pada penelitian ini menggunakan jumlah angkot yang lewat pada satu titik. Representasi skor dapat disempurnakan dengan mempertimbangkan beberapa faktor lain seperti, seberapa sering titik tersebut dilewati, tingkat kemacetan, atau seberapa populer titik tersebut di Kota Surabaya. Sehingga merepresentasikan tiap titik dengan lebih baik.

DAFTAR PUSTAKA

[1] Jakarta Globe, "Jakarta: World's Worst for Traffic Gridlock," Jakarta Globe, [Online]. Available: http://jakartaglobe.id/news/jakarta-worlds-worst-traffic-gridlock/. [Accessed 1 February 2017].

[2] Kementrian Dalam Negeri, "Permendagri No. 39 Tahun 2015," Kementrian Dalam Negeri, [Online]. Available: http://www.kemendagri.go.id/pages/profil-daerah/provinsi/detail/31/dki-jakarta. [Accessed 1 February 2017].

[3] Kementrian Dalam Negeri, "Permendagri No.66 Tahun 2011," Kementrian Dalam Negeri, [Online]. Available: http://www.kemendagri.go.id/pages/profil-daerah/kabupaten/id/35/name/jawa-timur/detail/3578/kota-surabaya. [Accessed 1 February 2017].

[4] M. Grote, I. Williams, J. Preston and S. Kemp, "Including Congestion Efects in Urban Road Traffic CO2 Emissions Modelling: Do Local Government Authorities Have the Right Options?," *Transportation Research Part D: Transport and Environtment*, vol. 43, pp. 95-106, 2016.

[5] S. Webb, "'Horror' Traffic Jam in Indonesia Lasted 35 Hours - and 18 Have Died on Same Stretch of Road in A Week," Mirror News, 8 July 2016. [Online]. Available: http://www.mirror.co.uk/news/world-news/horror-traffic-jam-indonesia-lasted-8377678. [Accessed 2 February 2017].

[6] Tempo, "3 Program Utama DKI Jakarta Hadapi Kemacetan," Tempo, 15 February 2015. [Online]. Available: https://m.tempo.co/read/news/2015/02/15/0836425

73/3-program-utama-dki-jakarta-atasi-kemacetan. [Accessed 4 February 2017].

[7] R. D. Arifiyananta, "Strategi Dinas Perhubungan Kota Surabaya Untuk Mengurangi Kemacetan Jalan Raya Kota Surabaya," Universitas Negeri Surabaya, Surabaya.

[8] M. Zulfikar and H. E. P. Gultom, "Ini Cara Tri Rismaharini Antisipasi Kemacetan di Surabaya," Tribun News, 29 November 2013. [Online]. Available: http://www.tribunnews.com/regional/2013/11/29/ini-cara-tri-rismaharini-antisipasi-kemacetan-di-surabaya. [Accessed 27 February 2017].

[9] Dinas Perhubungan Kota Surabaya, "Dibalik Berdirinya SITS," Dinas Perhubungan Kota Surabaya, [Online]. Available: http://sits.dishub.surabaya.go.id/ver2/tentang-sits. [Accessed 3 February 2017].

[10] P. Vansteenwegen, W. Souffriau and D. V. Oudheusden, "The Orienteering Problem: A Survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1-10, 2011.

[11] V. Campos, R. Marti, J. Sanchez-Oro and A. Duarte, "GRASP with Path Relinking for the Orienteering Problem," *Journal of the Operational Research Society*, vol. 65, no. 12, pp. 1800-1813, 2014.

[12] P. Vansteenwegen, W. Souffriau, G. V. Berghe and D. V. Oudheusden, "Iterated Local Search for the Team Orienteering Problem with Time Windows," *Computers & Operations Research*, no. 36, pp. 3281 - 3290, 2009.

[13] A. Gunawan, H. C. Lau and K. Lu, "An Iterated Local Search Algorithm for Solving the Orienteering Problem with Time Windows," in *European*

Conference on Evolutionary Computation in Combinatorial Optimization, Singapore, 2015.

[14] A. Downs, Stuck in Traffic: Coping with Peak-hour Traffic Congestion, Washington, D.C.: The Brookings Institution, 1992.

[15] T. A. Litman, Transportation Cost and Benefit Analysis: Techniques, Estimates and Implications, Victoria: Victoria Transport Policy Institute, 2004.

[16] M. D. Aftabuzzaman, "Measuring Traffic Congestion-A Critical Review," Institute of Transport Studies, Melbourne, 2007.

[17] M. J. Rothenberg, "Urban Congestion in the United States: What Does the Future Hold?," *ITE Journal*, vol. 7, no. 55, pp. 22-39, 1985.

[18] M. A. Miller and K. Li, "An Investigation of the Costs of Roadway Traffic Congestion: A Preparatory Step fo IVHS Benefits Evaluation," Institute of Transportation Studies, California, 1994.

[19] T. T. Dimyati and A. Dimyati, "Operations Research: Model-model Pengambilan Keputusan," Bandung Sinar Baru, Bandung, 2002.

[20] Society for Industrial and Applied Mathematics, "Introduction to Process Optimization," [Online]. Available: www.siam.org/books/mo10/mo10_sample.pdf. [Accessed 5 February 2017].

[21] Business Dictionary, "Optimization Definition," Business Dictionary, [Online]. Available: http://www.businessdictionary.com/definition/optimization.html. [Accessed 22 February 2017].

[22] B. Santosa and N. Hardiansyah, "Cross Entropy Method for Solving Generalized Orienteering Problem," *iBusiness*, no. 2, pp. 342-347, 2010.

[23] M. Keshtkaran and K. Ziarati, "A Novel GRASP Solution Approach for the Orienteering Problem," *Journal of Heuristics*, vol. 22, no. 5, pp. 699-726, 2016.

[24] A. Gunawan, H. Chuin Lau and P. Vansteenwegen, "Orienteering Problem: A Survey of Recent Variants, Solution Approaches and Applications," *European Journal of Operational Research*, vol. 255, no. 2, pp. 315-332, 2016.

[25] J. Brownliee, "Iterated Local Search," Clever Algorithm, [Online]. Available: http://www.cleveralgorithms.com/nature-inspired/stochastic/iterated_local_search.html. [Accessed 20 June 2017].

[26] H. Lourenco, O. Martin and T. Stutzle, "Iterated Local Search," *Handbook of Metaheuristc*, pp. 321-353, 2003.

[27] North Dakota State University, "Visualization of Hill Climbing," North Dakota State University, [Online]. Available: http://wwwic.ndsu.edu/juell/vp/cs724s00/hill_climbing/hill_help.html. [Accessed 20 June 2017].

[28] Computer Science and Engineering UNT, "Dijkstra's Algorithm," [Online]. Available: http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Dijkstra%27s%20algorithm.pdf. [Accessed 10 June 2017].

[29] M. Puthuparampil, "Report Dijkstra's Algorithm," New York University, New York.

[30] Dinas Perhubungan Kota Surabaya, "Jumlah dan Rute (Trayek) Angkutan Umum," Dinas Perhubungan Kota Surabaya, [Online]. Available: http://dishub.surabaya.go.id/index.php/post/id/1840. [Accessed 8 February 2017].

[31] Departemen Perhubungan Direktur Jenderal Perhubungan Darat, "Pedoman Teknis Perekayasanaan Tempat Perhentian Kendaraan Penumpang Umum," 1998. [Online]. Available: https://www.scribd.com/doc/74879620/PEDOMAN-TEKNIS-PEREKAYASANAAN-TEMPAT-PERHENTIAN-KENDARAAN-PENUMPANG-UMUM-HALTE. [Accessed 21 April 2017].

[32] M. F. Tasgetiren, "A Genetic Algorithm with an Adaptive Penalty Function for the Orienteering Problem," *Journal of Economic and Social Research*, vol. 4, no. 2, pp. 1-26, 2001.

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis lahir di Denpasar pada tanggal 5 Maret 1995. Merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh beberapa pendidikan formal yaitu; SD Katolik Santo Yoseph 1 Denpasar, SMP Katolik Santo Yoseph Denpasar, dan SMAN 1 Denpasar.

Pada tahun 2013 pasca kelulusan SMA, penulis melanjutkan pendidikan ke jenjang perguruan tinggi di Jurusan Sistem Informasi FTIf – Institut Teknologi Sepuluh Nopember (ITS) Surabaya dan terdaftar sebagai mahasiswa dengan NRP 5213100177. Selama menjadi mahasiswa, penulis mengikuti berbagai kegiatan kemahasiswaan seperti beberapa organisasi dan kepanitiaan serta pernah menjabat sebagai Koordinator Departemen Pengembangan Sumber Daya Mahasiswa di TPKB ITS pada tahun kedua dan ketiga perkuliahan, menjabat sebagai Staff Hubungan Luar HMSI ITS pada tahun kedua, dan staff ahli Departemen Hubungan Luar HMSI ITS pada tahun ketiga. Penulis juga berperan sebagai Koordinator Acara Information Systems Expo 2015. Di bidang akademik, penulis aktif menjadi asisten dosen Perencanaan Sumber Daya Perusahaan. Selain itu, penulis juga pernah diberikan kesempatan melakukan kerja praktik di PT. Pertamina (Persero) Jakarta Pusat dengan bergabung di bagian IT Solution.

Pada tahun keempat, karena penulis memiliki ketertarikan di bidang perekayasaan data, maka penulis mengambil bidang minat Rekayasa Data dan Intelegensi Bisnis (RDIB). Penulis dapat dihubungi melalui *email* di jockey1213@gmail.com.

Halaman ini sengaja dikosongkan

LAMPIRAN A

Iterative Local Search

```
1. package pseudocodeop;
2.
3. import java.io.*;
4. import java.util.*;
5.
6. /**
7.  *
8.  * @author Jockey
9.  */
10.  public class PseudoCodeOP {
11.
12.  static int jumlNode = 199;
13.  static int nodeAwal = 1;
14.  static int nodeAkhir = 91;
15.  static int[][] arrNode = new
   int[jumlNode][jumlNode];
16.  static int[] scoreNode = new
   int[jumlNode];
17.
18.  static
   ArrayList<ArrayList<Integer>>
   feasibleSol = new ArrayList<>();
19.  static ArrayList<Integer>
   feasibleSolJarak = new
   ArrayList<>();
20.  static ArrayList<Integer>
   feasibleSolScore = new
   ArrayList<>();
21.
22.  static
   ArrayList<ArrayList<Integer>>
   pickSolution = new ArrayList();
23.  static ArrayList<Integer>
   pickSolJarak = new ArrayList<>();
```

```
24.  static ArrayList<Integer>
   pickSolScore = new ArrayList<>();
25.
26.  public static void
   main(String[] args) throws
   FileNotFoundException,
   IOException {
27.
28.  String thisLine;
29.  BufferedReader nodeMatrix = new
   BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Doc
   uments\\NetBeansProjects\\SolusiO
   P\\networkDijkstra.csv"));
30.
31.  ArrayList<String[]> lines = new
   ArrayList<>();
32.  while ((thisLine =
   nodeMatrix.readLine()) != null) {
33.  lines.add(thisLine.split(","));
34.          }
35.
36.  //Convert our list to a String
   array
37.  String[][] array = new
   String[lines.size()][0];
38.  lines.toArray(array);
39.
40.  //Convert String array to
   Integer array
41.  for (int i = 0; i <
   array.length; i++) {
42.  for (int j = 0; j <
   array.length; j++) {
43.  arrNode[i][j] =
   Integer.parseInt(array[i][j]);
44.              }
```

```
45.             }
46.
47.  BufferedReader nodeScore = new
   BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Doc
   uments\\NetBeansProjects\\SolusiO
   P\\scoreNode.csv"));
48.
49.  String barisIni;
50.  int isi = 0; //representasi isi
   array
51.  while ((barisIni =
   nodeScore.readLine()) != null) {
52.  scoreNode[isi] =
   Integer.parseInt(barisIni);
53.  isi++;
54.             }
55.
56.  //Define DMAX
57.  int tMax = 30;
58.  //Define maxloop
59.  int maxLoop = 1000000;
60.  //Set count=0 and loop=0;
61.  int loop = 1;
62.  do {
63.  //Produce a random number, N ,
   between R [1,N]
64.  int Rn = new
   Random().nextInt(jumlNode - 2) +
   1;
65.  ArrayList<Integer> listRl = new
   ArrayList();
66.  ArrayList<Integer> listRl1 =
   new ArrayList();//array yang
   digunakan untuk random diantara
   node awal dan akhir
```

```
67.  for (int i = 1; i < jumlNode +
     1; i++) { //fungsi looping
     mengambil node 1 sampai jumlNode
68.  if (i == nodeAwal || i ==
     nodeAkhir) {
69.  continue;
70.            }
71.  listRl1.add(i);
72.                }
73.  Collections.shuffle(listRl1);
74.
75.  listRl.add(nodeAwal);
     //merepresentasikan node awal
76.  for (int i = 0 ; i <
     listRl1.size(); i++) {
77.  listRl.add(listRl1.get(i));
     //merepresentasikan seluruh node
     dari no 1 ke node 199 (kecuali
     node awal dan node akhir
78.                }
79.  listRl.add(nodeAkhir);
     //merepresentasikan node akhir
80.
81.  //Generate a sub list, by
     taking the first part of the
     random list L R
82.  ArrayList<Integer> subListRs =
     new ArrayList();
83.  subListRs.add(listRl.get(0));
     //mengambil node awal dari array
     listRl
84.  for (int i = 1; i < (1 + Rn);
     i++) {
85.  subListRs.add(listRl.get(i));
     //memasukkan node antara node
     awal dan node akhir secara random
     menandakan jalur yang dilewati
```

```
86.                 }
87.  subListRs.add(listRl.get(listRl
   .size() - 1)); //mengambil node
   akhir dari array listRl
88.
89.  //Compute the total distance,
   RS , of the sub d list
90.  for (int i = 0; i <
   subListRs.size(); i++) {
91.   if (kalkulasiJarak(subListRs)
   < tMax &&
   !feasibleSol.contains(subListRs))
   { //true apabila feasibleSol
   tidak berisikan subListRs
92.  feasibleSol.add(subListRs);
93.                     }
94.                 }
95.  loop++;
96.  } while (loop <= maxLoop);
97.  for (int i = 0; i <
   feasibleSol.size(); i++) {
98.  feasibleSolJarak.add(kalkulasiJ
   arak(feasibleSol.get(i)));
99.  feasibleSolScore.add(kalkulasiS
   core(feasibleSol.get(i)));
100.         }
101. for (int i = 0; i <
   feasibleSol.size(); i++) {
102. System.out.println(feasibleSol.
   get(i) + "(" +
   feasibleSolJarak.get(i) + ", " +
   feasibleSolScore.get(i) + ")");
103.         }
104. System.out.println("Total
   solusi awal yang ditemukan:" +
   feasibleSol.size());
105.
```

```
106. System.out.println("---Hasil
   Roullete Wheel Selection---");
107. //Arraylist untuk mengambil
   beberapa solusi menggunakan
   roullete wheel selection
108.
109. int size = 30;
110. do {
111. pickSolution.add(feasibleSol.ge
   t(roulleteWheelSelect()));
112.          } while
   (pickSolution.size() < size);
113.
114. for (int i = 0; i <
   pickSolution.size(); i++) {
115. pickSolJarak.add(kalkulasiJarak
   (pickSolution.get(i)));
116. pickSolScore.add(kalkulasiScore
   (pickSolution.get(i)));
117. System.out.println(pickSolution
   .get(i) + "(" +
   pickSolJarak.get(i) + ", " +
   pickSolScore.get(i) + ")");
118.          }
119.
120. System.out.println("---Local
   search---");
121.
122. for (int i = 0; i <
   pickSolution.size(); i++) {
123. //looping untuk mengecek semua
   isi pickSolution
124. int counterSearch = 1;
125. int maxSearchLoop = 2000000;
126. do {
127. int randCase = new
   Random().nextInt(3) + 1;
```

```
128. switch (randCase) {
129. case 1: //swap
130. int swap1 = new
   Random().nextInt(pickSolution.get
   (i).size() - 2) + 1;
131. int swap2 = new
   Random().nextInt(pickSolution.get
   (i).size() - 2) + 1;
132.
133. Collections.swap(pickSolution.g
   et(i), swap1, swap2);
134.
135. int currentJarakSwap =
   kalkulasiJarak(pickSolution.get(i
   ));
136. int currentScoreSwap =
   kalkulasiScore(pickSolution.get(i
   ));
137. if (currentJarakSwap <
   pickSolJarak.get(i)) {
   //currentjarak lebih kecil atau
   currentscore lebih besar
138. pickSolJarak.set(i,
   currentJarakSwap);
139. pickSolScore.set(i,
   currentScoreSwap);
140. counterSearch = 1;
141.
142. } else {
143. Collections.swap(pickSolution.g
   et(i), swap2, swap1);
144.     }
145. break;
146. case 2: //insertion
147. int randIndex = new
   Random().nextInt(pickSolution.get
   (i).size() - 2) + 1; //nilai
   random index
```

```
148. int randNode = new
   Random().nextInt(jumlNode) + 1;
   //nilai random node yang masuk
149. if (randNode == nodeAwal ||
   randNode == nodeAkhir ||
   pickSolution.get(i).contains(rand
   Node)) { //biar yang di insert
   ngga sama kaya node awal dan node
   akhir
150. continue;// apabila kondisi di
   atas = true, maka di skip
151. }
152. pickSolution.get(i).add(randInd
   ex, randNode);
153. int currentJarakInsert =
   kalkulasiJarak(pickSolution.get(i
   ));
154. int currentScoreInsert =
   kalkulasiScore(pickSolution.get(i
   ));
155. if (currentJarakInsert <=
   pickSolJarak.get(i) &&
   currentScoreInsert >=
   pickSolScore.get(i)) {
156.
157. pickSolJarak.set(i,
   currentJarakInsert);
158. pickSolScore.set(i,
   currentScoreInsert);
159. counterSearch = 1;
160.
161. } else {
162. pickSolution.get(i).remove(rand
   Index);
163.                          }
164. break;
165. case 3: //deletion
```

```
166.  int delRandom = new
   Random().nextInt(pickSolution.get
   (i).size() - 2) + 1;
167. if (pickSolution.get(i).size()
   > 3 ){
168.
169. int temp =
   pickSolution.get(i).get(delRandom
   ); //temp menyimpan value node
   yang dihapus
170. pickSolution.get(i).remove(delR
   andom);
171. int currentJarakDel =
   kalkulasiJarak(pickSolution.get(i
   ));
172. int currentScoreDel =
   kalkulasiScore(pickSolution.get(i
   ));
173. if (currentJarakDel <=
   pickSolJarak.get(i) &&
   currentScoreDel >=
   pickSolScore.get(i)) {
174.
175. pickSolJarak.set(i,
   currentJarakDel);
176. pickSolScore.set(i,
   currentScoreDel);
177. counterSearch = 1;
178. } else {
179. pickSolution.get(i).add(delRand
   om, temp); //mengembalikan node
   ke seperti semula
180. }
181. }
182. break;
183.
184.     }
```

```
185.
   counterSearch++;
186.       } while (counterSearch <=
   maxSearchLoop);
187.         }
188. for (int i = 0; i <
   pickSolution.size(); i++) {
189. System.out.println("Solusi ke =
   " + (i + 1));
190. System.out.println(pickSolution
   .get(i) + "(" +
   pickSolJarak.get(i) + ", " +
   pickSolScore.get(i) + ")");
191.         }
192. int indexTerbaik = 0;
193. int scoreTerbaik =
   pickSolScore.get(0);
194. for (int i = 1; i <
   pickSolution.size(); i++) {
195. if (pickSolScore.get(i) >
   scoreTerbaik) {
196. indexTerbaik = i;
197. scoreTerbaik =
   pickSolScore.get(i);
198.                }
199.           }
200. System.out.println();
201. System.out.println("Solusi
   terbaik: " +
   pickSolution.get(indexTerbaik) +
   "(" +
   pickSolJarak.get(indexTerbaik) +
   ", " + scoreTerbaik + ")");
202.
203.     } //end of Main
204.
```

```
205. public static int
     kalkulasiJarak(ArrayList<Integer>
     hitungList) { //perhitungan waktu
     tempuh
206. int sum = 0; //inisiasi nilai
     sum menjadi 0 untuk penjumlahan
207. for (int i = 0; i <
     hitungList.size() - 1; i++) {
     //loop untuk mengambil isi dari
     isi arraylist
208. sum = sum +
     arrNode[hitungList.get(i) -
     1][hitungList.get(i + 1) - 1];
     //menjumlah setiap waktu tempuh
     pada 1 array
209.          }
210.          return sum;
211.      }
212.
213. public static int
     kalkulasiScore(ArrayList<Integer>
     varListRl) { //Perhitungan skor
214. int sum = 0; //Inisiasi nilai
     awal sum adalah 0 untuk
     penjumlahan
215. for (int i = 0; i <
     varListRl.size(); i++) { //Loop
     untuk mengambil isi dari isi list
     Rl
216. sum = sum +
     scoreNode[varListRl.get(i) - 1];
     //Mengambil nilai score pada
     index sebelum i
217.          }
218.          return sum;
219.      }
220.
```

```
221. public static int totalScore()
   {
222. int sum = 0; //Inisiasi nilai
   awal sum adalah 0 untuk
   penjumlahan
223. for (int i = 0; i <
   feasibleSolScore.size(); i++) {
   //Loop untuk mengambil isi dari
   isi list feasibleScore
224. sum = sum +
   feasibleSolScore.get(i);
   //Menjumlah setiap score pada 1
   list feasibleScore
225.          }
226.          return sum;
227.      }
228.
229. public static int
   roulleteWheelSelect() {
230. double rand = new
   Random().nextDouble(); //nilai
   random ini akan menentukan
231. int totScore = totalScore();
232. int k = 0;
233. double sum = (double)
   feasibleSolScore.get(k) /
   totScore;
234. while (sum < rand) {
235. k++;
236. sum = sum + (double)
   feasibleSolScore.get(k) /
   totScore;
237.          }
238.          return k;
239.      }
240. }
```

LAMPIRAN B

Dijkstra's Algorithm

```
1. package pseudocodeop;
2.
3. /**
4.  *
5.  * @author Jockey
6.  */
7. import java.util.*;
8. import java.lang.*;
9. import java.io.*;
10.
11.
12.  public class ShortestPath {
13.
14.      static int jumlNode = 199;
15.      static int src = 0;
16.
17.      int minDistance(int dist[],
   Boolean sptSet[]) {
18.          int min =
   Integer.MAX_VALUE, min_index = 0;
19.
20.          for (int v = 0; v <
   jumlNode; v++) {
21.              if (sptSet[v] == false
   && dist[v] <= min) {
22.                  min = dist[v];
23.                  min_index = v;
24.              }
25.          }
26.          return min_index;
27.      }
28.
29.      void printSolution(int dist[],
   int n) {
```

```
30.           for (int i = 0; i <
   jumlNode; i++) {
31.
   System.out.print(dist[i] + ",");
32.           }
33.       }
34.
35.       void dijkstra(int graph[][],
   int src) {
36.           int dist[] = new
   int[jumlNode];
37.
38.           Boolean sptSet[] = new
   Boolean[jumlNode];
39.
40.           for (int i = 0; i <
   jumlNode; i++) {
41.               dist[i] =
   Integer.MAX_VALUE;
42.               sptSet[i] = false;
43.           }
44.
45.           dist[src] = 0;
46.
47.           for (int count = 0; count
   < jumlNode - 1; count++) {
48.               int u =
   minDistance(dist, sptSet);
49.               sptSet[u] = true;
50.
51.               for (int v = 0; v <
   jumlNode; v++)
52.               {
53.                   if (!sptSet[v] &&
   graph[u][v] != 0
54.                           && dist[u]
   != Integer.MAX_VALUE
```

```
55.                             && dist[u]
   + graph[u][v] < dist[v]) {
56.                         dist[v] =
   dist[u] + graph[u][v];
57.                     }
58.                 }
59.             }
60.             printSolution(dist,
   jumlNode);
61.         }
62.
63.         static int[][] arrNode = new
   int[jumlNode][jumlNode];
64.
65.         public static void
   main(String[] args) throws
   FileNotFoundException, IOException {
66.
67.             String thisLine;
68.             BufferedReader nodeMatrix
   = new BufferedReader(new
   FileReader("C:\\Users\\Jodie\\Docume
   nts\\NetBeansProjects\\ShortestPath\
   \Network Model Dijkstra.csv"));
69.
70.             ArrayList<String[]> lines
   = new ArrayList<>();
71.             while ((thisLine =
   nodeMatrix.readLine()) != null) {
72.
   lines.add(thisLine.split(","));
73.             }
74.
75.             String[][] array = new
   String[lines.size()][0];
76.             lines.toArray(array);
77.
```

```
78.           for (int i = 0; i <
   array.length; i++) {
79.               for (int j = 0; j <
   array.length; j++) {
80.                   arrNode[i][j] =
   Integer.parseInt(array[i][j]);
81.               }
82.           }
83.
84.           ShortestPath t = new
   ShortestPath();
85.           for (int i = 0; i <
   jumlNode; i++) {
86.               t.dijkstra(arrNode,
   i);
87.               System.out.println();
88.           }
89.
90.       }
91.   }
```

LAMPIRAN C

Node	Lokasi	Skor
1	Terminal Joyoboyo	174
2	Jalan Raya Wonokromo dekat jembatan	143
3	Jembatan Jagir	143
4	Jalan Raya Ngagel dekat SPBU	143
5	Jalan Raya Ngagel dekat Carefour Kalimas	143
6	Jalan Raya Ngagel dekat LBB Einstein	143
7	Jalan Raya Ngagel dekat Jalan Sulawesi	278
8	Jalan Sulawesi arah Raya Gubeng	309
9	Jalan Raya Gubeng	229
10	Jalan Raya Gubeng dekat Taman Lansia	278
11	Jalan Stasiun Gubeng	198
12	Jalan Kusuma Bangsa dekat Jalan melati	198
13	Perempatan Jalan Kusuma Bangsa dan Jalan Ambengan	331
14	Perempatan Jalan Kusuma Bangsa dan Jalan Kalianyar	330
15	Jalan Kapasari	143
16	Jalan Simokerto	197
17	Jalan Sidotopo Lor dekat PT. Kereta Api	143
18	Jalan Sidotopo Lor dekat Indomaret	143
19	Jalan Sidorame	198
20	Jalan Karang Tembok	143
21	Jalan Wonosari Lor	143
22	Jalan Biliton	309
23	Jalan Sulawesi arah Raya Ngagel	198
24	Jalan Upa Jiwa	143
25	Jalan St. Wonokromo	143
26	Jalan Wonokromo	143

C-2

Node	Lokasi	Skor
27	Jalan Raya Darmo dekat Cagar Budaya	31
28	Perempatan Jalan Raya Darmo dan Jalan Bengawan	31
29	Jalan Raya Darmo dekat Mecure Hotel	31
30	Jalan Majapahit	31
31	Jalan Dr. Soetomo	31
32	Jalan Sriwijaya	31
33	Jalan Pandegling	31
34	Jalan Nias	31
35	Jalan Gubeng Masjid	31
36	Pertigaan Jalan Penataran dan Jalan Gerbong	31
37	Perempatan Jalan kalasan dan Jalan Pacar Keling	164
38	Jalan Jagiran dekat jalan Oro-oro	31
39	Jalan Gersikan	31
40	Perempatan Jalan Raya Karang Asem dengan Jalan Gersikan	164
41	Jalan Ploso Baru dekat Jalan Bronggalan II	31
42	Jalan Ploso Baru	31
43	Jalan Kalijudan Dekat Jalan ploso baru	31
44	Jalan Kalijudan	31
45	Jalan Kalijudan dekat SMK PGRI 4	31
46	Jalan Kalijudan dekat Jalan Kenjeran	31
47	Jalan Kenjeran dekat Jalan Wiratno	31
48	Jalan Wiratno	31
49	Jalan Sawentar	31
50	Jalan Dr. Moestopo dekat RS Husada Utama	164
51	Jalan Karang Menjangan	164

Node	Lokasi	Skor
52	Jalan Airlangga	164
53	Jalan Dharmawangsa	164
54	Jalan Karimun Jawa	111
55	Jalan Urip Sumoharjo	31
56	Jalan Urip Sumoharjo dekat Carls Jr	31
57	Jalan Diponegoro	31
58	Jalan Ciliwung	31
59	Jalan Adityawarman	31
60	Jalan Hayam Wuruk	31
61	Jalan Gunung Sari	31
62	Jalan Raya Pantai Lama	54
63	Jalan Pantai Kenjeran	54
64	Jalan Pantai Kenjeran dekat Kyai Tambak Deres	54
65	Jalan Kyai Tambak Deres A	54
66	Jalan Kyai Tambak Deres B	54
67	Jalan Kyai Tambak Deres C	54
68	Jalan Kedung Cowek	54
69	Jalan Kedung Cowek dekat Kantor Pos Suramadu	54
70	Jalan Pogot	54
71	Jalan Platuk	54
72	Jalan Sidotopo Wetan	54
73	Jalan Sidotopo Wetan dekat Jalan Kenjeran	54
74	Jalan Kenjeran dekat JNE Kenjeran Baru	54
75	Jalan Kenjeran dekat Jalan Rangkah	54
76	Perempatan Jalan Sidodadi dan Jalan Kampung Seng	109
77	Jalan Sidodadi V	54
78	Jalan Pegirian	54

C-4

Node	Lokasi	Skor
79	Jalan K.H. Mas Mansyur	54
80	Jalan Panggung	54
81	Jalan Jembatan Merah	187
82	Jalan Veteran	187
83	Jalan Indrapura	187
84	Jalan Kembangan Barat	187
85	Jalan Kembangan Timur	187
86	Jalan Kasuari	54
87	Jalan Kalimas Barat	54
88	U Turn Jalan Kedung Cowek	54
89	Jalan Kalimas Baru	54
90	Jalan Jakarta	54
91	Jembatan Merah Plaza	187
92	Jalan Kalimati Kulon	54
93	Jalan Kalimati Wetan	54
94	U Turn Jalan Nyamplungan	54
95	Jalan Kampung Seng	109
96	Jalan Kapasan	109
97	Jalan Gembong dekat Pasar Atom	109
98	Perempatan Jalan Gembong dan Semut Baru	242
99	Perempatan Jalan Undaan wetan, Jagalan, Kalianyar	242
100	Jalan Ngaglik	187
101	Jalan Tambak Adi	54
102	Jalan Donorejo Wetan	54
103	Perempatan Jalan Pragoto dan Bolodewo	55
104	Jalan Jaksa Agung Suprapto	55
105	Jalan Wijaya Kusuma dekat SMAN 9	55

Node	Lokasi	Skor
106	Jalan Walikota Mustajab dekat Grand City	135
107	Jalan Kalibokor I	135
108	Jalan Kalibokor Timur	55
109	Jalan Raya Ngagel Jaya	55
110	Jalan Ngagel Jaya Selatan dekat Ngagel Tama	55
111	Jalan Ngagel Madya	135
112	Jalan Ngagel Jaya Utara	135
113	Jalan Raya Manyar dekat perempatan Ngagel Jaya Selatan	135
114	Jalan Raya Manyar	135
115	Jalan Raya Nginden	135
116	Jalan Raya Prapen	55
117	Jalan Raya Jemur Sari	55
118	Jalan Prapen Indah	55
119	Jalan Tenggilis Lama II	55
120	Jalan Tenggilis Lama IV	55
121	Jalan Raya Tenggilis Mejoyo	55
122	U Turn Jalan Raya Panjang Jiwo Permai	55
123	Jalan Raya Kalirungkut dekat Kaliwaru I	55
124	Jalan Raya Kalirungkut	55
125	Jalan Rungkut Puskesmas	135
126	Jalan Rungkut Asri	55
127	U Turn Jalan Rungkut Asri Utara XIII	55
128	Jalan Rungkut Asri Tengah	55
129	Jalan Raya Rungkut Madya	55
130	Jalan Raya Rungkut Madya	55
131	Pangkalan Gunung Anyar	55
132	Jalan Rungkut Asri Barat VII	55
133	Jalan Rungkut Lor Gg III	55

C-6

Node	Lokasi	Skor
134	Jalan Rungkut Lor Gg X	55
135	Jalan Tenggilis Mejoyo dekat Ubaya	55
136	Jalan Raya Tenggilis dekat Metropolis Apt	55
137	Jalan Raya Tenggilis	55
138	Jalan Bratang Jaya XIX	135
139	Perempatan Jalan Bratang Jaya dan Bratang Binangun	135
140	Jalan Bratang Wetan	55
141	U Turn Jalan Ngagel Jaya Selatan Menuju Ngagel Jaya Barat	55
142	Jalan Ngagel Jaya Barat	55
143	Jalan Kalibokor II	55
144	Jalan Waspada	55
145	Jalan Bongkaran	55
146	Jalan Pasar Besar Wetan	133
147	Jalan Peneleh	133
148	Jalan Makam Peneleh	133
149	Jalan Mas Soenjoto	133
150	Jalan Kapas Krampung	133
151	Jalan Kapas Krampung dekat Jalan Ploso	133
152	Jalan Bronggalan	133
153	Perempatan Tambang Boyo dan Kedung Tarukan	133
154	Jalan Dharmahusadha dekat Graha Ara	133
155	Jalan Prof. Dr. Mustopo dekat Dharmahusadha Indah	133
156	Jalan Dharmahusadha Indah	133
157	Jalan Dharmahusadha Indah dekat jembatan	133
158	Jalan Raya Kertajaya Indah	133

Node	Lokasi	Skor
159	Jalan Manyar Kertoarjo	133
160	Jalan Raya Kertajaya Indah dekat perempatan Ir. Soekarno	133
161	U Turn Kertaya Indah dekat ITS	133
162	Jalan Manyar Kerta Adi	133
163	Jalan Gebang Putih	133
164	Jalan Arief Rachman Hakim	133
165	Depan Universitas Hang Tuah	133
166	Jalan Kejawan Gebang	133
167	Jalan Jolotundo	133
168	Jalan Tambak sari	133
169	Perempatan Jalan Bubutan dan Tembaan	213
170	Jalan Rungkut Harapan	80
171	Jalan Rungkut Asri Utara VI	80
172	Jalan Raya Kedung Asem	80
173	Jalan Kedungasem	80
174	Jalan Raya Kedung Baruk	80
175	Jalan Raya Kedung Baruk dekat Kali Jagir	80
176	Jalan Panjang Jiwo	80
177	Jalan Pucang Jajar	80
178	Perempatan Pucang Anom	80
179	Jalan Juwingan	80
180	Jalan Bagong Karimata	80
181	Jalan Lombok	80
182	Pertigaan embong cerme dan embong kemiri	80
183	Jalan Basuki Rahmat	80
184	Jalan Basuki Rahmat dekat McD	80
185	Jalan Embong Malang	80
186	Jalan Blauran	80

C-8

Node	Lokasi	Skor
187	Pertigaan Jalan Bubutan dan Raden Saleh	80
188	Jalan Pasar Turi	80
189	Jalan Semarang	80
190	Jalan Pahlawan	80
191	Jalan Gembongan	80
192	Jalan Tunjungan dekat pertigaan Genteng Besar	80
193	Jalan Genteng Besar	80
194	Jalan Geneteng Kali	80
195	Jalan Walikota Mustajab sebelum Jaksa Agung Suprapto	80
196	Jalan Walikota Mustajab sesudah Jaksa Agung Suprapto	80
197	U Turn Kertajaya arah Sulawesi	80
198	Pasar Pucang Anom	80
199	Jalan Sidotopo Kidul	55

LAMPIRAN D

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
1	2	900	2
1	27	700	2
2	1	140	1
2	3	450	1
3	4	450	1
3	25	650	2
4	3	450	1
4	5	650	1
5	4	650	1
5	6	900	2
6	7	550	1
6	24	650	1
7	6	550	1
7	8	500	2
7	107	550	2
8	9	500	1
8	54	600	1
9	10	400	1
9	34	870	2
10	11	600	2
10	22	500	1
11	10	600	2
11	12	500	1
12	11	500	1
12	13	550	2
13	12	550	2
13	14	750	2

D-2

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
13	104	700	3
14	13	750	2
14	15	650	2
14	99	350	2
14	100	400	2
15	14	650	2
15	16	500	2
16	15	500	2
16	17	600	1
16	76	850	3
17	16	600	1
17	18	450	1
18	17	450	1
18	19	500	1
19	18	500	1
19	20	850	2
19	199	600	3
20	19	850	2
20	21	600	2
21	20	600	2
22	23	500	1
22	54	950	2
22	197	950	2
23	7	600	2
24	5	600	2
25	26	750	2
26	2	450	2
27	28	450	1

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
28	29	550	1
28	57	750	2
29	28	550	1
29	30	450	1
30	31	500	2
31	32	450	2
32	33	650	3
33	8	650	4
34	35	450	2
34	22	850	2
35	34	450	2
35	36	500	1
36	37	550	1
36	50	800	2
37	38	500	1
37	167	700	2
38	39	500	1
38	49	270	1
39	38	500	1
39	40	500	2
40	41	480	1
40	151	700	2
40	152	400	1
41	42	600	2
41	152	700	2
42	41	600	2
42	43	450	2
43	42	450	2
43	44	550	1

D-4

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
44	43	550	1
44	45	450	2
45	44	450	2
45	46	450	1
46	45	450	1
46	47	600	2
47	46	600	2
47	48	450	1
48	47	900	2
49	36	750	2
50	51	650	1
50	153	850	3
50	154	800	2
51	52	500	2
52	53	500	2
53	35	600	1
53	50	450	2
54	55	650	1
54	182	700	2
55	56	500	1
56	29	600	1
57	58	450	1
58	59	400	1
59	60	350	1
60	61	650	2
61	1	500	1
62	63	700	2
63	62	700	2

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
63	64	800	2
64	63	800	2
64	65	700	2
65	64	700	2
65	66	700	3
66	65	1100	5
66	67	750	3
67	66	750	3
67	68	550	1
68	69	550	1
68	70	650	2
69	68	550	1
70	88	850	2
70	71	850	4
71	70	850	4
71	72	800	3
72	71	800	3
72	73	450	1
73	72	450	1
73	74	350	1
74	73	800	3
74	75	850	2
74	16	700	3
75	74	850	2
76	77	650	3
76	95	220	1
76	103	350	1
77	76	650	3
77	78	550	2

D-6

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
78	79	500	4
78	94	550	2
79	80	450	5
80	81	750	3
81	82	350	1
82	83	600	1
82	146	900	3
83	84	450	1
84	85	350	1
85	86	550	2
85	91	500	2
86	87	700	2
86	91	350	1
87	86	700	2
87	89	750	3
88	67	500	2
89	90	700	2
90	87	850	4
91	92	500	2
91	81	450	2
92	93	300	2
93	78	500	2
94	77	600	2
95	76	220	1
95	96	500	2
96	95	500	2
96	97	500	2
97	98	400	2

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
97	144	500	1
98	97	400	2
98	99	400	1
99	98	400	1
99	14	350	2
99	104	800	3
99	146	650	2
100	101	450	2
100	150	750	3
101	102	300	1
102	74	550	2
103	76	850	3
103	19	550	2
104	105	700	2
104	99	540	2
105	106	650	2
106	10	900	3
107	7	550	2
107	108	800	3
107	198	500	2
108	109	800	3
109	110	500	2
110	111	700	2
110	141	600	1
111	112	500	2
111	114	850	3
112	113	950	3
112	177	700	3
113	112	300	1

D-8

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
113	114	300	3
114	115	850	5
114	139	850	3
115	116	600	3
115	176	650	2
116	117	450	1
116	138	900	3
117	116	450	1
117	118	350	1
118	117	350	1
118	119	550	2
119	118	850	3
119	120	500	2
120	121	450	2
121	122	350	1
122	123	500	1
123	124	550	3
124	125	350	1
125	126	600	2
125	133	550	3
125	171	550	1
125	172	550	2
126	125	600	2
126	127	550	2
127	128	500	1
128	129	450	2
129	130	550	2
129	132	800	3

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
130	129	550	2
130	131	500	1
131	130	500	1
132	126	400	1
133	134	450	2
134	135	500	2
135	136	500	1
136	137	500	1
137	119	850	3
138	139	400	1
138	115	850	2
139	138	400	1
139	140	550	2
139	113	950	3
140	110	550	1
141	142	400	2
142	143	900	4
143	107	350	2
144	145	600	2
145	96	550	2
146	147	550	2
146	169	500	3
147	148	500	2
148	149	400	2
149	99	550	2
150	151	600	2
150	168	900	3
151	150	600	2
151	40	700	2

D-10

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
152	39	650	2
152	40	400	1
152	153	650	2
153	37	300	1
153	50	700	2
154	155	700	2
154	51	700	2
155	154	700	2
155	156	700	2
156	155	700	2
156	157	700	1
157	156	700	1
157	158	550	1
158	159	450	1
158	160	650	1
159	158	450	1
159	157	550	1
159	37	260	1
160	158	650	1
160	161	750	2
161	162	720	2
162	163	400	1
162	160	700	4
163	162	400	1
163	164	750	2
164	163	750	2
164	165	800	2
165	164	800	2

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
165	166	950	4
166	165	950	4
167	152	450	1
168	13	800	3
169	83	750	1
169	188	800	3
170	171	600	2
171	170	600	2
171	125	550	1
172	125	550	2
172	173	450	2
173	172	450	2
173	174	600	3
174	173	600	3
174	175	850	3
175	174	850	3
175	176	750	2
176	175	750	2
176	138	900	3
177	178	600	2
177	111	650	3
178	177	600	2
178	179	650	3
179	180	550	2
180	181	350	2
181	8	400	2
181	7	550	2
182	183	900	2
183	184	700	2

D-12

Node Awal	Node Akhir	Jarak (meter)	Waktu tempuh (menit)
184	185	650	2
185	186	750	2
186	187	700	1
187	169	550	2
187	190	500	1
188	189	500	3
189	187	700	2
190	191	500	1
191	192	450	2
192	193	450	2
193	194	450	1
194	195	800	2
195	196	700	2
196	106	400	1
197	181	900	3
198	178	350	2
199	103	600	3

LAMPIRAN E

Variabel	Deskripsi
x1.2	Kunjungan dari node 1 ke node 2
x1.27	Kunjungan dari node 1 ke node 27
x2.1	Kunjungan dari node 2 ke node 1
x2.3	Kunjungan dari node 2 ke node 3
x3.4	Kunjungan dari node 3 ke node 4
x3.25	Kunjungan dari node 3 ke node 25
x4.3	Kunjungan dari node 4 ke node 3
x4.5	Kunjungan dari node 4 ke node 5
x5.4	Kunjungan dari node 5 ke node 4
x5.6	Kunjungan dari node 5 ke node 6
x6.7	Kunjungan dari node 6 ke node 7
x6.24	Kunjungan dari node 6 ke node 24
x7.6	Kunjungan dari node 7 ke node 6
x7.8	Kunjungan dari node 7 ke node 8
x7.107	Kunjungan dari node 7 ke node 107
x8.9	Kunjungan dari node 8 ke node 9
x8.54	Kunjungan dari node 8 ke node 54
x9.10	Kunjungan dari node 9 ke node 10
x9.34	Kunjungan dari node 9 ke node 34
x10.11	Kunjungan dari node 10 ke node 11
x10.22	Kunjungan dari node 10 ke node 22
x11.10	Kunjungan dari node 11 ke node 10
x11.12	Kunjungan dari node 11 ke node 12
x12.11	Kunjungan dari node 12 ke node 11
x12.13	Kunjungan dari node 12 ke node 13
x13.12	Kunjungan dari node 13 ke node 12
x13.14	Kunjungan dari node 13 ke node 14
x13.104	Kunjungan dari node 13 ke node 104

E-2

Variabel	Deskripsi
x14.13	Kunjungan dari node 14 ke node 13
x14.15	Kunjungan dari node 14 ke node 15
x14.99	Kunjungan dari node 14 ke node 99
x14.100	Kunjungan dari node 14 ke node 100
x15.14	Kunjungan dari node 15 ke node 14
x15.16	Kunjungan dari node 15 ke node 16
x16.15	Kunjungan dari node 16 ke node 15
x16.17	Kunjungan dari node 16 ke node 17
x16.76	Kunjungan dari node 16 ke node 76
x17.16	Kunjungan dari node 17 ke node 16
x17.18	Kunjungan dari node 17 ke node 18
x18.17	Kunjungan dari node 18 ke node 17
x18.19	Kunjungan dari node 18 ke node 19
x19.18	Kunjungan dari node 19 ke node 18
x19.20	Kunjungan dari node 19 ke node 20
x19.199	Kunjungan dari node 19 ke node 199
x20.19	Kunjungan dari node 20 ke node 19
x20.21	Kunjungan dari node 20 ke node 21
x21.20	Kunjungan dari node 21 ke node 20
x22.23	Kunjungan dari node 22 ke node 23
x22.54	Kunjungan dari node 22 ke node 54
x22.197	Kunjungan dari node 22 ke node 197
x23.7	Kunjungan dari node 23 ke node 7
x24.5	Kunjungan dari node 24 ke node 5
x25.26	Kunjungan dari node 25 ke node 26
x26.2	Kunjungan dari node 26 ke node 2
x27.28	Kunjungan dari node 27 ke node 28
x28.29	Kunjungan dari node 28 ke node 29
x28.57	Kunjungan dari node 28 ke node 57

Variabel	Deskripsi
x29.28	Kunjungan dari node 29 ke node 28
x29.30	Kunjungan dari node 29 ke node 30
x30.31	Kunjungan dari node 30 ke node 31
x31.32	Kunjungan dari node 31 ke node 32
x32.33	Kunjungan dari node 32 ke node 33
x33.8	Kunjungan dari node 33 ke node 8
x34.35	Kunjungan dari node 34 ke node 35
x34.22	Kunjungan dari node 34 ke node 22
x35.34	Kunjungan dari node 35 ke node 34
x35.36	Kunjungan dari node 35 ke node 36
x36.37	Kunjungan dari node 36 ke node 37
x36.50	Kunjungan dari node 36 ke node 50
x37.38	Kunjungan dari node 37 ke node 38
x37.167	Kunjungan dari node 37 ke node 167
x38.39	Kunjungan dari node 38 ke node 39
x38.49	Kunjungan dari node 38 ke node 49
x39.38	Kunjungan dari node 39 ke node 38
x39.40	Kunjungan dari node 39 ke node 40
x40.41	Kunjungan dari node 40 ke node 41
x40.151	Kunjungan dari node 40 ke node 151
x40.152	Kunjungan dari node 40 ke node 152
x41.42	Kunjungan dari node 41 ke node 42
x41.152	Kunjungan dari node 41 ke node 152
x42.41	Kunjungan dari node 42 ke node 41
x42.43	Kunjungan dari node 42 ke node 43
x43.42	Kunjungan dari node 43 ke node 42
x43.44	Kunjungan dari node 43 ke node 44
x44.43	Kunjungan dari node 44 ke node 43
x44.45	Kunjungan dari node 44 ke node 45
x45.44	Kunjungan dari node 45 ke node 44

E-4

Variabel	Deskripsi
x45.46	Kunjungan dari node 45 ke node 46
x46.45	Kunjungan dari node 46 ke node 45
x46.47	Kunjungan dari node 46 ke node 47
x47.46	Kunjungan dari node 47 ke node 46
x47.48	Kunjungan dari node 47 ke node 48
x48.47	Kunjungan dari node 48 ke node 47
x49.36	Kunjungan dari node 49 ke node 36
x50.51	Kunjungan dari node 50 ke node 51
x50.153	Kunjungan dari node 50 ke node 153
x50.154	Kunjungan dari node 50 ke node 154
x51.52	Kunjungan dari node 51 ke node 52
x52.53	Kunjungan dari node 52 ke node 53
x53.35	Kunjungan dari node 53 ke node 35
x53.50	Kunjungan dari node 53 ke node 50
x54.55	Kunjungan dari node 54 ke node 55
x54.182	Kunjungan dari node 54 ke node 182
x55.56	Kunjungan dari node 55 ke node 56
x56.29	Kunjungan dari node 56 ke node 29
x57.58	Kunjungan dari node 57 ke node 58
x58.59	Kunjungan dari node 58 ke node 59
x59.60	Kunjungan dari node 59 ke node 60
x60.61	Kunjungan dari node 60 ke node 61
x61.1	Kunjungan dari node 61 ke node 1
x62.63	Kunjungan dari node 62 ke node 63
x63.62	Kunjungan dari node 63 ke node 62
x63.64	Kunjungan dari node 63 ke node 64
x64.63	Kunjungan dari node 64 ke node 63
x64.65	Kunjungan dari node 64 ke node 65
x65.64	Kunjungan dari node 65 ke node 64

Variabel	Deskripsi
x65.66	Kunjungan dari node 65 ke node 66
x66.65	Kunjungan dari node 66 ke node 65
x66.67	Kunjungan dari node 66 ke node 67
x67.66	Kunjungan dari node 67 ke node 66
x67.68	Kunjungan dari node 67 ke node 68
x68.69	Kunjungan dari node 68 ke node 69
x68.70	Kunjungan dari node 68 ke node 70
x69.68	Kunjungan dari node 69 ke node 68
x70.88	Kunjungan dari node 70 ke node 88
x70.71	Kunjungan dari node 70 ke node 71
x71.70	Kunjungan dari node 71 ke node 70
x71.72	Kunjungan dari node 71 ke node 72
x72.71	Kunjungan dari node 72 ke node 71
x72.73	Kunjungan dari node 72 ke node 73
x73.72	Kunjungan dari node 73 ke node 72
x73.74	Kunjungan dari node 73 ke node 74
x74.73	Kunjungan dari node 74 ke node 73
x74.75	Kunjungan dari node 74 ke node 75
x74.16	Kunjungan dari node 74 ke node 16
x75.74	Kunjungan dari node 75 ke node 74
x76.77	Kunjungan dari node 76 ke node 77
x76.95	Kunjungan dari node 76 ke node 95
x76.103	Kunjungan dari node 76 ke node 103
x77.76	Kunjungan dari node 77 ke node 76
x77.78	Kunjungan dari node 77 ke node 78
x78.79	Kunjungan dari node 78 ke node 79
x78.94	Kunjungan dari node 78 ke node 94
x79.80	Kunjungan dari node 79 ke node 80
x80.81	Kunjungan dari node 80 ke node 81
x81.82	Kunjungan dari node 81 ke node 82

E-6

Variabel	Deskripsi
x82.83	Kunjungan dari node 82 ke node 83
x82.146	Kunjungan dari node 82 ke node 146
x83.84	Kunjungan dari node 83 ke node 84
x84.85	Kunjungan dari node 84 ke node 85
x85.86	Kunjungan dari node 85 ke node 86
x85.91	Kunjungan dari node 85 ke node 91
x86.87	Kunjungan dari node 86 ke node 87
x86.91	Kunjungan dari node 86 ke node 91
x87.86	Kunjungan dari node 87 ke node 86
x87.89	Kunjungan dari node 87 ke node 89
x88.67	Kunjungan dari node 88 ke node 67
x89.90	Kunjungan dari node 89 ke node 90
x90.87	Kunjungan dari node 90 ke node 87
x91.92	Kunjungan dari node 91 ke node 92
x91.81	Kunjungan dari node 91 ke node 81
x92.93	Kunjungan dari node 92 ke node 93
x93.78	Kunjungan dari node 93 ke node 78
x94.77	Kunjungan dari node 94 ke node 77
x95.76	Kunjungan dari node 95 ke node 76
x95.96	Kunjungan dari node 95 ke node 96
x96.95	Kunjungan dari node 96 ke node 95
x96.97	Kunjungan dari node 96 ke node 97
x97.98	Kunjungan dari node 97 ke node 98
x97.144	Kunjungan dari node 97 ke node 144
x98.97	Kunjungan dari node 98 ke node 97
x98.99	Kunjungan dari node 98 ke node 99
x99.98	Kunjungan dari node 99 ke node 98
x99.14	Kunjungan dari node 99 ke node 14
x99.104	Kunjungan dari node 99 ke node 104

Variabel	Deskripsi
x99.146	Kunjungan dari node 99 ke node 146
x100.101	Kunjungan dari node 100 ke node 101
x100.150	Kunjungan dari node 100 ke node 150
x101.102	Kunjungan dari node 101 ke node 102
x102.74	Kunjungan dari node 102 ke node 74
x103.76	Kunjungan dari node 103 ke node 76
x103.19	Kunjungan dari node 103 ke node 19
x104.105	Kunjungan dari node 104 ke node 105
x104.99	Kunjungan dari node 104 ke node 99
x105.106	Kunjungan dari node 105 ke node 106
x106.10	Kunjungan dari node 106 ke node 10
x107.7	Kunjungan dari node 107 ke node 7
x107.108	Kunjungan dari node 107 ke node 108
x107.198	Kunjungan dari node 107 ke node 198
x108.109	Kunjungan dari node 108 ke node 109
x109.110	Kunjungan dari node 109 ke node 110
x110.111	Kunjungan dari node 110 ke node 111
x110.141	Kunjungan dari node 110 ke node 141
x111.112	Kunjungan dari node 111 ke node 112
x111.114	Kunjungan dari node 111 ke node 114
x112.113	Kunjungan dari node 112 ke node 113
x112.177	Kunjungan dari node 112 ke node 177
x113.112	Kunjungan dari node 113 ke node 112
x113.114	Kunjungan dari node 113 ke node 114
x114.115	Kunjungan dari node 114 ke node 115
x114.139	Kunjungan dari node 114 ke node 139
x115.116	Kunjungan dari node 115 ke node 116
x115.176	Kunjungan dari node 115 ke node 176
x116.117	Kunjungan dari node 116 ke node 117
x116.138	Kunjungan dari node 116 ke node 138

Variabel	Deskripsi
x117.116	Kunjungan dari node 117 ke node 116
x117.118	Kunjungan dari node 117 ke node 118
x118.117	Kunjungan dari node 118 ke node 117
x118.119	Kunjungan dari node 118 ke node 119
x119.118	Kunjungan dari node 119 ke node 118
x119.120	Kunjungan dari node 119 ke node 120
x120.121	Kunjungan dari node 120 ke node 121
x121.122	Kunjungan dari node 121 ke node 122
x122.123	Kunjungan dari node 122 ke node 123
x123.124	Kunjungan dari node 123 ke node 124
x124.125	Kunjungan dari node 124 ke node 125
x125.126	Kunjungan dari node 125 ke node 126
x125.133	Kunjungan dari node 125 ke node 133
x125.171	Kunjungan dari node 125 ke node 171
x125.172	Kunjungan dari node 125 ke node 172
x126.125	Kunjungan dari node 126 ke node 125
x126.127	Kunjungan dari node 126 ke node 127
x127.128	Kunjungan dari node 127 ke node 128
x128.129	Kunjungan dari node 128 ke node 129
x129.130	Kunjungan dari node 129 ke node 130
x129.132	Kunjungan dari node 129 ke node 132
x130.129	Kunjungan dari node 130 ke node 129
x130.131	Kunjungan dari node 130 ke node 131
x131.130	Kunjungan dari node 131 ke node 130
x132.126	Kunjungan dari node 132 ke node 126
x133.134	Kunjungan dari node 133 ke node 134
x134.135	Kunjungan dari node 134 ke node 135
x135.136	Kunjungan dari node 135 ke node 136
x136.137	Kunjungan dari node 136 ke node 137

Variabel	Deskripsi
x137.119	Kunjungan dari node 137 ke node 119
x138.139	Kunjungan dari node 138 ke node 139
x138.115	Kunjungan dari node 138 ke node 115
x139.138	Kunjungan dari node 139 ke node 138
x139.140	Kunjungan dari node 139 ke node 140
x139.113	Kunjungan dari node 139 ke node 113
x140.110	Kunjungan dari node 140 ke node 110
x141.142	Kunjungan dari node 141 ke node 142
x142.143	Kunjungan dari node 142 ke node 143
x143.107	Kunjungan dari node 143 ke node 107
x144.145	Kunjungan dari node 144 ke node 145
x145.96	Kunjungan dari node 145 ke node 96
x146.147	Kunjungan dari node 146 ke node 147
x146.169	Kunjungan dari node 146 ke node 169
x147.148	Kunjungan dari node 147 ke node 148
x148.149	Kunjungan dari node 148 ke node 149
x149.99	Kunjungan dari node 149 ke node 99
x150.151	Kunjungan dari node 150 ke node 151
x150.168	Kunjungan dari node 150 ke node 168
x151.150	Kunjungan dari node 151 ke node 150
x151.40	Kunjungan dari node 151 ke node 40
x152.39	Kunjungan dari node 152 ke node 39
x152.40	Kunjungan dari node 152 ke node 40
x152.153	Kunjungan dari node 152 ke node 153
x153.37	Kunjungan dari node 153 ke node 37
x153.50	Kunjungan dari node 153 ke node 50
x154.155	Kunjungan dari node 154 ke node 155
x154.51	Kunjungan dari node 154 ke node 51
x155.154	Kunjungan dari node 155 ke node 154
x155.156	Kunjungan dari node 155 ke node 156

Variabel	Deskripsi
x156.155	Kunjungan dari node 156 ke node 155
x156.157	Kunjungan dari node 156 ke node 157
x157.156	Kunjungan dari node 157 ke node 156
x157.158	Kunjungan dari node 157 ke node 158
x158.159	Kunjungan dari node 158 ke node 159
x158.160	Kunjungan dari node 158 ke node 160
x159.158	Kunjungan dari node 159 ke node 158
x159.157	Kunjungan dari node 159 ke node 157
x159.37	Kunjungan dari node 159 ke node 37
x160.158	Kunjungan dari node 160 ke node 158
x160.161	Kunjungan dari node 160 ke node 161
x161.162	Kunjungan dari node 161 ke node 162
x162.163	Kunjungan dari node 162 ke node 163
x162.160	Kunjungan dari node 162 ke node 160
x163.162	Kunjungan dari node 163 ke node 162
x163.164	Kunjungan dari node 163 ke node 164
x164.163	Kunjungan dari node 164 ke node 163
x164.165	Kunjungan dari node 164 ke node 165
x165.164	Kunjungan dari node 165 ke node 164
x165.166	Kunjungan dari node 165 ke node 166
x166.165	Kunjungan dari node 166 ke node 165
x167.152	Kunjungan dari node 167 ke node 152
x168.13	Kunjungan dari node 168 ke node 13
x169.83	Kunjungan dari node 169 ke node 83
x169.188	Kunjungan dari node 169 ke node 188
x170.171	Kunjungan dari node 170 ke node 171
x171.170	Kunjungan dari node 171 ke node 170
x171.125	Kunjungan dari node 171 ke node 125
x172.125	Kunjungan dari node 172 ke node 125

Variabel	Deskripsi
x172.173	Kunjungan dari node 172 ke node 173
x173.172	Kunjungan dari node 173 ke node 172
x173.174	Kunjungan dari node 173 ke node 174
x174.173	Kunjungan dari node 174 ke node 173
x174.175	Kunjungan dari node 174 ke node 175
x175.174	Kunjungan dari node 175 ke node 174
x175.176	Kunjungan dari node 175 ke node 176
x176.175	Kunjungan dari node 176 ke node 175
x176.138	Kunjungan dari node 176 ke node 138
x177.178	Kunjungan dari node 177 ke node 178
x177.111	Kunjungan dari node 177 ke node 111
x178.177	Kunjungan dari node 178 ke node 177
x178.179	Kunjungan dari node 178 ke node 179
x179.180	Kunjungan dari node 179 ke node 180
x180.181	Kunjungan dari node 180 ke node 181
x181.8	Kunjungan dari node 181 ke node 8
x181.7	Kunjungan dari node 181 ke node 7
x182.183	Kunjungan dari node 182 ke node 183
x183.184	Kunjungan dari node 183 ke node 184
x184.185	Kunjungan dari node 184 ke node 185
x185.186	Kunjungan dari node 185 ke node 186
x186.187	Kunjungan dari node 186 ke node 187
x187.169	Kunjungan dari node 187 ke node 169
x187.190	Kunjungan dari node 187 ke node 190
x188.189	Kunjungan dari node 188 ke node 189
x189.187	Kunjungan dari node 189 ke node 187
x190.191	Kunjungan dari node 190 ke node 191
x191.192	Kunjungan dari node 191 ke node 192
x192.193	Kunjungan dari node 192 ke node 193
x193.194	Kunjungan dari node 193 ke node 194

E-12

Variabel	Deskripsi
x194.195	Kunjungan dari node 194 ke node 195
x195.196	Kunjungan dari node 195 ke node 196
x196.106	Kunjungan dari node 196 ke node 106
x197.181	Kunjungan dari node 197 ke node 181
x198.178	Kunjungan dari node 198 ke node 178
x199.103	Kunjungan dari node 199 ke node 103

LAMPIRAN F

Tabel F-1 Parameter Uji Coba 1

Uji Coba 1				
Node	1 ke 91		IFS	1 juta iterasi
tMax	60		ILS	2 juta iterasi
RWS	30			

Hasil:

Tabel F-2 Hasil Uji Coba 1

Percobaan	Waktu Tempuh	Skor
1	51	5391
2	53	4423
3	53	5299
4	37	4540
5	39	4750
6	53	5166
7	52	5133
8	54	4428
9	56	5433
10	47	5036

Tabel F-3 Parameter Uji Coba 2

Uji Coba 2				
Node	165 ke 52		IFS	1 juta iterasi
tMax	60		ILS	2 juta iterasi
RWS	30			

Hasil:

Tabel F-4 Hasil Uji Coba 2

Percobaan	Waktu Tempuh	Skor
1	59	4862
2	59	4675
3	59	4862
4	59	4345
5	59	4345
6	59	4345
7	59	5128
8	59	5128
9	59	5425
10	59	4995

Tabel F-5 Parameter Uji Coba 3

Uji Coba 3				
Node	165 ke 52		IFS	1 juta iterasi
tMax	30		ILS	2 juta iterasi
RWS	30			

Hasil:

Tabel F-6 Hasil Uji Coba 3

Percobaan	Waktu Tempuh	Skor
1	28	2119
2	28	2119
3	28	2119
4	28	2119
5	28	2119

Tabel F-7 Parameter Uji Coba 4

Uji Coba 4				
Node	1 ke 91		IFS	1 juta iterasi
tMax	60		ILS	2 juta iterasi
RWS	10,30,50			

Hasil:

Tabel F-8 Hasil Uji Coba 4 RWS:10

Percobaan	Waktu Tempuh	Skor
RWS: 10		
1	58	4626
2	57	5186
3	56	5246
4	45	4227
5	45	4788

Tabel F-9 Hasil Uji Coba 4 RWS: 30

Percobaan	Waktu Tempuh	Skor
RWS: 30		
1	58	5421
2	49	5156
3	45	4870
4	52	5016
5	56	4512

Tabel F-10 Hasil Uji Coba 4 RWS: 50

Percobaan	Waktu Tempuh	Skor
RWS: 50		
1	56	5433
2	44	5394
3	53	5988
4	52	5203
5	53	5486

Tabel F-11 Parameter Uji Coba 5

Uji Coba 5				
Node	1 ke 91		IFS	1 juta iterasi
tMax	40		ILS	2 juta iterasi
RWS	30			

Hasil:

Tabel F-12 Hasil Uji Coba 5 Swap Method

Percobaan	Waktu Tempuh	Skor
Swap		
1	39	1219
2	31	1022
3	31	1055
4	33	979
5	31	878

Tabel F-13 Hasil Uji Coba 5 Insertion Method

Percobaan	Waktu Tempuh	Skor
Insertion		
1	37	4220
2	39	4563
3	39	4750
4	39	4883
5	39	4716

Tabel F-14 Hasil Uji Coba 5 Deletion Method

Percobaan	Waktu Tempuh	Skor
Deletion		
1	33	963
2	31	826
3	39	1001
4	39	934
5	31	1000

Tabel F-15 Hasil Uji Coba 5 All Method

Percobaan	Waktu Tempuh	Skor
All Method		
1	35	4145
2	39	4563
3	39	4750
4	39	3567
5	31	4002

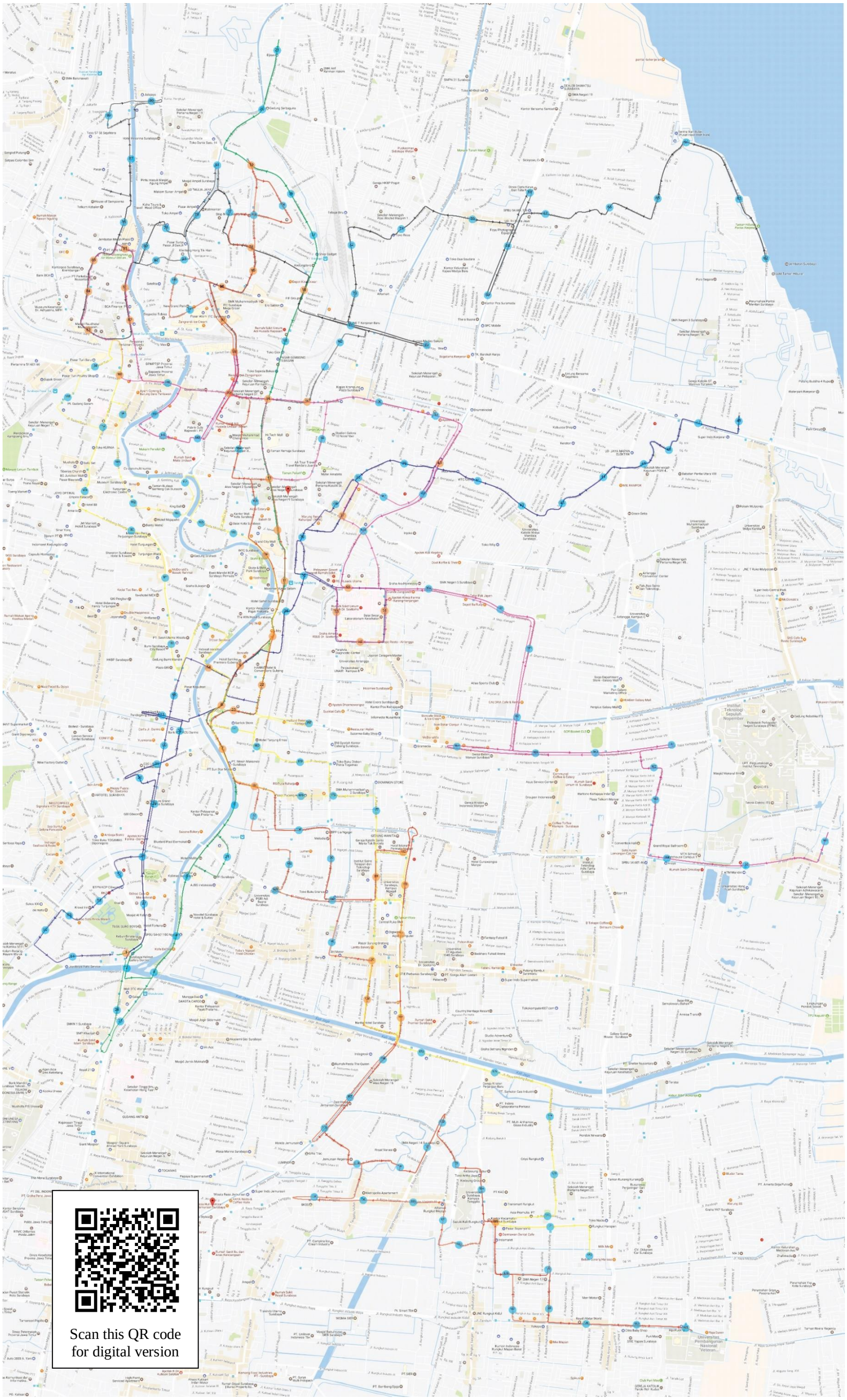
Halaman ini sengaja dikosongkan

LAMPIRAN G

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

Halaman ini sengaja dikosongkan

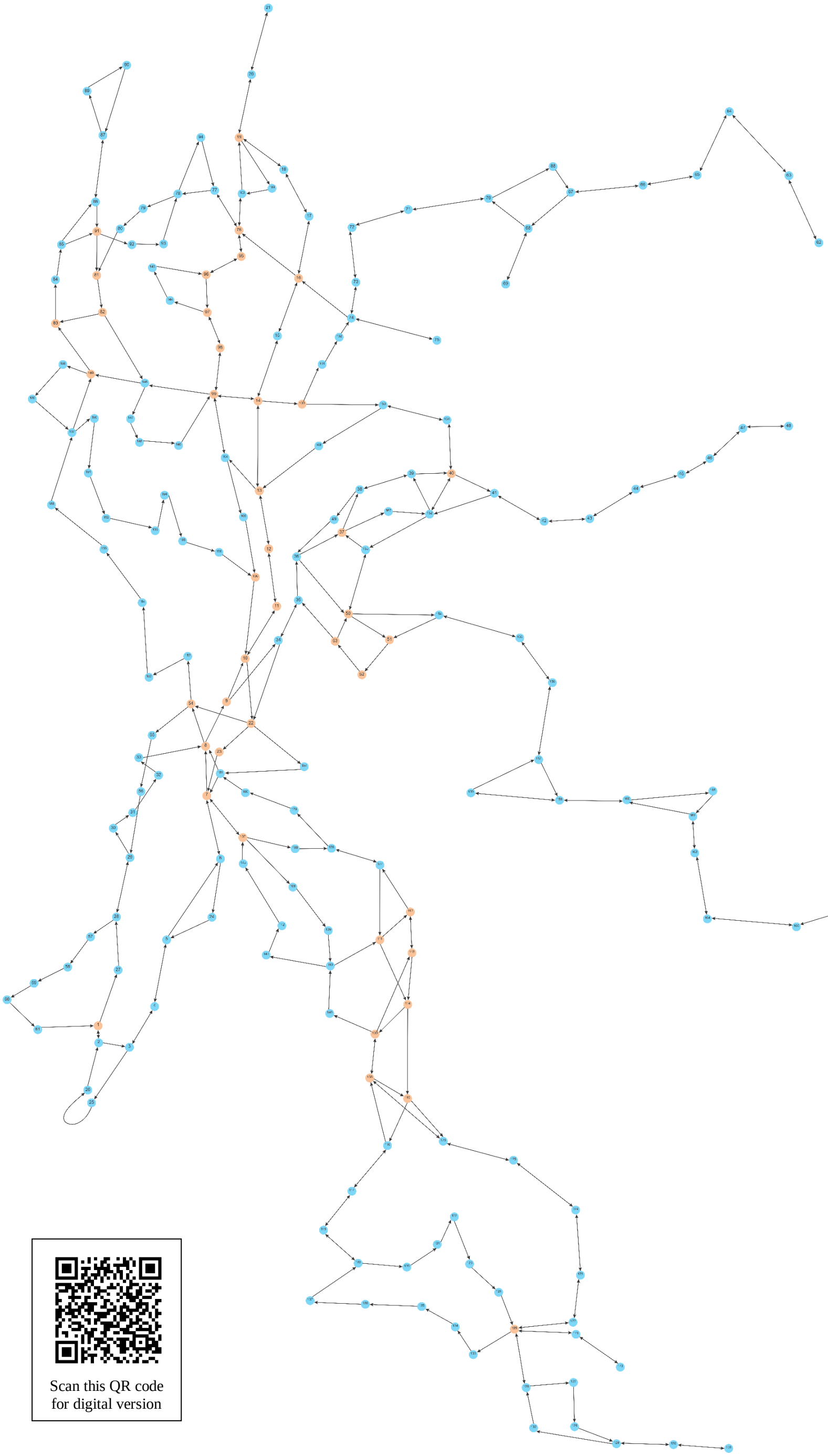
LAMPIRAN H



Scan this QR code
for digital version

Halaman ini sengaja dikosongkan

LAMPIRAN I



Scan this QR code
for digital version