HIBRIDISASI ALGORITME GENETIKA DAN *SIMULATED ANNEALING* UNTUK OPTIMASI *MULTI-TRIP VEHICLE ROUTING PROBLEM WITH TIME WINDOWS* (STUDI KASUS: PARIWISATA KABUPATEN BANYUWANGI)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Amalia Kartika Ariyani
NIM: 135150201111212

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2017

PENGESAHAN

HIBRIDISASI ALGORITME GENETIKA DAN *SIMULATED ANNEALING* UNTUK OPTIMASI *MULTI-TRIP VEHICLE ROUTING PROBLEM WITH TIME WINDOWS* (STUDI KASUS: PARIWISATA KABUPATEN BANYUWANGI)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Amalia Kartika Ariyani
NIM: 135150201111212

Skripsi ini telah diuji dan dinyatakan lulus pada
19 Juli 2017
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D
NIP: 19720919 199702 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 13 Juni 2017

Amalia Kartika Ariyani
NIM. 135150201111212

KATA PENGANTAR

Dengan menyebut nama Allah SWT Yang Maha Pengasih dan Maha Penyayang. Puji syukur kehadirat Allah SWT karena limpahan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan skripsi dengan baik yang berjudul "Hibridisasi Algoritme Genetika dan *Simulated Annealing* untuk Optimasi *Multi-Trip Vehicle Routing Problem With Time Windows* (Studi Kasus: Pariwisata Kabupaten Banyuwangi)". Shalawat serta salam senantiasa tercurahkan kepada junjungan Nabi besar Muhammad SAW beserta keluarga dan para sahabatnya. Skripsi ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Program Studi Teknik Informatika Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Pada kesempatan ini penulis juga ingin menyampaikan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan bantuan dan dukungan dalam menyelesaikan skripsi ini, antara lain :

1. Wayan Firdaus Mahmudy, S.Si, M.T., Ph.D, selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang sekaligus selaku Dosen Pembimbing I yang telah membimbing, mengarahkan dan memberikan banyak saran kepada penulis sehingga dapat menyelesaikan skripsi ini.
2. Seluruh jajaran pimpinan dan civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Keluarga Penulis diantaranya Suroyo, Martini, Ika, Bagus, Rara dan Brilliant serta keluarga besar atas segala do'a, nasihat, dukungan baik moril maupun materiil yang begitu besar terhadap kelancaran dalam menyelesaikan skripsi ini.
4. Seluruh keluarga BIOS FILKOM UB serta kawan-kawan Teknik Informatika FILKOM UB 2013 yang telah memberikan semangat dan dukungan kepada penulis.

Dengan kerendahan hati, penulis menyadari bahwa skripsi ini masih memiliki banyak kekurangan. Oleh karena itu kritik dan saran yang bersifat konstruktif sangat dibutuhkan sebagai pedoman untuk menyempurnakan skripsi ini agar lebih baik. Penulis berharap semoga skripsi ini dapat bermanfaat bagi diri sendiri maupun bagi semua pihak.

Malang, 13 Juni 2017

Penulis
Kartikaamalia95@gmail.com

ABSTRAK

Amalia Kartika Ariyani. 2017. Hibridisasi Algoritme Genetika dan Simulated Annealing untuk Optimasi Multi-Trip Vehicle Routing Problem with Time Windows (Studi Kasus: Pariwisata Kabupaten Banyuwangi). Fakultas Ilmu Komputer, Universitas Brawijaya, Malang. Dosen Pembimbing I : Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D

Seiring dengan pesatnya peningkatan jumlah wisatawan serta destinasi wisata yang berada di Banyuwangi, timbul beberapa permasalahan dari segi wisatawan. Dengan waktu yang terbatas, mereka ingin mengunjungi destinasi wisata sebanyak-banyaknya tanpa harus membuang waktu di perjalanan. Namun kendala lain muncul ketika pengunjung dan objek wisata yang dikunjungi masing-masing memiliki *time windows* atau waktu kunjungan yang ditentukan oleh wisatawan dan jam buka objek wisata itu sendiri. Dari permasalahan ini maka diperlukan suatu solusi untuk mengatasi permasalahan penjadwalan wisata tersebut dengan waktu tempuh yang paling optimal berdasarkan *time windows* dari wisatawan dan objek wisata. Permasalahan tersebut secara umum biasa disebut dengan *Vehicle Routing Problem with Time Windows* (VRPTW). Permasalahan VRPTW dapat diselesaikan dengan beberapa metode optimasi, diantaranya menggunakan hibridisasi algoritme genetika dan *simulated annealing*. Algoritme genetika (GA) akan digunakan untuk mengeksplorasi permasalahan global optimum, sedangkan *Simulated annealing* (SA) akan digunakan untuk mengeksploitasi permasalahan lokal optimum. Nilai *fitness* mendekati optimal yaitu sebesar 1.0616 didapatkan dengan menggunakan parameter GA-SA pada ukuran populasi sebesar 400, banyak generasi sebesar 800, kombinasi *cr* dan *mr* sebesar 0.3 dan 0.1, temperatur awal 0.9, faktor pendinginan sebesar 0.9, dan koefisien penerimaan solusi baru sebesar 200.

Kata kunci: m*ulti-trip* VRPTW, algoritme genetika, *simulated annealing*, hibridisasi GA-SA

ABSTRACT

Amalia Kartika Ariyani. 2017. Hybridization of Genetic Algorithm and Simulated Annealing for Optimization of Multi-Trip Vehicle Routing Problem with Time Windows (Case Study: Tourism of Banyuwangi Regency). Faculty of Computer Science, Brawijaya University, Malang. Supervising Professor I: Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D.

Along with the rapidly excalation in the number of tourists as well as tourism which are located in Banyuwangi, problems arise from the tourists. With limited time, they would like to visit as many destinations without wasting time in traveling. However, another problem happened when the visitor and each visited destination has time windows or visiting time which has been established. From this issue, it needs a solution to solve the problems regarding with the tourism scheduling with the optimal time based on time windows from the tourists and the destinations. Generally, these problems called Vehicle Routing Problem with Time Windows (VRPTW). VRPTW problems can be solved with several optimization methods, including using hybridizing genetic algorithm and simulated annealing. Genetic algorithms (GA) will be used to explore the global optimum problem, while Simulated annealing (SA) will be used to exploit a local optimum problem. The fitness value approach optimal i.e. of 1.0616 obtained by using parameters of GA-SA on the size of population of 400, amount of generation by 800, combination of cr and the mr of 0.3 and 0.1, initial temperature of 0.9, cooling factor of 0.9, and the coefficient of acceptance of a new solution amounting to 200.

Keywords: *multi-trip VRPTW, genetic algorithms, simulated annealing, hybridization GA-SA*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Pariwisata merupakan perjalanan keluar domisili asli yang dilakukan sesekali yang dilakukan sendiri maupun bersama-sama dalam lingkup ilmu, social, budaya maupun alam (Spilane, 1987 dalam Soebagyo, 2012). Pariwisata menjadi suatu pilihan sebagian besar orang untuk berekreasi dan menghilangkan kepenatan rutinitas pekerjaan yang padat. Destinasi wisata yang dipilih untuk berwisata sangat beragam, namun banyak orang yang memilih keluar domisilinya demi mendapatkan suasana baru. Banyuwangi merupakan salah satu kabupaten yang mempunyai beragam destinasi wisata, mulai dari pegunungan hingga pantai dan laut. Hal ini menjadikan Indonesia sebagai destinasi wisata yang cukup bersaing bagi wisatawan nusantara maupun mancanegara.

Berdasarkan data yang berasal dari Dinas Kebudayaan dan Pariwisata Kabupaten Banyuwangi, pada tahun 2014 saja total wisatawan nusantara maupun mancanegara yang berkunjung ke Kabupaten Banyuwangi mencapai 1.495.629 wisatawan, dan setahun setelahnya yaitu pada tahun 2015, total wisatawan nusantara maupun mancanegara meningkat 31% menjadi 1.972.393 wisatawan (Anggodo *et al*, 2016). Perbaikan tata kelola wisata, promosi, dan perbaikan infrastruktur terus digencarkan oleh pemerintah Kabupaten Banyuwangi demi meningkatkan angka kunjungan wisata di Kabupaten Banyuwangi.

Seiring dengan pesatnya peningkatan jumlah wisatawan serta jumlah destinasi wisata yang berada di Banyuwangi, timbul beberapa permasalahan dari segi wisatawan. Dengan waktu yang terbatas, mereka ingin mengunjungi destinasi wisata sebanyak-banyaknya tanpa harus membuang waktu di perjalanan. Namun kendala lain muncul ketika pengunjung dan objek wisata yang dikunjungi masing-masing memiliki *time windows* atau waktu kunjungan yang ditentukan oleh wisatawan dan jam buka objek wisata itu sendiri. Beberapa teknologi misalnya Google Maps telah mampu melakukan perhitungan jarak dan waktu tempuh dari satu titik ke titik lain, namun teknologi ini belum memberikan fasilitas untuk menentukan objek wisata mana saja yang akan dikunjungi dalam durasi waktu yang telah ditentukan oleh wisatawan yang paling optimal tanpa mengesampingkan jam buka objek wisata.

Dari permasalahan ini maka diperlukan suatu solusi untuk mengatasi permasalahan penjadwalan wisata tersebut dengan waktu tempuh yang paling optimal berdasarkan *time windows* dari wisatawan dan objek wisata. Permasalahan tersebut secara umum biasa disebut dengan *Vehicle Routing Problem* (VRP). VRP bertujuan untuk merancang penjadwalan rute yang paling optimal sebagai layanan untuk pelanggan dari satu atau lebih depot (titik awal) ke sejumlah node tujuan dengan seperangkat kendaraan (homogen) dengan kapasitas terbatas (Wassan dan Nagy, 2014). Terdapat beberapa pengembangan dari VRP, diantaranya adalah *Vehicle Routing Problem with Time Windows* (VRPTW). Pada VRPTW terdapat permasalahan tambahan berupa waktu yang

tersedia dari penyedia suatu barang atau jasa dan waktu yang disediakan pelanggan. Waktu ketersediaan tersebut disebut dengan *time windows* (Mahmudy, 2014).

Penyelesaian permasalahan dalam VRPTW ini sebenarnya sangat mudah jika hanya terdapat satu depot dan sedikit node tujuan. Namun permasalahan menjadi kompleks ketika terdapat beberapa depot, node, atau rute yang harus diselesaikan. Permasalahan VRP dengan lebih dari satu rute ini biasa disebut dengan *Multi-Trip* VRP. *Multi-Trip* VRP adalah salah satu pengembangan dari permasalahan VRP klasik yang dimana sebuah kendaraan melakukan perjalanan pada beberapa rute (*trip*) dalam jangka waktu tertentu (Wassan *et al*, 2016). Permasalahan VRPTW dapat diselesaikan dengan beberapa metode optimasi, diantaranya menggunakan algoritme genetika.

Algoritme Genetika (GA) banyak digunakan untuk permasalahan optimasi. Pada penelitian yang dilakukan oleh Chunhua (2010), Pitaloka *et al* (2014), serta Nugraha dan Mahmudy (2015), penerapan GA pada permasalahan VRPTW menunjukkan hasil optimasi yang optimal. Penerapan GA juga pernah digunakan untuk mengoptimasi permasalahan lain seperti TSP dengan batasan *time windows* (Priandani & Mahmudy, 2015). Pada penelitian lain yang dilakukan oleh Xu *et al* (2008) penggunaan GA dapat meningkatkan kualitas populasi yang dapat mempercepat proses evaluasi. Dari hasil penelitian tersebut, maka dapat disimpulkan bahwa GA dapat memberi solusi permasalahan VRPTW dengan optimal. Namun GA tidak mampu mengesploitasi solusi secara mendalam, maka dibutuhkan algoritme lain yang dapat menutupi kekurangan dari GA. Salah satu algoritme yang dapat mengeksploitasi solusi secara mendalam adalah *Simulated Annealing* (Nikjoo *et al*, 2010).

Simulated Annealing (SA) juga sudah banyak digunakan untuk pengoptimasian solusi dari permasalahan VRPTW. Pada penelitian yang yang dilakukan oleh Mahmudy (2014) SA digunakan untuk mengeksplorasi solusi *neighborhood* yang digunakan untuk pencarian solusi permasalahan VRPTW. Penggunaan SA pada penelitian ini menghasilkan solusi yang baik. Pada penelitian lain menunjukkan bahwa penggabungan SA dan GA pada permasalahan VRPTW menghasilkan rata-rata waktu komputasi yang lebih cepat dan solusi yang lebih baik dari metode GA (Shi *et al*, 2011).

Penelitian ini akan membahas tentang optimasi rute wisata dengan permasalahan *Multi-trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* (GA-SA). Algoritme genetika akan digunakan untuk mengeksplorasi permasalahan global, sedangkan *Simulated annealing* akan digunakan untuk mengeksploitasi permasalahan lokal optimum.

1.2 Rumusan masalah

Berikut adalah rumusan masalah yang didefinisikan berdasarkan latar belakang di atas.

1. Bagaimana mengimplementasikan hibridisasi algoritme genetika dan *simulated annealing* untuk optimasi *Multi-Trip* VRPTW?
2. Bagaimana hasil pengujian parameter terbaik hibridisasi algoritme genetika dan *simulated annealing* untuk optimasi *Multi-Trip* VRPTW?
3. Bagaimana hasil *fitness* dari pegujian parameter terbaik hibridisasi algoritme genetika dan *simulated annealing* untuk optimasi *Multi-Trip* VRPTW?

1.3 Tujuan

1. Mengimplementasikan hibridasi algoritme genetika dan simulated annealing untuk optimasi *Multi-Trip* VRPTW
2. Menguji pengaruh parameter dengan nilai *fitness* pada penerapan algoritme genetika dan simulated annealing untuk optimasi *Multi-Trip* VRPTW
3. Menghitung hasil nilai fitness dari parameter terbaik pada penerapan algoritme genetika dan simulated annealing untuk optimasi *Multi-Trip* VRPTW

1.4 Manfaat

Berdasarkan tujuan penelitian yang telah dituliskan di atas, penelitian ini diharapkan dapat memeri manfaat untuk berbagai pihak yang terlibat. Penelitian ini diharapkan bisa memberikan solusi permasalahan wisatawan seperti mempermudah dalam menentukan rute wisata yang paling optimal yang bertujuan untuk meminimalkan biaya perjalanan serta memaksimalkan kunjungan wisata dengan waktu yang telah ditentukan.

1.5 Batasan masalah

Untuk menghindari ketidakfokusan pembahasan dari penelitian ini, maka penulis memberikan batasan-batasan pembahasan sebagai berikut.

1. Time windows wisatawan diasumsikan pada hari pertama dan ketiga dimulai pada jam 05.00-19.00 serta pada hari kedua dimulai pada jam 01.00-19.00.
2. Data yang digunakan dalam penelitian ini adalah data objek wisata berjumlah 19 objek wisata dan data hotel berjumlah 14 hotel.
3. Data waktu tempuh objek wisata dan hotel di Kabupaten Banyuwangi diperoleh dari Google Maps menggunakan objek kendaraan mobil dan diasumsikan jalanan lancer serta waktu tempuh pulang pergi diasumsikan sama.
4. Representasi gen pada kromosom diasumsikan sebagai hotel dan objek wisata yang dikunjungi dengan asumsi waktu perjalanan 3 hari dengan hanya menempati satu hotel.
5. Waktu buka objek wisata ditentukan berdasarkan waktu ideal kunjungan yang telah ditentukan pada masing-masing objek wisata.

6. Pelanggaran waktu kedatangan setelah time windows yang telah ditentukan bagi wisatawan dan objek wisata akan dikenakan poin *penalty*.
7. Waktu tempuh antar objek wisata dan hotel diasumsikan sama untuk waktu pulang pergi.
8. Durasi kunjungan pada setiap objek wisata ditetapkan selama 3 jam, kecuali pada objek wisata kawah Ijen Banyuwangi dengan indeks objek wisata 13 ditetapkan selama 5 jam.

1.6 Sistematika pembahasan

Sistematika pembahasan yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut.

BAB I Pendahuluan

Bab ini membahas tentang latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah serta sistematika pembahasan terkait dengan optimasi *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

BAB II Landasan Kepustakaan

Bab ini membahas tentang definisi penjelasan dasar teori serta penjelasan tentang referensi penelitian yang mendukung penelitian optimasi *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

BAB III Metodologi Penelitian

Bab ini akan membahas tentang metode dan langkah-langkah yang akan digunakan dalam mengoptimasi permasalahan *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

BAB IV Perancangan

Bab ini menjelaskan tentang analisis kebutuhan serta perancangan sistem optimasi *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

BAB V Implementasi

Bab ini membahas tentang pengimplementasian dari sistem untuk optimasi *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

BAB VI Pengujian dan Analisis

Bab ini membahas tentang pengujian dan analisis sistem untuk optimasi VRPTW menggunakan hibridisasi algoritme genetika dan simulated annealing pada perjalanan wisata di Kabupaten Banyuwangi.

BAB VII Penutup

Bab ini membahas tentang kesimpulan dari analisis kebutuhan hingga pengujian dari perangkat lunak sistem untuk optimasi *Multi-Trip* VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi dan berisi saran-saran pengembangan sistem untuk kedepannya.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan akan membahas tentang metode dan langkah-langkah yang akan digunakan dalam mengoptimasi permasalahan VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi. Pada bab ini akan dibahas mengenai kajian pustaka, pengertian dari pariwisata, penjelasan dari penentuan rute wisata, algoritme genetika, *simulated annealing*, serta hibridisasi dari algoritme genetika dan *simulated annealing* untuk mengoptimasi permasalahan VRPTW menggunakan hibridisasi algoritme genetika dan *simulated annealing* pada perjalanan wisata di Kabupaten Banyuwangi.

2.1 Kajian Pustaka

Peneraan Algoritme Genetika (GA) banyak digunakan untuk permasalahan optimasi. Seperti contohnya pada penelitian yang dilakukan oleh Chunhua (2010). Penerapan GA pada permasalahan VRPTW menunjukkan hasil optimasi yang optimal. Pada penelitian lain yang dilakukan oleh Xu *et al* (2008) penggunaan GA dapat meningkatkan kualitas populasi yang dapat mempercepat proses evaluasi. Dari hasil penelitian tersebut, maka dapat disimpulkan bahwa GA dapat memberi solusi permasalahan VRPTW dengan optimal. Namun GA tidak mampu mengesploitasi solusi secara mendalam, maka dibutuhkan algoritme lain yang dapat menutupi kekurangan dari GA. Salah satu algoritme yng dapat mengeksploitasi solusi secara mendalam adalah *Simulated Annealing* (Nikjoo *et al*, 2010).

Algoritme *Simulated Annealing* (SA) juga sudah banyak digunakan untuk pengoptimasian solusi dari permasalahan VRPTW. Pada penelitian yang yang dilakukan oleh Mahmudy (2014) SA digunakan untuk mengeksplorasi solusi neighborhood yang digunakan untuk pencarian solusi permasalahan VRPTW. Penggunaan SA pada penelitian ini menghasilkan solusi yang baik. Pada penelitian lain menunjukkan bahwa penggabungan SA dan GA pada permasalahan VRPTW menghasilkan rata-rata waktu komputasi yang lebih cepat yaitu selama 82.29 detik dan solusi yang lebih baik dari metode GA (Shi *et al*, 2011).

Hibridisasi algoritme genetika dan *simulated annealing* telah digunakan pada beberapa penelitian. Pada penelitian yang dilakukan oleh Xu *et al* (2015), penggunaan Algoritme genetika menghasilkan hasil yang baik jika digunakan pada permasalahan optimasi untuk pencarian global, namun akan cepat menghasilkan konfergensi dini jika digunakan pada pencarian lokal. Algoritme *Simulated Annealing* dapat menghindari pemasalahan tersebut pada pencarian lokal. Hibridisasi algoritme genetika dan *simulated annealing* akan menggabungkan kelebihan dari kedua algoritme tersebut untuk mengoptimalkan pencarian solusi dari permasalahan yang diselesaikan. Hasil penelitian menyimpulkan bahwa penggunaan hibridisasi algoritme genetika dan *simulated annealing* menghasilkan solusi yang lebih optimal dibandingkan penggunaan masing-masing algoritme genetika dan *simulated annealing*. Dari 50 kali percobaan, hibridisasi algoritme

genetika dan *simulated annealing* menghasilkan rataan sukses sebesar 100%, lebih besar daripada penggunaan algoritme genetika yang hanya menghasilkan rataan sukses sebesar 6% dan *simulated annealing sebesar* 8%.

Penerapan *Simulated Annealing* diletakkan setelah proses seleksi pada tahap algoritme genetika selesai. Satu kromosom dengan nilai fitness terbaik akan diambil dan dilakukan proses penelusuran secara lokal dengan algoritme simulated annealing. Penggunaan algoritme *simulated annealing* ini diharapkan dapat lebih mengoptimalkan pencaran solusi di ruang lokal. Penelitian ini akan membahas tentang hibridisasi algoritme genetika dan *simulated annealing* untuk mengoptimasi permasalahan Multi-Trip VRPTW yaitu penentuan waktu tempuh tercepat pada rute wisata di Kabupaten Banyuwangi.

Tabel perbandingan tinjauan pustaka dari beberapa sumber terkait dengan penelitian ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Tinjauan Pustaka

Pustaka	Judul	Data	Metode	Metode Pembanding
Anggodo *et al*, 2016	Optimization of Multi-Trip Vehicle Routing Problem with Time Windows using Genetic Algorithm	Waktu Tempuh Wisata	Algoritme Genetika	Algoritme Genetika dan Simulated Annealing
Fenghe dan Yaping, 2010	Hybrid Genetic Algorithm for Vehicle Routing Problem with Time Windows	Disribusi Barang	Algoritme Genetika, Priority Rule Heuristic, dan Bi-directional Decoding	Algoritme Genetika dan Simulated Annealing
Mahmudy, 2014	Improved Simulated Annealing for Optimization of Vehicle Routing Problem with Time Windows (VRPTW)	Solomon Benchmark	Simulated Annealing	Algoritme Genetika dan Simulated Annealing
Minocha *et al*, 2011	Solving Vehicle Routing Problems using Hybrid Genetic Algorithm	Solomon Benchmark	Algoritme Genetika dan Local Search Heuristic	Algoritme Genetika dan Simulated Annealing

Priandani dan Mahmudy, 2015	Optimasi Travelling Salesman Problem with Time Windows (TSP-TW) pada Penjadwalan Paket Rute Wisata di Pulau Bali Menggunakan Algoritme Genetika	Jarak Tempuh Wisata	Algoritme Genetika	Algoritme Genetika dan Simulated Annealing
Zhang *et al*, 2010	Optimal Reservior Operation Using Hybrid Simulated Annealing Algorithm-Genetic Algorithm	Kapasitas Waduk	Algoritme Genetika dan Simulated Annealing	Algoritme Genetika dan Simulated Annealing

2.2 Pariwisata

Pariwisata merupakan perjalanan keluar domisili asli yang dilakukan sesekali yang dilakukan sendiri maupun bersama-sama dalam lingkup ilmu, social, budaya maupun alam (Spilane, 1987 dalam Soebagyo, 2012). Keuntungan dari pariwisata diantaranya dapat mempengaruhi pertumbuhan pada sektor ekonomi. Pesatnya pertumbuhan ekonomi akan mempercepat perkembangan pembangunan dan perbaikan infrastruktur di daerah sekitar tempat wisata yang akan mendorong percepatan laju ekonomi masyarakat sekitar (Pendit, 1990 dalam Soebagyo, 2012).

2.2.1 Pariwisata Kabupaten Banyuwangi

Banyuwangi adalah salah satu kabupaten di Jawa Timur yang berpotensi dalam bidang pengembangan industri pariwisata. Dengan banyak destinasi wisata mulai dari pantai, gunung, taman nasional, dan air terjun menjadikan Banyuwangi sebagai salah satu kabupaten dengan potensi wisata yang cukup besar. Hal tersebut menjadikan pariwisata menjadi salah satu sector yang dapat dikembangkan untuk peningkatan pemasukan regional.

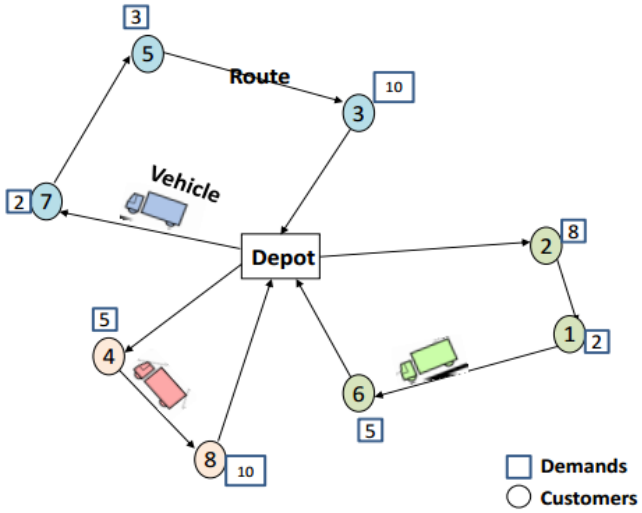
Berdasarkan data yang berasal dari Dinas Kebudayaan dan Pariwisata Kabupaten Banyuwangi, pada tahun 2014 saja total wisatawan nusantara maupun mancanegara yang berkunjung ke Kabupaten Banyuwangi mencapai 1.495.629 wisatawan, dan setahun setelahnya yaitu pada tahun 2015, total wisatawan nusantara maupun mancanegara meningkat 31% menjadi 1.972.393 wisatawan. Perbaikan tata kelola wisata, promosi, dan perbaikan infrastruktur terus digencarkan oleh pemerintah Kabupaten Banyuwangi demi meningkatkan angka kunjungan wisata di Kabupaten Banyuwangi.

2.3 Penentuan Rute

Penentuan rute menjadi proses yang penting ketika melakukan suatu pendistribusian barang. Penentuan rute yang tepat dapat mempengaruhi waktu tempuh perjalanan dan juga dapat menghindari keterlambatan pendistribusian barang sehingga dapat meminimalkan biaya dan waktu perjalanan. Beberapa faktor dapat mempengaruhi tepat atau tidaknya penjadwalan rute, sehingga perlu adanya suatu cara untuk mengoptimasi faktor-faktor tersebut agar menghasilkan solusi yang optimal dengan meminmalkan biaya dan waktu perjalanan. Sama halnya dengan penentuan rute wisata. Penjadwalan rute wisata harus dapat memperhitungkan beberapa hal seperti waktu yang disediakan wisatawan untuk berwisata, durasi kunjungan tiap wisata, waktu tempuh antar objek wisata, dan jam buka objek wisata. Penjadwalan harus mampu mengoptimalkan kunjungan wisata dengan meminimalkan waktu perjalanan tanpa melewati jam buka objek wisata. Oleh karena itu diperlukan suatu optimasi penjadwalan rute wisata agar mendapatkan solusi yang paling optimal.

2.4 *Vehicle Routing Problem*

Vehicle routing problem (VRP) merupakan salah satu permasalahan optimasi kombinatorial dalam hal sistem pendistribusian logistik di berbagai sektor terutama dalam bidang tranportasi. Pada VRP dasar terdiri dari kendaraan yang identik, berbasis pada satu depot pusat yang harus dioptimalkan untuk mendistribusikan barang kepada beberapa pelanggan dan hanya satu kendaraan yang memungkinkan untuk mengunjungi setiap pelanggan serta dimulai dan berahir di depot tunggal. Tujuan utama dari optimasi permasalahan VRP ini adalah untuk meminimalkan biaya pendistribusian (Bahri, 2012). Contoh ilustrasi VRP dasar dapat dilihat pada Gambar 2.1 dengan 8 pelanggan yang direpresentasikan sebagai node, 1 depot pusat, dan 3 kendaraan yang identic.



Gambar 2.1 VRP Dasar
(Bahri, 2012)

Berikut beberapa tujuan dari optimasi penentuan rute pendistribusian menurut (Bahri, 2012).

1. Meminimalkan total biaya pendistribusian
2. Meminimalkan jarak tempuh perndistribusian
3. Meminimalkan waktu tempuh, waktu keterlambatan, dan waktu tunggu
4. Meminimalkan jumlah penggunaan unit kendaraan
5. Meminimalkan kapasitas armada
6. Memaksimalkan kualitas layanan yang ditawarkan
7. Memaksimalkan manfaat bagi para pelanggan

Ada dua jenis pendekatan yang digunakan untuk optimasi permasalahan VRP menurut Bahri (2012).

1. Pencarian Lokal, meliputi *simulated annealing*, *deterministic annealing*, *descent method*, dan *Tabu search*
2. Pencarian populasi, meliputi algoritme evolusi seperti algoritme genetika dan algoritme *ant colony*

Terdapat beberapa pengembangan dari permasalahan VRP ini, diantaranya *Capacitated Vehicle Routing Problem* (CVRP), *Vehicle Routing Problem with Time Windows* (VRPTW), *Vehicle Routing Problem with Backhauls* (VRPB), *Vehicle Routing Problem with Pickup and Delivery* (VRPPD), *Vehicle Routing Problem with Heterogeneus Fleet* (VRPHF), *Multi-Depot Vehicle Routing Problem* (MDVRP), *Split Delivery Vehicle Routing Problem* (SDVRP), *Open Vehicle Routing Problem* (OVRP), *Dynamic Vehicle Routing Problem* (DVRP), dan *Stochastic Vehicle Routing Problem* (Stochastic-VRP) (Bahri, 2012).

2.5 *Vehicle Routing Problem with Time Windows*

Vehicle Routing Problem with Time Windows (VRPTW) adalah salah satu permasalahan VRP yang paling banyak dipelajari. Konsep dari VRPTW adalah pengembangan dari permasalahan VRP dengan tambahan kendala baru yaitu sebuah *time windows*. Dengan demikian, pelayanan setiap pelanggan harus dimulai dan diakhiri berdasarkan *time windows* yang telah ditentukan. Jika kendaraan distribusi sampai sebelum *time windows* pelanggan, maka dikenakan poin tunggu, sedangkan kendaraan distribusi yang dating setelah *time windows* pelanggan berakhir, maka dikenakan poin *penalty* (Bahri, 2012).

Tujuan dari optimasi permasalahan VRPTW adalah untuk menentukan rute dengan jarak terpendek untuk meminimalkan biaya perjalanan dan jumlah kendaraan tanpa melanggar *time windows* dan kapasitas kendaraan. Perjalanan dimulai dari depot tunggal lalu menuju ke masing-masing pelanggan yang tersebar. Kendaraan distribusi harus sampai ke tempat pelanggan diantara *time windows* pelanggan. Bobot total barang yang diangkut tidak boleh melebihi kapasitas kendaraan. Kendaraan yang telah selesai mengunjungi node pelanggan

harus kembali ke depot awal dalam *time windows* depot (titik awal dan akhir) yang telah ditetapkan (Lin *et al*., 2006).

2.6 *Multi-Trip* VRPTW

Multi-Trip Vehicle Routing Problem (MT-VRP) adalah salah satu pengembangan dari permasalahan VRP klasik yang dimana sebuah kendaraan melakukan perjalanan pada beberapa rute (*trip*) dalam jangka waktu tertentu (Wassan *et al*, 2016). MT-VRP merupakan permasalahan rute kendaraan khusus dimana setiap kendaraan dapat maelayani lebih dari satu perjalanan dimana setiap perjalanan mulai dari depot, melewati beberapa pelanggan dan berakhir di depot dan permintaan total pelanggan tidak boleh melebihi kapasitas (Yang dan Tang, 2010). *Multi-Trip* VRPTW menerapkan konsep MT-VRP namun terdapat penambahan batasan *time windows* di dalamnya.

2.7 Algoritme Genetika

Algoritme Genetika adalah salah satu algoritme pencarian dan optimasi berdasarkan mekanisme seleksi alam dan genetika yang telah dikembangkan sebagai pendekatan optimasi yang efektif untuk memecahkan permasalahan yang kompleks (Zhang *et al*, 2010). Algoritme genetika pertama kali diperkenalkan oleh professor Holland dari University of Michigan. Algoritme ini bersifat *stochastic* dan tidak tergantung pada area spesifik suatu permasalahan serta mampu memecahkan optimasi permasalahan yang kompleks (Xu *et al*, 2015).

Algoritme genetika memungkin suatu populasi terdiri dari banyak individu yang melakukan reproduksi berdasarkan aturan tertentu untuk memaksimalkan nilai *fitness* (Kale *et al*, 2014). Fungsi *fitness* membantu dalam evaluasi kromosom yang merupakan bagian dari algoritme genetika. Fungsi *fitness* merepresentasikan kualitas suatu kromosom (Alabsi dan Naoum, 2012).

Algoritme ini menggabungkan operasi seleksi, *crossover*, dan mutasi gen. Prosedur seleksi digunakan untuk menghasilkan populasi yang lebih baik sementara nilai *cr crossover* menggunakan gen *parent* (orang tua) terpilih untuk menghasilkan *offspring* (keturunan) baru yang akan membentuk generasi berikutnya. Mutasi digunakan untuk menghindari daerah minima. Pendekatan algoritme genetika dapat membantu menemukan solusi yang cukup baik untuk permasalahan matematis yang kompleks seperti pada permasalahan VRP (Minocha *et al*, 2011).

Permasalahan *Multi-Trip* VRPTW merupakan permasalahan optimasi kombinatorial dengan membentuk reprepentasi solusi yang sesuai maka akan didapatkan solusi yang optimal dalam penggunaan algoritme genetika untuk permasalahan *Multi-Trip* VRPTW. Pada penelitian ini pengkodean kromosom pada algoritme genetika menggunakan pengkodean permutasi.

Secara umum, struktur algoritme genetika dapat dideskripsikan dengan *pseudocode* pada Gambar 2.2 (Gen & Cheng, 1997 dalam Mahmudy, 2015).

```
Begin Algoritme Genetika
t = 0
inisialisasi P(t)
while (bukan kondisi berhenti) do
   reproduksi C(t) dari P(t)
   evaluasi P(t) dan C(t)
   seleksi P(t+1) dari P(t) dan C(t)
   t = t + 1
end while
end Algoritme Genetika
```

Gambar 2.2 *Pseudocode* Algoritme Genetika

2.7.1 Representasi Kromosom

Pada tahap representasi kromosom solusi permasalahan disajikan dalam serangkaian integer dengan panjang *n*, dimana *n* adalah jumlah pelanggan yang akan dilayani (Fenghe dan Yaping, 2010). Representasi kromosom dirancang untuk menghasilkan solusi yang dapat meminimalkan waktu komputasi yang dibutuhkan dalam perhitungan algoritme genetika untuk memindah suatu populasi kedalam ruang pencarian yang memungkinkan atau memperbaiki kromosom yang kurang layak (Marian, 2012 dalam Mahmudy *et al*, 2013a). Representasi kromosom yang tepat dapat menentukan kebehasilan penerapan algoritme genetika (Marian, 2006 dalam Mahmudy *et al*, 2012a).

2.7.2 Fungsi Fitness

Fungsi objektif dari suatu optimasi permasalahan harus dikonversi ke dalam fungsi *fitness* yang digunakan untuk mengukur kualitas suatu solusi (Mahmudy *et al*, 2012a, 2013a). Fungsi *fitness* membantu dalam proses evaluasi suatu kromosom. Kualitas kromosom dapat dilihat dari nilai *fitness* kromosom tersebut. Secara umum fungsi fitness mempertimbangkan 2 poin, *reward* dan *penalty*. Semakin besar nilai *reward* maka kualitas kromosom akan lebih baik, sebaliknya semakin besar nilai *penalty* maka kualitas kromosom akan semakin jelek (Alabsi dan Naoum, 2012). Fungsi fitness dapat dihitung dengan Persamaan 2.1 (Anggodo *et al*, 2016).

$$\boldsymbol{fitness} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{time}} + \boldsymbol{penalty} \qquad \textbf{(2.1)}$$

Dimana *time* adalah total waktu tempuh dari titik asal menuju titik tujuan dan *penalty* merupakan nilai pelanggaran. *Penalty* akan bernilai negatif jika terjadi pelanggaran pada *time windows* objek wisata maupun wisatawan. Inisialisasi Populasi

Inisialisasi populasi dibangun sedemikian rupa sehingga membentuk solusi yang layak (Minocha *et al*, 2011). Inisialisasi populasi dibangkitkan secara random dengan jumlah populasi sebesar *pop_size* (Mahmudy *et al*, 2012a). Sebagai solusi awal, populasi dengan jumlah individu yang diinginkan yang dihasilkan untuk memulai eksplorasi didekat solusi optimal (Azadeh *et al*, 2017).

2.7.3 Reproduksi

Pada setiap generasi baru kromosom diproduksi dengan menggunakan operator *crossover* dan mutasi. Jumlah kromosom baru ditentukan berdasarkan parameter *crossover rate* (*cr*) dan *mutation rate* (*mr*) (Mahmudy *et al*, 2012a). Jika terdapat jumlah populasi sebesar *pop_size*, maka jumlah keturunan (*offspring*) yang dihasilkan dari proses *crossover* proses berjumlah *pop_size* x *cr* dan *offspring* berjumlah *pop_size* x *mr* pada proses mutasi (Mahmudy, 2015).

2.7.3.1 *Crossover*

Crossover adalah operator algoritme genetika yang digunakan untuk memproduksi individu baru dengan menggabungkan karakteristik dari kedua *parent* yang dipilih secara acak (Azadeh *et al*, 2017). *Crossover* berperan penting dalam proses pertukatan informasi antar kromosom. Jika pada proses *crossover* dapat menghasilkan kombinasi yang baik maka dapat mempercepat proses pencarian suatu solusi yang optimal (Chunhua, 2010). *Crossover* juga disebut sebagai operator eksploitasi (Malhotra *et al*, 2011).

2.7.3.2 Mutasi

Penggunaan mutasi bertujuan untuk memperluas ruang pencarian pada algorima genetika (Azadeh *et al*, 2017). Mutasi dapat mencari solusi pada daerah-daerah baru. Berbeda dengan *crossover* yang mengesploitasi suatu solusi, mutasi mengeksplorasi solusi dengan mencari soluasi pada daerah-daerah baru secara mendalam yang tidak bisa dilakukan dengan operator *crossover* (Malhotra *et al*, 2011).

2.7.4 Seleksi

Tahap seleksi merupakan tahap yang penting dalam algoritme genetika karena akan berefek pada laju konfergensi (Wijayaningrum dan Mahmudy, 2016). Prosedur seleksi digunakan untuk memilih kromosom sebanyak *pop_size* dari populasi sebelumnya (*parent*) beserta dengan *offspring* yang telah dihasilkan dari proses *crossover* dan mutasi untuk membentuk generasi berikutnya (Mahmudy *et al*, 2012a, 2013a, 2013b).

2.8 *Simulated Annealing*

Simulated annealing (SA) terdiri dari pengoptimalan proses “pelelehan” dengan temperatur yang sesuai, kemudian menurunkan suhu dengan perlahan sampai “membeku” dan tidak ada proses selanjutnya (Kirkpatrick *et al*, 1983). Pada awal proses, logam akan dipanaskan. Kemudian, suhu diturunkan perlahan-lahan untuk menghindari keretakan agar meminimalkan energi yang digunakan (Tung *et al*, 2016). Semakin tinggi suhu dan semakin rendah nilai solusi, semakin besar kesempatan untuk menerima solusi yang kurang optimal (Chagas *et al*, 2016).

Parameter yang digunakan pada *simulated annealing* adalah sebagai berikut:

a. Temperatur awal (*temp0*): parameter yang digunakan untuk inisialisasi suhu awal
b. Temperatur Akhir (*temp1*): parameter yang digunakan untuk inisialisasi suhu akhir
c. Solusi awal (*S*): Inisialisasi solusi awal
d. Solusi tetangga (*Sn*): Solusi tetangga dari *S*
e. Faktor Pendinginan (*cooling factor*): parameter yang digunakan untuk menurunkan nilai suhu (*t*) secara bertahap

Secara umum langkah-langkah algoritme simulated annealing menurut Mahmudy (2014) adalah sebagai berikut:

Langkah 1 : Inisialisasi nilai awal *temp0*

Langkah 2 : Membangkitkan solusi awal *S* secara acak sebagai *current solution*

Langkah 3 : Cek nilai *t*, jika nilai *t* sama atau lebih dari nilai *temp1*, maka lanjut ke Langkah 4, jika tidak lanjut ke Langkah 10

Langkah 4 : Ulangi Langkah 5 sampai dengan Langkah 9 sampai dengan batas *inner_itr*

Langkah 5 : Inisialisasi nilai d_1, dimana d_1 adalah nilai fungsi objektif dari solusi *S*

Langkah 6 : Membangkitkan solusi *Sn* sebagai solusi baru

Langkah 7 : Inisialisasi nilai d_2, dimana d_2 adalah nilai fungsi objektif dari solusi *Sn*

Langkah 8 : Jika telah memperoleh seleksi terhadap solusi *Sn*, selanjutnya membandingkan nilai fitness d_1 dan d_2

Prinsip 1 : Jika nilai d_2 kurang dari d_1, terima perubahan dengan memasukkan nilai solusi baru ke solusi sekarang

Prinsip 2 : Jika nilai d_2 lebih besar dari d_1, menuju proses *probability acceptance* dengan menghitung probabilitas dan membangkitkan angka secara acak (α) antara 0 sampai 1.

$$\boldsymbol{prob} = \frac{\mathbf{1}}{\boldsymbol{e}^{\frac{(d_2 - d_1)k}{d_1 t}}} \qquad \textbf{(2.2)}$$

Dimana nilai k digunakan untuk menghitung probabilitas penerimaan solusi baru yang kurang baik. Setelah dilakukan beberapa percobaan oleh Mahmudy (2014) nilai k paling optimal ditentukan sebesar 25.

Langkah 9 : Menurunkan temperatur *t* dengan mengalikan temperatur *t* dengan variabel *cooling factor*

Langkah 10: Keluaran solusi baru mendekati optimal

2.9 Hibridisasi Algoritme Genetika dan *Simulated Annealing*

Algoritme genetika memiliki kemampuan pebarian global yang efektif dan dapat mencari dengan cepat semua solusi dalam ruang solusi. Namun, algoritme genetika belum mampu secara efektif melakukan percarian lokal sehingga menimbulkan konvergensi dini (Zhou dan Li, 2006 dalam Xu *et al*, 2015). Meskipun algoritme *simulated annealing* mampu menelusuri solusi pada ruang solusi lokal, namun kurang efektif dalam pencarian solusi di ruang solusi global (Yan dan Bao, 2004 dalam Xu *et al*, 2015). Berdasarkan kelebihan dan kekurangan dari masing-masing algoritme, penggabungan algoritme genetika dan *simulated annealing* akan melengkapi kekurangan dari masing-masing algoritme. Berikut langkah-langkah hibridisasi algoritme genetika dan *simulated annealing*:

Langkah 1 : Inisialisasi parameter algoritme genetika dan *simulated annealing*

Langkah 2 : Membangkitkan kromosom sebanyak *pop_size* secara random

Langkah 3 : Hitung *fitness* masing-masing kromosom

Langkah 4 : Lakukan tahap *crossover* dan mutasi dengan jumlah *offspring* masing masing tahap *crossover* sebanyak *cr* x *pop_size* dan mutasi sebanyak *mr* x *pop_size*

Langkah 5 : Evaluasi masing-masing *offsring* dengan menghitung *fitness* masing-masing *offspring*

Langkah 6 : Lakukan proses seleksi pada seluruh populasi sebelumnya (*parent*) beserta *offspring* yang telah dihasilkan dari Langkah 4 dan pilih kromosom sejumlah *pop_size* untuk menjadi generasi baru

Langkah 7 : Pilih satu kromosom sebagai solusi awal simulated annealing

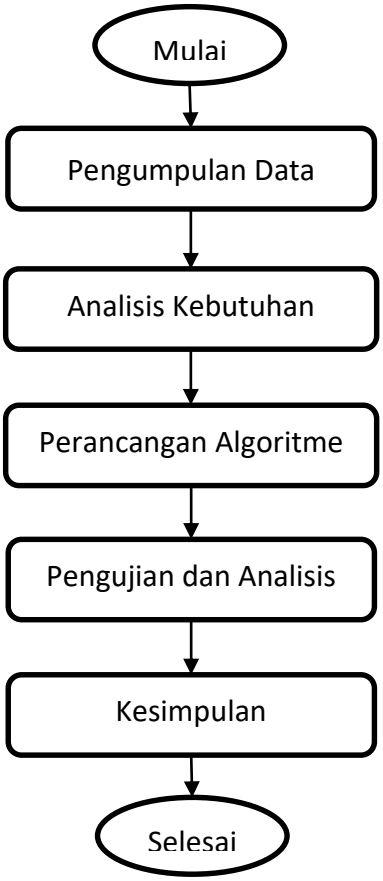
Langkah 8 : Menelusuri kromosom yang dipilih dari Langkah 7 secara mendalam dengan algoritme *simulated annealing*

Langkah 9 : Ulangi Langkah 2-8 sejumlah generasi

Langkah 10: *Output* hasil solusi mendekati optimal

BAB 3 METODOLOGI

Bab ini menjabarkan tahapan penelitian mengenai hibridisasi algoritme genetika dan *simulated annealing* untuk optimasi *Multi-Trip* VRPTW mulai dari ahap pengumpulan data, analisis kebutuhan, perancangan aloritma, pengujian dan analisis, dan kesimpulan yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Tahapan Penelitian

3.1 Pengumpulan Data

Data yang dipakai pada penelitian ini adalah 19 data objek wisata dan 14 data hotel. Data waktu tempuh dari masing-masing objek wisata dan hotel didapat dari *Google Maps* yang dijadikan tabel matriks objek wisata dengan objek wisata dan objek wisata dengan hotel. Pada kasus ini diasumsikan waktu tempuh pulang pergi antar objek wisata dan hotel sama (simetris). Data *time windows* wisatawan dan objek wisata disesuaikan dengan jam berkunjung normal. Untuk *time windows* wisatawan ditetapkan yaitu pada hari pertama dan ketiga dimulai pukul 05.00-19.00 serta pad ahari kedua dimulai pukul 01.00-19.00.

Tabel 3.1 Tabel *time windows* objek wisata

No	Nama Hotel	Jam Buka	Jam Tutup
1	Wedhi Ireng	5.00	17.00
2	Pulau Merah	5.00	19.00
3	Teluk Hijau	5.00	17.00
4	Boom	5.00	19.00
5	Bangsring	5.00	19.00
6	Grajagan	5.00	19.00
7	Watu Dodol	5.00	19.00
8	Pancer	5.00	19.00
9	Blimbing Sari	5.00	19.00
10	Pancur	5.00	19.00
11	Bedul	5.00	19.00
12	Cemara	5.00	19.00
13	Kawah Ijen	1.00	15.00
14	Jagir	8.00	17.00
15	Tirto Kemanten	8.00	17.00
16	Selendang Arum	8.00	17.00
17	Kali Bendo	8.00	17.00
18	Meru Betiri	8.00	17.00
19	Alas Purwo	8.00	17.00

Tabel 3.2 Tabel Daftar Hotel

No	Nama Hotel
1	Mirah
2	New Surya Hotel
3	Watu Dodol
4	Ketapang Indah
5	Santika
6	Mangir Asri
7	Baru
8	Manyar
9	Minak Jinggo
10	Blambangan
11	Agung Jaya Mahkota
12	Selamet
13	Mahkota Plengkung
14	Kalibaru Cottage

3.2 Analisis Kebutuhan

Pada optimasi *Multi-Trip* VRPTW dengan hibridisasi algoritme genetika dan *simulated annealing* terdapat beberapa kebutuhan yang dapat mendukung pengerjaan sistem. Diantaranya terdapat kebutuhan *software* dan *hardware*. Berikut spesifikasi kebutuhan yang digunakan untuk mendukung perancangan sistem:

1. Kebutuhan *hardware*
    - Laptop dengan spesifikasi Intel® Core™ i3-2348M CPU @2.30GHz RAM 4.00 GB
2. Kebutuhan *software*
    - Sistem Operasi Microsoft Windows 10 64-bit
    - Netbeans IDE V.8.0
    - JDK V.8.0
    - Microsoft Office Excel 2016
    - Astah Community 6.7.0

3.3 Perancangan Algoritme

Optimasi *Multi-Trip* VRPTW menggunakan gabungan algoritme genetika dan *simulated annealing*. Penggunaan *simulated annealing* diharapkan bisa menutupi kekurangan algoritme genetika dalam pencarian lokal. Pada penelitian ini tahapan algoritme *Simulated annealing* akan dilakukan setelah tahap seleksi pada algoritme genetika selesai dilakukan. Kromosom dengan nilai *fitness* terbaik dan terburuk akan akan digunakan sebagai solusi awal *S* pada algoritme *simulated annealing* untuk ditelusuri secara mendalam. Penggabungan algoritme genetika dan *simulated annealing* diharapkan dapat menelusuri solusi pada ruang lokal maupun global secara optimal.

3.4 Pengujian dan Analisis

Pada penelitian ini pengujian dilakukan untuk mencari parameter agloritma genetika dan *simulated annealing* yang paling tepat untuk menghasilkan nilai *fitness* yang paling optimal. Pengujian pada parameter algoritme genetika dilakukan terhadap ukuran populasi, banyak generasi, dan kombinasi nilai *cr* dan *mr* serta parameter pada simulated annealing yaitu nilai temperatur akhir (*temp1*) dan nilai *cooling factor*.

3.4.1 Pengujian Ukuran Populasi

Pengujian ukuran populasi (*popSize*) dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh ukuran populasi yang diujikan. Pengujian dilakukan dengan menggunakan jumlah populasi mulai dari popSize 100 sampai dengan popSize 500 dengan kelipatan 100 (Anggodo *et al*, 2016). Pada pengujian ukuran

populasi akan dilakukan percobaan sebanyak 10 kali. Tabel rancangan pengujian ukuran populasi dapat dilihat pada Tabel 3.3.

Tabel 3.3 Pengujian Ukuran Populasi

popSize	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
100											
200											
300											
400											
500											

3.4.2 Pengujian Banyak Generasi

Pengujian banyak generasi dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh banyak generasi yang diujikan. Pengujian dilakukan dengan menggunakan jumlah generasi mulai dari 200 sampai dengan 1000 dengan kelipatan 200 (Anggodo *et al*, 2016). Pada pengujian banyak generasi akan dilakukan percobaan sebanyak 10 kali. tabel rancangan pengujian banyak generasi dapat dilihat pada tabel 3.4.

Tabel 3.4 Pengujian Banyak Generasi

Banyak Generasi	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
200											
400											
600											
800											
1000											

3.4.3 Pengujian Kombinasi *cr* dan *mr*

Pengujian kombinasi *cr* dan *mr* dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh kombinasi *cr* dan *mr* yang diujikan. Pengujian dilakukan dengan menggunakan kombinasi *cr* dan *mr* pada kisaran nilai antara 0-0.4 dengan mengatur kombinasi *cr* dan *mr* sehingga menghasilkan nilai *cr* + *mr* = 0.4 untuk mendapatkan hasil yang lebih optimal (Mahmudy *et al*, 2012a). Pada pengujian kombinasi *cr* dan *mr* akan dilakukan percobaan sebanyak 10 kali. Tabel rancangan pengujian ukuran kombinasi *cr* dan *mr* dapat dilihat pada Tabel 3.5.

Tabel 3.5 Pengujian Kombinasi *cr* dan *mr*

Kombinasi		Nilai *fitness*										Rata-rata *fitness*
		Iterasi ke-										
cr	***mr***	1	2	3	4	5	6	7	8	9	10	
0.00	0.40											
0.05	0.35											
0.10	0.30											
0.15	0.25											
0.20	0.20											
0.25	0.15											
0.30	0.10											
0.35	0.05											
0.40	0.00											

3.4.4 Pengujian Ukuran Temperatur Awal

Pengujian ukuran temperatur awal (*temp0*) dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh ukuran temperatur awal yang diujikan. Pengujian dilakukan dengan menggunakan ukuran temperatur awal mulai dari 0.1 sampai dengan 0.9 dengan kelipatan 0.1. Pada pengujian ukuran temperatur akhir akan dilakukan percobaan sebanyak 10 kali. Tabel rancangan pengujian ukuran temperatur akhir dapat dilihat pada Tabel 3.6.

Tabel 3.6 Pengujian Ukuran Temperatur Awal

temp0	Nilai *fitness*										Rata-rata *fitness*
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
0.1											
0.2											
0.3											
0.4											
0.5											
0.6											
0.7											
0.8											
0.9											

3.4.5 Pengujian Nilai Faktor Pendinginan

Pengujian nilai faktor pendinginan (*cooling factor*) dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh nilai *cooling foctor* yang diujikan. Pengujian dilakukan dengan menggunakan nilai *cooling foctor* mulai dari 0.5 sampai dengan 0.9 dengan kelipatan 0.1 (Novitasari & Mahmudy, 2016). Pada

pengujian nilai *cooling foctor* akan dilakukan percobaan sebanyak 10 kali. Tabel rancangan pengujian nilai *cooling foctor* dapat dilihat pada Tabel 3.7.

Tabel 3.7 Pengujian Nilai Faktor Pendinginan

cooling factor	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
0.5											
0.6											
0.7											
0.8											
0.9											

3.4.6 Pengujian Nilai Koefisien Probabilitas

Pengujian nilai koefisien probabilitas penerimaan solusi baru (*k*) dilakukan untuk mencari rata-rata nilai *fitness* terbesar dari seluruh nilai *k* yang diujikan. Pengujian dilakukan dengan menggunakan nilai *k* mulai dari 50 sampai dengan 300 dengan kelipatan 50. Pada pengujian nilai *k* akan dilakukan percobaan sebanyak 10 kali. Tabel rancangan pengujian nilai *k* dapat dilihat pada Tabel 3.8.

Tabel 3.8 Pengujian Nilai Koefisien Probabilitas

k	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
50											
100											
150											
200											
250											
300											

3.4.7 Pengujian Perbandingan Algoritme

Pada pengujian ini akan membandingkan hasil optimasi penggunaan hibridisasi GA-SA serta penggunaan algoritme genetika dan simulated annealing secara terpisah pada permasalahan dan objek yang sama. Pengujian akan dilakukan dengan waktu komputasi yang sama yang ditetapkan berdasarkan waktu komputasi terlama dari ketiga algoritme tersebut dengan 10 kali pengujian pada masing-masing algoritme. Tabel rancangan pengujian perbandingan algoritme dapat dilihat pada Tabel 3.9.

Tabel 3.9 Pengujian Perbandingan Algoritme

Iterasi Ke-	Algoritme		
	GA	SA	GA-SA
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
Rata-rata Fitness			

3.5 Kesimpulan

Penarikan kesimpulan diambil setelah tahapan perancangan, implementasi, dan pengujian telah dilakukan. Penarikan kesimpulan akan menjawab rumusan masalah serta pengujian dan analisis yang telah dijabarkan sebelumnya. Tahapan selanjutnya setelah mendapat suatu kesimpulan penelitian, tahap selanjutnya adalah pemberian saran perbaikan dan pengembangan sistem yang dapat digunakan sebagai acuan untuk penelitian selanjutnya pada permasalahan yang sama.

BAB 4 PERANCANGAN

4.1 Formulasi Permasalahan

Pada penelitian ini data yang digunakan berupa data primer yaitu data waktu tempuh antara objek wisata dengan objek wisata lain maupun dengan hotel serta data koordinat masing-masing objek wisata dan hotel yang diperoleh dari *Google Maps*. Data yang digunakan berjumlah 19 data objek wisata dan 14 data hotel. Wisatawan diasumsikan berkunjung selama 3 hari. Tiap wisatawan memiliki *time windows* yaitu pada hari pertama dan ketiga dimulai pukul 05.00-19.00 serta pada hari kedua imulai pukul 01.00-19.00, selain itu objek wisata juga memiliki *time windows* yang ditetntukan berdasarkan waktu berkunjung normal pada objek wisata yang telah ditunjukkan pada Tabel 3.1. Selama 3 hari masa kunjungan, wisatawan diasumsikan hanya menginap di satu hotel yang sama.

Data waktu tempuh antar objek wisata pada arus pulang pergi diasumsikan sama sehingga matriks waktu tempuh akan menghasilkan tabel matriks simetris. Data waktu tempuh antar objek wisata dapat dilihat pada Tabel 4.1.

Tabel 4.1 Data Waktu Tempuh Antar Objek Wisata

No Objek Wisata	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0.11	1.10	2.18	2.55	1.19	2.44	0.20	1.55	2.00	1.24	2.10	3.30	2.26	2.10	2.18	1.32	1.07	2.09
2	0.11	0	1.02	2.07	2.48	1.11	2.32	0.12	1.46	2.15	1.16	2.02	3.21	2.17	2.04	2.09	1.24	1.01	1.38
3	1.10	1.02	0	2.50	3.32	1.48	2.03	0.50	2.26	2.29	1.53	2.44	4.02	2.58	2.43	2.46	2.01	0.20	2.15
4	2.18	2.07	2.50	0	0.42	1.40	0.31	1.57	0.35	2.07	1.46	0.24	1.40	0.33	2.03	1.23	1.04	2.42	1.53
5	2.55	2.48	3.32	0.42	0	2.24	0.11	2.31	1.08	2.42	2.19	1.00	2.11	1.05	2.36	1.56	1.39	2.16	0.43
6	1.19	1.11	1.48	1.40	2.24	0	2.09	0.58	1.19	1.13	0.33	1.23	2.55	1.51	1.54	1.51	0.52	1.44	0.59
7	2.44	2.32	2.03	0.31	0.11	2.09	0	2.26	1.01	2.36	2.15	0.52	2.02	0.56	2.29	1.51	1.34	3.15	2.23
8	0.20	0.12	0.50	1.57	2.31	0.58	2.26	0	1.37	1.40	1.04	1.54	3.13	2.09	1.54	1.58	1.12	0.47	1.26
9	1.55	1.46	2.26	0.35	1.08	1.19	1.01	1.37	0	1.45	1.22	0.28	1.49	0.45	1.40	1.00	0.43	2.20	1.31
10	2.00	2.15	2.29	2.07	2.42	1.13	2.36	1.40	1.45	0	0.58	2.04	3.23	2.19	2.33	2.19	1.05	2.25	0.14
11	1.24	1.16	1.53	1.46	2.19	0.33	2.15	1.04	1.22	0.58	0	1.41	3.00	1.56	1.58	1.55	0.44	1.50	0.48
12	2.10	2.02	2.44	0.24	1.00	1.23	0.52	1.54	0.28	2.04	1.41	0	1.45	0.40	1.55	1.19	1.02	2.44	1.51
13	3.30	3.21	4.02	1.40	2.11	2.55	2.02	3.13	1.49	3.23	3.00	1.45	0	0.46	2.52	2.16	1.59	2.40	2.48
14	2.26	2.17	2.58	0.33	1.05	1.51	0.56	2.09	0.45	2.19	1.56	0.40	0.46	0	2.14	1.35	1.18	2.59	2.07
15	2.10	2.04	2.43	2.03	2.36	1.54	2.29	1.54	1.40	2.33	1.58	1.55	2.52	2.14	0	1.49	1.36	2.36	2.21
16	2.18	2.09	2.46	1.23	1.56	1.51	1.51	1.58	1.00	2.19	1.55	1.19	2.16	1.35	1.49	0	1.16	2.43	2.05
17	1.32	1.24	2.01	1.04	1.39	0.52	1.34	1.12	0.43	1.05	0.44	1.02	1.59	1.18	1.36	1.16	0	1.58	0.52
18	1.07	1.01	0.20	2.42	2.16	1.44	3.15	0.47	2.20	2.25	1.50	2.44	2.40	2.59	2.36	2.43	1.58	0	2.11
19	2.09	1.38	2.15	1.53	0.43	0.59	2.23	1.26	1.31	0.14	0.48	1.51	2.48	2.07	2.21	2.05	0.52	2.11	0

Waktu tempuh antara masing-masing objek wisata dan masing-masing hotel dapat dilihat pada Tabel 4.2.

Tabel 4.2 Data Waktu Tempuh Antara Objek Wisata dan Hotel

No Hotel	No Objek Wisata																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2.16	2.08	2.46	0.15	0.29	1.40	0.18	1.56	0.41	2.09	1.45	0.33	1.19	0.35	2.01	1.29	1.08	2.43	1.36
2	1.08	1.00	1.37	1.09	1.44	0.43	1.31	0.48	0.47	1.23	0.28	1.05	2.02	1.20	1.08	1.11	0.35	1.34	0.48
3	2.40	2.31	3.10	0.30	0.13	2.02	0.07	2.19	0.56	2.31	2.06	0.48	1.36	0.51	2.20	1.47	1.24	3.08	1.56
4	2.21	2.13	2.57	0.19	0.29	1.46	0.16	2.06	0.45	2.17	1.52	0.37	1.23	0.40	2.07	1.33	1.12	2.53	1.41
5	1.57	1.49	2.28	0.13	0.46	1.22	0.35	1.37	0.21	1.51	1.26	0.13	1.10	0.27	1.41	1.09	0.48	2.25	1.16
6	1.37	1.29	2.08	0.37	1.27	1.00	1.04	1.19	0.15	1.28	1.07	0.34	1.30	0.48	1.33	0.59	0.26	2.05	0.54
7	2.09	2.02	2.41	0.07	0.44	1.31	0.32	1.50	0.28	2.01	1.36	0.20	1.11	0.27	1.52	1.17	0.56	2.38	1.25
8	2.28	2.20	2.58	0.18	0.27	1.49	0.14	2.07	0.44	2.19	1.55	0.36	1.24	0.39	2.10	1.35	1.13	2.55	1.43
9	1.41	1.33	2.09	1.30	2.04	1.20	1.52	1.22	1.07	2.00	1.25	1.26	2.16	1.37	0.30	1.20	1.04	2.07	1.27
10	2.11	2.03	2.03	0.09	0.43	1.32	0.31	1.51	0.28	2.02	1.37	0.22	1.11	0.27	1.52	1.17	0.57	2.39	1.26
11	1.22	1.14	1.51	1.06	1.43	1.00	1.30	1.02	0.46	1.41	1.05	1.03	2.01	1.18	0.48	1.00	0.43	1.48	1.06
12	2.11	2.03	2.39	0.06	0.58	1.34	0.34	1.52	0.29	2.01	1.40	0.19	1.12	0.28	1.55	1.17	0.59	2.37	1.27
13	2.23	2.16	2.51	0.14	0.32	1.46	0.21	2.04	0.41	2.13	1.52	0.32	1.19	0.35	2.06	1.30	1.12	2.49	1.39
14	2.00	1.53	2.28	1.49	2.22	1.48	2.12	1.42	1.28	2.19	1.46	1.45	2.35	1.56	0.20	1.39	1.25	2.26	1.45

4.2 Siklus Hibridisasi Algoritme Genetika dan *Simulated Annealing*

4.2.1 Representasi Kromosom

Pada kontek VRPTW, kromosom merupakan *encode* permutasi dari permutasi objek wisata yang dikunjungi. Representasi dalam bentuk permutasi dirasa akan memberikan hasil yang optimal karena mencakup keseluruhan tempat wisata sehingga algoritme genetika dapat melakukan penelusuran solusi secara optimal pula. Gambar 4.1 menunjukkan contoh representasi kromosom.

8	17	1	12	2	3	14	10	7	19	13	4	5	15	18	16	9	6	11

Gambar 4.1 Representasi Kromosom

Setelah mendapatkan hasil kromosom, selanjutnya adalah menghitung nilai *fitness* dari kromosom yang telah terbentuk. Perhitungan *fitness* dimulai dengan seolah-olah menyisipkan indeks hotel pada kromosom. Pada gen pertama kromosom disisipkan indeks hotel yang terpilih, misalkan indeks hotel yang terpilih adalah 4, maka indeks hotel 4 disisipkan diawal kromosom. Setelah indeks hotel disisipkan, maka selanjutnya adalah memasukkan gen objek wisata satu per satu dalam kromosom, jika total durasi kunjungan wisata telah melebihi jam 14.45 maka dilakukan penyisipan indeks hotel 4 kembali. Setelah dilakukan penyisipan indeks hotel proses pemasukan gen objek wisata kembali dilakukan dengan proses yang sama hingga batas hari yang dimasukkan. Pada penelitian ini diasumsikan perjalanan wisata dilakukan dalam 3 hari sehingga proses pemasukan gen

kromosom dilakukan dalam 3 kali perjalanan. Jika proses memasukkan gen telah selesai, maka diakhir kromosom disisipkan kembali indeks hotel 4.

Perhitungan *fitness* didapatkan berdasarkan Persamaan 2.1, dimana total waktu tempuh (menit) digunakan sebagai inisialisasi variabel *time* dan total *penalty* (menit) digunakan sebagai inisialisasi variabel *penalty*. Proses penyisipan indeks hotel hanya bertujuan untuk menghitung total waktu perjalanan yang dimulai dari hotel dan berakhir di hotel yang sama tanpa mengubah struktur kromosom asli. Proses perhitungan nilai *fitness* dapat dilihat pada Tabel 4.3.

Tabel 4.3 Perhitungan *Fitness*

	V	Node	Starting Time	Waktu Tempuh	Tiba	Earliest Time	Tunggu	Mulai	Durasi Kunjungan	Selesai	Latest Time	Penalty
Individu P1	1	4,8	5:00	2:06	7:06	5:00	0:00	7:06	3:00	10:06	19:00	0:00
		8,17	10:06	1:12	11:18	8:00	0:00	11:18	3:00	14:18	17:00	0:00
		17,1	14:18	1:32	15:50	5:00	0:00	15:50	3:00	18:50	17:00	1:50
		1,4	18:50	2:21	21:11	5:00	0:00	21:11	0:00	21:11	19:00	2:11
	2	4,12	1:00	0:37	1:37	5:00	3:23	5:00	3:00	8:00	19:00	0:00
		12,2	8:00	2:02	10:02	5:00	0:00	10:02	3:00	13:02	19:00	0:00
		2,3	13:02	1:02	14:04	5:00	0:00	14:04	3:00	17:04	17:00	0:04
		3,4	17:04	2:57	20:01	5:00	0:00	20:01	0:00	20:01	19:00	1:01
	3	4,14	5:00	0:40	5:40	8:00	2:20	8:00	3:00	11:00	19:00	0:00
		14,10	11:00	2:19	13:19	5:00	0:00	13:19	3:00	16:19	19:00	0:00
		10,4	16:19	2:17	18:36	5:00	0:00	18:36	0:00	18:36	19:00	0:00
Total				19:05	5:06							5:06
fitness												**-4.2101**

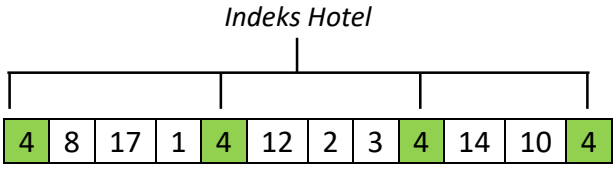
Perhitungan nilai *fitness* dipengaruhi oleh parameter waktu tempuh (*time*), *penalty*, dan total objek wisata yang dikunjungi. Pada parameter *time* semakin besar nilai *time* maka nilai *fitness* akan semakin kecil sehingga digunakan persamaan 1 dibagi dengan 1+*time*, penambahan nilai 1 pada parameter *time* digunakan untuk menghindari nilai tak terhingga jika terdapat kejiadian nilai *time* sama dengan 0. Pada parameter *penalty* semakin besar nilai *penalty* maka akan memperkecil nilai *fitness* sehingga digunakan persamaan *penalty* dikalikan dengan nilai -1 agar nilai *penalty* menjadi minus, jika tidak terdapat *penalty* maka nilai *penalty* sama dengan 0. Pada parameter jumlah objek wisata, semakin sedikit jumlah objek wisata yang dikunjungi maka akan mengurangi nilai *fitness* sehingga nilai jumlah objek wisata dibagi dengan jumlah objek wisata ideal yang dapat dikunjungi dalam waktu 3 hari yaitu sebanyak 10 objek wisata (Anggodo *et al*, 2016). Pada penelitian ini Persamaan 4.1 menunjukkan fungsi untuk menghitung nilai *fitness*.

$$\boldsymbol{fitness} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{time}} + (\boldsymbol{penalty}\ \boldsymbol{x} - \mathbf{1}) + \frac{\boldsymbol{jumlah\ wisata}}{\mathbf{10}} \qquad \textbf{(4.1)}$$

$$fitness = \frac{1}{1+19.05} + (5.06\ x - 1) + \frac{8}{10}$$

$$fitness = -4.2101$$

Hasil penambahan gen dalam kromosom dapat dilihat pada Gambar 4.2. Penggambaran indeks hotel hanya sebagai ilustrasi penyisipan saja dan bukan bagian dari kromosom. Gambar 4.2 juga dapat menggambarkan jalur yang akan dituju oleh wisatawan.



Gambar 4.2 Ilustrasi Proses Perhitungan Fitness

4.2.2 Inisialisasi Populasi Awal

Inisialisasi populasi awal jumlah populasi (*popSize*) diasumsikan sebesar 3 individu. Kromosom dibangkitkan secara acak untuk tiap gen dengan nilai permutasi indeks objek wisata dengan panjang masing-masing kromosom sebesar 19 gen. Inisialisasi populasi awal dapat dilihat pada Tabel 4.4.

Tabel 4.4 Inisialisasi Populasi Awal

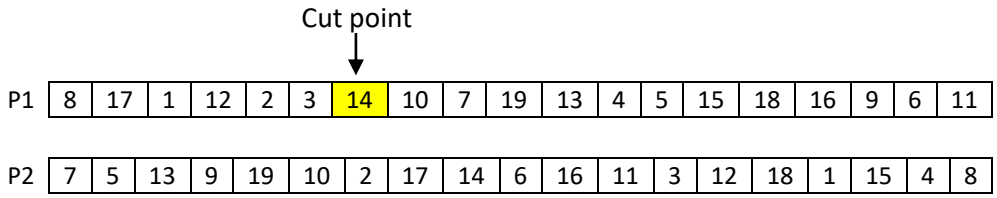
Individu	Kromosom																		
P1	8	17	1	12	2	3	14	10	7	19	13	4	5	15	18	16	9	6	11
P2	7	5	13	9	19	10	2	17	14	6	16	11	3	12	18	1	15	4	8
P3	18	10	13	9	7	1	15	4	3	12	11	8	19	17	6	5	14	16	2

4.2.3 Reproduksi

Pada proses reproduksi digunakan dua operator untuk menghasilkan keturunan baru (*offspring*) hasil reproduksi, yaitu *crossover* dan mutasi.

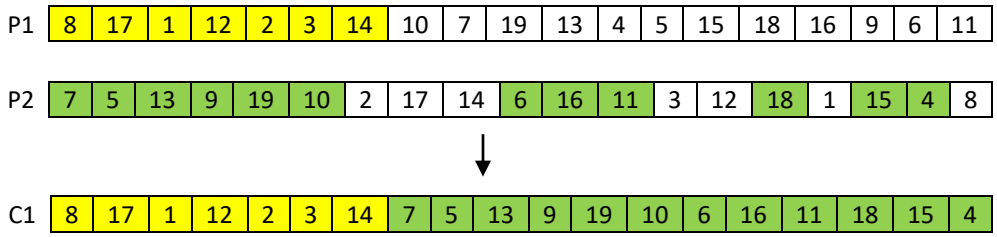
4.2.3.1 *Crossover*

Pada penelitian ini proses *crossover* menggunakan metode *one-cut point crossover* (Mahmudy, 2015). Jumlah *offspring* hasil *crossover* diperoleh dari *crossover rate* x *popsize*. Pada penelitian ini diasumsikan nilai *cr* = 0,3 dan *popSize* = 3 sehingga jumlah *offspring* yang dihasilkan sebesar 0,3 x 3 = 0,9 = 1 *offspring*. Pada proses *crossover* langkah pertama adalah memilih 2 individu secara acak yang akan digunakan sebagai *parent*. Pada metode *one-cut point crossover*, nilai *cut point* akan dibangkitkan secara acak dengan rentang nilai 0 hingga panjang gen – 1. Pada penelitian ini nilai *cut point* yang digunakan adalah 6 dengan *parent* yang terpilih adalah P1 dan P2. Proses *crossover* dapat dilihat pada Gambar 4.3.



Gambar 4.3 Proses *Crossover*

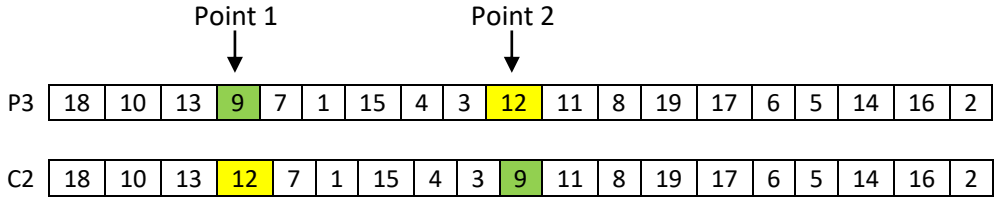
Kromosom individu baru pada gen indeks 0-6 diperoleh dari individu P1 pada indeks gen 0-6, sedangkan gen 7-18 pada kromosom baru diperoleh dari kromosom P2 dengan dilakukan pencarian mulai dari gen pertama, jika kode gen tersebut belum tercantum pada gen kromosom baru, maka gen tersebut dimasukkan pada gen kromosom baru. Hasil proses crossover dapat dilihat pada Gambar 4.4.



Gambar 4.4 *Offspring* Hasil *Crossover*

4.2.3.2 Mutasi

Pada penelitian ini proses mutasi menggunakan metode *Reciprocal Exchange Mutation* (Mahmudy, 2015). Jumlah *offspring* hasil mutasi diperoleh dari *mutation rate* x *popsize*. Pada penelitian ini diasumsikan nilai *mr* = 0,2 dan *popSize* = 3 sehingga jumlah *offspring* yang dihasilkan sebesar 0,2 x 3 = 0,6 = 1 *offspring*. Pada proses mutasi langkah pertama adalah memilih 1 individu secara acak. Pada metode *Reciprocal Exchange Mutation*, akan dipilih 2 indeks secara acak dengan rentang nilai 0 – panjang kromosom-1 dan menukar kedua nilai dari indeks gen terpilih. Misalkan idividu yang tepilih adalah P3 dan nilai kedua indeks masing-masing adalah 3 dan 9 sehingga memperoleh individu baru seperti pada Gambar 4.5.



Gambar 4.5 Hasil Proses Mutasi

4.2.4 Evaluasi

Pada proses optimasi, tahap evaluasi terus berjalan sepanjang proses untuk mengetahui kualitas suatu individu yang dapat menentukan individu yang dapat bertahan dengan menghitung nilai fitness tiap individu. Nilai fitness dapat dihitung dengan Persamaan 4.1.

Setelah melalui proses reproduksi, maka populasi baru yang terbentuk dapat dilihat pada Tabel 4.5.

Tabel 4.5 Populasi Baru

Individu	Kromosom																		
P1	8	17	1	12	2	3	14	10	7	19	13	4	5	15	18	16	9	6	11
P2	7	5	13	9	19	10	2	17	14	6	16	11	3	12	18	1	15	4	8
P3	18	10	13	9	7	1	15	4	3	12	11	8	19	17	6	5	14	16	2
C1	8	17	1	12	2	3	14	7	5	13	9	19	10	6	16	11	18	15	4
C2	18	10	13	12	7	1	15	4	3	9	11	8	19	17	6	5	14	16	2

Langkah selanjutnya yaitu menghitung nilai *fitness* masing-masing individu. Nilai *fitness* diperoleh dengan melakukan perhitungan berdsarkan Persamaan 4.1. Tahapan perhitungan nilai *fitness* dapat dilihat pada Tabel 4.5. Perhitungan *fitness* dilakukan pada seluruh kromosom individu dalam populasi. Hasil perhitungan *fitness* seluruh kromosom dapat dilihat dalam Tabel 4.6.

Tabel 4.6 Hasil Perhitungan *fitness*

Individu	*fitness*
P1	-4.2101
P2	-3.5235
P3	0.8515
C1	-4.2013
C2	0.8525

4.2.5 Seleksi

Pada tahapan seleksi, metode yang akan digunakan adalah *Replacement Selection*. Prinsip dari operator *replacement selection* adalah jika *offspring* hasil reproduksi memiliki nilai *fitness* lebih besar daripada *parent* maka *offspring* tersebut akan menggantikan *parent Offspring* yang didapat dari proses mutasi akan menggantikan *parent* jika *offspring* tersebut memiliki *fitness* yang lebih besar dari pada *parent*, jika pada proses *crossover* yang memiliki 2 *parent*, maka dipilih 1 *parent* dengan nilai *fitness* terendah (Mahmudy et al, 2012a, 2013a, 2013b).

Seleksi dilakukan pada *parent* dan *child* yang diperoleh dari tahap *crossover* dan mutasi. Pada seleksi yang dilakukan pada proses *crossover*, *fitness* pada *child* (C1) dibandingkan dengan *fitness* kedua *parent* (P1 dan P2), jika *fitness* C1 lebih besar daripada *fitness* P1 dan atau P2, maka kromosom C1 akan menggantikan kromosom kedua *parent* dengan nilai *fitness* terkecil, jika *fitness* kromosom C1 lebih kecil daripada kedua *parent*, maka kromosom C1 akan dieliminasi. Seleksi pada proses mutasi dilakukan dengan membandingan *fitness* kromosom *child* (C2) dan kromosom *parent* (P3), jika nilai *fitness* C2 lebih besar dari P3 maka kromosom C2 menggantikan kromosom P3, sebaliknya jika nilai *fitness* C2 lebih kecil dari P3 maka kromosom C2 akan dieliminasi. Setelah semua kromosom diseleksi maka akan dihasilkan populasi baru. Populasi baru hasil seleksi dapat dilihat dalam Tabel 4.7.

Tabel 4.7 Populasi Baru

Individu	Kromosom																			*fitness*
C1	8	17	1	12	2	3	14	7	5	13	9	19	10	6	16	11	18	15	4	-4.2013
P2	7	5	13	9	19	10	2	17	14	6	16	11	3	12	18	1	15	4	8	-3.5235
C2	18	10	13	12	7	1	15	4	3	9	11	8	19	17	6	5	14	16	2	0.8525

Pada penelitian kali ini, proses seleksi dilakukan setelah proses masing-masing *crossover* dan mutasi selesai sehingga *fitness offspring* dari *crossover* dan mutasi akan langsung dibandingkan dengan *parent* masing-masing.

4.2.6 Algoritme *Simulated Annealing*

Proses algoritme *simulated annealing* dijalankan setelah proses seleksi pada algoritme genetika selesai dilakukan. Kromosom yang telah melalui seleksi pada algoritme genetika dengan nilai *fitness* tertinggi dan terendah digunakan sebagai inisialisasi solusi awal pada algoritme *simulated annealing*. Pada penelitian ini juga tidak digunakan variabel *inner_itr* sehingga pengguaan variabel *inner_itr* dihilangkan dan digantikan dengan prinsip-prinsip tertentu yang digunakan untuk pengambilan keputusan pada perhitungan probabilitas penerimaan solusi baru yang akan dijelaskan pada tahapan simulated annealing berikut:

Langkah 1:

Inisialisasi parameter *temp0* sebesar 0.9, *temp1* sebesar 0.01, dan *cooling_factor* sebesar 0.1.

Langkah 2:

Inisialisasi solusi awal dengan mengambil individu hasil seleksi algoritme genetika dengan *fitness* terbaik. Pada kasus ini individu C2 dipilih sebagai solusi awal untuk fitness terbaik.

Langkah 3:

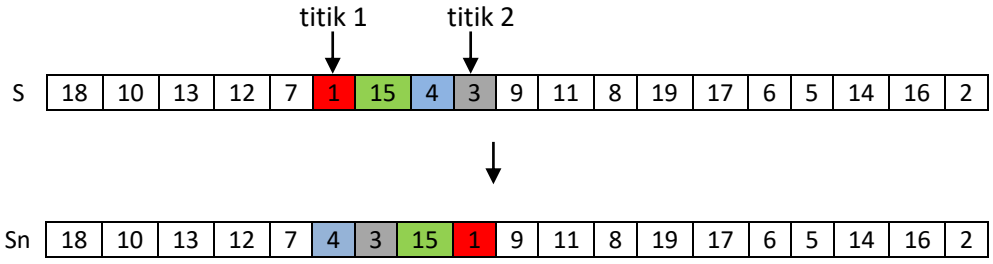
Cek nilai *t*, jika nilai *t* sama atau lebih dari nilai *temp1*, maka lanjut ke Langkah 4, jika tidak lanjut ke Langkah 8. Pada iterasi pertama nilai *t*=1, maka lanjut ke Langkah 4

Langkah 4:

Inisialisasi nilai d_1 = fitness C2 = 0.8525

Langkah 5:

Membangkitkan solusi *Sn* dengan menggunakan operator *neighborhood Scramble Mutation* (de Oliveira *et al*, 2007). Pada operator ini akan dipilih 2 titik secara acak dan menukar posisi kromosom diantara dua titik secara acak.



Gambar 4.6 Pembangkitan Solusi *Neighborhood*

Langkah 6:

Selanjutnya yaitu menghitung nilai *fitness* solusi *Sn* menggunakan Persamaan 4.1 dan dimasukkan kedalam variabel d_2.

Tabel 4.8 Perhitungan *Fitness* Solusi *Sn*

Solusi *Sn*	V	Node	Starting Time	Waktu Tempuh	Tiba	Earliest Time	Tunggu	Mulai	Durasi Kunjungan	Selesai	Latest Time	Penalty
	1	4,18	5:00	2:53	7:53	8:00	0:07	8:00	3:00	11:00	17:00	0:00
		18,10	11:00	2:25	13:25	5:00	0:00	13:25	3:00	16:25	19:00	0:00
		10,4	16:25	1:23	17:48	5:00	0:00	17:48	0:00	17:48	19:00	0:00
	2	4,13	1:00	1:52	2:52	1:00	0:00	2:52	5:00	7:52	19:00	0:00
		13,12	7:52	1:45	9:37	5:00	0:00	9:37	3:00	12:37	19:00	0:00
		12,7	12:37	0:52	13:29	5:00	0:00	13:29	3:00	16:29	19:00	0:00
		7,4	16:29	0:16	16:45	5:00	0:00	16:45	0:00	16:45	19:00	0:00
	3	4,4	5:00	0:19	5:19	5:00	0:00	5:19	3:00	8:19	19:00	0:00
		4,3	8:19	2:50	11:09	5:00	0:00	11:09	3:00	14:09	17:00	0:00
		3,15	14:09	2:43	16:52	5:00	0:00	16:52	3:00	19:52	19:00	0:52
		15,4	19:52	2:07	21:59	5:00	0:00	21:59	0:00	21:59	19:00	2:59
Total (Menit)				16:32								3.51
fitness												**-2.6523**

Langkah 7:

Selanjutnya yaitu membandingkan selisih nilai *fitness* solusi *Sn* dan *S*.

Fitness *S* : 0.8525

Fitness *Sn* : -2.6523

Karena nilai *fitness* solusi *S* lebih besar daripada nilai *fitness* solusi *Sn*, maka masuk ke dalam perhitungan probabilitas penerimaan solusi baru untuk nilai fitness yang lebih rendah. Pertama membangkitkan nilai random α dari 0 – 1. Diasumsikan jika nilai α yang terpilih adalah 0,3. Selanjutnya adalah menghitung nilai probabilitasi penerimaan solusi baru, dikarenakan pada penelitian ini tidak menggunakan variabel energy atau *cost* melainkan fitness, maka perhitungan nilai probabilitas penerimaan disesuaikan dengan variabel yang dipakai sehingga menghasilkan fungsi yang dapat dilihat pada Persamaan 4.2.

$$prob = \frac{1}{e^{\frac{(d_1 - d_2)k}{t}}} \quad \textbf{(4.2)}$$

$$prob = \frac{1}{e^{\frac{(0.8525 - (-2.6523))25}{0{,}9}}}$$

$$prob = 0$$

Sebagai pengganti penggunaan variabel *inner_itr*, maka digunakan prinsip pengambilan keputusan solusi baru yang sebelumnya digunakan dalam penelitian Tung et al (2016) sebagai berikut:

Prinsip 1 : Jika *α* lebih kecil dari *prob* maka solusi *Sn* diterima sebagai solusi baru

Prinsip 2 : Jika *α* lebih besar dari *prob* maka solusi *Sn* ditolak sebagai solusi baru

Pada tahap ini nilai *α* lebih besar daripada nilai probabilitas penerimaan solusi baru sehingga solusi *Sn* ditolak sebagai solusi baru.

Langkah 8:

Menurunkan temperatur *t* dengan mengalikan temperatur *t* dengan variabel *cooling factor* sehingga menghasilkan *t* baru denngan perhitungan sebagai berikut:

t = 0.9 x 0.1

t = 0.09

Langkah 9:

Keluaran solusi mendekati optimal pada iterasi kedua dengan *t*=0.09 dapat dilihat pada Tabel 4.9.

Tabel 4.9 Individu Baru Optimasi SA *Fitness* Terbaik Perulangan Terakhir

Individu	Kromosom																		fitness
S	18	10	13	12	7	1	15	4	3	9	11	8	19	17	6	5	14	16	2	0.8525
Sn	18	10	13	12	15	7	1	4	3	9	11	8	19	17	6	5	14	16	2	-1.2046

Fitness *S* : 0.8525

Fitness *Sn* : -1.2046

Karena nilai *fitness* solusi *Sn* lebih kecil sama dengan daripada nilai *fitness* solusi *S*, maka masuk ke dalam perhitungan probabilitas penerimaan solusi baru untuk nilai fitness yang lebih rendah. Pertama membangkitkan nilai random *α* dari 0 – 1. Diasumsikan jika nilai *α* yang terpilih adalah 0,1. Selanjutnya adalah menghitung nilai probabilitasi penerimaan solusi baru menggunakan Persamaan 4.2.

$$prob = \frac{1}{e^{\frac{(0.8525 - (-1.2046))25}{0{,}09}}}$$

$$prob = 4.16568\text{E} - 24$$

Pada tahap ini nilai α lebih besar daripada nilai probabilitas penerimaan solusi baru sehingga solusi *Sn* ditolak sebagai solusi baru.

Setelah proses SA dengan nilai solusi awal berupa *fitness* terbaik terlah selesai, maka proses SA dengan nilai solusi awal berupa *fitness* terburuk akan dihitung dengan tahapan sesuai Langkah 1-9 algoritme *simulated annealing*.

Kromosom yang terpilih sebagai solusi awal algoritme SA dengan *fitness* terburuk adal kromosom C1 dengan nilai *fitness* sebesar -4.2013. Berikut hasil perhitungan SA dengan nilai solusi awal berupa *fitness* terburuk dengan nilai *t* sebesar 0.9 dapat dilihat pada Tabel 4.10.

Tabel 4.10 Hasil Perhitungan SA *Fitness* Terburuk Perulangan Pertama

Individu	Kromosom																		*fitness*
C1/S	8	17	1	12	2	3	14	7	5	13	9	19	10	6	16	11	18	15	4	-4.2013
Sn	8	17	1	12	2	3	14	7	5	13	9	19	10	16	11	18	6	15	4	-4.2013

Dari hasil perhitungan SA pada Tabel 4.12, dapat dilihat bahwa nilai *fitness S* dan *Sn* sama, maka dilakukan proses perhitungan probabilitas penerimaan solusi baru yang menghasilkan nilai probabilitas sebesar 0.9 dengan nilai random α sebesar 0.5 sehingga solusi *Sn* ditolak sebagai solusi baru.

Berikut hasil perhitungan SA dengan nilai solusi awal berupa *fitness* terburuk dengan nilai *t* sebesar 0.09 dapat dilihat pada Tabel 4.11.

Tabel 4.11 Hasil Perhitungan SA *Fitness* Terburuk Perulangan Terakhir

Individu	Kromosom																		*fitness*
S	8	17	1	12	2	3	14	7	5	13	9	19	10	6	16	11	18	15	4	-4.2013
Sn	17	1	8	12	2	3	14	7	5	13	9	19	10	6	16	11	18	15	4	-2.1635

Tabel 4.13 menunjukkan bahwa solusi *Sn* memiliki nilai *fitness* lebih besar dari pada nilai *fitness* solusi *S*, maka solusi *Sn* diterima sebagai solusi baru dari proses SA ini.

Solusi mendekati optimal dari keluaran proses algoritme *simulated anneling* menghasilkan populasi baru yang dapat dilihat pada Tabel 4.12.

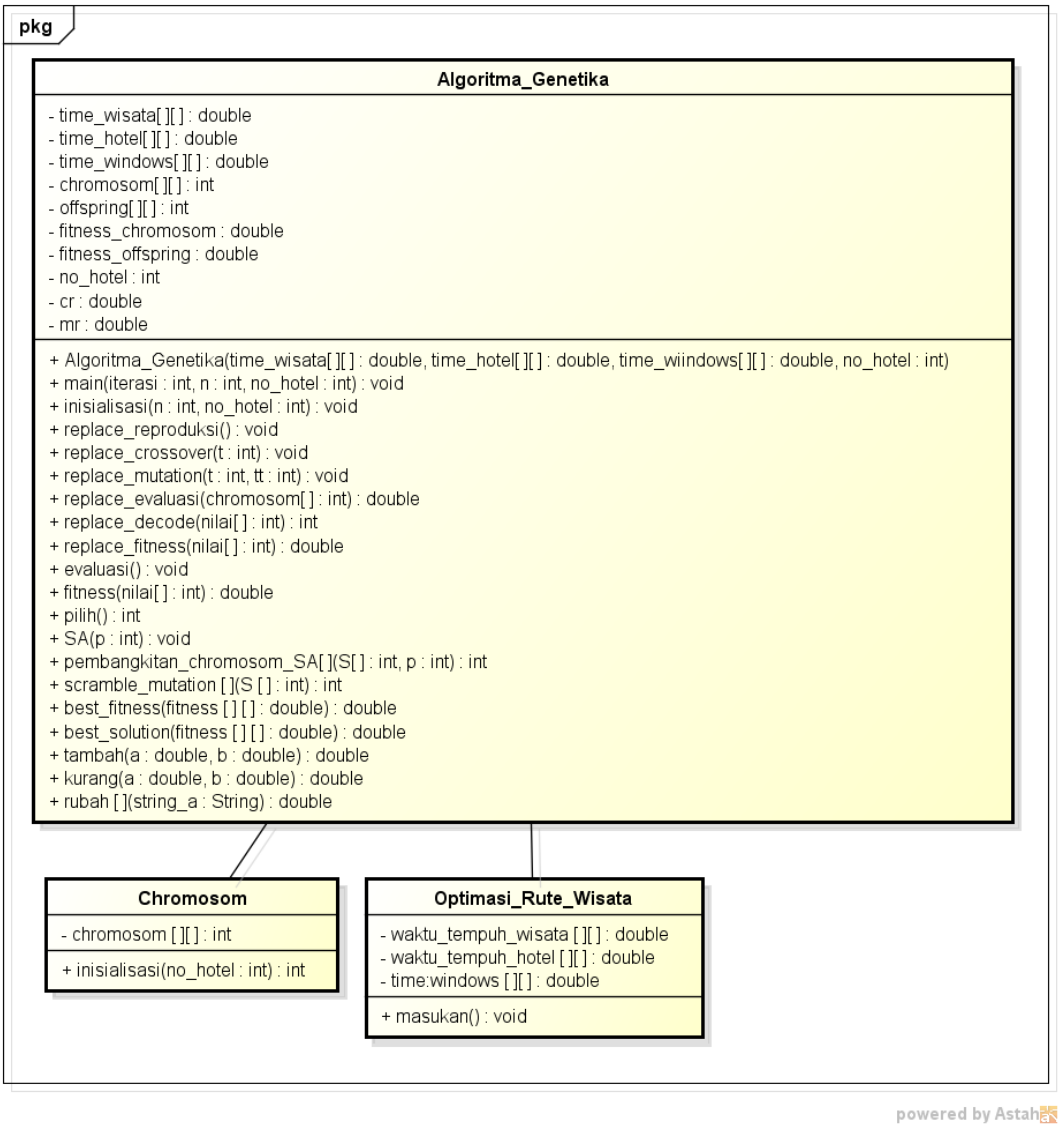
Tabel 4.12 Populasi Baru Optimasi GA-SA

Individu		*fitness*
Sn	P1	-2.1635
P2	P2	-3.5235
C2	P3	0.8525

BAB 5 IMPLEMENTASI

5.1 Struktur *Class Diagram*

Pada sub-bab ini ditampilkan struktur *class diagram* dari implementasi program optimasi *Multi-trip* VRPTW. Gambar 5.1 menampilkan *class diagram* yang terdiri dari 3 kelas, yaitu kelas Algoritme_Genetika sebagai kelas perhitungan algoritme genetika dan algoritme *simulated annealing*, kelas Chromosom yang digunakan sebagai inisialisasi kromosom, dan kelas Optimasi_Rute Wisata sebagai kelas untuk masukan data-data eksternal yang dibutuhkan untuk perhitungan optimasi yaitu data waktu tempuh antar objek wisata, waktu tempuh objek wisata dan hotel, serta *time windows* wisatawan.



Gambar 5.1 *Class Diagram*

5.2 Implementasi Kode Program

Optimasi *Multi-trip* VRPTW menggunakan algoritme genetika dan algoritme *Simulated Annealing* yang diimplementasikan menggunakan bahasa pemrograman Java dengan tahapan penyelesaian masalah sesuai tahapan pada sub-bab 4.1 sampai dengan 4.2.

5.2.1 Implementasi Alogoritma Genetika

Pada sub-bab ini menampilkan implementasi kode program dari algoritme genetika.

5.2.1.1 Implementasi Inisialisasi Populasi Awal

Pada tahap pertaman algoritme genetika akan menginisialisasi populasi awal berjumlah *n* kromosom dengan panjang masing-masing kromosom berjumlah 19 gen. Kode program implementasi inisialisasi populasi awal dapat dilihat pada Gambar 5.2.

```
1  public int[][] inisialisasi(int no_hotel) {
2      for (int i = 0; i < chromosom.length; i++) {
3          int[] hasil = new int[20];
4          int ii = 0, x = 0;
5          // membangkitkan segmen pertama secara random
6          while (ii < 19) {
7              // batasan nilai [1 19]
8              double random = Math.random() * (20 - 0) + 0;
9              Double r = new Double(random);
10             int tanda = 0;
11             for (int j = 0; j < hasil.length; j++) {
12                 if (r.intValue() == hasil[j]) {
13                     tanda++;
14                 }
15             }
16             if (tanda == 0) {
17                 hasil[ii++] = r.intValue();
18             }
19             tanda = 0;
20         }
21         // membangkitkan segmen kedua secara random
22         // inisialisasi nilai no. hotel
23         hasil[hasil.length - 1] = no_hotel;
24         // mengurangi dengan 1
25         for (int j = 0; j < hasil.length; j++) {
26             if (hasil[j] == 0) {
27
28             } else {
29                 hasil[j] -= 1;
30             }
31         }
32         chromosom[i] = hasil;
33     }
34     return chromosom;
35 }
```

Gambar 5.2 Implementasi Inisialisasi Populasi Awal

Berikut adalah penjelasan source code pada Gambar 5.2:

Baris 1 : Inisialisasi *method* Inisiaslisasi

Baris 2 : Perulangan selama panjang kromosom

Baris 3-4 : Inisialisasi parameter yang digunakan pada proses inisiasasi

Baris 6-20 : Perulangan untuk memasukkan indeks objek wisata secara acak dengan *range* 1-19 selama panjang kromosom

Baris 23-31 : Membangkitkan segmen kedua pada array kromosom untuk meletakkan indeks hotel

5.2.1.2 Implementasi Fungsi *Fitness*

Setelah menginisialisasi populasi awal, selanjutnya adalah menghitung *fitness* masing-masing kromosom. Perhitungan kromosom sesuai dengan tahapan perhitungan pada sub-bab 4.2.1. Kode program implementasi fungsi *fitness* dapat dilihat pada Gambar 5.3.

```
 1  public double fitness(int[] gen) {
 2      double hasil = 0;
 3      double time = 0; // variabel lama tempuh
 4      double penalty = 0;
 5      // menghitung nilai time tiap chromosom
 6      // menghitung waktu tempuh
 7      int tt = 0;
 8      // looping wisata
 9      double timer = 5;
10      int indek_wisata = 0;
11      // looping keseluruhan chromosome hari pertama
12      // untuk hari pertama hotel ke wisata ke-0
13      String S_time = String.valueOf(time);
14      String S_time_hotel =
    String.valueOf(time_hotel[no_hotel][gen[0]]);
15      time = tambah(S_time, S_time_hotel);
16      String S_timer = String.valueOf(timer);
17      timer = tambah(S_timer, S_time_hotel);
18      if (gen[0] == 12) {
19          S_timer = String.valueOf(timer);
20          timer = tambah(S_timer, "5.00");
21      } else {
22          S_timer = String.valueOf(timer);
23          timer = tambah(S_timer, "3.00");
24      }
25      indek_wisata++;
26      for (int i = 0; i < gen.length; i++) {
27          // untuk waktu selanjutkanya
28          S_time = String.valueOf(time);
29          S_timer = String.valueOf(timer);
30          String S_time_wisata =
    String.valueOf(time_wisata[gen[i]][gen[i + 1]]);
31          time = tambah(S_time, S_time_wisata);
32          timer = tambah(S_timer, S_time_wisata);
33          // lama berwisata 3 jam
34          boolean kondisi = false;
35          if (timer > time_windows[gen[i]][1]) {
36              kondisi = true;
```

```
37              S_timer = String.valueOf(timer);
38              String S_time_windows =
String.valueOf(time_windows[gen[i]][1]);
39              String S_penalty = String.valueOf(penalty);
40              penalty = tambah(String.valueOf(kurang(S_timer,
S_time_windows)), S_penalty);
41              if (gen[i + 1] == 12) {
42                  S_timer = String.valueOf(timer);
43                  timer = tambah(S_timer, "5.00");
44              } else {
45                  S_timer = String.valueOf(timer);
46                  timer = tambah(S_timer, "3.00");
47              }
48          } else if (timer < time_windows[gen[i]][0]) {
49              timer = (time_windows[gen[i]][0]);
50              if (gen[i + 1] == 12) {
51                  S_timer = String.valueOf(timer);
52                  timer = tambah(S_timer, "5.00");
53              } else {
54                  S_timer = String.valueOf(timer);
55                  timer = tambah(S_timer, "3.00");
56              }
57          } else {
58              if (gen[i + 1] == 12) {
59                  S_timer = String.valueOf(timer);
60                  timer = tambah(S_timer, "5.00");
61              } else {
62                  S_timer = String.valueOf(timer);
63                  timer = tambah(S_timer, "3.00");
64              }
65          }
66          indek_wisata++;
67          if (kondisi == false && timer >
time_windows[gen[i]][1]) {
68              S_timer = String.valueOf(timer);
69              String S_time_windows =
String.valueOf(time_windows[gen[i]][1]);
70              String S_penalty = String.valueOf(penalty);
71              penalty = tambah(String.valueOf(kurang(S_timer,
S_time_windows)), S_penalty);
72          }
73          // lihat waktu
74          if (timer > 14.45) {
75              break;
76          }
77      }
78      // kembali ke hotel
79      S_time = String.valueOf(time);
80      S_timer = String.valueOf(timer);
81      S_time_hotel =
String.valueOf(time_hotel[no_hotel][gen[indek_wisata -
1]]);
82      time = tambah(S_time, S_time_hotel);
83      timer = tambah(S_timer, S_time_hotel);
84      // penalty
85      if (timer > 19) {
86          S_timer = String.valueOf(timer);
87          String S_penalty = String.valueOf(penalty);
88
```

```
        penalty = tambah(String.valueOf(kurang(S_timer,
89  "19.00")), S_penalty);
90      }
91      // untuk hari kedua hotel ke wisata ke-1
92      timer = 1;
93      S_time = String.valueOf(time);
94      S_timer = String.valueOf(timer);
        S_time_hotel =
95  String.valueOf(time_hotel[no_hotel][gen[indek_wisata]]);
96      time = tambah(S_time, S_time_hotel);
97      timer = tambah(S_timer, S_time_hotel);
98      if (timer < time_windows[gen[indek_wisata]][0]) {
99          timer = (time_windows[gen[indek_wisata]][0]);
100         if (gen[indek_wisata] == 12) {
101             S_timer = String.valueOf(timer);
102             timer = tambah(S_timer, "5.00");
103         } else {
104             S_timer = String.valueOf(timer);
105             timer = tambah(S_timer, "3.00");
106         }
107     } else {
108         if (gen[indek_wisata] == 12) {
109             S_timer = String.valueOf(timer);
110             timer = tambah(S_timer, "5.00");
111         } else {
112             S_timer = String.valueOf(timer);
113             timer = tambah(S_timer, "3.00");
114         }
115     }
116     indek_wisata++;
117     for (int i = indek_wisata; i < gen.length; i++) {
118         // untuk waktu selanjutkanya
119         S_time = String.valueOf(time);
120         S_timer = String.valueOf(timer);
            String S_time_wisata =
121 String.valueOf(time_wisata[gen[i - 1]][gen[i]]);
122         time = tambah(S_time, S_time_wisata);
123         timer = tambah(S_timer, S_time_wisata);
124         // lama berwisata 3 jam
125         boolean kondisi = false;
126         if (timer > time_windows[gen[i]][1]) {
127             kondisi = true;
128             S_timer = String.valueOf(timer);
                String S_time_windows =
129 String.valueOf(time_windows[gen[i]][1]);
130             String S_penalty = String.valueOf(penalty);
                penalty = tambah(String.valueOf(kurang(S_timer,
131 S_time_windows)), S_penalty);
132             if (gen[i + 1] == 12) {
133                 S_timer = String.valueOf(timer);
134                 timer = tambah(S_timer, "5.00");
135             } else {
136                 S_timer = String.valueOf(timer);
137                 timer = tambah(S_timer, "3.00");
138             }
139         } else if (timer < time_windows[gen[i]][0]) {
140             timer = (time_windows[gen[i]][0]);
141             if (gen[i + 1] == 12) {
142                 S_timer = String.valueOf(timer);
```

```
143                 timer = tambah(S_timer, "5.00");
144             } else {
145                 S_timer = String.valueOf(timer);
146                 timer = tambah(S_timer, "3.00");
147             }
148         } else {
149             if (gen[i + 1] == 12) {
150                 S_timer = String.valueOf(timer);
151                 timer = tambah(S_timer, "5.00");
152             } else {
153                 S_timer = String.valueOf(timer);
154                 timer = tambah(S_timer, "3.00");
155             }
156         }
            if (kondisi == false && timer >
157 time_windows[gen[i]][1]) {
158             S_timer = String.valueOf(timer);
                String S_time_windows =
159 String.valueOf(time_windows[gen[i]][1]);
160             String S_penalty = String.valueOf(penalty);
                penalty = tambah(String.valueOf(kurang(S_timer,
161 S_time_windows)), S_penalty);
162         }
163         indek_wisata++;
164         if (timer > 14.45) {
165             break;
166         }
167     }
168     // kembali ke hotel
169     S_time = String.valueOf(time);
170     S_timer = String.valueOf(timer);
        S_time_hotel =
    String.valueOf(time_hotel[no_hotel][gen[indek_wisata -
171 1]]);
172     time = tambah(S_time, S_time_hotel);
173     timer = tambah(S_timer, S_time_hotel);
174     // penalty
175     if (timer > 19) {
176         S_timer = String.valueOf(timer);
177         String S_penalty = String.valueOf(penalty);
            penalty = tambah(String.valueOf(kurang(S_timer,
178 "19.00")), S_penalty);
179     }
180     // untuk hari ketiga hotel ke wisata ke-
181     timer = 5;
182     S_time = String.valueOf(time);
183     S_timer = String.valueOf(timer);
        S_time_hotel =
184 String.valueOf(time_hotel[no_hotel][gen[indek_wisata]]);
185     time = tambah(S_time, S_time_hotel);
186     timer = tambah(S_timer, S_time_hotel);
187     if (timer < time_windows[gen[indek_wisata]][0]) {
188         timer = (time_windows[gen[indek_wisata]][0]);
189         if (gen[indek_wisata] == 12) {
190             S_timer = String.valueOf(timer);
191             timer = tambah(S_timer, "5.00");
192         } else {
193             S_timer = String.valueOf(timer);
194             timer = tambah(S_timer, "3.00");
```

```
195             }
196         } else {
197             if (gen[indek_wisata] == 12) {
198                 S_timer = String.valueOf(timer);
199                 timer = tambah(S_timer, "5.00");
200             } else {
201                 S_timer = String.valueOf(timer);
202                 timer = tambah(S_timer, "3.00");
203             }
204         }
205         indek_wisata++;
206         for (int i = indek_wisata; i < gen.length; i++) {
207             // untuk waktu selanjutkanya
208             S_time = String.valueOf(time);
209             S_timer = String.valueOf(timer);
                String S_time_wisata =
210 String.valueOf(time_wisata[gen[i - 1]][gen[i]]);
211             time = tambah(S_time, S_time_wisata);
212             timer = tambah(S_timer, S_time_wisata);
213             // lama berwisata 3 jam
214             boolean kondisi = false;
215             if (timer > time_windows[gen[i]][1]) {
216                 kondisi = true;
217                 S_timer = String.valueOf(timer);
                    String S_time_windows =
218 String.valueOf(time_windows[gen[i]][1]);
219                 String S_penalty = String.valueOf(penalty);
                    penalty = tambah(String.valueOf(kurang(S_timer,
220 S_time_windows)), S_penalty);
221                 if (gen[i + 1] == 12) {
222                     S_timer = String.valueOf(timer);
223                     timer = tambah(S_timer, "5.00");
224                 } else {
225                     S_timer = String.valueOf(timer);
226                     timer = tambah(S_timer, "3.00");
227                 }
228             } else if (timer < time_windows[gen[i]][0]) {
229                 S_timer = String.valueOf(timer);
                    String S_time_windows =
230 String.valueOf(time_windows[gen[i]][0]);
                    timer = tambah(S_timer,
231 String.valueOf(kurang(S_time_windows, S_timer)));
232                 if (gen[i + 1] == 12) {
233                     S_timer = String.valueOf(timer);
234                     timer = tambah(S_timer, "5.00");
235                 } else {
236                     S_timer = String.valueOf(timer);
237                     timer = tambah(S_timer, "3.00");
238                 }
239             } else {
240                 if (gen[i] == 12) {
241                     S_timer = String.valueOf(timer);
242                     timer = tambah(S_timer, "5.00");
243                 } else {
244                     S_timer = String.valueOf(timer);
245                     timer = tambah(S_timer, "3.00");
246                 }
247             }
```

```
248         if (kondisi == false && timer >
249 time_windows[gen[i]][1]) {
                S_timer = String.valueOf(timer);
250             String S_time_windows =
251 String.valueOf(time_windows[gen[i]][1]);
                String S_penalty = String.valueOf(penalty);
252             penalty = tambah(String.valueOf(kurang(S_timer,
253 S_time_windows)), S_penalty);
254         }
255         indek_wisata++;
256         if (timer > 14.45) {
257             break;
258         }
259     }
260     S_time = String.valueOf(time);
        S_timer = String.valueOf(timer);
        S_time_hotel =
261 String.valueOf(time_hotel[no_hotel][gen[indek_wisata -
262 1]]);
263     time = tambah(S_time, S_time_hotel);
264     timer = tambah(S_timer, S_time_hotel);
265     // penalty
266     if (timer > 19) {
267         S_timer = String.valueOf(timer);
            String S_penalty = String.valueOf(penalty);
268         penalty = tambah(String.valueOf(kurang(S_timer,
269 "19.00")), S_penalty);
270     }
271     double aa = indek_wisata;
272     hasil = (1 / (1 + time)) + (penalty * (-1)) + (aa /
    10);
        return hasil;
    }
```

Gambar 5.3 Implementasi Fungsi *Fitness*

Berikut adalah penjelasan source code pada Gambar 5.3:

Baris 1 : Inisialisasi *method fitness*

Baris 2-17 : Inisialisasi parameter yang digunakan pada proses perhitungan *fitness*

Baris 18-89 : Perhitungan fitness untuk hari ke-1

Baris 91-179 : Perhitungan fitness untuk hari ke-2

Baris 181-270 : Perhitungan fitness untuk hari ke-3

Baris 91-179 : Mengembalikan nilai hasil perhitungan fitness

5.2.1.3 Implementasi Reproduksi

Langkah selanjutnya adalah melakukan proses reproduksi. Pada algoritme genetika digunakan dua jenis reproduksi yaitu *crossover* dan mutasi. Pada tahap ini dilakukan perhitungan jumlah *offspring* yang dihasilkan dari perkalian dengan masing-masing operator reproduksi. Kode program implementasi reproduksi dapat dilihat pada Gambar 5.4.

```
1  public void replace_reproduksi() {
2      double child_crossover = cr * chromosom.length;
3      Double cc = new Double(child_crossover);
4      double child_mutation = mr * chromosom.length;
5      Double cm = new Double(child_mutation);
6      // inisiaslisasi offspring
7      offspring = new int[cc.intValue() + cm.intValue()][20];
8      replace_crossover(cc.intValue());
9      replace_mutation(cc.intValue(), offspring.length);
10 }
```

Gambar 5.4 Implementasi Reproduksi

Berikut adalah penjelasan source code pada Gambar 5.4:

Baris 1 : Inisialisasi method replace_reproduksi

Baris 2-5 : Inisialisasi parameter dan instansiasi objek untuk menghitung banyak *offspring* yang akan dihasilkan dengan nilai *cr* dan *mr* yang telah ditentukan

Baris 6-9 : perhitungan banyak *offspring* yang dihasilkan dari proses *crossover* dan mutasi

(a) Implementasi *Crossover* dan Seleksi

Pada proses *crossover* digunakan operator *one cut-point crossover* dimana akan dipilih satu titik dari kromosom pertama lalu pada gen setelah titik tersebut akan dimasukkan gen pada kromosom kedua satu per satu sesuai dengan urutan gen kromosom kedua yang belum tercantum pada kromosom pertama. Kode program implementasi *crossover* dapat dilihat pada Gambar 5.5.

```
1  public void replace_crossover(int t) {
2      // random 2 individu, batasan 0 - n
3      for (int x = 0; x < t; x++) {
4          int r = 0, random1 = 0, random2 = 0;
5          while (r != 1) {
6              random1 = (int) (Math.random() *
   (chromosom.length - 1));
7              random2 = (int) (Math.random() *
   (chromosom.length - 1));
8              if (random1 != random2) {
9                  r = 1;
10             }
11         }
12         // random satu titik dalam gen chromosom
13         int randomT = (int) (Math.random() *
   (chromosom[0].length - 1));
14         // nilai dikiri titik
15         int k = 0;
16         for (int i = 0; i <= randomT; i++) {
17             offspring[x][k++] = chromosom[random1][i];
18         }
19         // nilai dikanan titik
20         for (int l = 0; l < chromosom[random2].length - 1;
   l++) {
21             if (k == chromosom[0].length) {
22                 break;
23             }
```

```
24              int hasil = 0;
25              for (int j = 0; j < offspring[x].length; j++) {
26                  if (chromosom[random2][l] ==
    offspring[x][j]) {
27                      hasil = 1;
28                      break;
29                  }
30              }
31              // apakah kosong, jika ya masukan
32              if (hasil == 0) {
33                  offspring[x][k] = chromosom[random2][l];
34                  if (l != chromosom[0].length - 1) {
35                      k++;
36                  }
37              } else {
38                  hasil = 0;
39              }
40          }
41          offspring[x][offspring[0].length - 1] =
    chromosom[x][chromosom[0].length - 1];
42          // melakukan replacement selection
43          // melihat nilai fitness offspring[x],
    chromosom[random1], chromosom[random2].
44          double fitness_parent1 =
    replace_evaluasi(chromosom[random1]);
45          double fitness_parent2 =
    replace_evaluasi(chromosom[random2]);
46          double fitness_offspring =
    replace_evaluasi(offspring[x]);
47          double fitness_parent_kecil = 0;
48          int tanda = 0;
49          if (fitness_parent1 < fitness_parent2) {
50              fitness_parent_kecil = fitness_parent1;
51              tanda = random1;
52          } else {
53              fitness_parent_kecil = fitness_parent2;
54              tanda = random2;
55          }
56          if (fitness_offspring >= fitness_parent_kecil) {
57              for (int i = 0; i < chromosom[0].length; i++) {
58                  chromosom[tanda][i] = offspring[x][i];
59              }
60          }
61      }
62  }
```

Gambar 5.5 Implementasi *Crossover*

Berikut adalah penjelasan source code pada Gambar 5.5:

Baris 1 **:** Inisialisasi method replace_crossover

Baris 3 **:** Perulangan sebanyak *offspring crossover*

Baris 5-11 **:** Memilih dua individu sebagai *parent* secara acak

Baris 13 **:** Menentukan nilai *cut point* secara acak

Baris 15-18 **:** Memasukkan gen kromosom pada kiri *cut point* P1 (*parent* 1) kedalam kromosom *offspring*

Baris 19-40 : Memasukkan gen kromosom pada P2 ke dalam kromosom *offspring* dengan kondisi jika indeks gen ke-i pada kromosom P2 belum terdapat pada kromosom *offspring* maka gen tersebut akan dimasukkan ke dalam kromosom *offspring*

Baris 42-60 : Melakukan proses seleksi dengan prinsip *Replacement Selection*

(b) Implementasi Mutasi dan Seleksi

Pada proses mutasi digunakan operator *reciprocal exchange mutation*. Operator ini akan memilih dua titik pada satu kromosom secara acak dan menukar urutan kedua gen tersebut. Kode program implementasi mutasi dapat dilihat pada Gambar 5.6.

```
1  public void replace_mutation(int t, int tt) {
2      for (int i = t; i < tt; i++) {
3          int r = 0, random1 = 0, random2 = 0, random_kromosom
   = 0;
4          // memilih 2 titik gen secara acak
5          while (r == 0) {
6              random1 = (int) (Math.random() *
   (chromosom[i].length - 1));
7              random2 = (int) (Math.random() *
   (chromosom[i].length - 1));
8              if (random1 != random2) {
9                  r = 1;
10             }
11         }
12         // memilih 1 chromosom secara acak
13         random_kromosom = (int) (Math.random() *
   (chromosom.length));
14         for (int j = 0; j < chromosom[i].length; j++) {
15             offspring[i][j] = chromosom[i][j];
16         }
17         int sementara = offspring[i][random1];
18         offspring[i][random1] = offspring[i][random2];
19         offspring[i][random2] = sementara;
20         // melakukan replacement selection
21         double fitness_offspring =
   replace_evaluasi(offspring[i]);
22         double fitness_parent =
   replace_evaluasi(chromosom[random_kromosom]);
23         if (fitness_offspring > fitness_parent) {
24             chromosom[random_kromosom] = offspring[i];
25         }
26     }
27 }
```

Gambar 5.6 Implementasi Mutasi

Berikut adalah penjelasan source code pada Gambar 5.6:

Baris 1 **:** Inisialisasi method replace_mutasi

Baris 2 **:** Perulangan sebanyak *offspring* mutasi

Baris 4-11 **:** Memilih dua titik secara acak dengan batasan 1-19

Baris 13 **:** Memilih satu individu secara acak

Baris 14-19 : menukar gen pada kedua titik

Baris 20-25 : Melakukan proses seleksi dengan prinsip *Replacement Selection*

5.2.1.4 Implementasi Evaluasi

Setelah dilakukan proses reproduksi, langkah selanjutnya adalah melakukan evaluasi terhadap populasi. Pada tahap ini nilai fitness dari masing-masing *offspring* yang dihasilkan dari proses reproduksi akan dihitung. Langkah selanjutnya akan dilakukan proses seleksi. Pada proses seleksi operator seleksi yang digunakan adalah *replacement selection*. Kromosom *offspring* dengan nilai *fitness* lebih tinggi daripada kromosom *parent* yang digunakan saat proses reproduksi akan menggantikan kromosom *parent* yang mempunyai nilai *fitness* terendah. Kode program implementasi evaluasi dan seleksi dapat dilihat pada Gambar 5.7.

```
 1  public double replace_evaluasi(int[] chromosom) {
 2      double fitness = 0;
 3      fitness = replace_fitness(chromosom);
 4      return fitness;
 5  }
 6
 7  // menghitung nilai fitness [replacement selection]
 8  public double replace_fitness(int[] nilai) {
 9      return (fitness(nilai));
10  }
11
12  // menghitung nilai fitness tiap chromsom
13  public void evaluasi() {
14      for (int i = 0; i < fitness_chromosom.length; i++) {
15          for (int j = 0; j < fitness_chromosom[i].length - 1;
    j++) {
16              fitness_chromosom[i][j] = chromosom[i][j];
17          }
18      }
19      for (int i = 0; i < fitness_offspring.length; i++) {
20          for (int j = 0; j < fitness_offspring[i].length - 1;
    j++) {
21              fitness_offspring[i][j] = offspring[i][j];
22          }
23      }
24      for (int i = 0; i < fitness_chromosom.length; i++) {
25          fitness_chromosom[i][fitness_chromosom[i].length -
    1] = fitness(chromosom[i]);
26      }
27      for (int i = 0; i < fitness_offspring.length; i++) {
28          fitness_offspring[i][fitness_offspring[i].length -
    1] = fitness(offspring[i]);
29      }
30  }
```

Gambar 5.7 Implementasi Evaluasi

Berikut adalah penjelasan source code pada Gambar 5.7:

Baris 1-5 : Inisialisasi *method* replace_evaluasi untuk memanggil method replace_fitness

Baris 8-10 : Inisialisasi *method* replace_fitness untuk memanggil *method* fitness

Baris 13 : Inisialisasi *method* evaluasi

Baris 14-29 : Perulangan untuk menghitung seluruh *fitness* dalam populasi beserta *offspring*

5.2.2 Implementasi Algoritme *Simulated Annealing*

Pada sub-bab ini menampilkan implementasi kode program dari algoritme *simulated annealing*.

5.2.2.1 Implementasi Algoritme *Simulated Anneling*

Setelah didapat hasil seleksi dari algoritme genetika, langkah selanjutnya adalah mengambil kromosom dengan *fitness* terbaik untuk dilakukan optimasi menggunakan algoritme *simulated annealing*. Tahapan kode program *simulated annealing* sesuali dengan tahapan penyelesaian masalah pada sub-bab 4.2.6. Kode program implementasi algoritme *simulated annealing* dapat dilihat pada Gambar 5.8.

```
1  public void SA(int p) {
2      // chromosome
3      int[] S = chromosom[p];
4      int[] Sn = new int[chromosom[0].length];
5      // temperature
6      double temp0 = 0.9;
7      double temp1 = 0.01;
8      double t = temp0;
9      double cooling_factor = 0.995;
10     // fitness
11     double d2 = 0; // fitness dari Sn
12     // proses SA
13     while (t >= temp1) {
14         double d1 = fitness(S); // fitness dari S
15         Sn = pembangkitan_chromosome_SA(S, p);
16         // hitung d2
17         d2 = fitness(Sn);
18         // kondisi
19         if (d2 > d1) {
20             S = Sn;
21         } else {
22             int k = 25;
23             double prob = 1/(Math.exp((d1 - d2)*k / (t)));
24             // kondisi d2
25             double alpa = Math.random();
26             if (alpa < prob) {
27                 S = Sn;
28             }
29         }
30         t = t * cooling_factor;
31     }
32     chromosom[p] = S; // add new solution in GA's from SA
33 }
```

Gambar 5.8 Implementasi Algoritme *Simulated Annealing*

Berikut adalah penjelasan source code pada Gambar 5.8:

Baris 1 : Inisialisasi *method* SA

Baris 3-11 : Inisialisasi parameter yang digunakan pada proses SA

Baris 13 : Perulangan selama temperatur saat ini lebih dari temperatur akhir

Baris 15 : Membangkitkan kromosom *Sn* dengan operator *scramble mutation*

Baris 19-29 : Pengkondisian penerimaan solusi tetangga, jika *fitness* solusi tetangga lebih baik dari pada *fitness* solusi awal maka solusi tetangga diterima sebagai solusi baru, jika tidak maka akan dilakukan perhitungan probabilitas penerimaan solusi baru.

Baris 30 : Menurunkan suhu berdasarkan parameter *cooling factor*

5.2.2.2 Implementasi Inisialisasi Solusi Awal

Solusi awal yang digunakan pada tahap ini diambil dari kromosom dengan nilai *fitness* terbaik dan terburuk dari hasil seleksi dari aloritma genetika. Kode program inisialisasi solusi awal dapat dilihat pada Gambar 5.9.

```
1  public int pilih(int kondisi) {
2      int t = 0;
3      if (kondisi == 0) {//fitness terbaik
4          double besar =
   fitness_chromosom[0][fitness_chromosom[0].length - 1];
5          for (int i = 1; i < fitness_chromosom.length; i++) {
6              if
   (fitness_chromosom[i][fitness_chromosom[0].length - 1] >
   besar) {
7                  besar =
   fitness_chromosom[i][fitness_chromosom[0].length - 1];
8                  t = i;
9              }
10         }
11     } else if (kondisi == 1) {//fitness terburuk
12         double kecil =
   fitness_chromosom[0][fitness_chromosom[0].length - 1];
13         for (int i = 1; i < fitness_chromosom.length; i++) {
14             if
   (fitness_chromosom[i][fitness_chromosom[0].length - 1] <
   kecil) {
15                 kecil =
   fitness_chromosom[i][fitness_chromosom[0].length - 1];
16                 t = i;
17             }
18         }
19     } else if (kondisi == 2) {//random
20         double random = Math.random() * (chromosom.length);
21         Double tt = new Double(random);
22         t = tt.intValue();
23     }
24     return t;
25 }
```

Gambar 5.9 Implementasi Inisialisasi Solusi Awal

Berikut adalah penjelasan source code pada Gambar 5.9:

Baris 1 : Inisialisasi *method* pilih_kondisi untuk menentukan solusi awal SA berasal dari kromosom dengan *fitness* terendah atau tertinggi

Baris 2-10 : Kondisi jika solusi awal SA yang dipilih berasal dari fitness kromosom tertinggi

Baris 11-18 : Kondisi jika solusi awal SA yang dipilih berasal dari fitness kromosom terendah

Baris 19-23 : Kondisi jika solusi awal SA yang dipilih secara acak

Baris 24 : Mengembalikan indeks kromosom terpilih

5.2.2.3 Implementasi Inisialisasi Solusi Tetangga

Solusi tetangga pada algoritme *simulated annealing* diperoleh menggunakan operator *Scramble Mutation*. Pada tahap pertama, dua titik akan dipilih dari kromosom yang menjadi solusi awal secara acak sebagai batasan. Gen dalam dua titik tersebut akan diacak secara acak. Kode program inisialisasi solusi tetangga dapat dilihat pada Gambar 5.10.

```
1  public int[] scramble_mutation(int[] S) {
2      int[] new_S = new int[S.length];
3      // bangkitkan dua titik q dan r
4      int pp = 0;
5      int q = 0; // batas depan
6      int r = 0; // batas belakang
7      while (pp != 1) {
8          Double qq = new Double(Math.random() * (S.length -
   1));
9          q = qq.intValue();
10         Double rr = new Double(Math.random() * (S.length -
   1));
11         r = rr.intValue();
12         if (r > q) {
13             pp = 1;
14         }
15     }
16     int[] kondisi = new int[1 + r - q];
17     for (int i = 0; i < kondisi.length; i++) {
18         kondisi[i] = 999;
19     }
20     int s = 0;
21     for (int i = 0; i < new_S.length; i++) {
22         if (i >= q && i <= r) {
23             // random
24             int boll = 0;
25             int kk = 0;
26             while (boll != 1) {
27                 Double k = Math.random() * (1 + r - q) + q;
28                 kk = k.intValue();
29                 for (int j = 0; j < kondisi.length; j++) {
30                     if (kondisi[j] == kk) {
31                         break;
32                     }
33                     if (j == kondisi.length - 1) {
```

```
34                          boll = 1;
35                          kondisi[s++] = kk;
36                      }
37                  }
38              }
39              int nilai = S[kk];
40              new_S[i] = nilai;
41          } else {
42              new_S[i] = S[i];
43          }
44      }
45      return new_S;
46  }
```

Gambar 5.10 Implementasi Inisialisasi Solusi Tetangga

Berikut adalah penjelasan source code pada Gambar 5.10:

Baris 1 **:** Inisialisasi *method* scramble_mutation

Baris 2-6 **:** Inisialisasi parameter yang digunakan dalam perhitungan *scramble mutation*

Baris 7-15 **:** Perulangan untuk mendapatkan nilai 2

Baris 16-19 **:** Inisialisasi array baru sepanjang kromosom dianara dua titik

Baris 21-44 **:** Perulangan untuk mengacak gen diantara kedua titik

BAB 6 PENGUJIAN DAN ANALISIS

6.1 Pengujian Parameter Algoritme Genetika

Pengujian pada algoritme genetika dilakukan pada parameter ukuran populasi, banyak generasi, dan kombinasi *cr* dan *mr*. Nomor urut hotel yang digunakan dalam pengujian parameter algoritme genetika adalah nomor hotel 8 yang dapa dilihat pada Tabel 3.2. Pengujian parameter algoritme genetika dilakukan untuk mencari nilai parameter mendekati optimal yang dapat menghasilkan nilai *fitness* tertinggi.

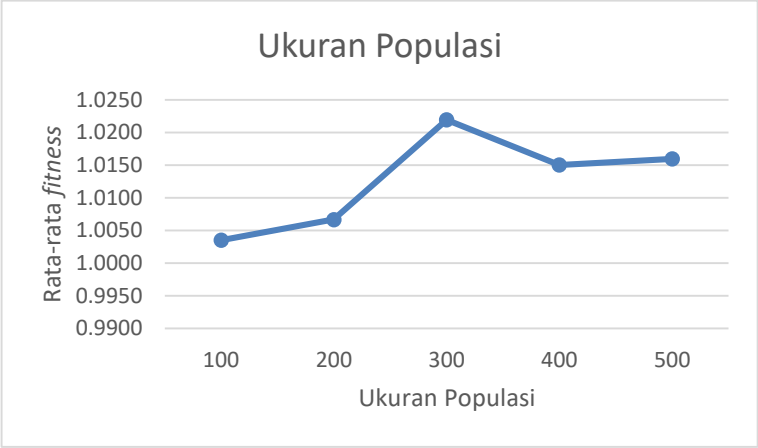
6.1.1 Pengujian Parameter Ukuran Populasi

Pengujian parameter ukuran populasi pada algoritme genetika dilakukan dengan menggunakan parameter awal banyak generasi sebesar 1000, *cr* sebesar 0.8, dan *mr* sebesar 0.2. Parameter algoritme genetika awal yang digunakan pada pengujian parameter ukuran populasi mengacu pada parameter terbaik yang dihasilkan pada penelitian sebelumnya mengenai permasalahan yang sama dari Anggodo *et al* (2016). Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter ukuran populasi mulai dari 100 hingga 500 dengan kelipatan 100 (Anggodo *et a*l, 2016). Hasil pengujian parameter ukuran populasi dapat dilihat pada Tabel 6.1.

Tabel 6.1 Tabel Pengujian Parameter Ukuran Populasi

pop Size	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
100	1.0047	1.0053	1.0085	1.0101	0.9987	0.9973	1.0093	1.0046	1.0064	0.9903	1.0035
200	1.0064	1.0101	1.0064	1.0093	1.0059	0.9988	1.0094	1.0047	1.0064	1.0093	1.0067
300	**1.0074**	**1.0883**	**1.0059**	**1.0064**	**1.0867**	**1.0074**	**1.0050**	**0.9988**	**1.0085**	**1.0048**	**1.0219**
400	1.0101	1.0900	1.0080	1.0050	1.0064	0.9988	1.0060	1.0093	1.0101	1.0064	1.0150
500	0.9985	1.0101	1.0093	1.0101	1.0050	1.0085	1.0888	1.0096	1.0094	1.0101	1.0160

Representasi hasil pengujian parameter ukuran populasi algoritme genetika ditunjukkan pada Gambar 6.1.



Gambar 6.1 Grafik Pengujian Parameter Ukuran Populasi

Berdasarkan Gambar 6.1 nilai rata-rata *fitness* dengan nilai kenaikan paling signifikan didapatkan pada ukuran populasi 300 dengan rata-rata fitness sebesar 1,0219. Hasil keluaran salah satu kromosom pada pengujian parameter ukuran populasi algoritme genetika sebesar 300 yaitu dengan urutan indeks wisata 12, 9, 17, 10, 19, 5, 14, 4, 7, 13, 8, 11, 6, 16, 18, 2, 3, 15, 1 yang menghasilkan nilai *fitness* sebesar 1.0064 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Melalui grafik di atas juga dapat disimpulkan bahwa semakin besar ukuran populasi yang digunakan, solusi yang dihasilkan akan lebih optimal karena kromosom yang dihasilkan pada satu populasi lebih beragam sehingga peluang mendapatkan solusi lebih tinggi menjadi lebih besar.

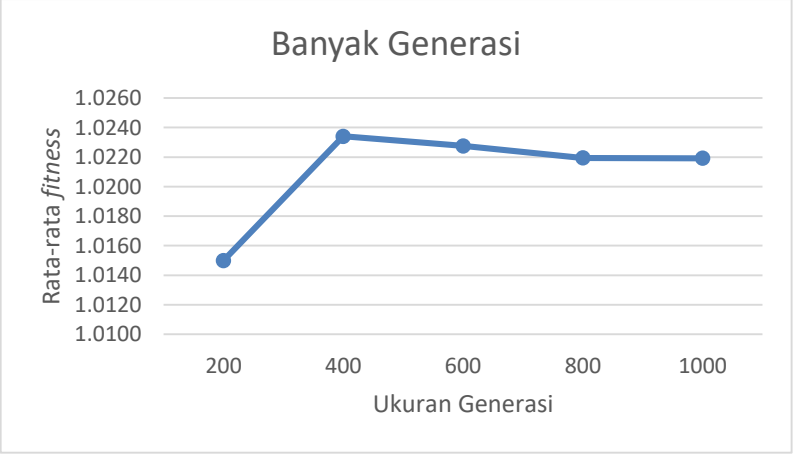
6.1.2 Pengujian Parameter Banyak Generasi

Pengujian parameter banyak generasi pada algoritme genetika dilakukan dengan menggunakan parameter ukuran populasi sebesar 300 yang diperoleh dari hasil pengujian ukuran populasi sebelumnya yang dapat dilihat pada Tabel 6.1, *cr* sebesar 0.8, dan *mr* sebesar 0.2 (Anggodo *et al*, 2016). Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter Banyak Generasi mulai dari 200 hingga 1000 dengan kelipatan 100 (Anggodo *et al*, 2016). Hasil pengujian parameter ukuran populasi dapat dilihat pada Tabel 6.2.

Tabel 6.2 Tabel Pengujian Parameter Banyak Generasi

Banyak Generasi	Nilai *fitness*										Rata-rata *fitness*
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
200	0.9988	1.0096	1.0093	1.0053	1.0065	1.0050	1.0093	1.0074	1.0093	1.0894	1.0150
400	**1.0094**	**1.0093**	**1.0829**	**1.0054**	**1.0050**	**1.0074**	**1.0050**	**1.0101**	**1.0894**	**1.0101**	**1.0234**
600	1.0059	1.0064	1.0050	1.0052	1.0900	1.0054	1.0053	1.0093	1.0867	1.0085	1.0228
800	1.0883	0.9987	1.0101	1.0050	1.0064	1.0093	0.9970	1.0059	1.0093	1.0894	1.0219
1000	1.0074	1.0883	1.0059	1.0064	1.0867	1.0074	1.0050	0.9988	1.0085	1.0048	1.0219

Representasi hasil pengujian parameter banyak generasi algoritme genetika ditunjukkan pada Gambar 6.2.



Gambar 6.2 Grafik Pengujian Parameter Banyak Generasi

Berdasarkan Gambar 6.2 nilai rata-rata *fitness* tertinggi didapatkan pada banyak generasi sebesar 400 dengan rata-rata *fitness* sebesar 1.0234. Hasil keluaran salah satu kromosom pada pengujian parameter banyak generasi algoritme genetika sebesar 400 generasi yaitu dengan urutan indeks wisata 4, 5, 7, 8, 2, 17, 14, 9, 12, 3, 11, 19, 18, 6, 16, 15, 13, 10, 1 yang menghasilkan nilai *fitness* sebesar 1.0054 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Semakin banyak banyak generasi yang digunakan maka cenderung didapatkan nilai *fitness* yang semakin besar, tetapi setelah generasi tertentu tidak didapatkan perbaikan hasil yang signifikan. Penggunaan banyak generasi yang terlalu besar hanya akan menambah waktu komputasi namun solusi yang dihasilkan tetap atau konvergen dari solusi terbaik sebelumnya (Mahmudy, 2013b).

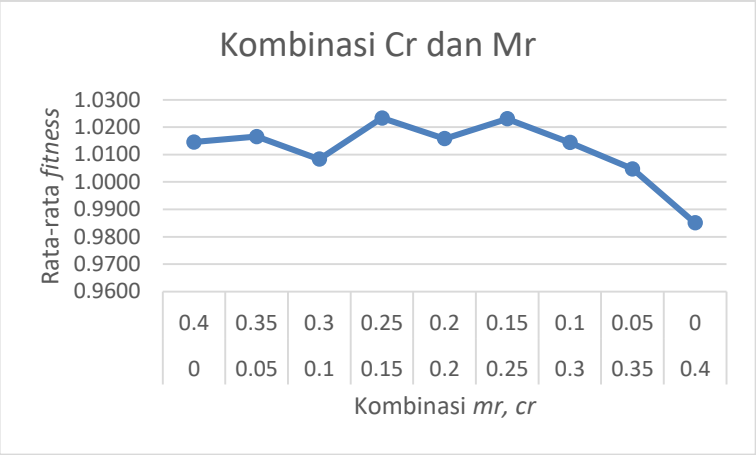
6.1.3 Pengujian Parameter Kombinasi *cr* dan *mr*

Pengujian parameter kombinasi *cr* dan *mr* pada algoritme genetika dilakukan dengan menggunakan parameter ukuran populasi sebesar 300 dan banyak generasi sebesar 400 yang diperoleh dari hasil pengujian ukuran populasi sebelumnya yang dapat dilihat pada Tabel 6.1 dan 6.2. Pengujian dilakukan dengan menggunakan kombinasi *cr* dan *mr* pada kisaran nilai antara 0-0.4 dengan mengatur kombinasi *cr* dan *mr* sehingga menghasilkan nilai *cr* + *mr* = 0.4 untuk mendapatkan hasil yang lebih optimal (Mahmudy *et al*, 2012a). Pada pengujian kombinasi *cr* dan *mr* akan dilakukan percobaan sebanyak 10 kali. Hasil pengujian parameter kombinasi *cr* dan *mr* dapat dilihat pada Tabel 6.3.

Tabel 6.3 Tabel Pengujian Parameter Kombinasi *cr* dan *mr*

Kombinasi		**Nilai *fitness***										**Rata-rata *fitness***
		Iterasi ke-										
Cr	***mr***	1	2	3	4	5	6	7	8	9	10	
0	0.4	1.0054	1.0101	1.0094	1.0064	1.0064	1.0064	1.0429	1.0406	1.0096	1.0085	1.0146
0.05	0.35	1.0094	1.0096	1.0096	1.0050	1.0064	1.0101	1.0074	1.0093	1.0094	1.0891	1.0166
0.1	0.3	1.0059	1.0074	1.0093	1.0064	1.0101	1.0094	1.0064	1.0094	1.0094	1.0096	1.0083
0.15	**0.25**	**1.0093**	**1.0064**	**1.0864**	**1.0064**	**1.0054**	**1.0900**	**1.0093**	**1.0093**	**1.0054**	**1.0054**	**1.0233**
0.2	0.2	1.0047	1.0074	1.0085	1.0054	1.0900	1.0085	1.0054	1.0094	1.0093	1.0094	1.0158
0.25	0.15	1.0047	1.0093	1.0093	1.0864	1.0057	1.0064	1.0050	1.0894	1.0050	1.0094	1.0231
0.3	0.1	1.0093	1.0064	1.0074	1.0888	1.0093	1.0093	1.0096	1.0064	0.9988	0.9988	1.0144
0.35	0.05	1.0064	1.0096	1.0074	0.9953	1.0064	1.0094	1.0064	1.0094	0.9984	0.9988	1.0048
0.4	0	0.9752	0.9883	0.9759	0.9687	0.9994	0.9897	0.9768	1.0000	0.9822	0.9945	0.9851

Representasi hasil pengujian parameter kombinasi *cr* dan *mr* algoritme genetika ditunjukkan pada Gambar 6.3.



Gambar 6.3 Grafik Pengujian Parameter Kombinasi *cr* dan *mr*

Berdasarkan Gambar 6.3 nilai rata-rata *fitness* tertinggi didapatkan pada kombinasi *cr* dan *mr* sebesar 0.15 dan 0.25 dengan rata-rata *fitness* sebesar 1.0233. Hasil keluaran salah satu kromosom pada pengujian parameter kombinasi *cr* dan *mr* algoritme genetika masing-masing sebesar 0.15 dan 0.25 yaitu dengan urutan indeks wisata 14, 12, 9, 6, 11, 17, 7, 5, 4, 10, 16, 19, 3, 8, 15, 18, 13, 2, 1 yang menghasilkan nilai *fitness* sebesar 1.0093 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Penggunaan nilai *cr* yang terlalu tinggi akan mengurangi keragaman kromosom dalam suatu populasi yang akan mengakibatkan konvergensi dini, sedangkan penggunaan nilai *mr* yang terlalu tinggi akan membuat algoritme genetika menjadi metode acak dan tidak dapat secara efektif melakukan pencarian solusi pada ruang pencarian (Mahmudy et al, 2013b).

Dari ketiga parameter yang digunakan pada pengujian algoritme genetika pada permasalahan VRPTW, didapat parameter mendekati optimal pada ukuran populasi 300, banyak generasi 400, serta kombinasi *cr* dan *mr* sebesar 0.15 dan 0.25 dengan nilai rata-rata *fitness* sebesar 1.0233.

6.2 Pengujian Parameter *Simulated Annealing*

Pengujian pada algoritme *simulated annealing* dilakukan pada parameter *temp0*, *cooling factor*, dan koefisien penerimaan solusi baru. Pengujian parameter *simulated annealing* dilakukan untuk mencari nilai parameter mendekati optimal yang dapat menghasilkan nilai *fitness* tertinggi. Pengujian dilakukan dengan inisialisasi parameter *temp1* sebesar 0.01 dan nomor hotel 8. Kromosom solusi awal yang digunakan pada pengujian parameter algoritme *simulated annealing* ditentukan dengan urutan indeks permutasi gen 4, 9, 14, 5, 19, 10, 11, 17, 6, 7, 12, 13, 18, 16, 2, 3, 15, 8, 1. Penggunaan kromosom yang telah ditentukan bertujuan untuk mengurangi hasil keluaran solusi yang terlalu acak sehingga akan membuat pengujian parameter *simulated annealing* menjadi tidak dapat menemukan parameter yang mendekati optimal.

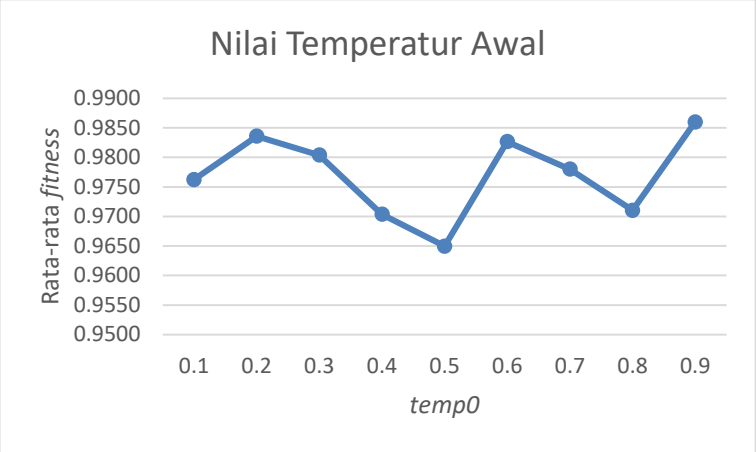
6.2.1 Pengujian Parameter Temperatur Awal

Pengujian parameter temperature awal (*temp0*) pada algoritme *simulated annealing* dilakukan dengan menggunakan parameter awal *cooling factor* sebesar 0.8 dan nilak *k* sebesar 25 (Mahmudy, 2014). Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter *temp0* mulai dari 0.1 hingga 0.9 dengan kelipatan 0.11. Hasil pengujian parameter *temp0* dapat dilihat pada Tabel 6.4.

Tabel 6.4 Tabel Pengujian Parameter Temperatur Awal

temp0	Nilai *fitness*										Rata-rata *fitness*
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
0.1	0.9894	0.9815	0.9811	0.9970	0.9878	0.9749	0.8864	0.9975	0.9908	0.9762	0.9762
0.2	0.9811	0.9698	0.9867	0.9797	0.9888	0.9737	0.9889	0.9962	0.9878	0.9831	0.9836
0.3	0.9900	0.9814	1.0046	0.9737	0.9906	0.9266	0.9703	0.9905	0.9867	0.9895	0.9804
0.4	0.9875	0.9815	0.9825	0.9764	0.8760	0.9645	0.9709	0.9952	0.9816	0.9875	0.9704
0.5	0.9896	0.9812	0.9707	0.9999	0.9883	0.8704	0.8865	0.9898	0.9907	0.9825	0.9650
0.6	1.0046	0.9893	0.9704	0.9983	0.9805	0.9961	0.9686	0.9742	0.9706	0.9739	0.9826
0.7	0.9831	0.9891	0.9950	0.9875	0.9896	0.9873	0.9950	0.9813	0.8756	0.9965	0.9780
0.8	0.9703	0.9868	1.0043	0.8887	0.9710	0.9819	0.9809	0.9737	0.9762	0.9766	0.9710
0.9	**0.9903**	**0.9894**	**0.9815**	**0.9864**	**0.9713**	**1.0080**	**0.9749**	**0.9871**	**0.9820**	**0.9887**	**0.9860**

Representasi hasil pengujian parameter temperatur awal algoritme *simulated annealing* ditunjukkan pada Gambar 6.4.



Gambar 6.4 Grafik Pengujian Parameter Temperatur Awal

Berdasarkan Gambar 6.4 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *temp0* sebesar 0.9 dengan rata-rata *fitness* sebesar 0.9860. Hasil keluaran salah satu kromosom pada pengujian parameter temperatur awal algoritme *simulated annealing* sebesar 0.9 yaitu dengan urutan indeks wisata 9, 5, 4, 11, 10, 19, 14, 12, 7, 15, 16, 8, 17, 3, 2, 6, 18, 13, 1, 7 yang menghasilkan nilai *fitness* sebesar 0.9903 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Penggunaan nilai *temp0* yang terlalu rendah membuat algoritme *simulated annealing* kurang optimal dalam mengeksplorasi ruang solusi sehingga menyebabkan konvergensi dini, sedangkan penggunaan *temp0* yang terlalu tinggi akan menambah waktu komputasi (Novitasari dan Mahmudy, 2016).

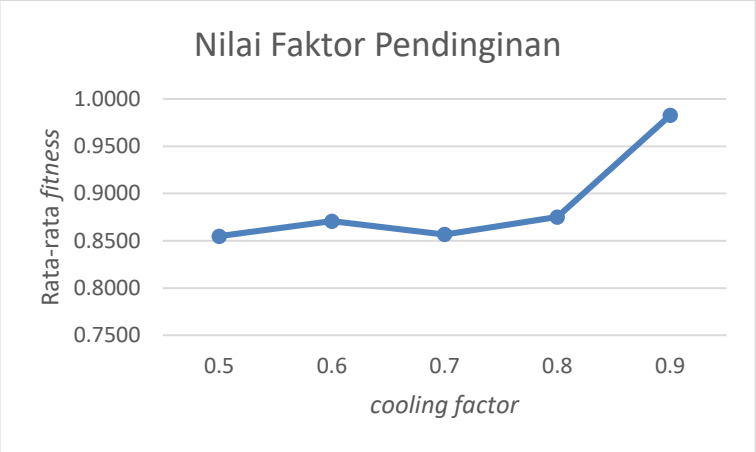
6.2.2 Pengujian Parameter Faktor Pendinginan

Pengujian parameter factor pendinginan (*cooling factor*) pada algoritme *simulated annealing* dilakukan dengan menggunakan parameter temperature awal *temp0* sebesar 0.9 yang diperoleh dari pengujian parameter *temp0* sebelumnya yang dapat dilihat pada Tabel 6.4 dan nilai *k* sebesar 25 (Mahmudy, 2014). Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter *cooling factor* mulai dari 0.5 hingga 0.9 dengan kelipatan 0.1 (Novitasari & Mahmudy, 2016). Hasil pengujian parameter *cooling factor* dapat dilihat pada Tabel 6.5.

Tabel 6.5 Tabel Pengujian Parameter *Cooling Factor*

cooling factor	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
0.5	0.9805	0.7820	0.8994	0.9294	0.7820	0.7820	0.7820	0.8741	0.8743	0.8626	0.8548
0.6	0.8957	0.9806	0.9265	0.9887	0.7825	0.7800	0.7820	0.9890	0.7820	0.8000	0.8707
0.7	0.7820	0.8864	0.9887	0.8769	0.8169	0.7820	0.8804	0.8958	0.8760	0.7820	0.8567
0.8	0.9740	0.8000	0.9000	0.8763	0.7820	0.9895	0.8664	0.8794	0.9015	0.7820	0.8751
0.9	**0.9753**	**0.9876**	**0.9950**	**0.9795**	**0.9688**	**0.9899**	**0.9739**	**0.9962**	**0.9863**	**0.9748**	**0.9827**

Representasi hasil pengujian parameter *temp0* algoritme *simulated annealing* ditunjukkan pada Gambar 6.5.



Gambar 6.5 Grafik Pengujian Parameter Faktor Pendinginan

Berdasarkan Gambar 6.5 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *cooling factor* sebesar 0.9 dengan rata-rata *fitness* sebesar 0.9827. Hasil keluaran salah satu kromosom pada pengujian parameter faktor pendinginan algoritme *simulated annealing* sebesar 0.9 yaitu dengan urutan indeks wisata 4, 17, 10, 9, 2, 11, 7, 5, 12, 3, 19, 14, 8, 13, 1, 6, 16, 18, 15 yang menghasilkan nilai *fitness* sebesar 0.9753 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Penggunaan nilai cooling factor yang terlalu rendah membuat algoritme *simulated annealing* kemungkinan gagal untuk menemukan solusi lokal optimal, mengulang-ulang perhitungan sehingga menambah waktu komputasi, sedangkan penggunaan *cooling factor* yang terlalu tinggi akan menambah waktu komputasi (Novitasari dan Mahmudy, 2016).

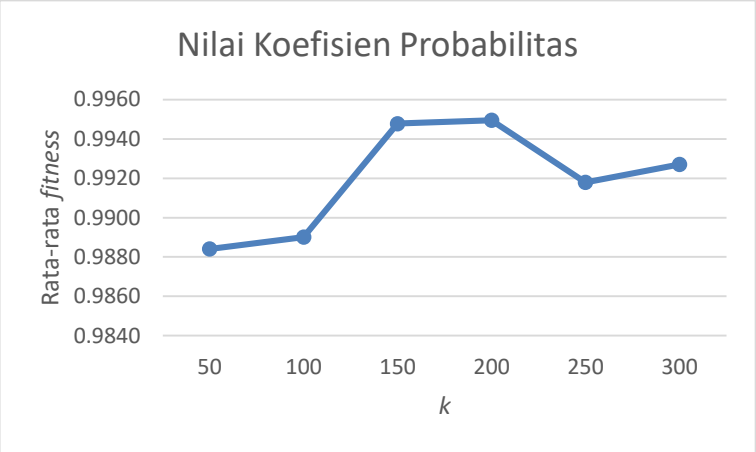
6.2.3 Pengujian Nilai Koefisien Probabilitas

Pengujian nilai koefisien probabilitas penerimaan solusi baru pada algoritme *simulated annealing* dilakukan dengan menggunakan parameter temperature awal *temp0* sebesar 0.9, cooling factor sebesar 0.9 yang diperoleh dari pengujian parameter temperatur awal dan *cooling factor* sebelumnya yang dapat dilihat pada Tabel 6.4 dan Tabel 6.5. Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter *cooling factor* mulai dari 50 hingga 300 dengan kelipatan 50. Hasil pengujian nilai koefisien probabilitas penerimaan solusi baru dapat dilihat pada Tabel 6.6.

Tabel 6.6 Tabel Pengujian Nilai Koefisien Probabilitas

k	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
50	0.9966	0.9739	0.9815	0.9806	0.9872	1.0047	0.9758	0.9977	0.9967	0.9894	0.9884
100	0.9972	0.9944	0.9906	0.9762	0.9763	0.9894	0.9960	0.9885	0.9948	0.9868	0.9890
150	0.9952	0.9888	0.9951	0.9980	0.9985	0.9894	0.9895	0.9989	0.9895	1.0047	0.9948
200	**0.9893**	**0.9992**	**0.9947**	**0.9971**	**0.9948**	**0.9873**	**0.9976**	**0.9870**	**1.0082**	**0.9944**	**0.9949**
250	0.9713	0.9995	0.9829	1.0059	1.0043	0.9812	0.9893	0.9988	0.9867	0.9979	0.9918
300	0.9966	0.9870	0.9812	1.0093	0.9977	0.9969	0.9887	0.9829	0.9894	0.9975	0.9927

Representasi hasil pengujian nilai koefisien probabilitas penerimaan solusi baru algoritme *simulated annealing* ditunjukkan pada Gambar 6.6.



Gambar 6.6 Grafik Pengujian Nilai Koefisien Probabilitas

Berdasarkan Gambar 6.6 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *k* sebesar 200 dengan rata-rata *fitness* sebesar 0.9949. Hasil keluaran salah satu kromosom pada pengujian parameter nilai koefisien probabilitas penerimaan solusi baru algoritme *simulated annealing* sebesar 200 yaitu dengan urutan indeks wisata 9, 14, 4, 10, 19, 5, 11, 17, 12, 6, 13, 3, 2, 7, 15, 1, 16, 8, 18, 7 yang menghasilkan nilai *fitness* sebesar 0.9893 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 9 objek wisata. Penetapan nilai *k* yang terlalu rendah menyebabkan tingginya perubahan yang acak saat menerima solusi baru dengan *fitness* yang lebih buruk, sedangkan penetapan nilai *k* yang terlalu tinggi akan menghasilkan nilai probabilitas penerimaan solusi baru dengan nilai yang lebih

buruk akan rendah sehingga akan menghilangkan kemampuan dari algoritme *simulated annealing* untuk keluar dari lokal optimal (Mahmudy, 2014).

Dari ketiga parameter yang digunakan pada pengujian algoritme simulated annealing pada permasalahan VRPTW, didapat parameter mendekati optimal pada nilai *temp0* sebesar 0.9, nilai *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200 dengan nilai rata-rata *fitness* sebesar 0.9949.

6.3 Pengujian Parameter GA-SA

Pengujian pada hibridisasi GA-SA dilakukan pada parameter algoritme genetika yaitu ukuran populasi, banyak generasi serta *cr* dan *mr*, sedangkan pada parameter *simulated annealing* dilakukan pada parameter *temp0*, *cooling factor*, dan koefisien probabilitas penerimaan solusi baru. Pengujian parameter GA-SA dilakukan untuk mencari nilai parameter mendekati optimal yang dapat menghasilkan nilai *fitness* tertinggi. Nomor hotel yang dipakai dalam perhitungan pengyjian parameter GA-SA adalah nomor notel 8. Pengujian parameter GA-SA juga akan menunjukkan apakah penggunaan hibridisasi GA-SA menghasilkan solusi yang lebih baik dari pada penggunaan GA da SA secara terpisah.

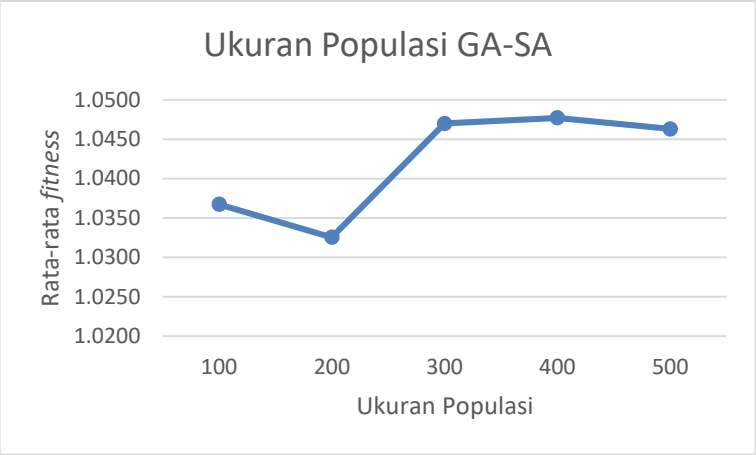
6.3.1 Pengujian Parameter Ukuran Populasi GA-SA

Pengujian parameter ukuran populasi pada hibridisasi GA-SA dilakukan dengan menggunakan parameter awal banyak generasi sebesar 400, *cr* sebesar 0.15, dan *mr* sebesar 0.25, *temp0* sebesar 0.9, *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200. Parameter awal pengujian ukuran populasi didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.2 sampai dengan Tabel 6.6. Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter ukuran populasi mulai dari 100 hingga 500 dengan kelipatan 100 (Anggodo *et a*l, 2016). Hasil pengujian parameter ukuran populasi dapat dilihat pada Tabel 6.7.

Tabel 6.7 Tabel Pengujian Parameter Ukuran Populasi GA-SA

pop Size	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
100	1.0868	0.9953	1.0093	1.0093	1.0074	1.0868	1.0867	1.0064	1.0699	1.0094	1.0367
200	1.0101	1.0888	1.0093	1.0074	1.0868	1.0093	1.0085	1.0093	1.0867	1.0093	1.0325
300	1.0888	1.0093	1.0047	1.0894	1.0074	1.0812	1.0868	1.0101	1.0054	1.0868	1.0470
400	**1.0888**	**1.0812**	**1.0888**	**1.0884**	**1.0057**	**1.0888**	**1.0074**	**1.0093**	**1.0094**	**1.0093**	**1.0477**
500	1.0891	1.0047	1.0093	1.0093	1.0868	1.0699	1.0884	1.0094	1.0094	1.0867	1.0463

Representasi hasil pengujian parameter ukuran populasi hibridisasi GA-SA ditunjukkan pada Gambar 6.7.



Gambar 6.7 Grafik Pengujian Parameter Ukuran Populasi GA-SA

Berdasarkan Gambar 6.7 nilai rata-rata *fitness* tertinggi didapatkan pada ukuran populasi 400 dengan rata-rata *fitness* sebesar 1.0477. Hasil keluaran salah satu kromosom pada pengujian parameter ukuran populasi hibridisasi GA-SA sebesar 500 populasi yaitu dengan urutan indeks wisata 6, 11, 19, 5, 7, 4, 12, 9, 17, 14, 13, 15, 18, 3, 10, 16, 2, 8, 1 yang menghasilkan nilai *fitness* sebesar 1.0888 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata *fitness* yang diperoleh pada pengujian parameter ukuran populasi hibriisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata *fitness* pengujian ukuran populasi pada algoritm genetika yang hanya mencapai 1.0219.

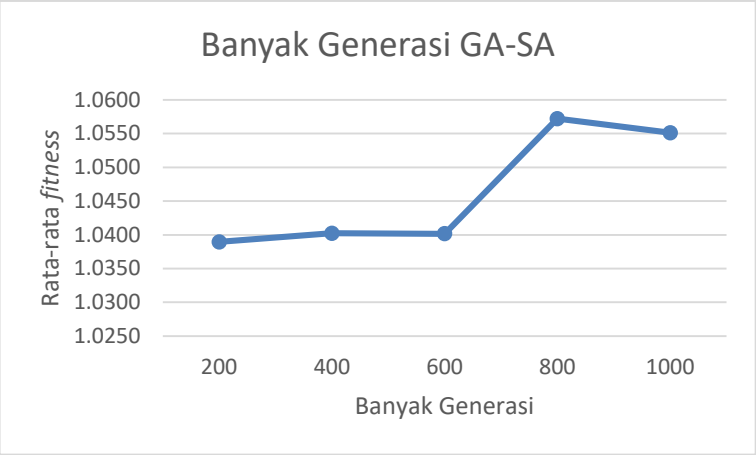
6.3.2 Pengujian Parameter Banyak Generasi GA-SA

Pengujian parameter banyak generasi pada hibridisasi GA-SA dilakukan dengan menggunakan parameter ukuran populasi sebesar 400, *cr* sebesar 0.15, dan *mr* sebesar 0.25, *temp0* sebesar 0.9, *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200. Parameter pengujian banyak generasi didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.3 sampai dengan Tabel 6.7. Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter banyak generasi mulai dari 200 hingga 1000 dengan kelipatan 200 (Anggodo *et al*, 2016). Hasil pengujian parameter banyak generasi dapat dilihat pada Tabel 6.8.

Tabel 6.8 Tabel Pengujian Parameter Banyak Generasi GA-SA

Banyak Generasi	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
200	1.0897	1.0900	1.0060	1.0064	1.0057	1.0900	1.0093	1.0765	1.0064	1.0094	1.0389
400	1.0884	1.0064	1.0096	1.0900	1.0053	1.0057	1.0096	1.0900	1.0888	1.0085	1.0402
600	1.0093	1.0094	1.0864	1.0094	1.0894	1.0096	1.0897	1.0064	1.0867	1.0053	1.0402
800	**1.0888**	**1.0900**	**1.0094**	**1.0900**	**1.0894**	**1.0093**	**1.0868**	**1.0093**	**1.0894**	**1.0096**	**1.0572**
1000	1.0888	1.0093	1.0094	1.0864	1.0093	1.0900	1.0888	1.0064	1.0765	1.0864	1.0551

Representasi hasil pengujian parameter banyak generasi hibridisasi GA-SA ditunjukkan pada Gambar 6.8.



Gambar 6.8 Grafik Pengujian Parameter banyak generasi GA-SA

Berdasarkan Gambar 6.8 nilai rata-rata *fitness* tertinggi didapatkan pada banyak generasi 800 dengan rata-rata *fitness* sebesar 1.0572. Hasil keluaran salah satu kromosom pada pengujian parameter banyak generasi hibridisasi GA-SA sebesar 800 generasi yaitu dengan urutan indeks wisata 6, 11, 19, 5, 7, 4, 12, 9, 17, 14, 1, 15, 13, 18, 10, 3, 2, 16, 8 yang menghasilkan nilai *fitness* sebesar 1.0888 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata *fitness* yang diperoleh pada pengujian parameter banyak generasi hibridisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata *fitness* pengujian banyak generasi pada algoritme genetika yang hanya mencapai 1,0234.

6.3.3 Pengujian Parameter Kombinasi *cr* dan *mr* GA-SA

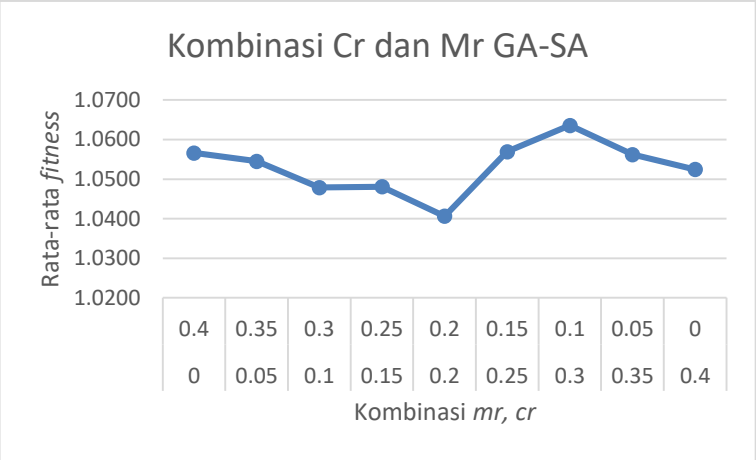
Pengujian parameter kombinasi *cr* dan *mr* pada hibridisasi GA-SA dilakukan dengan menggunakan parameter ukuran populasi sebesar 400, banyak generasi sebesar 800, *temp0* sebesar 0.9, *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200. Parameter pengujian kombinasi *cr* dan *mr* didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.4 sampai dengan Tabel 6.8. Pengujian dilakukan dengan menggunakan kombinasi *cr* dan *mr* pada kisaran nilai antara 0-0.4 dengan mengatur kombinasi *cr* dan *mr* sehingga menghasilkan nilai *cr* + *mr* = 0.4 untuk mendapatkan hasil yang lebih optimal (Mahmudy *et al*, 2012a). Pada pengujian kombinasi *cr* dan *mr* akan dilakukan percobaan sebanyak 10 kali. Hasil pengujian parameter kombinasi *cr* dan *mr* dapat dilihat pada Tabel 6.8.

Tabel 6.9 Tabel Pengujian Parameter Kombinasi *cr* dan *mr* GA-SA

Kombinasi		Nilai *fitness* Iterasi ke-										Rata-rata *fitness*
Cr	***mr***	1	2	3	4	5	6	7	8	9	10	
0	0.4	1.0867	1.0094	1.0864	1.0867	1.0894	1.0101	1.0888	1.0094	1.0094	1.0900	1.0566
0.05	0.35	1.0057	1.0096	1.0888	1.0894	1.0057	1.0101	1.0900	1.0868	1.0868	1.0724	1.0545
0.1	0.3	1.0096	1.0085	1.0864	1.0085	1.0868	1.0888	1.0897	1.0085	1.0060	1.0864	1.0479
0.15	0.25	1.0888	1.0096	1.0880	1.0049	1.0867	1.0064	1.0094	1.0888	1.0888	1.0094	1.0481
0.2	0.2	1.0096	1.0064	1.0085	1.0093	1.0085	1.0900	1.0868	1.0093	1.0894	1.0888	1.0407
0.25	0.15	1.0864	1.0094	1.0085	1.0897	1.0093	1.0894	1.0894	1.0900	1.0900	1.0074	1.0569

0.3	**0.1**	**1.0085**	**1.0900**	**1.0094**	**1.0060**	**1.0868**	**1.0891**	**1.0894**	**1.0876**	**1.0880**	**1.0812**	**1.0636**
0.35	0.05	1.0868	1.0891	1.0063	1.0093	1.0093	1.0897	1.0867	1.0096	1.0888	1.0868	1.0562
0.4	0	1.0816	1.0827	1.0828	1.0047	1.0900	1.0054	1.0060	1.0824	0.9998	1.0894	1.0525

Representasi hasil pengujian parameter kombinasi *cr* dan *mr* hibridisasi GA-SA ditunjukkan pada Gambar 6.9.



Gambar 6.9 Grafik Pengujian Parameter Kombinasi *cr* dan *mr* GA-SA

Berdasarkan Gambar 6.9 nilai rata-rata *fitness* tertinggi didapatkan pada kombinasi *cr* dan *mr* masing-masing sebesar 0.3 dan 0.1 dengan rata-rata *fitness* sebesar 1.0636. Hasil grafik nilai rata-rata fitness yan diperoleh pada pengujian parameter *cr* dan *mr* menghasilkan grafik yang naik turun, hal tersebut dipengaruhi oleh penggunaan algoritme SA yang bersifat acak. Hasil keluaran salah satu kromosom pada pengujian parameter kombinasi *cr* dan *mr* hibridisasi GA-SA masing-masing sebesar 0.15 dan 0.25 yaitu dengan urutan indeks 17, 9, 12, 13, 14, 4, 7, 5, 19, 10, 6, 3, 2, 11, 1, 18, 15, 8, 16 yang menghasilkan nilai *fitness* sebesar 1.0900 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata fitness yang diperoleh pada pengujian parameter banyak generasi hibridisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata *fitness* pengujian kombinasi cr dan mr pada algoritme genetika yang hanya mencapai 1,0233.

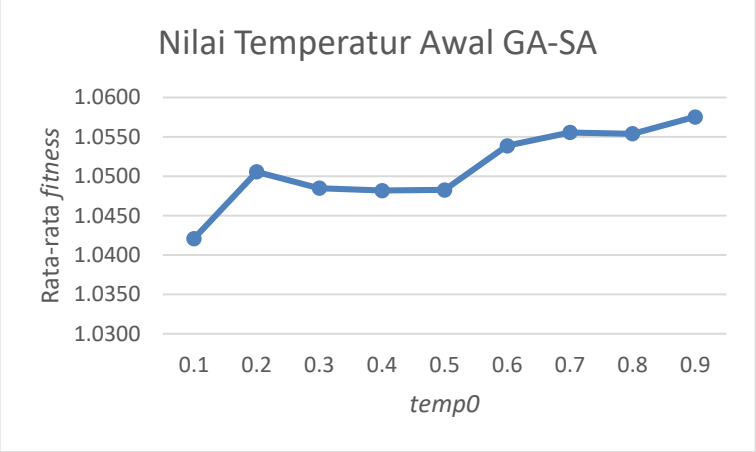
6.3.4 Pengujian Parameter Temperatur Awal GA-SA

Pengujian parameter nilai temperatur awal (*temp0*) pada hibridisasi GA-SA dilakukan dengan menggunakan parameter ukuran populasi sebesar 400, banyak generasi sebesar 800, *cr* sebesar 0.3, dan *mr* sebesar 0.1, *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200. Parameter pengujian ukuran populasi didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.5 sampai dengan Tabel 6.9. Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter nilai *temp0* mulai dari 0.1 hingga 0.9 dengan kelipatan 0.1. Hasil pengujian parameter nilai *temp0* dapat dilihat pada Tabel 6.10.

Tabel 6.10 Tabel Pengujian Parameter Temperatur Awal GA-SA

temp0	Nilai *fitness*										Rata-rata *fitness*
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
0.1	1.0064	1.0085	1.0868	1.0093	1.0868	1.0287	1.0894	1.0894	1.0093	1.0065	1.0421
0.2	1.0096	1.0093	1.0868	1.0864	1.0867	1.0900	1.0101	1.0287	1.0880	1.0101	1.0506
0.3	1.0888	1.0093	1.0101	1.0096	1.0888	1.0867	1.0894	1.0093	1.0867	1.0064	1.0485
0.4	1.0868	1.0054	1.0064	1.0096	1.0894	1.0894	1.0054	1.0900	1.0096	1.0900	1.0482
0.5	1.0894	1.0888	1.0096	1.0900	1.0053	1.0101	1.0867	1.0868	1.0096	1.0064	1.0483
0.6	1.0054	1.0868	1.0867	1.0724	1.0884	1.0894	1.0060	1.0052	1.0101	1.0884	1.0539
0.7	1.0101	1.0867	1.0096	1.0868	1.0867	1.0096	1.0888	1.0812	1.0868	1.0094	1.0556
0.8	1.0812	1.0894	1.0894	1.0101	1.0085	1.0888	1.0054	1.0867	1.0049	1.0897	1.0554
0.9	**1.0287**	**1.0888**	**1.0829**	**1.0867**	**1.0868**	**1.0894**	**1.0093**	**1.0883**	**1.0093**	**1.0053**	**1.0575**

Representasi hasil pengujian parameter nilai *temp0* hibridisasi GA-SA ditunjukkan pada Gambar 6.10.



Gambar 6.10 Grafik Pengujian Parameter Temperatur Awal GA-SA

Berdasarkan Gambar 6.10 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *temp0* sebesar 0.9 dengan rata-rata *fitness* sebesar 1.0575. Hasil keluaran salah satu kromosom pada pengujian parameter temperatur awal hibridisasi GA-SA sebesar 0.9 yaitu dengan urutan indeks wisata 6, 11, 17, 13, 14, 4, 12, 9, 5, 7, 3, 15, 1, 19, 10, 16, 18, 2, 8 yang menghasilkan nilai *fitness* sebesar 1,1332 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata *fitness* yang diperoleh pada pengujian parameter nilai *temp0* hibridisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata *fitness* pengujian nilai temp0 pada algoritme simulated annealing yang hanya mencapai 0.9860.

6.3.5 Pengujian Parameter Faktor Pendinginan GA-SA

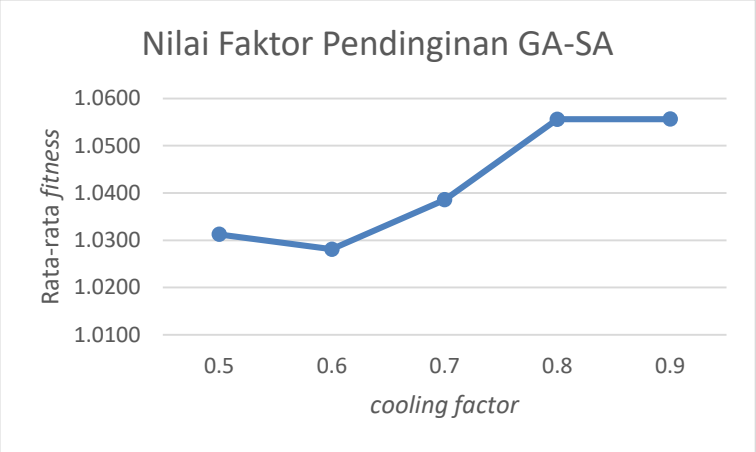
Pengujian parameter nilai factor pendinginan (*cooling factor*) pada hibridisasi GA-SA dilakukan dengan menggunakan parameter ukuran populasi sebesar 400, banyak generasi sebesar 800, *cr* sebesar 0.3, dan *mr* sebesar 0.1, *temp0* sebesar 0.9, dan nilai *k* sebesar 200. Parameter pengujian ukuran populasi didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.6 sampai dengan Tabel 6.10. Pengujian dilakukan sebanyak sepuluh kali dengan nilai parameter nilai

cooling factor mulai dari 0.5 hingga 0.9 dengan kelipatan 0.1 (Novitasari & Mahmudy, 2016). Hasil pengujian parameter nilai *cooling factor* dapat dilihat pada Tabel 6.11.

Tabel 6.11 Tabel Pengujian Parameter Faktor Pendinginan GA-SA

cooling factor	Nilai *fitness* Iterasi ke-										Rata-rata *fitness*
	1	2	3	4	5	6	7	8	9	10	
0.5	1.0101	1.0057	1.0064	1.0829	1.0094	1.0829	1.0093	1.0093	1.0101	1.0864	1.0312
0.6	1.0052	1.0050	1.0093	1.0093	1.0699	1.0888	1.0052	1.0093	1.0064	1.0724	1.0281
0.7	1.0054	1.0900	1.0085	1.0724	1.0894	1.0074	1.0094	1.0085	1.0054	1.0894	1.0386
0.8	1.0868	1.0888	1.0093	1.0884	1.0094	1.0812	1.0096	1.0880	1.0891	1.0054	1.0556
0.9	**1.0096**	**1.0867**	**1.0101**	**1.0888**	**1.0868**	**1.0868**	**1.0810**	**1.0868**	**1.0101**	**1.0094**	**1.0556**

Representasi hasil pengujian parameter nilai *cooling factor* hibridisasi GA-SA ditunjukkan pada Gambar 6.11.



Gambar 6.11 Grafik Pengujian Parameter Faktor Pendinginan GA-SA

Berdasarkan Gambar 6.11 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *cooling factor* sebesar 0.9 dengan rata-rata fitness sebesar 1.0556. Hasil keluaran salah satu kromosom pada pengujian parameter faktor pendinginan hibridisasi GA-SA sebesar 0.9 yaitu dengan urutan indeks wisata 6, 11, 17, 5, 7, 4, 12, 9, 10, 19, 8, 13, 3, 1, 16, 18, 14, 2, 15 yang menghasilkan nilai *fitness* sebesar 1.0867 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata *fitness* yang diperoleh pada pengujian parameter nilai *cooling factor* hibridisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata fitness pengujian nilai *cooling factor* pada algoritme *simulated annealing* yang hanya mencapai 0.9827.

6.3.6 Pengujian Nilai Koefisien Probabilitas GA-SA

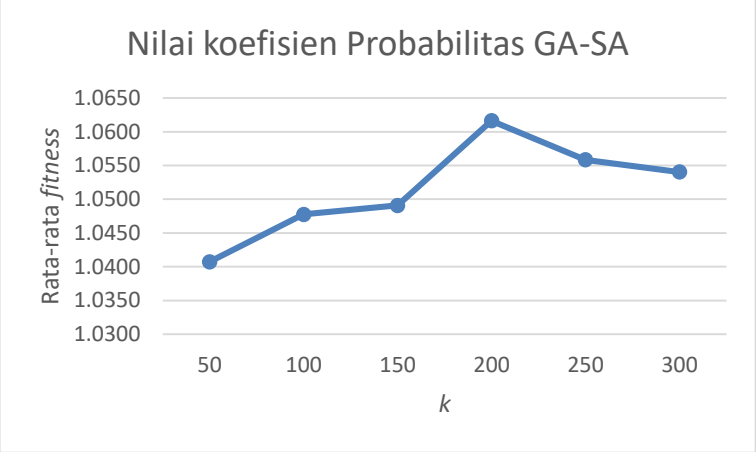
Pengujian nilai koefisien probabilitas penerimaan solusi baru pada algoritme *simulated annealing* dilakukan dengan menggunakan parameter ukuran populasi sebesar 400, banyak generasi sebesar 800, *cr* sebesar 0.3, dan *mr* sebesar 0.1, nilai *temp0* sebesar 0.9, dan nilai *cooling factor* sebesar 0.9. Parameter

pengujian ukuran populasi didapat dari pengujian parameter algoritme genetika dan algoritme *simulated annealing* sebelumnya yang dapat dilihat pada Tabel 6.6 sampai dengan Tabel 6.10. Pengujian dilakukan sebanyak sepuluh kali dengan nilai nilai koefisien probabilitas penerimaan solusi baru mulai dari 50 hingga 300 dengan kelipatan 50. Hasil pengujian nilai koefisien probabilitas penerimaan solusi baru dapat dilihat pada Tabel 6.12.

Tabel 6.12 Tabel Pengujian Nilai Koefisien Probabilitas GA-SA

k	**Nilai *fitness***										**Rata-rata *fitness***
	Iterasi ke-										
	1	2	3	4	5	6	7	8	9	10	
50	1.0074	1.0094	1.0093	1.0897	1.0085	1.0897	1.0085	1.0094	1.0888	1.0867	1.0407
100	1.0888	1.0096	1.0094	1.0868	1.0054	1.0074	1.0897	1.0812	1.0900	1.0093	1.0478
150	1.0894	1.0864	1.0094	1.0897	1.0093	1.0096	1.0884	1.0096	1.0894	1.0096	1.0491
200	**1.0900**	**1.0094**	**1.0894**	**1.0826**	**1.0888**	**1.0101**	**1.0868**	**1.0054**	**1.0724**	**1.0812**	**1.0616**
250	1.0891	1.0064	1.0900	1.0894	1.0085	1.0868	1.0093	1.0868	1.0868	1.0054	1.0558
300	1.0085	1.0868	1.0054	1.0891	1.0096	1.0897	1.0724	1.0868	1.0868	1.0052	1.0540

Representasi hasil pengujian nilai koeisien probabilitas penerimaan solusi baru algoritme *simulated annealing* ditunjukkan pada Gambar 6.12.



Gambar 6.12 Grafik Pengujian Nilai Koefisien Probabilitas GA-SA

Berdasarkan Gambar 6.12 nilai rata-rata *fitness* tertinggi didapatkan pada nilai *k* sebesar 200 dengan rata-rata *fitness* sebesar 1.0616. Hasil keluaran salah satu kromosom pada pengujian parameter nilai koefisien probabilitas hibridisasi GA-SA sebesar 200 yaitu dengan urutan indeks wisata 5, 19, 10, 13, 14, 4, 7, 12, 9, 17, 16, 6, 11, 2, 18, 3, 15, 8, 1 yang menghasilkan nilai *fitness* sebesar 1.0900 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Nilai rata-rata *fitness* yang diperoleh pada pengujian parameter nilai *k* hibridisasi GA-SA menghasilkan nilai yang lebih baik daripada nilai rata-rata fitness pengujian nilai *k* pada algoritme *simulated annealing* yang hanya mencapai 0.9927.

Dari keenam parameter yang digunakan pada pengujian hibridisasi GA-SA pada permasalahan VRPTW, didapat parameter mendekati optimal pada ukuran populasi 400, banyak generasi 800, kombinasi *cr* dan *mr* sebesar 0.3 dan 0.1, nilai *temp0* sebesar 0.9, nilai *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200 dengan nilai rata-rata *fitness* sebesar 1.0616.

6.4 Pengujian Perbandingan Algoritme

Pengujian perbandingan algoritme pada hibridisasi GA-SA dilakukan dengan membandingkan hasil rata-rata fitness tertinggi dari masing-masing pengujian algoritme genetika, algoritme simulated annealing dan hibridisasi GA-SA pada permasalahan *Multi-trip* VRPTW dengan waktu komputasi yang sama. Waktu komputasi ditetapkan selama 60 detik pada masing-masing *running* algoritme. Hasil tersebut diperoleh dari rata-rata waktu komputasi terlama dari ketiga algoritme tersebut yaitu waktu komputasi pada algoritme GA-SA. Parameter yang digunakan pada pengujian pebandingan algoritme diperoleh dari hasil pengujian parameter algoritme genetika pada subbab 6.1, hasil pengujian parameter algoritme *simulated annealing* pada subbab 6.2, dan hasil pengujian parameter hibridisasi GA-SA pada subbab 6.3. Pengujian dilakukan sebanyak 10 kali pada masing-masing algoritme. Hasil pengujian perbandingan algoritme dapat dilihat pada Tabel 6.13.

Tabel 6.13 Tabel Pengujian Perbandingan Algoritme

Iterasi Ke-	Algoritme		
	GA	SA	GA-SA
1	1.0900	1.0816	1.0884
2	1.0101	1.0876	1.0868
3	1.0888	1.0816	1.0900
4	0.9982	1.0867	1.0894
5	1.0060	1.0868	1.0868
6	1.0094	1.0812	1.0894
7	1.0883	1.0868	1.0894
8	1.0054	1.0064	1.0900
9	1.0891	1.0868	1.0897
10	0.9984	1.0867	1.0884
Rata-rata Fitness	**1.0384**	**1.0772**	**1.0888**

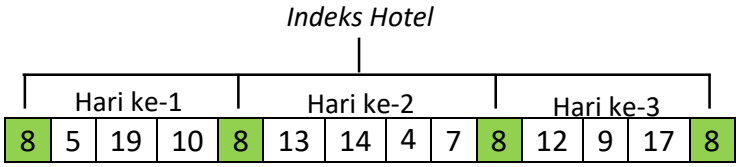
Berdasarkan Tabel 6.13 penggunaan hibridisasi GA-SA pada optimasi permasalahan *Multi-trip* VRPTW dapat menghasilkan nilai rata-rata fitness yang lebih baik daripada penggunaan GA dan algoritme SA secara terpisah. Pada hibridisasi GA-SA nilai rata-rata *fitness* mencapai 1,0888 lebih tinggi daripada nilai rata-rata *fitness* algoritme SA sebesar 1.0772 dan nilai rata-rata *fitness* GA yang hanya mencapai 1.0382.

Pada pengujian perbandingan algoritme, hasil *fitness* algoritme SA cenderung lebih besar dari hasil pengujian parameter algoritme SA sebelumnya, bahkan nilai rata-rata *fitness* algorima SA lebih besar dari pada nilai rata-rata *fitness* GA, hal ini dikarenakan waktu komputasi pada SA yang lebih lama akan membuat perulangan pencarian solusi tetangga menjadi lebih banyak sehingga kesempatan menemukan solusi yang lebih baik menjadi lebih besar. Namun semakin banyak perulangan yang dilakukan oleh algoritme SA akan menyebabkan

temperatur semakin rendah, sedangkan temperatur yang rendah akan menyebabkan kromosom dengan solusi yang lebih buruk akan memiliki probabilitas yang semakin kecil untuk diterima sebagai solusi baru, hal ini akan mengakibatkan algoritme SA terjebak pada keadaan lokal optimal pada keadaan tertentu.

6.5 Solusi Rute menggunakan Parameter Terbaik

Berdasarkan hasil pengujian menggunakan parameter terbaik hibridisasi GA-SA yaitu pada ukuran populasi 400, banyak generasi 800, kombinasi *cr* dan *mr* sebesar 0.3 dan 0.1, nilai *temp0* sebesar 0.9, nilai *cooling factor* sebesar 0.9, dan nilai *k* sebesar 200, didapatkan salah satu solusi rute mendekati optimal dengan nilai fitness sebesar 1.0900 dengan urutan indeks wisata 5, 19, 10, 13, 14, 4, 7, 12, 9, 17, 16, 6, 11, 2, 18, 3, 15, 8, 1 dan jumlah wisata yang dapat dikunjungi dalam 3 hari berjumlah 10 objek wisata. Urutan kunjungan wisata dari solusi tersebut dapat dilihat pada Gambar 6.13.



Gambar 6.13 Solusi rute menggunakan Parameter Terbaik

Berikut Penjelasan rute dari Gambar 6.13:

a. Rute hari pertama: 8-5-19-10-8

   Hotel Manyar → Pantai Bangsring → Alas Purwo → Pantai Pancur → Hotel Manyar

b. Rute hari kedua: 8-13-14-4-7-8

   Hotel Manyar → Kawah Ijen → Air Terjun Jagir → Pantai Boom → Pantai Watu Dodol → Hotel Manyar

c. Rute hari ketiga: 8-12-9-17-8

   Hotel Manyar → Pantai Cemara → Pantai Blimbing Sari → Air Terun Kali Bendo → Hotel Manyar

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dibahas pada bab sebelumnya maka dapat disimpulkan beberapa hal sebagai berikut:

1. Hibridisasi algoritme genetika dan *simulated annealing* mampu menyelesaikan permasalahan *Multi-trip* VRPTW lebih optimal dari pada penggunaan algoritme genetika dan *simulated annealing* secara terpisah.
2. Hibridisasi algoritme genetika dan *simulated annealing* mampu memberikan fitness mendekati optimal dengan ukuran populasi sebesar 400, banyak generasi sebesar 800, kombinasi *cr* dan *mr* sebesar 0.3 dan 0.1, temperatur awal sebesar 0.9, faktor pendinginan sebesar 0.9, dan koefisien probabilitas penerimaan solusi baru sebesar 200.
3. Penggunaan hibridisasi GA-SA pada optimasi permasalahan *Multi-trip* VRPTW dapat menghasilkan nilai rata-rata *fitness* yang lebih baik daripada penggunaan GA dan algoritme SA secara terpisah. Pada hibridisasi GA-SA nilai rata-rata *fitness* mencapai 1.0888 lebih tinggi daripada nilai rata-rata *fitness* algoritme SA sebesar 1,0772 dan nilai rata-rata *fitness* GA yang hanya mencapai 1,0384.

7.2 Saran

Berdasarkan hasil pengujian dan analisis pada bab sebelumnya didapatkan beberapa saran yang dapat dipertimbangkan untuk pengembangan penelitian selanjutnya, diantaranya:

1. Penggunaan probabilitas pada algoritme *simulated annealing* membuat solusi yang dihasilkan menjadi kurang stabil, oleh karena itu diperlukan penelitian lebih lanjut mengenai perhitungan probabilitas yang sesuai dengan kasus ini agar solusi yang dihasilkan tidak terlalu acak.
2. Hibridisasi algoritme genetika dan *simulated annealing* pada permasalahan *multi-trip* VRPTW diharapkan dapat dikembangkan menjadi produk aplikasi rekomendasi perjalanan wisata yang disertai pula dengan peta rute rekomendasi wisata agar manfaat dari penelitian ini dapat dirasakan langsung oleh para wisatawan khususnya yang akan berkunjung ke Kabupaten Banyuwangi.

DAFTAR PUSTAKA

Alabsi, F. & Naoum, R. 2012. “Fitness Function for Genetic Algorithm used in Intrusion Detection System”. International Journal of Applied Science and Technology, Vol. 2, No. 4, pp. 129-134.

Anggodo, Y. P., Ariyani, A. K., Ardi, M. K., Mahmudy, W. F. 2016. “Optimization Of Multi-Trip Vehicle Routing Problem With Time Windows Using Genetic Algorithm”. Journal of Enviromental Engineering and Sustainable Technology, Vol. 3, No. 2, pp 92-97.

Azadeh, A., Elahi, S., Farahani, M. H., & Nasirian, B. 2017. “A Genetic Algorithm-Taguchi Based Approach to Inventory Routing Problem of A Single Perishable Product with Transshipment”. Computers & Industrial Engineering 104, pp. 124-133.

Bahri, O. “Multi-Objective Optimiation of Vehicle Routing Problem with uncertainty: An Approach Based on Possibility Theory”. 2012. Tersedia di: < http://www.larodec.com/download/file/fid/543> [Diakses 25 Januari 2017]

Chagas, J. B. C., Silviera, U. E. F., Benedito, M. P. L., & Santos, A. G. 2016. “Simulated Annealing Metaheuristic for the Double Vehicle Routing Problem with Multiple Stacks”. International Conference on Intelligent Transportation Systems, pp. 1311-1316.

Chunhua, Tang. 2010. “An Improving Genetic Algorithm for Vehicle Routing Problem with Time Windows”. International Conference on Intelligent Computation Technology and Automation, pp. 603-606.

Fenghe, J. & Yaping, F. 2010. “Hybrid Genetic Agorithm for Vehicle Routing Problem with Time Windows”. International Conference on Natural Computation, Vol. 4, pp. 502-506.

Kale, Dr. V. S., Agarwal, M., Kesarkar, P. D., Regmi, D. R., Chaudhary, A., & Killawala, C. 2014. “Optimal Coordination of Overcurrent Relays Using Genetic Algorithm and Simulated Annealing”. International Conference on Control, Instrumentation, Energy & Communication, pp. 361-365.

Kickpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983. “Optimization by Simulated Annealing”. Science, New Series, Vol. 220, No. 4598, pp. 671-680.

Lin, S. W., Ying, K. C., Lee, Z. J., & Chen, H. S. 2006. “Vehicle Routing Problems with Time Windows Using Simulated Annealing”. Systems, Man, and Cybernetics, pp. 645-650.

Mahmudy, W. F. 2014. “Improved Simulated Annealing for Optimization of Vehicle Routing Problem with Time Windows (VRPTW)”. Kursor Journal, Vol. 7, No. 3, pp. 109-116.

Mahmudy, W. F. 2015. “Modul Kuliah Semester Ganjil 2015-2016: Dasar-dasar Algoritme Evolusi”. Universitas Brawijaya.

Mahmudy, W. F., Marian, R. M., & Luong, L. H. S. 2012a. “Solving Part Type Selection and Loading Problem in Flexible Manufacturing System Using Real Coded Genetic Algorithms – Part I: Modeling”. International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering, Vol. 6, No. 9, pp. 1922-1928.

Mahmudy, W. F., Marian, R. M., & Luong, L. H. S. 2012b. “Solving Part Type Selection and Loading Problem in Flexible Manufacturing System Using Real Coded Genetic Algorithms – Part II: Modeling”. International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering, Vol. 6, No. 9, pp. 1922-1928.

Mahmudy, W. F., Marian, R. M., & Luong, L. H. S. 2013a. “Optimization of Part Type Selection and Loading Problem with Alternative Production Plans in Flexible Manufacturing System using Hybrid Genetic Algorithms – Part 1: Modelling and Representation”. International Conference on Knowledge and Smart Technology, pp. 75-80.

Mahmudy, W. F., Marian, R. M., & Luong, L. H. S. 2013b. “Optimization of Part Type Selection and Loading Problem with Alternative Production Plans in Flexible Manufacturing System using Hybrid Genetic Algorithms – Part 2: Genetic Operators and Results”. International Conference on Knowledge and Smart Technology, pp. 81-85.

Malhotra, R., Singh, N., & Singh, Y. 2011. “Genetic Algorithms: Concepts, Design for Optimization of Process Controllers”. Computer and Information Science, Vol. 4, No. 2, pp. 39-54.

Minocha, B., Tripathi, S., & Mohan, C. 2011. “Solving Vehicle Routing and Scheduling Problems using Hybrid Genetic Algorithm”. International Conference on Electronics Computer Technology, Vol. 2, pp. 189-193.

Nikjoo, R. S., Sadeghi, S. H. H., & Moini, R. 2010. “A Hybrid Genetic- Simulated Annealing Algorithm for Optimal Design of Grounding Grids with Rods”. Asia-Pasific Power and Enegry Engineering Conference, pp. 1-4.

Novitasari, D., Cholisodin, I., & Mahmudy, W.F. 2016. “Hybridizing PSO with SA for Optimizing SVR Applied to Software Effort Estimation”. TELKOMNIKA, Vol. 14, No.1, pp. 245-253.

Nugraha, D. C. A. & Mahmudy, W. F. 2015. “Optimasi vehicle routing problem with time windows pada distribusi katering menggunakan algoritme genetika”. Seminar Nasional Sistem Informasi Indonesia (SESINDO), Institut Teknologi Sepuluh Nopember (ITS), Surabaya, 2-3 November, pp. 275-282.

Pitaloka, D. A., Mahmudy, W. F., & Sutrisno. 2014. “Penyelesaian vehicle routing problem with time windows (VRPTW) menggunakan algoritme genetika hybrid”. Journal of Environmental Engineering & Sustainable Technology, Vol. 1, No. 2, pp. 103-109.

Priandani, N. D. & Mahmudy, W. F. 2015. "Optimasi travelling salesman problem with time windows (TSP-TW) pada penjadwalan paket rute wisata di pulau Bali menggunakan algoritme genetika". Seminar Nasional Sistem Informasi Indonesia (SESINDO), Institut Teknologi Sepuluh Nopember (ITS), Surabaya, 2-3 November, pp. 259-266.

Shi, M., Lin, Q., Zhang, Q., & Zhao, H. 2011. "Hybrid Genetic Tabu Search Simulated Annealing Algorithm and its Application in Vehicle Routing Problem with Time Windows". International Conference on Artificial Intelegence, Management Science and Electronic Commerse, pp. 4022-4025.

Soebagyo. 2012. "Strategi Pengembangan Pariwisata di Indonesia". Jurnal Liquidity, Vol. 1, No.2, pp. 153-158.

Tung, K. T., Chen, C. Y., & Hung, Y. F. 2016. "Solving Cutting Scheduling Problem by Simulated Annealing Search Method". International Conference on Industrial Engineering and Engineering Management, pp. 907-911.

Wassan, N. A. & Nagy, Gabor. 2014. "Vehicle Routing Problem with Deliveries and Pickups: Modelling Issues and Meta-heuristics Solution Approaches". International Journal of Transportation, Vol. 2, No.1, pp. 95-110.

Wassan, N., Wassan, N., Nagy, G., & Salhi, S. 2016. "The Multiple Trip Vehicle Routing Problem with Backhauls: Formulation and a Two-Level Variable Neighborhood Search". Computer & Operations Research 78, pp. 454-467.

Wijayaningrum, V. N. & Mahmudy, W. F. 2016. "Optimization of Ship's Route Scheduling Using Genetic Algorithm". Indonesian Journal of Electrical Engineering and Computer Science, Vol. 2, No. 1, pp. 180-186.

Xu, H., Fan, W., Wei, T., & Yu, L. 2008. "An Or-opt NSGA-II Algorithm for Multi-Objective Vehicle Routing Problem with Time Windows". Automation Science and Engineering, pp. 309-314.

Xu, P., Sui, S., & Du, Z. 2015. "Application of Hybrid Genetic Algorithm Based on Simulated Annealing in Function Optimization". International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering, Vol. 9, No. 11, pp. 685-688.

Yang, Y. & Tang, L. 2010. "A Filter-and-fan Approach to the Multi-trip Vehicle Routing Problem". International Conference on Logistics Systems and Intelligent Management, Vol. 3, pp. 1713-1717.

Zhang, Y. Y., Huang, Q., Gao, Fan., & Sun, X. Y. 2010. "Optimal Reservoir Operation Using a Hybrid Simulated Annealing Algorithm-Genetic Algorithm". International Conference on Bio-Inspired Computing: Theories and Applications, pp. 454-458.