Analisa Performa Hybrid Ant Colony Optimization dalam Memecahkan Vehicle Routing Problem with Time Windows.

Timothy Handi Wibawa[1], Rolly Intan[2]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) - 8417658
Email: timothyhw33@gmail.com[1],rintan@petra.ac.id[2]

ABSTRAK

Di zaman modern ini, manusia sudah banyak dibantu oleh perkembangan teknologi. Salah satu bidang tersebut adalah bidang pengiriman barang, yang dinamakan Vehicle Routing Problem (VRP). Vehicle Routing Problem adalah masalah dimana terdapat sebuah armada pengiriman barang yang harus mengirimkan sejumlah barang kepada berbagai pelanggan. Solusi yang dicari adalah rute paling pendek yang bisa dilakukan armada tersebut.

Hybrid Ant Colony Optimization (HACO) adalah algoritma yang meniru cara semut mencari makanan, yang merupakan perkembangan dari *Ant Colony Optimization*. Algoritma tersebut dipublikasikan oleh Q. Ding et. al pada tahun 2012 untuk menyelesaikan *Vehicle Routing Problem with Time Windows*(VRPTW). Namun dari penelitian yang dipublikasikan tersebut, terdapat beberapa ketidakjelasan seperti waktu proses dan memori yang dipakai.

Oleh karena itu dalam jurnal ini ada pengkajian ulang untuk menjelaskan performa dari algoritma HACO dalam memecahkan VRPTW, yang akan dilakukan dengan unity. Didalam skripsi ini juga dibahas beberapa pengembangan dalam HACO yang diharapkan dapat memperbaiki dan meningkatkan hasil dari HACO.

Kata Kunci: Ant Colony Optimization, Vehicle Routing Problem with Time Windows, C#, Unity

ABSTRACT

In modern times, humans have been helped by many technological developments. One such field is the goods delivery field, called the Vehicle Routing Problem (VRP). Vehicle Routing Problem is a matter of where there is a freight fleet that has to deliver a certain amount of goods to various customers. The solution sought is the shortest route the fleet can take.

,Hybrid Ant Colony Optimization (HACO)is an algorithm that is inspired from the foraging behaviour of ant species, that is one of the result of development for Ant Colony Optimization. HACO has been published by Q. Ding et. al in 2012 to solve Vehicle Routing Problem with Time Windows. However, in that published journal, there were vagueness in some points like process time and memory usage.

Therefore in this journal there is a need to review the performance of HACO to solve VRPTW, which will be done using unity. In this thesis is also discussed some development of HACO which are expected to fix and improve the result of HACO..

Keywords: *Ant Colony Optimization, Vehicle Routing Problem with Time Windows, C#, Unity*

1. PENDAHULUAN

Di jaman modern ini, teknologi semakin berkembang dimana manusia semakin banyak dibantu dalam menjalankan berbagai aktivitas. Salah satu bidang yang sudah terbantu oleh teknologi jaman modern adalah masalah pengiriman barang, dimana terdapat program yang dapat membantu pemilihan rute dalam mengirimkan barang. Vehicle Routing Problem (VRP) adalah masalah dimana terdapat sebuah armada pengiriman barang yang harus mengirimkan barang kepada berbagai pelanggan. Solusi yang dicari adalah rute paling pendek yang bisa dilakukan oleh armada pengiriman tersebut. Seiring berjalannya waktu, agar semakin mirip dengan permasalahan riil, beberapa perkembangan dari VRP muncul, seperti *multi-compartment*[7], *Multi-objective*[6], *heterogenous vehicle*[10],Vehicle Routing Problem with Time Windows (VRPTW) muncul sebagai perkembangan dari Vehicle Routing Problem. VRPTW menambahkan pertimbangan waktu service dan jangkauan waktu customer dari setiap pengiriman barang.

Ant Colony Optimization (ACO) sebagai salah satu bagian dari Swarm Intelligence Algorithm (SIA) adalah sebuah algoritma yang meniru koloni semut dalam mencari makanan. Ketika semut melewati suatu lokasi, semut akan mengeluarkan suatu feromon, yang dapat menarik semut yang lain untuk melewati daerah tersebut. Jika jalur tersebut semakin populer, maka feromon yang ada juga akan semakin pekat, sehingga membutuhkan waktu yang lebih lama untuk pudar. Dengan demikian, kepekatan feromon dipakai oleh semut untuk mencari jalur yang terpopuler yang dapat diasumsikan sebagai jalur yang terbaik bagi semut.

Penelitian tentang Ant Colony Optimization untuk ACO sudah banyak dilakukan. Pada tahun 2004, penelitian mengenai penggunaan ACO dengan penambahan candidate list dan 2-opt heuristic untuk menyelesaikan VRP telah diajukan dan dipublikasikan oleh Bell dan McMullen [1]. Kemudian pada tahun 2007, ACO berkembang menjadi suatu algoritma yang dikenal dengan nama Improved Ant Colony Optimization (IACO). IACO melakukan permutasian beberapa semut yang ada sehingga membuat hasil semakin teracak dan tidak terjebak pada local solution yang mungkin bukan solusi yang optimal[2]. Terdapat juga penelitian menggunakan multiple ant colony[9] dan juga hybrid method yang menggabungkan ant colony dengan algoritma yang lain[4].

Setelah itu, Hybridized Ant Colony Optimization (HACO) yang merupakan dasar dari penelitian ini dikembangkan melalui penggabungan beberapa algoritma yang telah ada untuk mendapatkan hasil yang lebih optimal [3]. Beberapa Ant Colony yang dipakai dalam penggabungan ini adalah IACO dan Max-Min Ant System[8]. Dalam penelitian yang sudah dilakukan, hasil kerja dari HACO dibandingkan dengan beberapa algoritma yang lain dengan tolok ukur jarak yang dilalui dan jumlah kendaraan yang dipakai. Dari hasil perbandingan tersebut, HACO memiliki jarak yang terpendek dan jumlah kendaraan yang paling sedikit. Namun para peneliti tidak mencantumkan kecepatan proses eksekusi dan jumlah memory yang dipakai untuk memproses HACO. Maka dari itu perlu ada penelitian lebih lanjut yang dilakukan untuk membuktikan bahwa HACO adalah metode paling optimal dalam memecahkan masalah ini jika dilihat pada kecepatan proses dan jumlah memori yang dipakai dalam proses.

Pada jurnal ini akan dilakukan penelitian ulang mengenai cara kerja dari HACO, lalu melakukan penelitian lanjut dengan mencari tahu waktu proses eksekusi dan memory space yang digunakan untuk memecahkan problem yang ada. Selain itu, penelitian ini juga menjajaki kemungkinan untuk melakukan pengembangan metode HACO agar dapat lebih meningkatkan performanya.

2. LANDASAN TEORI

2.1 Hybrid Ant Colony Optimization

Dalam jurnal karya Ding et al.(2012) dijelaskan bahwa algoritma HACO adalah algoritma ACO yang telah dimodifikasi sedemikian rupa dengan menggabungkan beberapa metode yang pernah digunakan untuk mengembangkan ACO. Berikut adalah beberapa pengembangan yang dilakukan oleh HACO.

2.1.1 Pendekatan Pengaturan Feromon.

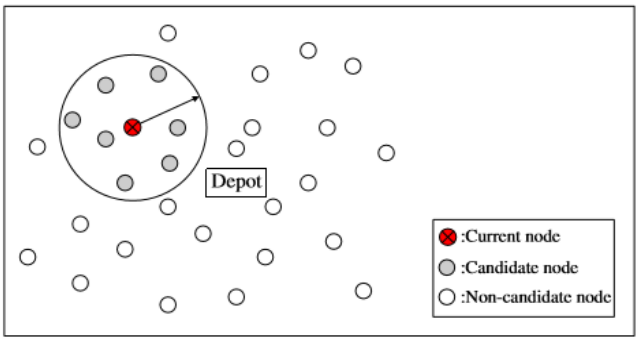
Dari proses ACO, dapat dilihat bahwa proses penggunaan feromon memiliki dampak yang besar dalam menemukan hasil pencarian. Oleh sebab itu, terdapat beberapa penyesuaian dalam fungsi yang mengatur feromon untuk menjaga proses HACO agar tidak terjebak di *local optimal solution*. Pertama, dalam proses pemilihan rute, ada peluang untuk memilih jalur tidak berdasarkan seleksi mutlak menggunakan rumus, namun juga menggunakan seleksi *random*. Kedua, memajukan batas nilai maksimum dan minimum dari value jejak feromon. Ketiga, jejak feromon akan diinisialisasikan sama seperti nilai maksimum feromon. Keempat, mengubah rumus pemudaran jejak feromon menjadi dinamis dan tidak mutlak.

2.1.2 Penggunaan Disaster Operator.

Agar proses HACO tidak terjebak pada local optimal solution, maka metode *disaster operator* digunakan. Disaster operator adalah metode yang mengubah value feromon pada beberapa bagian dari *local optimization route* menjadi minimum. Metode ini bekerja mirip seperti proses mutasi, dilakukan dengan peluang kecil yang random, karena jika peluang terlalu besar, jalur feromon akan hancur karena terlalu banyak disaster yang terjadi.

2.1.3 Strategi Candidate List.

Dalam Proses ACO, proses pencarian rute memerlukan waktu yang cukup lama karena dalam memilih rute dari *node* i ke j , program harus menghitung probabilitas semua *node* yang belum pernah dikunjungi. Oleh karena itu, digunakan *candidate list* yang mengelompokkan beberapa node terdekat dari i agar program tidak perlu menghitung semua node yang tersedia. Lihat Gambar 1.



Gambar 1. Candidate List

2.1.4 Penambahan Saving Algorithm

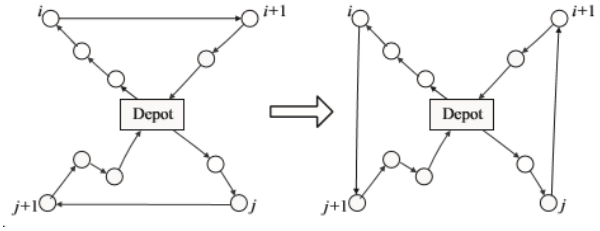
Saving algorithm adalah cara yang simpel dan efisien dalam memilih rute. Algoritma ini pertama memisahkan setiap customer menjadi rute yang berbeda, lalu menghitung nilai yang didapat (*saving value*) jika menggabungkan 2 dari rute-rute tersebut. Cara menghitung *saving value* tersebut ada pada rumus 1, yaitu:

$$S_{ij} = d_{0i} + d_{0j} - d_{ij} \quad (1)$$

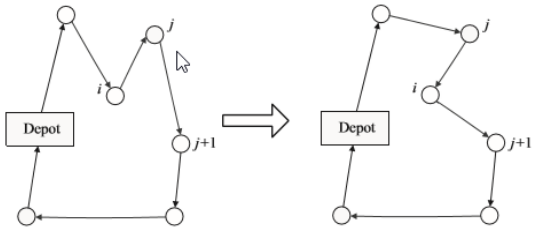
Dimana S adalah *saving value* dari node i menuju node j, d menandakan jarak dari 1 node ke node yang lain. 0 adalah depot, i dan j adalah node customer.

2.1.5 λ – interchange mechanism

λ – interchange mechanism adalah cara yang diajukan oleh Osman [9] untuk mempercepat proses HACO. Cara tersebut adalah penggunaan 2-*interchange* dan *or-interchange mechanism* untuk mengurangi waktu proses HACO dan membantu mencari hasil yang optimal. Dasar dari *or-interchange* adalah dengan menukar *node* pada 1 jalur yang sama pada *solution* yang ada jika memenuhi syarat dari vrptw, jika nilai yang dihasilkan lebih bagus, maka jalur akan diambil. Sedangkan or-*interchange* adalah proses menukar 2 *node* dari 2 jalur yang berbeda pada *solution* yang ada jika memenuhi syarat dari vrptw, jika nilai yang dihasilkan lebih bagus, maka jalur akan diambil. Lihat Gambar 2 dan 3.



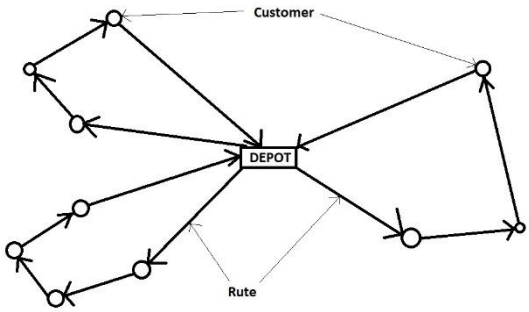
Gambar 2. 2-*interchange*



Gambar 3. Or-interchange

2.2 Vehicle Routing Problem with Time Windows

Vehicle routing problem adalah masalah yang mencari rute optimal dari sejumlah kendaraan dengan kapasitas tertentu untuk mengunjungi pelanggan yang memiliki permintaan masing masing. Rute tersebut harus dimulai dan diakhiri di suatu tempat yang bernama *depot*. Semua pelanggan dikunjungi sekali dan total kapasitas kendaraan harus diperhitungkan dalam mengunjungi pelanggan tersebut. Lihat gambar 4.



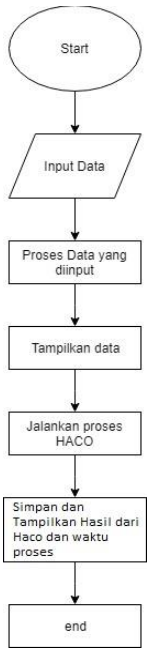
Gambar 4. Contoh penyelesaian VRP 3 rute

Sedangkan VRPTW adalah sebuah VRP yang juga memperhitungkan faktor waktu dari proses pengirimaan barang. Setiap kendaraan harus menyuplai barang ke setiap pelanggan yang masing-masing memiliki jangka waktu tertentu. Proses pengiriman barang juga memiliki waktu service, dan kendaraan juga harus kembali pada jam tertentu.

3. DESAIN SISTEM

3.1 Alur Kerja Program

Flowchart dibawah ini menggambarkan alur program berjalan dari awal hingga akhir. Program dibuat dengan menggunakan Unity dan Bahasa pemrograman yang dipakai adalah C#.



Gambar 5. *Flowchart Program*

a. Pada saat program dijalankan, program akan mengambil data problem yang ada. Data berisi daftar dari customer dan depot. Masing-masing customer dan depot memiliki lokasi (x,y),demand barang, time windows (waktu minimum service, waktu maksimum service, lama service).

b. Data yang telah masuk kedalam program akan disimpan dalam array of object node-node baik customer ataupun depot, lalu object akan diletakkan di berbagai lokasi sesuai data.

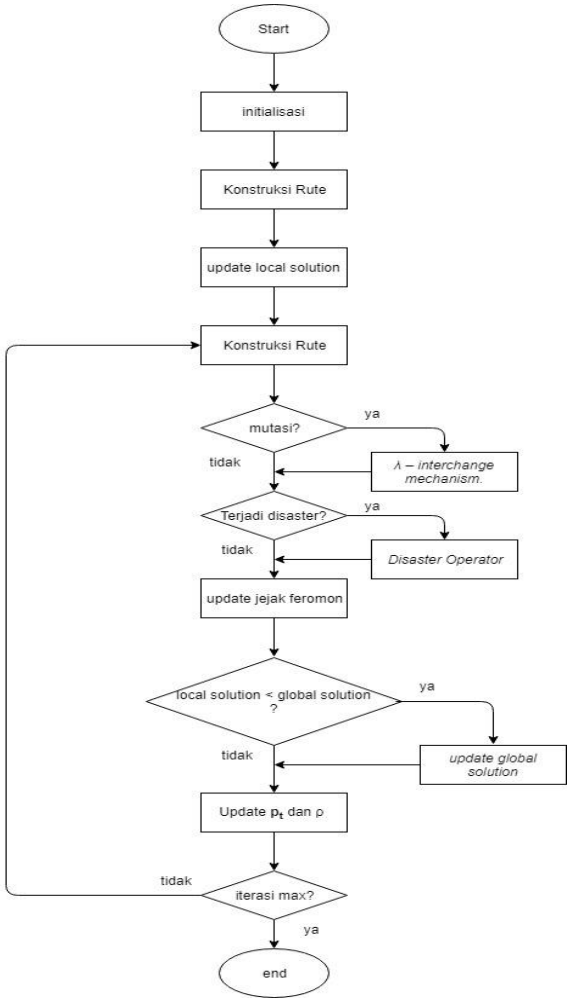
c. Tampilkan data yang sudah proses kedalam layar dalam bentuk titik-titik customer dan depot.

d. Jalankan proses HACO untuk menyelesaikan problem yang ada.

e. Setelah proses HACO selesai, tampilkan lama proses dari HACO, dan gambarkan hasil rute terbaik yang dihasilkan pada UI.

3.2 Proses HACO-VRPTW

Flowchart dibawah ini menjelaskan langkah-langkah pengerjaan HACO untuk menyelesaikan VRPTW.



Gambar 6. Flowchart HACO-VRPTW

a. Inisialisasi program, menginput data problem yang akan dikerjakan, memasukan semua variabel dan konstanta yang

diperlukan seperti kapasitas kendaraan, jumlah semut(m),pengendali intensitas jejak semut(α), pengendali visibilitas jarak antar node(β), pengendali intensitas penggunaan saving value(γ), konstanta Q, variabel pengontrol kecepatan penguapan jejak feromon(ρ), variabel probabilitas seleksi mutlak(p0), dan total iterasi. Jika mengacu pada jurnal yang dipublikasikan Ding et al.(2012), nilai m = 0.35 dari customer, α = [2,3], β =[4,6], γ = [3,5], Q= 400, ρ = 1, p0 = 1. Hitung feromon maksimum dan minimum yang didapat dari:

$$\tau_{min} = \frac{Q}{\sum_i 2d_{0i}} \quad (2)$$

$$\tau_{max} = \frac{Q}{\sum_i d_{0i}} \quad (3)$$

Rumus tersebut diambil dari laporan penelitian oleh Bin,Y. et al.(2008). D_{0i} merupakan jarak dari depot ke node i. Setelah menghitung nilai feromon maksimum, inisialisasi semua rute dengan $\tau_0 = \tau_{max}$.

b. Membuat candidate list untuk setiap node yang ada. Jumlah candidate list yang dipakai untuk 1 node adalah n/4 , dimana n adalah jumlah customer yang diinput.

c. Pilih semua node yang ada dalam candidate list yang belum pernah dilewati dan juga memenuhi syarat dari VRPTW, jika i merupakan lokasi saat ini, pilih node selanjutnya (j) berdasarkan rumus 4:

$$j = \begin{cases} \text{argmax}_{j \notin tabu_k} [\tau_{ij}(t)]^{\alpha} [\eta_{ij}(t)]^{\beta} [\mu_{ij}]^{\gamma} & \text{if } q \le p_t \\ \text{random } j \notin tabu_k, & \text{otherwise} \end{cases} \quad (4)$$

$$\eta_{ij} = \left(\frac{1}{d_{ij}+w_{ij}}\right) \quad (5)$$

$$\eta_{ij} = \left(\frac{1}{d_{ij}}\right) \quad (6)$$

Dimana tabuk merupakan tabel tabu yang merekam semua *node* yang pernah dilalui semut k. τ_{ij} melambangkan feromon yang ada pada jalur i menuju j dan η_{ij} , didapat dari Rumus 6, adalah invers dari total waktu tunggu w_{ij} dan jarak dari i menuju j d_{ij} , yang didapat setelah sebelumnya mencoba menggunakan Rumus 6, namun setelah pengujian pada Sub Bab 4.1, dapat dilihat hasil yang didapat kurang memuaskan. Oleh karena itu ditambahkannya total waktu tunggu W_{ij} yang berdasarkan pengujian kedua pada Sub Bab 4.1, hasil yang didapat jauh lebih memuaskan. μ_{ij} merupakan *saving value* dari *saving algorithm.* **α** ,β , γ merupakan *relative influence* setiap variabel. q adalah value yang dipilih secara acak di interval [0,1]. p_t (0<pt<=1) merupakan variabel yang mengontrol apakah menggunakan rumus atau menggunakan random, dimana diawali dengan p0 = 1 dan akan disesuaikan secara dinamis seiring berjalannya proses.

d. jika jumlah total semut yang melakukan proses pencarian lebih kecil dari m, kembali ke langkah III dan sisa semut akan melanjutkan proses pencarian, jika sudah, pergi ke langkah V.

e. Hitung solution L_k untuk setiap semut, masukan L_k

$$L_{local} = \sum_{k=1}^{m} L_k \quad (7)$$

sebagai local optimal solution lalu simpan data rute dari L_{local}.

f. Melakukan 2-interchange dan or-interchange mechanism pada Llocal dengan probabilitas z, dimana z diberi initial value yang besar lalu akan semakin dikurangi seiring berjalannya proses, yang dihitung menggunakan rumus 8.

$$z = \frac{1}{n} + \left(\frac{1}{n_v} - \frac{1}{n}\right)^{1-t/T} \quad (8)$$

dimana n adalah jumlah customer, n_v adalah jumlah kendaraan yang ada, t adalah nomor iterasi keberapa yang sedang dijalankan dan T adalah total iterasi yang dijalankan. Jika hasil dari mutasi (Lopt) tidak melanggar rule vrptw dan value $L_{opt} < L_{local}$, set L_{local} menjadi L_{opt} , kemudian update rute.

g. Pilih sebagian dari L_{local} dengan probabilitas z, jika berjalan, set jejak feromon yang ada menjadi nilai minimal feromon, kemudian hitung ulang solusi, jika value $L_{opt} < L_{local}$, set L_{local} menjadi L_{opt} , kemudian update rute.

h. Update semua jejak feromon dengan rumus 9 dan 10 yaitu:

$$\tau_{ij}^{new} = \rho\tau_{ij}^{old} + \Delta\tau_{ij} \quad (9)$$

$$\Delta\tau_{ij} = \begin{cases} \frac{Q}{L_{local}}, & arc(ij)\ belongs\ to\ L_{local} \\ 0, & otherwise \end{cases} \quad (10)$$

Dimana Q merupakan konstan yang berhubungan dengan jejak yang dikeluarkan semut. Q diset di interval [300,500] untuk mencapai hasil yang maksimal. ρ (0< ρ <=1) merupakan keguguhan jejak yang diinitialisasikan ρ=1, dan akan diubah seiring berjalannya proses. τ_{ij} akan diubah menjadi τ_{max} jika $\tau_{ij} > \tau_{max}$, atau diubah menjadi $(\tau_{min} + \tau_{\max})/2$ ketika $\tau_{ij} < \tau_{min}$.

i. Bandingkan $\mathbf{L_{local}}$ dengan $\mathbf{L_{global}}$. Jika $\mathbf{L_{local} < L_{global}}$. Maka set $\mathbf{L_{global} = L_{local}}$, kemudian *update* rute

j. Update $\mathbf{p_t}$ dan ρ dengan rumus 10 dan 11:

$$P_t = \begin{cases} 0.95P_{t-1}, & \text{if } 0.95P_{t-1} \ge P_{min} \\ P_{min}, & \text{otherwise} \end{cases} \quad (10)$$

$$\rho_t = \begin{cases} 0.95\rho_{n-1}, & \text{if } 0.95\rho_{n-1} \ge \rho_{min} \\ \rho_{min}, & \text{otherwise} \end{cases} \quad (11)$$

Dimana p_{min} digunakan untuk memastikan pemilihan rute meskipun p_t semakin kecil, dan ρ_{min} digunakan untuk mencegah kecepatan perubahan yang terlalu kecil jika ρ terlalu kecil.

k. Jika jumlah iterasi yang telah disetujui diawal sudah memenuhi, maka algoritma selesai, jika belum, kembali ke langkah b.

4. PENGUJIAN SISTEM

Pengujian dilakukan dengan menguji total hasil panjang rute, menguji waktu yang diperlukan untuk menyelesaikan program, menguji pemakaian resource (memory), menguji beberapa perkembangan yang diusulkan dengan perubahan waktu tempuh dan hasil. Program diujikan pada computer berspesifikasi: Intel Core i7 7700, NVIDIA GeForce GTX 1060, 16gb RAM.

4.1 Implementasi HACO

Pada bagian ini dilakukan pengujian program HACO-VRPTW. Dalam pengujian ini, parameter yang dipakai adalah:

- α = 1-10
- β = 1-10
- γ = 1-10
- Q = 400
- Jumlah semut = 35
- Jumlah iterasi = 5
- Data yang dipakai data R-101 dengan 100 customer
- Total kapasitas kendaraan 200

Dari parameter diatas, tabel ini adalah 5 hasil terbaik yang didapat dari pengujian tersebut.

Tabel 1. Implementasi HACO

A	β	γ	Panjang rute	Total Vehicle	Banyak rute
1	9	3	1946,806118	21	122
3	4	8	2088,844316	22	123
3	8	1	2133,642363	25	126
3	6	10	2186,085897	30	122
2	9	5	2198,753068	28	127

Rata-rata panjang rute dari semua hasil yang didapat adalah 2501,23 , dengan hasil terbaik 1946,80 dengan 37 kendaraan. Namun hasil terbaik yang diberikan oleh pembuat data adalah 1637 dengan menggunakan 20 kendaraan. Kelemahan yang ditemukan setelah melihat hasil tersebut adalah kendaraan memilih customer yang dekat, meskipun memiliki waktu tunggu yang lama untuk dapat dilayani.

Oleh karena itu dilakukan percobaan dengan menambahkan faktor *waiting time* pada rumus pemilihan *node* selanjutnya, untuk menghindari pemilihan kota yang dekat tetapi waiting time lama. Tabel dibawah ini adalah 5 hasil terbaik dari penelitian tersebut.

Tabel 2. Implementasi HACO + *Waiting Time*

α	B	γ	Panjang rute	Total Kendaraan	Jumlah rute
1	8	10	1843,706894	21	118
4	6	8	1868,244748	21	114
2	3	4	1877,180068	21	114
3	6	8	1880,509086	21	113
1	9	8	1880,684941	21	116

Rata-rata dari semua hasil yang didapat adalah 2047,025618, dengan hasil terbaik 1843,706894 dengan 21 kendaraan. Dari dua penelitian yang sudah dilakukan, dengan menambahkan pertimbangan waiting time, hasil yang didapat cenderung lebih baik, dengan rata-rata hasil turun 454 unit.

4.2 Analisa Performa Mutasi

Dalam penelitian yang dilakukan, performa dari mutasi disaster operator lebih baik dari *Or-interchange* dan 2-*interchange*. *Or-interchange* dan 2-*interchange* tidak dapat merubah value terlalu tinggi karena *constraint* dari *time windows* dan *capacity* yang membuat perubahan yang terlalu besar menjadi sulit. Sedangkan *disaster operator* mengubah value feromon pada suatu jalur, lalu menghitung ulang rute dari awal, tidak hanya menukar 1 *node*.

Tabel 3 dan 4 adalah beberapa contoh hasil perubahan value dari *Disaster operator* dan *interchange*. Nilai yang di *highlight bold* merupakan nilai asli sebelum dimutasikan.

Tabel 3. Hasil Disaster

Jumlah Kendaraan	Jumlah Rute	Panjang Rute
21	**122**	**1934,877166**
21	122	1926,843637
17	118	1895,891366
16	117	1822.555551

Tabel 4. Hasil *Interchange*

Jumlah Kendaraan	Jumlah Rute	Panjang Rute
21	**122**	**1934,877166**
21	122	1917,591454
21	122	1923,343142
21	122	1927,891759

4.3 Analisa Waktu Proses dan Memory

Dengan menggunakan maksimum iterasi = 5, total waktu yang terpakai untuk mengerjakan program adalah 5 sekon, dimana 2 sekon pertama dipakai untuk inisialisasi program dan variabel awal. Sedangkan *total memory usage* secara keseluruhan tidak berubah ketika dites dengan berbagai konfigurasi. Total memory usage secara lebih detail dapat dilihat di tabel 3.

Tabel 5. Hasil Memory Profiler

Memory Type	Besar Pemakaian
Unity	71.4 MB
Mono	9.3 MB
GfxDriver	17.4 MB
FMOD	1.3 MB
Profiler	14.8 MB

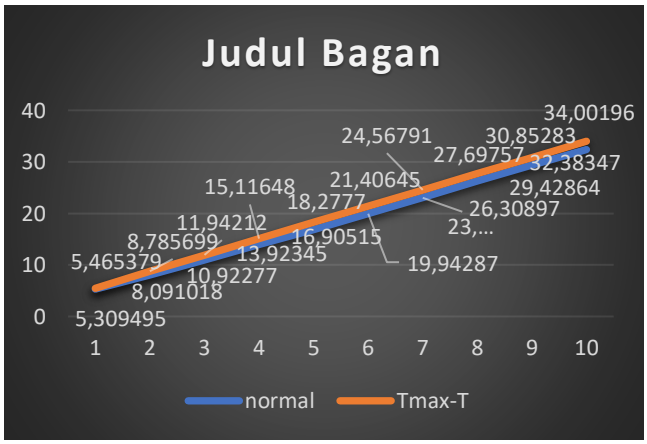
Used Total	103.6 MB

memori yang digunakan dipisahkan menjadi beberapa kategori. Unity adalah memory yang digunakan untuk menjalankan code yang ada diprogram Unity. Mono adalah memori yang direserve ketika program berjalan. GfxDriver adalah estimasi penggunaan memory yang digunakan untuk *texture*, *render targets*, *Shaders* dan *Mesh data*. FMOD adalah memory yang digunakan oleh *audio driver*. Profiler adalah memory yang digunakan untuk mencatat hasil memory usage ini.

4.4 Perkembangan Rate Mutasi

Dari hasil penelitian yang sudah dilakukan di Sub Bab 4.1, setelah nilai menjadi stagnan dalam suatu iterasi, nilai dari rute tersebut akan dapat menjadi lebih baik dengan adanya mutasi. Maka dari itu dilakukan percobaan dimana ditambahnya jumlah mutasi yang dilakukan per iterasi, dengan mengorbankan waktu proses sesedikit mungkin. Jumlah mutasi per iterasi yang sebelumnya 1, dijadikan Tmax-T , dimana Tmax adalah jumlah iterasi total dan T adalah iterasi saat ini, karena semakin bertambahnya iterasi, nilai probailitas dari mutasi akan semakin berkurang, maka dari itu jumlah mutasi juga akan dikurangi seiring berjalannya proses iterasi.

Gambar 7 merupakan data graph perbandingan waktu proses antara mutasi yang normal dan yang jumlah mutasi yang ditambah, program dijalankan 10x berturut turut dan total waktu ditambahkan untuk melihat perbedaan.



Gambar 7. Perbandingan waktu – mutasi

Dapat dilihat hasil perbedaan waktu yang ada tidak terlalu signifikan, apalagi ketika program dijalankan hanya sekali. Oleh karena itu, jika ingin menaikan probabilitas terjadinya perubahan pada hasil, cara ini dapat dilakukan dengan mengorbankan sedikit waktu proses.

4.5 Pengujian Jumlah Iterasi dengan Waktu Proses dan Hasil Panjang Rute

Pengujian ini membandingkan pengaruh jumlah iterasi terhadap waktu proses dan hasil pegujian berupa hasil panjang rute. Dari hasil pengujian tersebut, setiap pertambahan 1 iterasi menambahkan waktu proses sekitar 0.2-1 detik. Untuk hasil proses, bertambahnya jumlah iterasi akan membuat hasil cenderung lebih baik, tetapi akan memakan waktu yang cukup lama. Hasil dari 5 sampai dengan 20 iterasi mengeluarkan nilai rata-rata panjang 1930 dengan waktu 3 detik sampai dengan 15 detik. Sedangkan rata-rata hasil dari 91-100 iterasi menunjukan rata rata hasil sekitar 1860, namun waktu yang dipakai adalah 72-80 detik.

5. KESIMPULAN

Setelah dilakukan perancangan sistem, pengimplementasian, pengujian, dan analisa terhadap aplikasi yang telah dibuat, dapat ditarik kesimpulan sebagai berikut:

1. Dalam penelitian dan analisa teori serta percobaan yang dilakukan, berikut adalah perkembangan yang dilakukan HACO dari ACO beserta fungsinya:
   - Feromon pada semua jalur diinisialisasikan dengan *value* maksimum untuk dapat meraih nilai semi optimal yang cepat.
   - Feromon memiliki nilai maksimum dan minimum yang dipersempit(bukan [0,1]) untuk mengurangi range stagnansi dan membuat hasil lebih dinamis.
   - Candidate List digunakan untuk mempersingkat range pencarian, sehingga mengurangi waktu proses.
   - Rumus pemilihan node selanjutnya diubah dan ditambahkannya saving algorithm agar hasil node yang dipilih memiliki jarak yang lebih pendek.
   - Cara pemilihan node dibagi menjadi 2, yaitu menggunakan rumus pemilihan node atau pemilihan secara random. Pemilihan secara random dimasukan agar jika hasil stagnan, maka jalur bisa dirandom untuk keluar dari stagnansi.
   - Mutasi *2-interchange* , *or-interchange*, dan juga *disaster operator* berfungsi untuk mengurangi stagnansi dan berpeluang untuk mengoptimalkan hasil yang sudah dihitung sebelumnya. *2-interchange* dan *or-interchange* memiliki perubahan yang cenderung lebih kecil dibanding disaster *operator*, tetapi membutuhkan waktu proses yang lebih sedikit.
2. Dalam implementasi HACO-VRPTW dengan unity, waktu proses untuk menjalankan program sekali adalah sekitar 4-5 detik sudah terhitung dengan inisialisasi dan penggambaran user interface. Sedangkan total memori yang dipakai total memakan sekitar 100 MB memori.
3. Berdasarkan hasil pengujian, untuk dapat mengembangkan hasil yang didapat dari proses HACO agar dapat lebih baik, diajukan 2 *improvement* yang dapat dipakai. Improvement pertama adalah penambahan faktor waiting time dalam rumus pemilihan node agar semut tidak cenderung memilih lokasi customer yang dekat namun memiliki waktu tunggu yang lama baru dapat diservice. Improvement kedua adalah menambah jumlah terjadinya mutasi dalam 1 iterasi, karena mutasi dapat memperbaiki hasil yang didapat dari konstruksi rute secara manual.

6. REFERENSI

[1] Bell, J., & McMullen, P. 2004. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*

[2] Bin, Y., Yang, Z., Yao, B. 2007. An improved ant colony optimization for vehicle routing problem, European Journal of Operational Research 196 171–176

[3] Ding, Q., Hu, X., Sun, L., & Wang, Y. 2012. An improved ant colony optimization and its application to vehicle routing problem with time windows. *Neurocomputing*, 98, 101-107.

[4] Mahi, M., Baykan, Ö., & Kodaz, H. 2015. A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing*, *30*, 484-490.

[5] Osman, I. 1993. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals Of Operations Research*, *41*(4), 421-451.

[6] Pala, O., & Aksaraylı, M. 2018. An Ant Colony Optimization Algorithm Approach for Solving Multi-objective Capacitated Vehicle Routing Problem. *Alphanumeric Journal*.

[7] Reed, M., Yiannakou, A., & Evering, R. 2014. An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing*, *15*, 169-176.

[8] Stutzle, T., Hoos,H. 1997. improvements on the ant system: introducing the MAX-MIN ant system. *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pp 245 - 249

[9] Ting, C., & Chen, C. 2013. A multiple ant colony optimization algorithm for the capacitated location routing problem. *International Journal Of Production Economics*, *141*(1), 34-44

[10] Yao, B., Yu, B., Hu, P., Gao, J., & Zhang, M. 2015. An improved particle swarm optimization for carton heterogeneous vehicle routing problem with a collection depot. *Annals Of Operations Research*, *242*(2), 303-320.