RANCANG BANGUN *SOFTWARE* SIMULASI PENDUKUNG KEPUTUSAN DENGAN MENGGUNAKAN ALGORITMA *ANT COLONY SYSTEM* PADA KASUS *TRAVELING SALESMAN PROBLEM*

Rindra Yusianto[1], Budi Setyo Utomo[2]

[1] Program Studi Teknik Industri
Universitas Dian Nuswantoro Semarang
Jl. Nakula I No 5-11 Semarang
Email : rindra@staff.dinus.ac.id

Abstrak

Dalam kehidupan sehari-hari seringkali manajer atau pengambil keputusan dihadapkan pada suatu permasalahan yang membutuhkan metode tertentu untuk memecahkannya. Salah satu permasalahan tersebut adalah Traveling Salesman Problem (TSP). TSP secara alami muncul sebagai sub masalah dalam berbagai aplikasi transportasi, sebagai contoh masalah dalam menyusun rute kota yang harus dilalui oleh seorang kurir (salesman) agar tidak terlalu jauh jarak yang ditempuh.

Adapun algoritma yang digunakan untuk memecahkan permasalahan TSP tersebut adalah Ant Colony System. Dimana algoritma tersebut akan dibandingkan dengan beberapa algoritma lain yaitu algoritma simple insertion, local search, nearest neighbor with local search. Hasil dari penelitian ini adalah sebuah software simulasi yang dapat membantu manajer atau pengambil keputusan untuk memecahkan permasalahan TSP.

Dari hasil penelitian dapat disimpulkan bahwa algoritma Ant Colony System mampu menghasilkan solusi yang lebih optimum dibandingkan dengan algoritma lainnya untuk jumlah kota yang lebih banyak.

Keywords : *Traveling Salesman Problem*, Algoritma *Ant Colony System*

1. PENDAHULUAN

Teknologi komputer terus berkembang, baik secara sadar maupun tidak, telah mempengaruhi seluruh aspek kehidupan manusia. Dengan kata lain, komputer telah diterima dan dirasakan kegunaannya oleh masyarakat sebagai bagian dari kehidupan manusia. Berbicara tentang komputer, maka tidak terlepas dari istilah *hardware* dan *software*. Berbagai perusahaan *hardware* terus berlomba untuk menciptakan teknologi baru, demikian juga perusahaan *software* yang terus berlomba untuk menciptakan berbagai perangkat lunak yang menarik dan berguna. (Utomo, 2007).

Merancang bangun *software*, untuk memecahkan suatu permasalahan yang cukup kompleks memerlukan sebuah metode khusus. Metode khusus untuk memecahkan suatu permasalahan disebut sebagai algoritma. Menurut Sismoro (1992), algoritma adalah kumpulan instruksi/perintah/langkah yang berhingga jumlahnya, dituliskan secara sistematis dan digunakan untuk menyelesaikan masalah/persoalan logika dan matematika dengan bantuan komputer.

Salah satu permasalahan yang cukup kompleks sehingga memerlukan algoritma khusus untuk memecahkannya adalah *Traveling Salesman Problem* (TSP). TSP adalah persoalan bagaimana menemukan rute terpendek yang akan dilalui salesman untuk mengunjungi sejumlah kota dengan syarat setiap kota hanya boleh dikunjungi sekali, kecuali kota terakhir. Beberapa algoritma yang telah diterapkan pada aplikasi TSP, diantaranya algoritma *genetik,* algoritma *simulated anneling,* algoritma *elastik net,* algoritma *simple insertion,* algoritma *local search,* dan algoritma *nearest neighbor with local search*.

Selain algoritma di atas, Algoritma *Ant Colony System* juga dapat diterapkan untuk memecahkan TSP. Algoritma koloni semut ini diilhami dari tingkah laku sebuah koloni semut, dimana sekelompok semut mampu menemukan jalur terpendek untuk membawa makanan dari sumber makanan ke sarangnya. Hal ini bisa mereka lakukan karena mereka menggunakan suatu sistem kerja sama yang unik, yaitu dengan memanfaatkan informasi *pheromone* yang

dikumpulkan semut-semut pada jalur yang dilewatinya. Dengan cara ini semut bisa mengetahui kadar *pheromone* pada jalur-jalur yang ada, dan semut akan memilih jalur yang kaya akan *pheromone*. Proses ini terus diulang sampai pada tahap tertentu dimana semua semut akan memilih jalur yang sama. Dengan mengambil contoh dari tingkah laku semut, seorang peneliti berkebangsaan Italia bernama Marco Dorigo pada tahun 1995 berhasil menciptakan algoritma ini.

Penelitian ini dimaksudkan untuk merancang bangun sebuah *software* simulasi pendukung keputusan dengan menggunakan algoritma *Ant Colony System* pada kasus TSP. *Software* simulasi yang dirancang bangun, juga dapat memperbandingkan Algoritma *Ant Colony System* dengan algoritma *local search,* algoritma *simple insertion* dan algoritma *nearest neighbor with local search* terutama ditinjau dari segi optimalisasi jarak tempuh. *Software* simulasi ini juga dapat digunakan untuk membantu pengambilan keputusan dengan memberikan solusi algoritma yang lebih baik digunakan, jika jarak tempuh TSP adalah sama.

2. TINJAUAN PUSTAKA

Menurut Applegate et al. (2001), permasalahan TSP berhubungan dengan permasalahan matematika. TSP secara umum mulai dipelajari oleh para ahli matematika pada tahun 1930-an oleh Karl Menger di Vienna dan Harvard. Permasalahan ini kemudian diperkenalkan lagi oleh Hassler Whitney dan Merrill Flood di Princeton. TSP telah banyak digemari sejak lama sebagai bahan pertimbangan dari hasil riset. TSP juga memainkan peranan penting dalam *Ant Colony Optimization* (ACO) sejak algoritma ACO pertama yang disebut *Ant System* yang pada awalnya diaplikasikan pada TSP, begitu juga dengan algoritma-algoritma ACO yang dikemukakan setelah *Ant System*, semuanya diaplikasikan pada TSP (Stutzle dan Dorigo, 1999).

2.1 Definisi TSP

Menurut Jonhson dan McGeoch (1995), TSP dapat didefinisikan sebagai berikut : ada satu set kota{1,c2,c3,…,c_N} dan d{c_i,c_j} adalah jarak antara kota ke-i dan kota ke-j. Tujuannya adalah menemukan urutan π dari rumus di bawah ini untuk mendapatkan nilai yang paling minimal

$$\sum_{i=1}^{N-1} d(c_{\pi(i)}.c_{\pi(i+1)}) + d(c_{\pi(N).}c_{\pi(1)}) \qquad (1)$$

Hasil dari rumus di atas disebut sebagai panjang dari perjalanan seorang *salesman* mengunjungi kota-kota sesuai urutan π dari rumus di atas, dimana setelah mengunjungi semua kota *salesman* tersebut akan kembali ke kota awal. Dalam hal ini jika kasusnya *symmetric* TSP maka d{c_i,c_j}= d{c_j,c_i} jika *asymmetric* TSP maka d{c_i,c_j}□ d{c_j,c_i} untuk 1≤ i, j≤N.

Permasalahan TSP ini termasuk dalam kategori permasalahan *non determistik polynomial time* artinya belum ditemukan waktu polinomial yang dibutuhkan untuk memecahkan masalah tersebut, misal: untuk algoritma A, jumlah kota N, waktu yang dibutuhkan T. Maka untuk jumlah kota 2N tidak bisa dikatakan membutuhkan waktu 2T untuk algoritma A. Hal ini mengakibatkan 2 alternatif pengembangan algoritma. Yang pertama algoritma yang berusaha menemukan perjalanan yang mendekati optimal tetapi dapat dilakukan dengan waktu yang singkat. Yang kedua mengembangkan optimasi algoritma yang benar-benar mencari solusi yang terbaik dari permasalahan TSP.

Jika hanya dua kota yang menjadi tujuan dalam suatu perjalanan, maka masalah perjalanan ini bisa dikatakan mudah sekali. Untuk *symmetric* TSP perjalanan tiga kota juga masalah yang mudah dipecahkan. Jumlah kombinasi perjalanan yang dihasilkan untuk kasus *Asymmetric* adalah (n-1)!. Untuk melihat kenapa itu bisa terjadi, pilihlah satu kota yang mana saja secara bebas sebagai kota pertama, kemudian ada (n-1) kota yang bisa dipilih sebagai kota yang kedua untuk dikunjungi, (n-2) kota yang bisa dipilih sebagai kota yang ke tiga untuk dikunjungi, begitu seterusnya sampai semua kota habis dikunjungi dan kembali ke kota awal.

Menurut Dakin (1997), pada kasus *symmetric* TSP, jumlah kombinasi perjalanannya adalah setengah dari jumlah kombinasi perjalanan *asymmetric* TSP yaitu $\frac{(n-1)!}{2}$.

2.2 Implementasi TSP

Menurut Applegate et al. (2001), sebagian besar pekerjaan yang menggunakan TSP, tidak dimotivasi oleh aplikasi-aplikasi yang secara langsung menggunakan TSP dalam kegiatan sehari-hari, tetapi oleh fakta bahwa TSP menyediakan *platform* yang ideal untuk mempelajari metode-metode umum yang dapat diaplikasikan pada berbagai macam masalah-masalah optimasi.

TSP secara alami muncul sebagai sub masalah dalam berbagai aplikasi transportasi dan logistik, sebagai contoh masalah dalam menyusun rute bis sekolah untuk menjemput anak-anak dalam area sekolah. Aplikasi bis ini merupakan aplikasi lama yang penting dan sangat berarti dalam sejarah TSP, terlebih ketika menyediakan motivasi untuk Merrill Flood, salah seorang dari perintis riset TSP pada tahun 1940. Aplikasi kedua TSP pada tahun 1940-an yang melibatkan transportasi alat pertanian dari satu lokasi ke lokasi lainnya untuk mengecek keadaan tanah, memelopori pelajaran matematika di Bengal oleh P.C. Mahalanobis dan di Iowa oleh R. J. Jessen. Aplikasi lain yang baru-baru ini melibatkan pengantaran makanan ke rumah-rumah orang, rute perjalanan truk untuk pengambilan pos bingkisan parcel dan untuk yang lainnya.

Aplikasi berikutnya yaitu permasalahan arah pergerakan kendaraan pada kompetisi Whizzkids'96. Permasalahannya terdiri dari pencarian rute terbaik untuk 4 pengantar koran dalam mengantarkan koran ke 120 pelanggan mereka. Team yang beranggotakan David Applegate, William Cook, Sanjeeb Dash, dan Andre Rohe menerima hadiah 5,000 Gulden dari firma teknologi informasi untuk solusi mereka pada Februari 2001.

3. HASIL PENELITIAN DAN PEMBAHASAN

3.1 Pemecahan dengan Beberapa Algoritma untuk masalah TSP

Menurut Jonhson and McGeoch (1995), *Traveling Salesman Problem* (TSP) adalah suatu permasalahan mencari sebuah rute tertutup untuk mengunjungi sejumlah kota, dimana setiap kota hanya dikunjungi sekali dan kembali ke kota awal setelah semua kota dikunjungi. Yang menjadi masalah adalah bagaimana menemukan rute yang total jarak perjalanannya paling pendek. TSP dibedakan menjadi 2 jenis yaitu *symetric* TSP dan *asymetric* TSP (ATSP). Dalam penelitian ini yang dibahas adalah masalah *simetric travelling salesman problem* atau yang biasa disebut TSP saja, dimana jarak dari kota A ke kota B, sama dengan jarak dari kota B ke kota A.

Jumlah kombinasi n kota untuk masalah TSP adalah ½ kali jumlah kombinasi n kota untuk masalah ATSP. Jumlah semua kombinasi untuk n kota pada kasus TSP adalah (n-1)!/2. Untuk memecahkan masalah TSP, maka diperlukan suatu algoritma yang pada intinya adalah mengurangi pengecekan terhadap semua kombinasi yang mungkin. Beberapa algoritma yang pernah dipakai untuk memecahkan masalah TSP menurut Jonhson dan Mc.Geoch (1995) adalah algoritma *genetic,*algoritma *simulated annealing,* algoritma *tabu search,* algoritma *lin Kernighan,* algoritma *neutral network,* algoritma *local search,* algoritma *simple insertion* dan algoritma *nearest neighbor with local search.*

3.2 Algoritma *Ant Colony System*

Algoritma *Ant Colony System* (ACS), merupakan bagian dari algoritma *Ant Colony* Optimization (ACO). ACO adalah kumpulan algoritma-algoritma yang meniru koloni semut (*ant*) dalam mencari makanan melalui rute yang dilaluinya. Yang termasuk ke dalam ACO yaitu *Ant system, Ant-Q System* dan ACS (Stutzle. and Dorigo, 1999).

Sesuai dengan namanya, algoritma ini terinspirasi oleh tingkah laku koloni semut, terutama tingkah laku dalam mencari makanan. Salah satu pokok pikiran dari algoritma ini adalah komunikasi tidak langsung dari agen koloninya (yang pada algoritma ini disebut semut-semut buatan) berdasarkan jejak *pheromone* (*pheromone* juga digunakan oleh semut asli untuk berkomunikasi). Jejak *pheromone* buatan tersebut merupakan informasi dalam bentuk angka yang dimodifikasi oleh semut-semut (buatan) untuk menunjukkan jalan yang telah mereka lalui dalam memecahkan masalah tertentu.

Menurut Gould (2001) *pheromone* ialah bau yang diproduksi oleh seekor binatang yang mempengaruhi tingkah laku dari binatang lain. Cara kerja *pheromone* dapat dianalogikan dengan cara kerja dari hormon-hormon di dalam tubuh yang mengirimkan sinyal kimia khusus dari sekelompok sel ke lainnya, yang menyebabkan mereka melakukan kegiatan tertentu. *Pheromone* ditemukan di seluruh dunia kehidupan dan merupakan bentuk komunikasi binatang yang paling kuno.

Algoritma ACO yang pertama disebut *ant system* (AS), yang telah diaplikasikan pada TSP. Dimulai dari AS, kemudian dikemukakan beberapa perbaikan dari algoritma dasar tersebut. Secara khusus, algoritma-algoritma hasil perbaikan tersebut telah dites ulang dengan TSP. Semua versi perbaikan dari AS ini memiliki kesamaan yaitu ditemukannya suatu eksploitasi yang lebih kuat dari solusi yang terbaik, yang penemuan eksploitasi tersebut digunakan untuk mengarahkan proses pencarian oleh semut. Semua algoritma tersebut memiliki perbedaan utama yaitu dalam beberapa aspek yang mengontrol pencarian. Pada ACO, *performance* terbaik dalam memecahkan TSP, berguna untuk memperbaiki solusi yang dihasilkan oleh pencarian semut-semut sebelumnya, yaitu dengan menggunakan algoritma *Local Search* (Stutzle dan Dorigo, 1999).

3.3 Cara Kerja Algoritma ACS

Pada awal program sebanyak m-semut ditempatkan di sebanyak n kota yang dipilih dengan beberapa aturan inisialisasi (misalnya *random*). Setiap semut membangun suatu perjalanan (perjalanan yang sesuai dengan aturan) dengan mengaplikasikan rumus transisi secara berulang. Ketika membangun sebuah perjalanan, semut juga mengubah jumlah *pheromone* di jalur yang dilaluinya dengan melakukan *local updating rule*. Ketika semua semut telah menyelesaikan perjalanannya, jumlah *pheromone* pada jalur-jalur yang dilewatinya diperbaharui kembali. Sama halnya dengan *ant system*, semut-semut dipandu dalam menyelesaikan perjalanannya dengan informasi *heuristic* (akan memilih jalur yang pendek), dan dengan informasi *pheromone*: jalur yang memiliki *pheromone* yang kuat merupakan pilihan yang menarik untuk dilewati. *Pheromone updating rule* dirancang sedemikian rupa sehingga akan menambah *pheromone* pada jalur-jalur yang seharusnya dipilih oleh semut-semut.

Secara umum semua algoritma ACO, termasuk ACS mengacu pada 2 tahap : Tahap pertama adalah inisialisasi *pheromone* dan beberapa parameter yang digunakan dalam program utama. Tahap kedua adalah proses perulangan utama yang akan berulang terus sampai kondisi tertentu, misalnya jumlah iterasi telah mencapai nilai yang ditentukan atau waktu pemrosesan telah mencapai batas waktu.

Rumus yang Digunakan dalam Algoritma ACS

Rumus eq1:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)].[\eta(r,s)]^{\beta}}{\sum_{u \ \varepsilon \ j_k(r)} [\tau(r,u)].[\eta(r,u)]^{\beta}} & if \quad s \varepsilon j_k(r), \ jikatidak \\ 0 & \end{cases}$$

Dimana τ adalah *pheromone*, η=1/δ adalah kebalikan dari jarak δ(r,s), $J_k(r)$ adalah kota-kota yang belum dikunjungi oleh semut k di kota r, dan *ß* adalah parameter yang menentukan seberapa besar pengaruh dari jumlah *pheromone* atau jarak (*ß*>0). Pada rumus eq1 ini, kita akan memilih jalur yang lebih pendek dan memiliki jumlah *pheromone* yang besar.

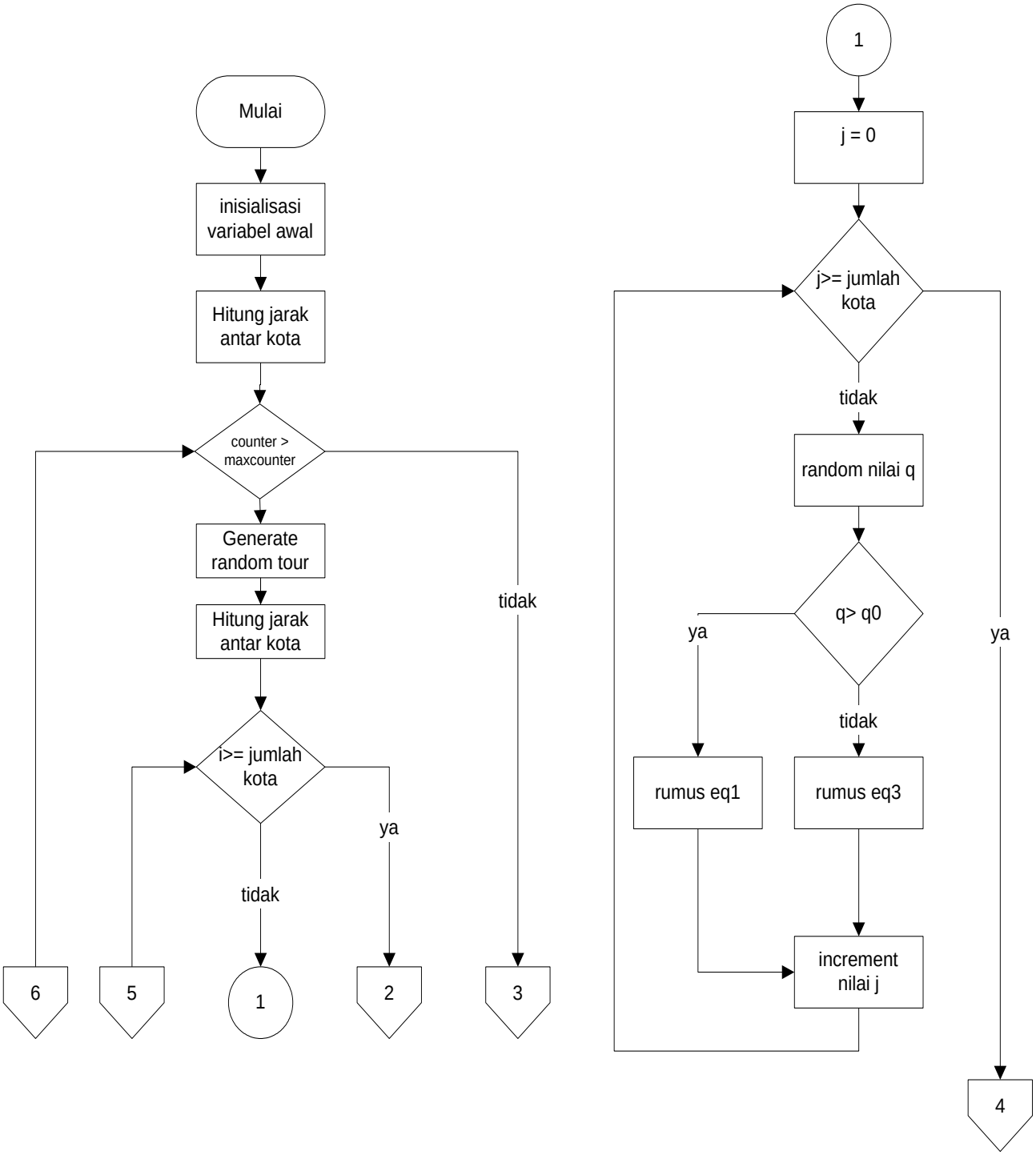
Rumus eq3 :

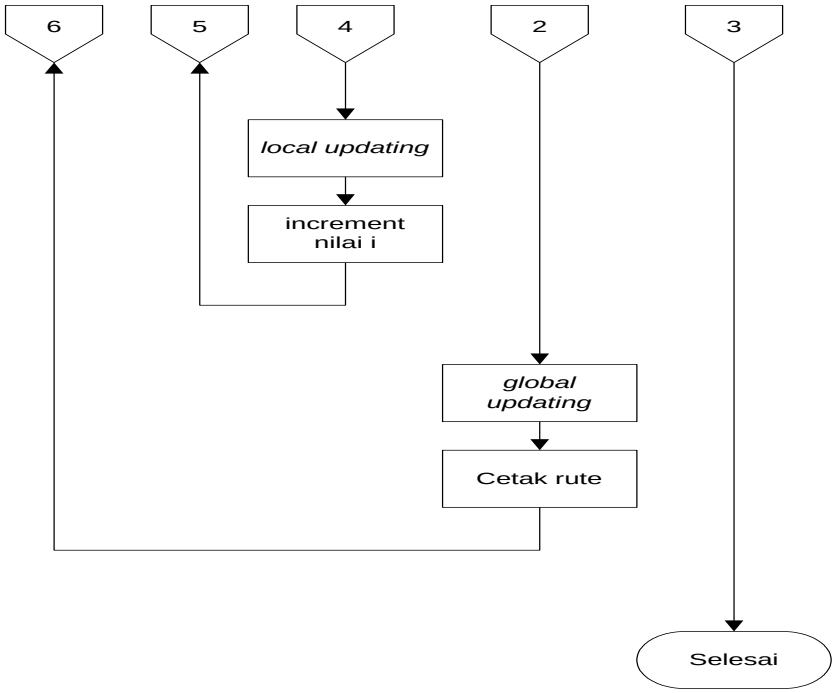
$$s = \begin{cases} \arg\max_{u \ \varepsilon \ j_k(r)} \{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \} & if \ q \le q_0 \quad \text{exploitation} \\ s \quad \text{jika tidak} \quad \text{(biased exploration)} & \end{cases}$$

Keterangan rumus eq 3 :
q random[0..1], q_o adalah *parameter* ($0 \le q_o \le 1$), s adalah variabel random, dari rumus eq1

Dimana τ adalah *pheromone*, η=1/δ adalah kebalikan dari jarak δ(r,s), $J_k(r)$ adalah kota-kota yang belum dikunjungi oleh semut k di kota r, dan *ß* adalah parameter yang menentukan seberapa besar pengaruh dari jumlah *pheromone* atau jarak (*ß*>0).

Diagram Alir Algoritma ACS





Gambar 1. Diagram alir algoritma *Ant Colony System*
Sumber : Kumalasari, 2002

Sebelum membangun sebuah software simulasi, terlebih dahulu merancang struktur database yang akan menampung hasil pengolahan data. Adapun struktur database tersebut adalah sebagai berikut :

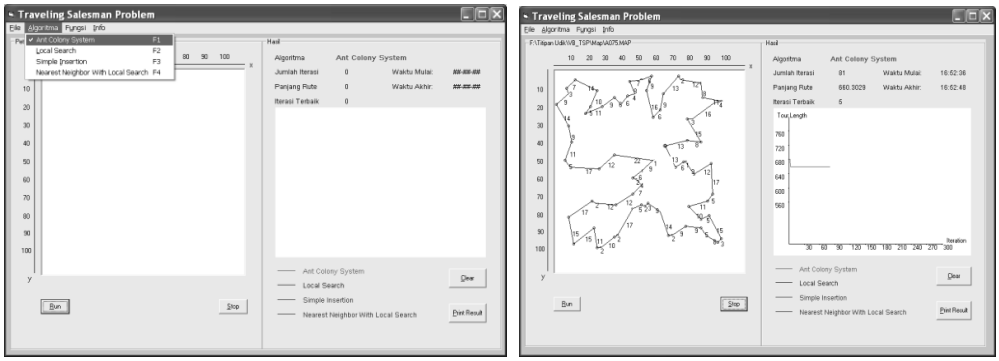
Tabel 1. Struktur database file output

Nama Field	Tipe Data	Panjang	Keterangan
Algoritma	String	36	Nama algoritma yang digunakan
Nama File	String	20	Nama *file* Input yang digunakan
Panjang Rute	String	12	Panjang rute yang dihasilkan
Iterasi	String	12	Jumlah iterasi yang dilakukan
Waktu	String	15	Waktu yang dibutuhkan
Iterasi terbaik	String	14	Iterasi terakhir yang terbaik

Gambar 2. Rancangan Tampilan Form

Rancang Bangun *Software* Simulasi

ACS berkaitan erat dengan panjang rute. Dalam *software* simulasi ini, penentuan panjang rute disediakan 4 macam algoritma. Nama algoritma yang dipilih akan muncul pada bagian "Hasil" di sebelah kanan rancangan tampilan form dengan warna sesuai dengan warna yang telah ditentukan untuk masing-masing algoritma. Proses perhitungan diawali dengan *generate* peta secara random sesuai dengan kebutuhan jumlah kota.

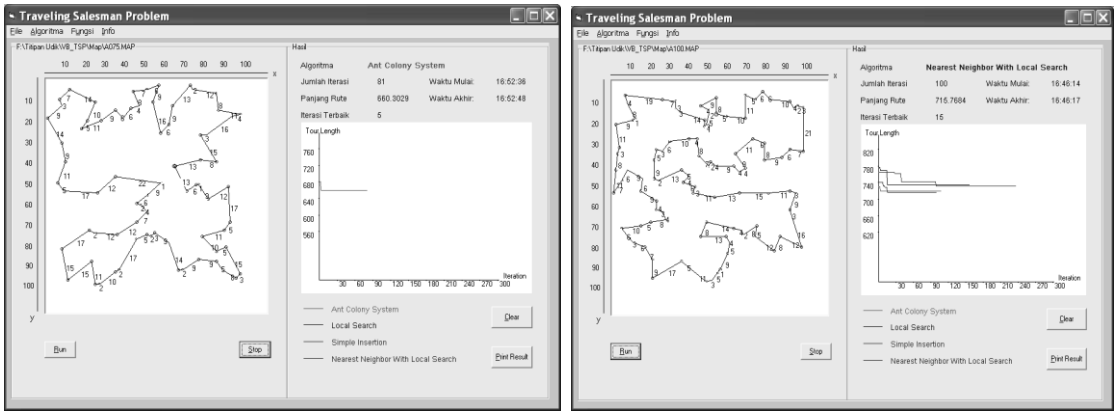


Gambar 3. Fasilitas Memilih Algortitma dan *Generate* Peta

Hasil yang didapat ditampilkan dalam bentuk garis-garis yang menghubungkan kota-kota yang di dalam gambar (dilambangkan dengan lingkaran-lingkaran kecil). Selain ditampilkan dalam bentuk gambar, hasil perhitungan juga ditampilkan dalam bentuk tulisan dan grafik yang dimunculkan pada bagian sebelah kanan pada program aplikasi. Informasi yang disajikan dalam *software* simulasi ini terdiri dari :

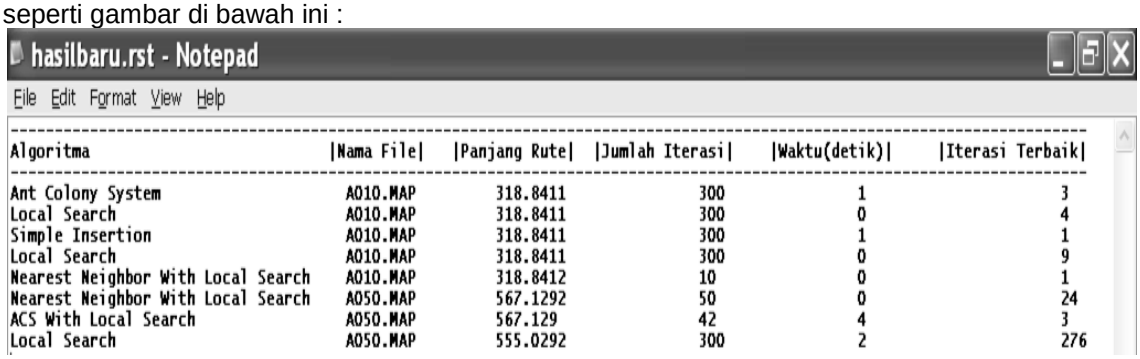
1. Algoritma (nama algoritma yang digunakan)
2. Jumlah iterasi (iterasi pada saat tersebut)
3. Perhitungan yang dilakukan dalam suatu proses perhitungan untuk mencari panjang rute terpendek tidak hanya sekali tetapi berkali-kali supaya dapat menemukan jarak yang lebih pendek lagi. Angka pada *sum of iteration* menunjukkan perhitungan yang telah dilakukan dalam satu proses perhitungan (dalam satu proses perhitungan ada banyak perhitungan).
4. Panjang rute
5. Angka yang ditunjukkan bukan merupakan angka yang dihasilkan oleh perhitungan pada iterasi saat tersebut. Tetapi merupakan angka terkecil (jarak rute terpendek) yang didapat selama proses perhitungan.
6. Iterasi terakhir
7. Iterasi terakhir menunjukkan pada iterasi ke berapa perhitungan sudah tidak menghasilkan panjang rute yang lebih pendek lagi atau dengan kata lain pada iterasi ke berapa grafik sudah tidak menurun lagi
8. Waktu Mulai (waktu saat proses dimulai)
9. Waktu Akhir (waktu saat proses berakhir)

Informasi lain yang disajikan adalah panjang rute, yang ditampilkan dalam bentuk grafik garis sesuai dengan panjang rute yang dihasilkan pada setiap kali perhitungan atau iterasi. Garis-garis yang muncul pada grafik tersebut diberi warna sesuai dengan warna yang telah ditentukan untuk masing-masing algoritma. Dimana perhitungan akan terus dilakukan sampai iterasi ke-300, namun dalam rancang bangun *software* simulasi ini iterasi dapat dihentikan dengan menekan tombol "Stop".



Gambar 4. Grafik pada Proses Perhitungan dengan Berbagai Algoritma

Bila ingin mencetak hasil yang diperoleh ke dalam suatu file, disediakan tombol *"Print Result".* Hasil analisa dan penambahan data ke dalam form "*Print Result",* dapat dilihat kemudian dicetak secara otomatis akan terekam pada folder result dengan format notepad seperti gambar di bawah ini :



hasilbaru.rst - Notepad

File Edit Format View Help

Algoritma	Nama File	Panjang Rute	Jumlah Iterasi	Waktu(detik)	Iterasi Terbaik
Ant Colony System	A010.MAP	318.8411	300	1	3
Local Search	A010.MAP	318.8411	300	0	4
Simple Insertion	A010.MAP	318.8411	300	1	1
Local Search	A010.MAP	318.8411	300	0	9
Nearest Neighbor With Local Search	A010.MAP	318.8412	10	0	1
Nearest Neighbor With Local Search	A050.MAP	567.1292	50	0	24
ACS With Local Search	A050.MAP	567.129	42	4	3
Local Search	A050.MAP	555.0292	300	2	276

Gambar 5. Tampilan Hasil Analisa

3.4 Implementasi Algoritma ke dalam Program Aplikasi

Penentuan letak titik-titik kota yang akan ditampilkan, didasarkan dari peta yang dibuat pada *For*m Buat Peta. Adapun source kode pembuatan random untuk letak kota tersebut dapat dilihat dibawah ini:

```
Set ftsp = fobj.CreateTextFile(strfile)
For i = 0 To Val(Text1.Text) - 1
   Do
     test = 1
     ax = Int(99 * Rnd) + 1
     ay = Int(99 * Rnd) + 1
     If i > 0 Then
       For j = 0 To i - 1
         If x(j) = ax And y(j) = ay Then
           test = 0
     Exit
```

Dalam penentuan titik-titik kota yang dilakukan secara random atau acak, akan tercipta jarak antar titik-titik kota tersebut. Untuk menentukan jarak tersebut dalam program ini akan digunakan rumus *Phytagoras.* Rumus tersebut adalah sebagai berikut:

```
distc(i, j) = Sqr((city(i).X - city(j).X) ^ 2 + (city(i).Y - city(j).Y) ^ 2)
```
(2)

Keterangan:

Dist(i,j) = jarak antara titik i dan j

Sqr = *Square* (akar)

City(i).X = koordinat titik Xi

City(j).X = koordinat titik Xj

City(i).Y = koordinat titik Yi

City(j).Y = koordinat titik Yj

Adapun algoritma ACS yang digunakan dalam simulasi ini adalah :

Memberi nilai parameter-parameter yang digunakan
Menghitung jarak masing-masing kota
Menghitung *Pheromone* awal di tiap-tiap jalur antar kota
Perulangan selama ctr < jumlah iterasi
    Perulangan sampai semua kota dikunjungi
        Perulangan sampai semua semut sudah berpindah kota
            Pilih kunjungan kota berikutnya dengan menerapkan rumus eq1 dan rumus eq3
            Tambahkan kota yang dipilih pada *array* tour
        Selesai perulangan
            Update array Pheromone pada jalur yang baru saja dilewati oleh masing-masing semut
        Selesai perulangan
            Selesai perulangan

3.5 Hasil Percobaan Perbandingan Kinerja Algoritma-algoritma yang digunakan dalam Pemecahan Beberapa Kasus TSP

Dalam penelitian dilakukan percobaan untuk menghitung panjang rute dengan mebandingkan ACS dengan ke-3 algoritma lain. Hasil-hasil percobaan dapat dilihat pada tabel berikut ini.

Tabel 2. Tabel hasil percobaan untuk kasus 25 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	**b**	**c**	**d**	**a**	**b**	**c**	**d**	**a**	**b**	**c**	**d**	**a**	**b**	**c**	**d**
1	**430.696**	3	3	300	**430.696**	1	22	300	**430.696**	0	239	300	**430.696**	0	13	25
2	**430.696**	3	1	300	**430.696**	1	146	300	**430.696**	0	18	300	**430.696**	0	13	25
3	**430.696**	4	1	300	**430.696**	0	183	300	**430.696**	0	48	300	**430.696**	1	13	25
4	**430.696**	3	3	300	**430.696**	0	4	300	**430.696**	0	1	300	**430.696**	0	13	25
Rata-rata	**430.696**	3.25	2	300	**430.696**	0.5	88.75	300	**430.696**	0	76.5	300	**430.696**	0.25	13	25

Tabel 3. Tabel hasil percobaan untuk kasus 50 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	570.389	10	5	300	**555.029**	1	162	300	562.331	1	122	300	567.129	0	29	50
2	555.029	10	6	300	**555.029**	1	75	300	**555.029**	1	8	300	567.129	0	29	50
3	567.129	10	3	300	**555.029**	1	145	300	567.473	0	269	300	567.129	1	29	50
4	567.129	10	3	300	**555.029**	1	203	300	563.282	1	144	300	567.129	0	29	50
Rata-rata	564.919	10	4.25	300	**555.029**	1	146.25	300	562.029	0.75	135.75	300	567.129	0.25	29	50

Tabel 4. Tabel hasil percobaan untuk kasus 75 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	673.208	21	3	300	**666.319**	2	85	300	**663.319**	2	46	300	673.208	0	54	75
2	**657.136**	26	5	300	664.853	2	276	300	662.379	1	19	300	673.208	1	54	75
3	**658.736**	23	6	300	667.348	3	64	300	664.343	1	64	300	673.208	0	54	75
4	**666.804**	20	5	300	668.076	3	53	300	667.510	2	92	300	673.208	1	54	75
Rata-rata	**663.971**	22.5	4.75	300	666.649	2.5	119.5	300	664.388	1.5	55.25	300	673.208	0.5	54	75

Tabel 5. Tabel hasil percobaan untuk kasus 100 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	**714.6435**	158	3	300	715.9755	13	11	300	732.665	6	143	300	715.7684	4	46	100
2	**713.8851**	165	2	300	722.818	15	48	300	722.58	6	257	300	715.7684	3	52	100
3	**713.8852**	172	4	300	730.8488	14	277	300	727.167	6	202	300	715.7684	3	3	100
4	718.4748	132	2	300	725.5053	14	111	300	725.7	6	35	300	**715.4877**	3	71	100
Rata-rata	**715.222**	156.8	2.75	300	723.787	14	111.75	300	727.028	6	159.25	300	715.698	3.25	43	100

Tabel 6. Tabel hasil percobaan untuk kasus 150 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	**956.3019**	647	6	300	972.0409	34	253	300	980.829	13	190	300	964.6393	12	49	150
2	**946.6044**	476	10	300	974.7029	43	46	300	970.882	13	229	300	962.3762	13	87	150
3	**951.6656**	493	6	300	977.7162	33	83	300	986.321	13	103	300	966.7644	13	37	150
4	**951.7402**	522	9	300	979.0405	33	29	300	982.741	13	48	300	969.8222	13	106	150
Rata-rata	**951.578**	534.5	7.75	300	975.875	35.8	102.75	300	980.193	13	142.5	300	965.901	12.8	69.8	150

Tabel 7. Tabel hasil percobaan untuk kasus 200 kota dengan 4 macam algoritma

P	ACS				Local Search				Simple Insertion				Nearest Neighbor with Local Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	**1029.256**	1609	10	300	1074.583	64	214	300	1074.999	23	104	300	1048.901	30	13	200
2	**1025.054**	2150	12	300	1072.726	63	67	300	1058.964	23	69	300	1045.519	30	179	200
3	**1029.014**	2318	8	300	1069.638	62	138	300	1073.839	23	113	300	1049.803	31	52	200
4	**1035.475**	1466	9	300	1075.895	63	101	300	1074.002	23	69	300	1049.803	30	144	200
Rata-rata	**1029.700**	1886	9.8	300	1073.211	63	130	300	1070.451	23	88.75	300	1048.507	30.3	97	200

Keterangan tabel :
P = Percobaan
a = Panjang rute
b = Waktu (dalam satuan detik)
c = Iterasi Terbaik
d = Jumlah iterasi maksimum
Angka yang berwarna tebal menunjukkan hasil terbaik dalam satu kali percobaan

3.6 Evaluasi Hasil Percobaan Kinerja Algoritma-algoritma yang Digunakan dalam Pemecahan Beberapa Kasus TSP

Melihat informasi-informasi pada tabel 2 sampai dengan tabel 7, dapat disimpulkan bahwa didapatkan panjang rute yang sama pada jumlah kota 25 (tabel 2). Karena jarak yang dihasilkan adalah sama, maka untuk memilih algoritma mana yang sebaiknya digunakan, adalah membandingkan iterasi terbaik dan waktu yang diperlukan. Sehingga akhirnya diperoleh bahwa Algoritma *Simple Insertion* lebih baik dari ke 3 algoritma yang lain.

Untuk jumlah kota lebih besar pada suatu kasus TSP, maka algoritma ACS akan menghasilkan hasil yang semakin baik ditinjau dari optimalisasi jarak tempuh. Itu terbukti dari contoh kasus dengan jumlah kota 75, 100, 150 dan 200 (tabel 3 sampai dengan tabel 7). Selisih panjang rute terbaik yang dihasilkan oleh algoritma ACS dibanding dengan beberapa algoritma lain yang digunakan dalam penelitian ini cukup besar.

4. KESIMPULAN

Berdasarkan hasil analisa menggunakan *software* simulasi pendukung keputusan yang telah dirancang bangun, maka dapat disimpulkan bahwa :

1. Algoritma yang baik digunakan jika ditinjau dari jarak tempuh yang dihasilkan adalah Algoritma ACS. Algoritma ACS menghasilkan solusi jarak yang lebih baik daripada algoritma yang lain. Hal ini dapat dilihat pada hasil analisa di kota yang berjumlah 75, 100, 150, dan 200.
2. Algoritma yang lebih baik digunakan jika jarak tempuh yang dihasilkan sama adalah Algoritma *Simple Insertion*. Hal ini didasarkan pada hasil analisa kota berjumlah 25, dengan hasil dari ke-4 algoritma menunjukkan jarak tempuh yang sama yaitu 430,696. Namun dilihat dari segi iterasi terbaik dan waktu yang digunakan pada percobaan ke-4, algoritma *Simple Insertion* menghasilkan solusi lebih baik dengan waktu yang digunakan 0, dan iterasi terbaiknya 1.

DAFTAR PUSTAKA

[1] Applegate, D., Bixby, R., Chvátal V. and Cook, W. 2001. " ***Application of the TSP*** ".

[2] Budi Setyo. 2008. " **Perbandingan algoritma *ant colony system d*engan *simple insertion, local search,* dan *nearest neighbor with local search* pada kasus *traveling salesman problem*** ". Universitas Dian Nuswantoro. Semarang (Skripsi), Tidak diterbitkan.

[3] Colorni, A., Dorigo, M., Maffioli, Maniezzo, F. V., Righini, G. and Trubian, M. 1996. " **Heuristics from Nature for Hard Combiatorial Optimization Problems** ". Dipartomento di Elettronica e Informazione. Politecnico di Milano. Italy.
[4] Dakin, Robert. 1997. " ***The Travelling Salesman Problem*** "
[5] Dorigo, M. and Gambardella, L. M.1997. " ***Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem*** ". *IEEE Transactions on Evolutionary Computation*, Vol.1(1). Universite Libre de Bruxelles,Belgium.
[6] Gould, C. G. 2001. *Pheromone*. Research Associate, Department of Biology. Princeton University.
[7] Jonhson, D. S. and Mc Geoch, L. A. 1995. " **The Travelling Salesman Problem : A Case Study in Local Optimization Aarts** "*, E. H. L. and Lenstra, J. K. (ed.) Local Search in Combinatorial Optimization.* John Wiley and Sons. Ltd., London.
[8] Kumalasari, S., Setiawan, B., Liong, J. 2002. " **Perbandingan Algoritma Ant Colony Dengan Beberapa Algoritma Lain** ". Universitas Bina Nusantara. Jakarta (Skripsi), Tidak diterbitkan.
[9] *Mertens, S. 1995.* " **TSP Algorithms in Action Animated Examples of Heuristi Algorithms** ".
[10] Rich, E. and Knight, K. 1991. " ***Artificial Intelligence*** ". International Edition. McGrow-Hill Book Co. New York.
[11] Sismoro, H. 2005. " **Pengantar Logika Informatika, Algoritma, dan Pemrograman Komputer** ", Penerbit ANDI. Yogyakarta.
[12] Stutzle, T. and Dorigo, M. 1999. " ***ACO* Algorithms for the Traveling Salesman Problem**". Iridia. Universite Libre de Bruxelles. Belgium.