ALGORITMA *ANT COLONY OPTIMIZATION* (ACO) UNTUK MENYELESAIKAN *TRAVELING SALESMAN PROBLEM* (TSP)

SKRIPSI

Oleh :

Agus Leksono

J2A 002 002

PROGRAM STUDI MATEMATIKA JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS DIPONEGORO

SEMARANG

2009

ALGORITMA *ANT COLONY OPTIMIZATION* (ACO) UNTUK MENYELESAIKAN *TRAVELING SALESMAN PROBLEM* (TSP)

Agus Leksono
J2A 002 002

Skripsi
Diajukan sebagai syarat untuk memperoleh gelar Sarjana Sains
pada
Program Studi Matematika

PROGRAM STUDI MATEMATIKA JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS DIPONEGORO
SEMARANG
2009

HALAMAN PENGESAHAN

Judul : Algoritma *Ant Colony Optimization* (ACO) untuk Menyelesaikan *Traveling Salesman Problem* (TSP)

Nama : Agus Leksono

NIM : J2A 002 002

Telah diujikan pada sidang Tugas Akhir tanggal 10 Juni 2009

dan dinyatakan **lulus** pada tanggal 16 Juni 2009

Semarang, 16 Juni 2009
Panitia Penguji Tugas Akhir
Ketua,

Drs. Sarwadi, MSc. PhD
NIP. 131 835 919

Mengetahui,
Ketua Jurusan Matematika
FMIPA UNDIP

Dr. Widowati, S.Si. M.Si
NIP. 132 090 819

Mengetahui,
Ketua Program Studi Matematika
Jurusan Matematika FMIPA UNDIP

Bambang Irawanto, S.Si. M.Si
NIP. 132 102 826

HALAMAN PENGESAHAN

Judul : Algoritma *Ant Colony Optimization* (ACO) untuk Menyelesaikan *Traveling Salesman Problem* (TSP)

Nama : Agus Leksono

NIM : J2A 002 002

Telah diujikan pada sidang Tugas Akhir tanggal 10 Juni 2009

Semarang, 16 Juni 2009

Pembimbing Utama,	Pembimbing Anggota,
Drs. Sarwadi, MSc. PhD	Drs. Bayu Surarso, MSc.PhD
NIP. 131 835 919	NIP. 131 764 886

ABSTRAK

Traveling Salesman Problem (TSP) merupakan persoalan yang penting dalam sistem distribusi. Masalah *traveling salesman* secara umum digambarkan sebagai suatu kasus dimana seseorang harus mengunjungi sejumlah kota dari suatu pusat fasilitas dan kembali lagi ke tempat pemberangkatan semula, dengan asumsi jarak diketahui. Tujuan dari masalah ini adalah untuk meminimalkan total jarak tempuh salesman. Masalah *traveling salesman* termasuk kelas *NP-Hard*, sehingga sangat sulit untuk diselesaikan menggunakan algoritma eksak. Untuk menyelesaikan masalah tersebut biasanya digunakan algoritma heuristik. Algoritma *Ant Colony Optimization* (ACO) merupakan salah satu metode metaheuristik yang menerapkan semut sebagai agen dengan update *Pheromone*-nya untuk dapat melakukan proses pencarian solusi yang efektif dan efisien. Algoritma ACO yang dibandingkan sebanyak lima yaitu *Ant System* (AS), *Elitist Ant System* (EAS), *Rank-based Ant System* (AS_{Rank}), *Max-min Ant System* (*MMAS*), dan *Ant Colony System* (ACS). Simulasi dilakukan dengan mencari solusi mendekati optimal dari beberapa kasus TSP dengan jumlah titik n = 20 sampai n = 115. Hasil mendekati optimal diperoleh dengan melakukan beberapa kali percobaan untuk setiap kasus. Selanjutnya hasil perhitungan untuk setiap kasus dan setiap algoritma dibandingkan. Hasil perbandingan kelima algoritma ACO tersebut, terlihat bahwa untuk jumlah titik sampai n = 40 solusi yang dihasilkan semua algoritma sama baiknya. Untuk kasus dengan jumlah titik yang lebih banyak algoritma ACS mempunyai solusi yang terbaik dan algoritma AS yang terjelek dari kelima algoritma tersebut.

Kata kunci : *Traveling Salesman Problem, Ant Colony Optimization*, *Ant System*, *Elitist Ant System*, *Rank-based Ant System*, *Max-min Ant System*, *Ant Colony System.*

ABSTRACT

The traveling salesman problem (TSP) is one of the important topics in distribution system. Generally, this problem is viewed as a case which a salesman must visit number of city from a centralized facility and back (return) to the initial city, such that the total travel distance is minimized. The traveling salesman problem is an NP-Hard class problem in optimization. It is very difficult to solve this problem using exact method. Usually, it is solved by heuristic approaches. *Ant Colony Optimization* Algorithms (ACO) is a meta-heuristic method, algorithms applies ant as agent with update Pheromone in search solution process with efficient and effectively. ACO algorithms are compared they are *Ant System* (AS), *Elitist Ant System* (EAS), *Rank-based Ant System* (AS_{Rank}), *Max-min Ant System* (MMAS), and *Ant Colony System* (ACS). A simulation of implementation ACO algorithms is performed to solve TSP cases ranging the size from n = 20 to n = 115. Near optimal solution are obtained by calculating every cases repeatedly. Futhermore, the process of calculating of the solution for every cases and every algorithm is compared. This comparison shows that the solutions of all algorithm are same for problem us to n = 40 nodes. ACS algorithm's solution is the best and AS algorithm's solution is the worst for problem with bigger number of nodes (cities).

Keywords: *Traveling Salesman Problem, Ant Colony Optimization*, *Ant System*, *Elitist Ant System*, *Rank-based Ant System*, *Max-min Ant System*, *Ant Colony System.*

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-NYA sehingga penulis dapat menyelesaikan tugas akhir ini. Sholawat dan salam penulis sampaikan kepada Rasulullah SAW beserta keluarganya, sahabatnya, dan orang-orang yang tetap istiqomah dalam mengikuti sunnahnya.

Tugas Akhir dengan judul **"Algoritma *Ant Colony Optimization* (ACO) Untuk Menyelesaikan *Traveling Salesman Problem* (TSP)"** disusun sebagai salah satu syarat untuk memperoleh gelar sarjana strata satu pada Fakultas Matematika dan Ilmu Pengetahuan Alam di Universitas Diponegoro Semarang.

Pada kesempatan ini penulis mengucapkan terima kasih kepada :

1. Dr. Widowati, S.Si. M.Si selaku ketua jurusan Matematika FMIPA UNDIP.
2. Drs. Sarwadi, M.Sc, PhD selaku dosen pembimbing I yang dengan sabar membimbing dan mengarahkan penulis hingga selesainya tugas akhir ini.
3. Drs. Bayu Surarso, M.Sc, PhD selaku dosen pembimbing II yang dengan sabar membimbing dan mengarahkan penulis hingga selesainya tugas akhir ini.
4. Bapak dan Ibu Dosen Jurusan Matematika FMIPA UNDIP dimana penulis mendapatkan ilmu pengetahuan dan pengalaman yang sangat berharga.
5. Bapak dan ibunda tercinta yang selalu memberikan segalanya untuk putranya tercinta. Juga kakak-kakakku dan keponakan – keponakanku yang lucu dan

buat orang gemes, serta dindaq yang selalu menemani dan memberi semangat.

6. Teman – temanku semua dan semua pihak yang telah membantu hingga selesainya tugas akhir ini. Yang tidak dapat penulis sebutkan satu persatu. Semoga Allah membalas segala kebaikan yang telah anda berikan kepada penulis. Amin.

Penulis menyadari bahwa tugas akhir ini masih jauh dari sempurna. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan. Semoga tulisan ini bermanfaat untuk semua.

Semarang, Juni 2009

Penulis

DAFTAR ISI

DAFTAR SIMBOL

1. α : parameter pengendali intensitas jejak semut (*Pheromone*)
2. β : parameter pengendali visibilitas
3. ρ : parameter penguapan (*evaporasi*) *Pheromone* global
4. ε : parameter penguapan (*evaporasi*) *Pheromone* lokal
5. e : parameter yang menyatakan adanya tour terbaik pada EAS
6. J : variabel random yang dipilih berdasarkan P_{rs}^{k}
7. k : indeks semut
8. m : jumlah semut
9. n : jumlah titik (kota)
10. w : parameter yang menyatakan tour terbaik pada AS_{Rank}
11. z : peringkat semut pada AS_{Rank}
12. q : bilangan random pemilihan titik berikutnya pada ACS
13. Q : tetapan iterasi semut
14. *avg* : rata-rata jumlah dari pilihan berbeda semut dalam menemukan tour terbaiknya
15. η_{rs} : visibilitas antara titik r dan titik s
16. τ_0 : intensitas jejak semut (*Pheromone*) awal
17. τ_{rs} : intensitas jejak semut (*Pheromone*) antara titik r dan titik s
18. $\tau_{\min}$: batas bawah nilai *Pheromone* pada algoritma *MMAS*
19. $\tau_{\max}$: batas atas nilai *Pheromone* pada algoritma *MMAS*

20. ρ_{best} : parameter evaporasi terbaik $\mathcal{MMAS}$ untuk menhitung $\tau_{\min}$

21. $\Delta\tau_{rs}^{bs}$: perubahan nilai *Pheromone* pada edge (r,s) dalam tour terbaik untuk algoritma EAS

22. $\Delta\tau_{rs}^{k}$: perubahan nilai *Pheromone* pada edge (r,s) yang dilakukan oleh semut k

23. $\Delta\tau_{rs}^{z}$: perubahan nilai *Pheromone* pada edge (r,s) yang dilakukan oleh semut dengan peringkat z pada AS_{Rank}

24. $\Delta\tau_{rs}^{best}$: perubahan nilai *Pheromone* edge (r,s) dalam tour terbaik

25. C^{k} : panjang tour yang dilalui semut k

26. C^{z} : panjang tour yang dilalui semut berperingkat z pada algoritma AS_{Rank}

27. C^{ib} : *iteration best-tour* pada $\mathcal{MMAS}$

28. C^{bs} : panjang tour terbaik keseluruhan yang ditemukan sejak awal algoritma dijalankan

29. C^{nn} : panjang tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic*

30. C^{best} : *best so-far-tour* pada $\mathcal{MMAS}$

31. T^{bs} : tour terbaik pada algoritma EAS

32. d_{rs} : jarak (panjang) antara titik r dan titik s

33. d_{sr} : jarak (panjang) antara titik s dan titik r

34. r_k : titik dimana semut k berada

35. $J_k\left(r_{k_1}\right)$: himpunan titik-titik yang belum dan akan dikunjungi oleh

semut k yang berada dititik r_{k_1}

36. $J_k(s_k)$: himpunan titik-titik yang belum dan akan dikunjungi oleh semut k yang berada dititik s_k

37. q_0 : parameter pembanding nilai random (q) nilainya $0 \le q_0 \le 1$

38. P_{rs}^k : probabilitas semut k pada titik r yang memilih titik s

39. J_r^k : himpunan titik yang akan dikunjungi semut k yang berada pada titik r

40. NC_{max} : jumlah iterasi maksimum

41. $Tabu_k$: *tabu list* untuk semut k

42. $Tabu_k(r)$: elemen ke r dari $Tabu_k$, yaitu titik ke-r yang dikunjungi semut k pada sebuah tour

43. $d_{tabu_k(s),tabu_k(s+1)}$: jarak edge dari titik s sampai s+1 pada tabu list yang diperoleh semut k

44. $d_{tabu_k(n),tabu_k(1)}$: jarak edge dari titik n sampai 1 pada tabu list yang diperoleh semut k

45. G = (V,E) : graph dengan himpunan vertek V dan himpunan edge E

46. E : himpunan edge E

47. V : himpunan vertek V

48. G'= (V',E') : subgraph dari graph G

49. E' : himpunan bagian edge dari edge E

50. V' : himpunan bagian vertek dari vertek V

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Traveling Salesman Problem (TSP) dikenal sebagai salah satu masalah optimasi yang banyak menarik perhatian para peneliti sejak beberapa dekade terdahulu. Pada mulanya, TSP dinyatakan sebagai permasalahan dalam mencari jarak minimal sebuah tour tertutup terhadap sejumlah *n* kota dimana kota-kota yang ada hanya dikunjungi sekali dengan kota awal juga merupakan kota akhir (tujuan). Meskipun TSP sepertinya mudah dinyatakan, akan tetapi sangat sulit untuk diselesaikan terutama untuk persoalan dengan jumlah kota yang banyak. Sampai saat ini belum ada suatu metode eksak yang dapat menjamin keberbhasilan nilai optimal untuk sebarang masalah dalam *Polynomial computation time*. TSP dikarakteristikan kedalam kelas **NP-*hard*** (*Nondeterministik Polynomial – hard*). Berdasarkan hal tersebut, banyak peneliti lebih memusatkan kepada pengembangan metode-metode pendekatan (*heuristic)* seperti *simulated annealing*, algoritma semut (*Ant Algorithm*), Algoritma Genetika, *Tabu search*, dan lain sebagainya.

Pada perkembangannya, ternyata TSP merupakan persoalan yang banyak diaplikasikan pada berbagai persoalan dunia nyata. Hingga saat ini, TSP diaplikasikan pada persoalan perencanaan pembangunan,

perencanaan produksi, rute pengambilan surat dari kotak pos, rute pengisian uang pada mesin ATM, rute patroli polisi, rute pesawat terbang dsb.

Ant Colony Optimization (ACO) diadopsi dari perilaku koloni semut yang dikenal sebagai system semut (Dorigo, M., dan Gambardella, L., 1996). Secara alamiah koloni semut mampu menemukan rute terpendek dalam perjalanan dari sarang ke tempat-tempat sumber makanan. Koloni semut dapat menemukan rute terpendek antara sarang dan sumber makanan berdasarkan jejak kaki pada lintasan yang telah dilalui. Semakin banyak semut yang melalui suatu lintasan, maka akan semakin jelas bekas jejak kakinya. Hal ini akan menyebabkan lintasan yang dilalui semut dalam jumlah sedikit, semakin lama akan semakin berkurang kepadatan semut yang melewatinya, atau bahkan akan tidak dilewati sama sekali. Sebaliknya, lintasan yang dilalui semut dalam jumlah banyak, semakin lama akan semakin bertambah kepadatan semut yang melewatinya, atau bahkan semua semut akan melalui lintasan tersebut.

Pada ACO setiap semut ditempatkan di semua titik graph (dalam hal ini titik – titik yang dikunjungi) yang kemudian akan bergerak mengunjungi seluruh titik. Setiap semut akan membuat jalur masing – masing sampai kembali ketempat semula dimana mereka ditempatkan pertama kali. Jika sudah mencapai keadaan ini, maka semut telah menyelesaikan sebuah siklus

(tour). Solusi akhir adalah jalur terpendek dari seluruh jalur yang dihasilkan oleh pencarian semut tersebut.

Algoritma ACO telah banyak diaplikasikan dalam berbagai bidang yang mencakup beberapa persoalan, yaitu :

1. *Traveling Salesman Problem* (TSP), yaitu mencari rute terpendek dalam sebuah graph menggunakan rute Hamilton.
2. *Quadratic Assignment Problem* (QAP), yaitu menugaskan sejumlah n resources untuk ditempatkan pada sejumlah m lokasi dengan meminimalisasi biaya penugasan (*assignment*).
3. *Job-shop Scheduling Problem* (JSP) juga salah satu contoh aplikasi *Ant Colony Optimization,* yaitu untuk mencari lintasan sejumlah n pekerjaan menggunakan sejumlah m mesin demikian sehingga seluruh pekerjaan diselesaikan dalam waktu yang seminimal mungkin.
4. *Vehicle Routing Problem (VRP)*
5. Pengaturan rute kendaraan
6. Pewarnaan graph
7. Implementasi pada jaringan komunikasi
8. *Network routing*, dll.

Mengingat prinsip algoritma yang didasarkan pada perilaku koloni semut dalam menemukan jarak perjalanan paling pendek tersebut, ACO sangat tepat digunakan untuk diterapkan dalam penyelesaian masalah

optimasi, salah satunya adalah untuk menentukan jalur terpendek pada permasalahan TSP.

1.2 Rumusan Masalah

Travelling Salesman Problem termasuk ke dalam masalah NP-hard, sehingga sangat sulit untuk mencari solusi dari masalah ini dengan pendekatan eksak. Untuk menyelesaikan TSP digunakan pendekatan secara heuristic. Salah satu metode heuristic yang dikenal adalah Algoritma *Ant Colony Optimization*. Permasalahan dari penulisan Tugas Akhir ini adalah mengetahui bagaimana performa dari algoritma – algoritma ACO (*Ant system, Elitist Ant System, Rank Based Ant System, Max-Min Ant System, dan Ant Colony System*) untuk menyelesaikan TSP simetrik dan mengetahui manakah yang terbaik diantara algoritma tersebut.

1.3 Batasan Masalah

Dalam tugas akhir ini masalah hanya akan dibatasi sebagai berikut :

1. Algoritma ACO yang dibandingkan adalah *Ant System, Elitist Ant System, Rank-Based Ant System, Max-Min Ant System,* dan *Ant Colony System.*
2. Permasalahan yang dibandingkan dalam semua algoritma adalah masalah *Travelling Salesman Problem* (TSP) Simetrik Euclidean dimana jarak antar 2 kota dan sebaliknya dianggap sama.

1.4 Tujuan Penulisan

Tujuan dari penulisan tugas akhir ini adalah untuk :

1. Menerapkan konsep dan cara kerja algoritma *Ant Colony Optimization* (ACO) untuk menyelesaikan masalah *Travelling Salesman Problem* (TSP).
2. Mengetahui perbandingan Algoritma *Ant system, Elitist Ant System, Rank Based Ant System, Max-Min Ant System,* dan *Ant Colony System*, dalam menyelesaikan TSP Simetrik.

1.5 Sistematika Penulisan

Sistematika penulisan Tugas Akhir ini terdiri dari BAB I sebagai pendahuluan yang berisi tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan masalah, dan sistematika penulisan. BAB II yaitu dasar teori yang terdiri dari teori graph, optimisasi, *Traveling Salesman Problem, Ant Colony Optimization, Ant system, Elitist Ant System, Rank-Based Ant System, Max-Min Ant System,* dan *Ant Colony System.* BAB III yaitu pembahasan yang terdiri dari Algoritma *Ant Colony Optimization, Ant system, Elitist Ant System, Rank Based Ant System, Max-Min Ant System,* dan *Ant Colony System* dalam menyelesaikan permasalahan TSP. Perbandingan Algoritma – algoritma ACO berdasarkan kinerjanya, cara kerjanya, parameter terbaik, performa hasil pada beberapa kasus, waktu CPU, jumlah iterasi, dan hasil simulasi algoritma ACO menggunakan

ACOTSP *version* 1.0. BAB IV sebagai penutup yang berisi tentang kesimpulan dari pembahasan pada bab sebelumnya dan saran.

BAB II

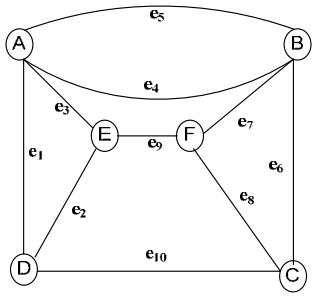
DASAR TEORI

2.1 Teori Graph

2.1.1 Definisi Graph

Definisi 2.1 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Sutatu graph G terdiri atas himpunan yang tidak kosong dari elemen – elemen yang disebut **titik** (vertek), dan suatu daftar pasangan vertek yang tidak terurut disebut **sisi** (edge). Himpunan vertek dari suatu graph G dinotasikan dengan V, dan daftar himpunan edge dari graph tersebut dinotasikan dengan E. Untuk selanjutnya suatu graph G dapat dinotasikan dengan G = (V, E).

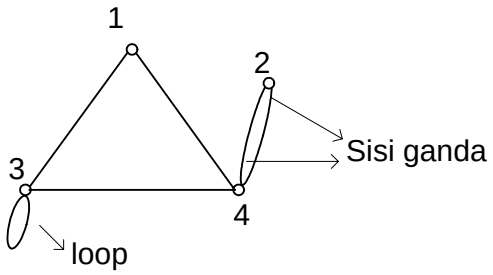


Gambar 2.1 Contoh graph G

Gambar 2.1, menunjukkan graph G dengan V = {A, B, C, D, E, F} dan E = $\{e_1, e_2, e_3, \ldots, e_{10}\}$.

Definisi 2.2 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Dua edge atau lebih yang menghubungkan pasangan vertek yang sama disebut **sisi ganda**, dan sebuah edge yang mengubungkan sebuah vertek ke dirinya sendiri disebut **loop**.

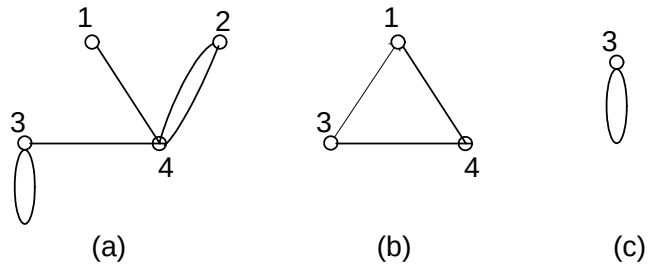


Gambar 2.2 Sisi ganda dan loop

Definisi 2.3 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Misal G suatu graph dengan himpunan vertek V dan himpunan edge E. Suatu **subgraph G'** adalah suatu himpunan pasangan berurutan (V', E') dimana V' merupakan himpunan bagian dari V dan E' adalah himpunan bagian dari E. Dengan kata lain, subgraph dari G adalah suatu graph yang semua verteknya anggota V dan semua edgenya anggota E.

Jika G suatu graph terhubung seperti pada gambar 2.2, dengan V = {1, 2, 3, 4} dan E = {(1,3), (1,4), (2,4), (3,3), (3,4), (4,2)}, maka berikutnya adalah contoh dari subgraph G' yang ditunjukkan pada gambar 2.3.



Gambar 2.3 Contoh graph G dan subgraph G'

Pada gambar 2.3 (a) merupakan subgraph G' dari graph G, dengan himpunan vertek V' = {1, 2, 3, 4} yang merupakan himpunan bagian dari V dan himpunan edge E' = {(1,3), (1,4), (2,4), (3,3), (3,4), (4,2) yang merupakan himpunan bagian dari E. Gambar 2.3 (b) juga merupakan subgraph G' dari graph G dengan himpunan vertek V' = {1, 3, 4} dan himpunan edge E' = {(1,3), (1,4), (3,4)} yang masing – masing merupakan himpunan bagian dari V dan E. Gambar 2.3 (c) juga merupakan subgraph G' dari graph G dengan himpunan vertek V' = {3} dan himpunan edge E' = {(3,3)} yang masing – masing merupakan himpunan bagian dari V dan E.

2.1.2 Definisi Walk

Definisi 2.4 **(Evans, J. R dan Edward, M, 1992)**

Suatu **walk** (jalan) dalam graph G adalah barisan vertek – vertek dan edge – edge yang dimulai dan diakhiri oleh suatu vertek. Panjang suatu walk dihitung berdasarkan jumlah edge dalam walk tersebut.

Walk juga dapat diartikan sebagai suatu perjalanan (dalam sebuah graph) dari vertek satu ke vertek lain yang terhubung dengan suatu edge.

2.1.3 Definisi Trail dan Path

Definisi 2.5 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Walk yang panjangnya k pada suatu graph G adalah urutan k edge G yang berbentuk

uv, vw, wx, …, yz.

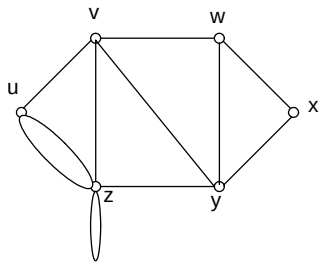
Walk ini dinotasikan dengan uvwx….yz, dan ditunjuk sebagai walk antara u dan z. Jika semua edge (tetapi tidak perlu semua vertek) suatu walk berbeda, maka walk itu disebut **trail**. Jika semua vertek pada trail itu berbeda, maka trail itu disebut **path**.

2.1.4 Definisi Cycle

Definisi 2.6 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Walk tertutup dalam graph G merupakan urutan edge G yang berbentuk uv, vw, wx, …, yz, zu.

Jika semua edgenya berbeda, maka walk itu disebut **trail tertutup** (*close trail*). Kemudian *close trail* dengan semua vertek berbeda disebut **cycle**.



Gambar 2.4 walk uvwxywvzzy

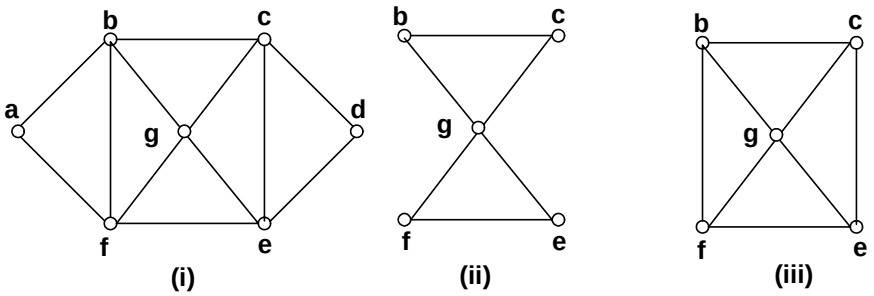
uvwxywvzzy adalah walk yang panjangnya 9 antara u dan y, yang memuat edge vw dua kali, titik v, w, y, dan z dua kali. Pada gambar 2.4 di atas, walk vzzywxy merupakan trail, sedang pada walk vwxyz merupakan path. Kemudian uvwyzvzu merupakan *close trail*, sedang *close trail* zz, vwxyv, dan vwxyzv semuanya adalah cycle.

2.1.5 Graph Eulerian dan Graph Hamiltonian

Definisi 2.7 **(Wilson, R. J dan Watkhins, J. J, 1990)**

Graph terhubung G merupakan **graph Euler** (*Eulerian*) jika ada *close trail* yang memuat setiap edge dari G. Trail semacam ini disebut **trail Euler**.

Graph terhubung G merupakan **graph Hamilton** (*Hamiltonian*) jika ada cycle yang memuat setiap vertek dari G. Cycle semacam ini disebut **cycle Hamilton**.



Gambar 2.5 Graph Hamilton dan graph Euler

Graph (i) merupakan graph Euler dan graph Hamilton,

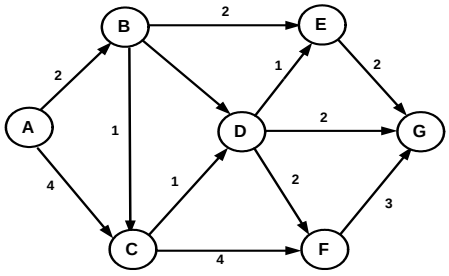
Graph (i) merupakan graph Euler dan trail Eulernya bcgfeb,

Graph (i) merupakan graph Hamilton dengan cycle Hamiltonnya bcgefb.

2.1.6 Macam – macam Graph Menurut arah dan Bobotnya

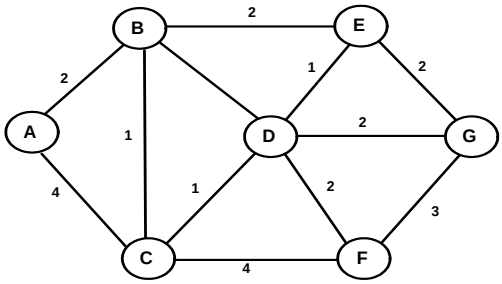
Menurut arah dan bobotnya, graph dibagi menjadi empat bagian, yaitu :

1. Graph berarah dan berbobot : setiap edge mempunyai arah (yang ditunjukkan dengan anak panah) dan bobot. Gambar 2.6 adalah contoh graph berarah dan berbobot yang terdiri dari tujuh vertek yaitu vertek A, B, C, D, E, F, G. Vertek A mempunyai dua edge yang masing – masing menuju ke vertek B dan vertek C, vertek B mempunyai tiga edge yang masing – masing menuju ke vertek C, vertek D dan vertek E. Bobot antara vertek A dan vertek B pun telah di ketahui.



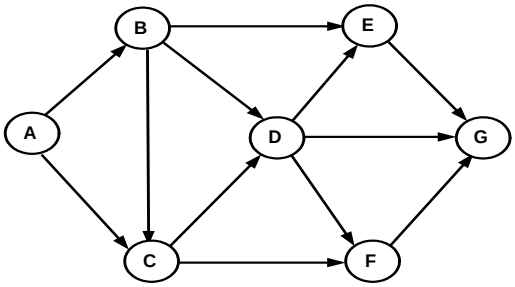
Gambar 2.6 Graph berarah dan berbobot

2. Graph tidak berarah dan berbobot : setiap edge tidak mempunyai arah tetapi mempunyai bobot. Gambar 2.7 adalah contoh graph tidak berarah dan berbobot. Graph terdiri dari tujuh vertek yaitu vertek A, B, C, D, E, F, G. Vertek A mempunyai dua edge yang masing – masing berhubungan dengan vertek B dan vertek C, tetapi dari masing – masing edge tersebut tidak mempunyai arah. Edge yang menghubungkan vertek A dan vertek B mempunyai bobot yang telah diketahui begitu pula dengan edge – edge yang lain.



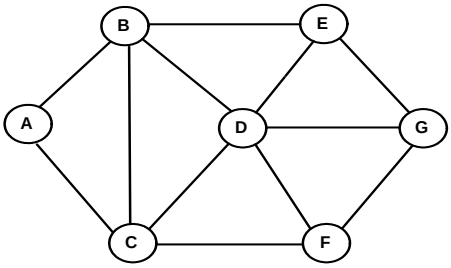
Gambar 2.7 Graph tidak berarah dan berbobot

3. Graph berarah dan tidak berbobot : setiap edge mempunyai arah tetapi tidak mempunyai bobot. Gambar 2.8 adalah contoh graph berarah dan tidak berbobot.



Gambar 2.8 Graph berarah dan tidak berbobot

4. Graph tidak berarah dan tidak berbobot : setiap edge tidak mempunyai arah dan tidak terbobot. Gambar 2.9 adalah contoh graph tidak berarah dan tidak berbobot.



Gambar 2.9 Graph tidak berarah dan tidak berbobot

2.2 Optimisasi

2.2.1 Definisi Masalah Optimisasi

Optimisasi adalah suatu proses untuk mencapai hasil yang optimal (nilai efektif yang dapat dicapai). Dalam disiplin matematika optimisasi

merujuk pada studi permasalahan yang mencoba untuk mencari nilai minimal atau maksimal dari suatu fungsi riil. Untuk dapat mencapai nilai optimal baik minimal atau maksimal tersebut, secara sistematis dilakukan pemilihan nilai variabel integer atau riil yang akan memberikan solusi optimal (Wardy, I.S, Jurnal Prodi TI ITB, 2007).

2.2.2 Definisi Nilai Optimal

Nilai optimal adalah nilai yang didapat melalui suatu proses dan dianggap menjadi solusi jawaban yang paling baik dari semua solusi yang ada (Wardy, I.S, Jurnal Prodi TI ITB, 2007).

2.2.3 Macam – macam Permasalahan Optimisasi

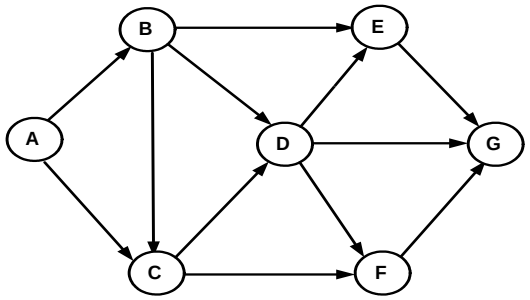
Permasalahan yang berkaitan dengan optimisasi sangat kompleks dalam kehidupan sehari-hari. Nilai optimal yang didapat dalam optimisasi dapat berupa besaran panjang, waktu, jarak, dan lain-lain. Berikut ini adalah termasuk beberapa persoalan optimisasi :

1. Menentukan lintasan terpendek dari suatu tempat ke tempat yang lain.
2. Menentukan jumlah pekerja seminimal mungkin untuk melakukan suatu proses produksi agar pengeluaran biaya pekerja dapat diminimalkan dan hasil produksi tetap maksimal.
3. Mengatur rute kendaraan umum agar semua lokasi dapat dijangkau.
4. Mengatur *routing* jaringan kabel telepon agar biaya pemasangan kabel tidak terlalu besar dan penggunaannya tidak boros.

Selain beberapa contoh di atas, masih banyak persoalan lainnya yang terdapat dalam berbagai bidang.

2.2.4 Permasalahan Rute Terpendek

Masalah rute terpendek merupakan masalah yang berkaitan dengan penentuan edge-edge dalam sebuah jaringan yang membentuk rute terdekat antara sumber dan tujuan. Tujuan dari permasalahan rute terpendek adalah mencari rute yang memiliki jarak terdekat antara titik asal dan titik tujuan. Gambar 2.10 merupakan suatu graph ABCDEFG yang berarah dan tidak berbobot.



Gambar 2.10. Graph ABCDEFG

Gambar 2.10 diatas, misalkan kita dari kota A ingin menuju Kota G. Untuk menuju kota G, dapat dipilih beberapa rute yang tersedia :

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow G$$

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow G$$

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow G$$

$$A \rightarrow B \rightarrow C \rightarrow F \rightarrow G$$

$$A \rightarrow B \rightarrow D \rightarrow E \rightarrow G$$

$$A \rightarrow B \rightarrow D \rightarrow F \rightarrow G$$

$$A \rightarrow B \rightarrow D \rightarrow G$$

$$A \rightarrow B \rightarrow E \rightarrow G$$

$$A \rightarrow C \rightarrow D \rightarrow E \rightarrow G$$

$$A \rightarrow C \rightarrow D \rightarrow F \rightarrow G$$

$$A \rightarrow C \rightarrow D \rightarrow G$$

$$A \rightarrow C \rightarrow F \rightarrow G$$

Berdasarkan data diatas, dapat dihitung rute terpendek dengan mencari jarak antara rute-rute tersebut. Apabila jarak antar rute belum diketahui, jarak dapat dihitung berdasarkan koordinat kota-kota tersebut, kemudian menghitung jarak terpendek yang dapat dilalui.

2.2.5 Penyelesaian Masalah Optimisasi

Secara umum, penyelesaian masalah pencarian rute terpendek dapat dilakukan dengan menggunakan dua metode, yaitu metode konvensional dan metode heuristik. Metode konvensional dihitung dengan perhitungan matematis biasa, sedangkan metode heuristik dihitung dengan menggunakan system pendekatan.

1. Metode Konvensional

   Metode konvensional adalah metode yang menggunakan perhitungan matematika eksak. Ada beberapa metode konvensional yang biasa digunakan untuk melakukan pencarian rute terpendek, diantaranya:

algoritma Djikstra, algoritma Floyd-Warshall, dan algoritma Bellman-Ford (Mutakhiroh, I., Saptono, F., Hasanah, N., dan Wiryadinata, R., 2007).

2. Metode Heuristik

   Metode Heuristik adalah suatu metode yang menggunakan system pendekatan dalam melakukan pencarian dalam optimasi. Ada beberapa algoritma pada metode heuristik yang biasa digunakan dalam permasalahan optimasi, diantaranya Algoritma Genetika, *Ant Colony Optimization*, logika *Fuzzy*, jaringan syaraf tiruan, *Tabu Search*, S*imulated Annealing*, dan lain-lain (Mutakhiroh, I., Saptono, F., Hasanah, N., dan Wiryadinata, R., 2007).

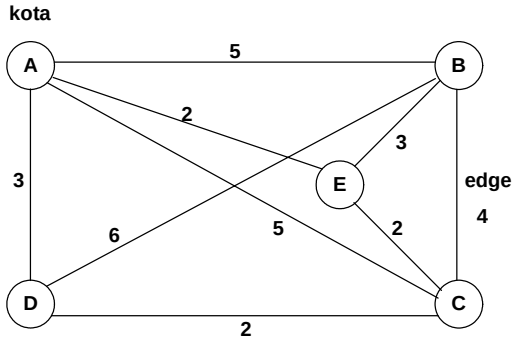
2.3 *Traveling Salesman Problem (TSP)*

Masalah *Travelling Salesman Problem* (TSP) adalah salah satu contoh yang paling banyak dipelajari dalam *combinatorial optimization.* Masalah ini mudah untuk dinyatakan tetapi sangat sulit untuk diselesaikan. TSP termasuk kelas NP-*Hard problem* dan tidak dapat diselesaikan secara optimal dalam *Polynomial computation time* dengan algoritma eksak. Bila diselesaikan secara eksak waktu komputasi yang diperlukan akan meningkat secara eksponensial seiring bertambah besarnya masalah.

TSP dapat dinyatakan sebagai permasalahan dalam mencari jarak minimal sebuah tour tertutup terhadap sejumlah *n* kota dimana kota-kota yang ada hanya dikunjungi sekali. TSP direpresentasikan dengan

menggunakan sebuah graph lengkap dan berbobot G = (V, E) dengan V himpunan vertek yang merepresentasikan himpunan titik - titik, dan E adalah himpunan dari edge. Setiap edge (r,s) $\in$ E adalah nilai (jarak) d_{rs} yang merupakan jarak dari kota r ke kota s, dengan (r,s) $\in$ V. Dalam TSP simetrik (jarak dari kota *r* ke titik s sama dengan jarak dari titik s ke titik r), $d_{rs} = d_{sr}$ untuk semua edge (r,s) $\in$ E. Misalkan terdapat n buah titik maka graph tersebut memiliki $\left(\frac{n!}{((n-2)!2!)}\right)$ buah edge, sesuai dengan rumus kombinasi, dan juga memiliki $\frac{(n-1)!}{2}$ buah tour yang mungkin.

Dalam sebuah graph, TSP digambarkan seperti gambar 2.11 dibawah ini :

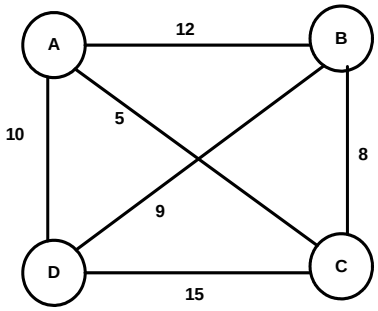


Gambar 2.11 Ilustrasi masalah TSP

Berikut adalah contoh kasus TSP: "Diberikan sejumlah kota dan jarak antar kota. Tentukan sirkuit terpendek yang harus dilalui oleh seorang pedagang bila pedagang itu berangkat dari sebuah kota asal dan menyinggahi setiap kota tepat satu kali dan kembali lagi ke kota asal keberangkatan."

Apabila kita mengubah contoh kasus tersebut menjadi persoalan pada graph, maka dapat dilihat bahwa kasus tersebut adalah bagaimana menentukan sirkuit Hamilton yang memiliki bobot minimum pada graph tersebut.

Seperti di ketahui, bahwa untuk mencari jumlah sirkuit Hamilton di dalam graph lengkap dengan n vertek adalah : (*n* - 1)!/2.



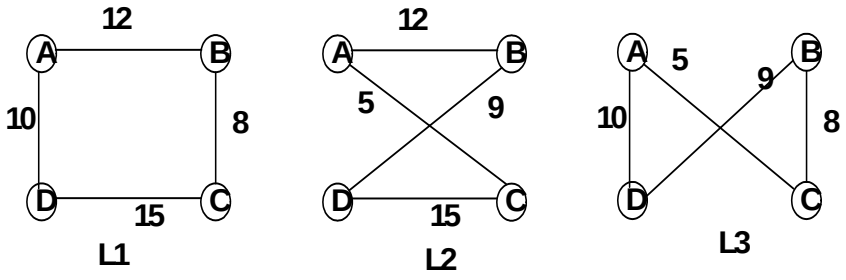
Gambar 2.12 Graph ABCD

Pada gambar 2.12 diatas, graph memiliki $\frac{(4-1)!}{2} = 3$ sirkuit Hamilton, yaitu:

L1 = (*A*,*B*,*C*,*D*,*A*) atau (*A*,*B*,*C*,*D*,*A*) => panjang = 10 + 12 + 8 + 15 = 45

L2 = (*A*,*C*,*D*,*B*,*A*) atau (*A*,*B*,*D*,*C*,*A*) => panjang = 12 + 5 + 9 + 15 = 41

L3 = (*A*,*C*,*B*,*D*,*A*) atau (*A*,*D*,*B*,*C*,*A*) => panjang = 10 + 5 + 9 + 8 = 32



Gambar 2.13 Sirkuit hamilton

Pada gambar 2.13 diatas terlihat jelas bahwa sirkuit Hamilton terpendek adalah L3 = (*A, C, B, D, A*) atau (*A, D, B, C, A*) dengan panjang sirkuit = 10 + 5 + 9 + 8 = 32. Jika jumlah vertek *n* = 20 akan terdapat (19!)/2 sirkuit Hamilton atau sekitar 6×10^{16} penyelesaian.

Dalam kehidupan sehari-hari, kasus TSP ini dapat diaplikasikan untuk menyelesaikan kasus lain, diantaranya yaitu :

1. Tukang Pos mengambil surat di kotak pos yang tersebar pada *n* buah lokasi di berbagai sudut kota.
2. Lengan robot mengencangkan *n* buah mur pada beberapa buah peralatan mesin dalam sebuah jalur perakitan.
3. Mobil pengangkut sampah mengambil sampah pada tempat – tampat pembuangan sampah yang berada pada n buah lokasi diberbagai sudut kota.
4. Petugas Bank melakukan pengisian uang pada sejumlah mesin ATM di n buah lokasi.
5. Dan lain sebagainya.

2.4 *Ant Colony Optimization (ACO)*

Ant Colony Optimization (ACO) diadopsi dari perilaku koloni semut yang dikenal sebagai sistem semut (Dorigo, et.al, 1996). Semut mampu mengindera lingkungannya yang kompleks untuk mencari makanan dan kemudian kembali ke sarangnya dengan meninggalkan zat *Pheromone* pada rute-rute yang mereka lalui.

Pheromone adalah zat kimia yang berasal dari kelenjar endokrin dan digunakan oleh makhluk hidup untuk mengenali sesama jenis, individu lain, kelompok, dan untuk membantu proses reproduksi. Berbeda dengan hormon, *Pheromone* menyebar ke luar tubuh dan hanya dapat mempengaruhi dan dikenali oleh individu lain yang sejenis (satu *spesies*).

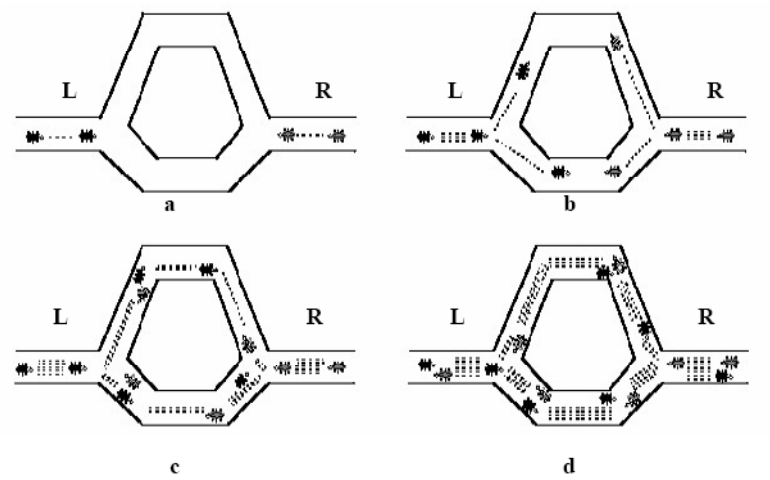
Proses peninggalan *Pheromone* ini dikenal sebagai *stigmery*, yaitu sebuah proses memodifikasi lingkungan yang tidak hanya bertujuan untuk mengingat jalan pulang ke sarang, tetapi juga memungkinkan para semut berkomunikasi dengan koloninya.

Seiring waktu, bagaimanapun juga jejak *Pheromone* akan menguap dan akan mengurangi kekuatan daya tariknya. Lebih cepat setiap semut pulang pergi melalui rute tersebut, maka *Pheromone* yang menguap lebih sedikit. Begitu pula sebaliknya jika semut lebih lama pulang pergi melalui rute tersebut, maka *Pheromone* yang menguap lebih banyak.

2.4.1 Cara kerja semut menemukan rute terpendek dalam ACO

Secara jelasnya cara kerja semut menemukan rute terpendek dalam ACO adalah sebagai berikut : Secara alamiah semut mampu menemukan rute terpendek dalam perjalanan dari sarang ke tempat-tempat sumber makanan. Koloni semut dapat menemukan rute terpendek antara sarang dan sumber makanan berdasarkan jejak kaki pada lintasan yang telah dilalui. Semakin banyak semut yang melalui suatu lintasan, maka akan semakin jelas bekas jejak kakinya. Hal ini akan menyebabkan lintasan

yang dilalui semut dalam jumlah sedikit, semakin lama akan semakin berkurang kepadatan semut yang melewatinya, atau bahkan akan tidak dilewati sama sekali. Sebaliknya lintasan yang dilalui semut dalam jumlah banyak, semakin lama akan semakin bertambah kepadatan semut yang melewatinya, atau bahkan semua semut akan melalui lintasan tersebut (Dorigo, M., Maniezzo, V., dan Colorni, A., 1991a).



Gambar 2.14. Perjalanan semut dari sarang ke sumber makanan.

Gambar 2.14.a di atas menunjukkan ada dua kelompok semut yang akan melakukan perjalanan. Satu kelompok bernama L yaitu kelompok yang berangkat dari arah kiri yang merupakan sarang semut dan kelompok lain yang bernama kelompok R yang berangkat dari kanan yang merupakan sumber makanan. Kedua kelompok semut dari titik awal keberangkatan sedang dalam posisi pengambilan keputusan jalan sebelah mana yang akan diambil. Kelompok semut L membagi dua kelompok lagi. Sebagian melalui jalan atas dan sebagian melalui jalan bawah. Hal ini juga

berlaku pada kelompok semut R. Gambar 2.14.b dan gambar 2.14.c menunjukkan bahwa kelompok semut berjalan pada kecepatan yang sama dengan meninggalkan *Pheromone* (jejak kaki semut) di jalan yang telah dilalui. *Pheromone* yang ditinggalkan oleh semut - semut yang melalui jalan atas telah mengalami banyak penguapan karena semut yang melalui jalan atas berjumlah lebih sedikit dari pada jalan yang di bawah. Hal ini dikarenakan jarak yang ditempuh lebih panjang daripada jalan bawah. Sedangkan *Pheromone* yang berada di jalan bawah, penguapannya cenderung lebih lama. Karena semut yang melalui jalan bawah lebih banyak daripada semut yang melalui jalan atas. Gambar 2.14.d menunjukkan bahwa semut-semut yang lain pada akhirnya memutuskan untuk melewati jalan bawah karena *Pheromone* yang ditinggalkan masih banyak. Sedangkan *Pheromone* pada jalan atas sudah banyak menguap sehingga semut-semut tidak memilih jalan atas tersebut. Semakin banyak semut yang melalui jalan bawah maka semakin banyak semut yang mengikutinya.

Demikian juga dengan jalan atas, semakin sedikit semut yang melalui jalan atas, maka *Pheromone* yang ditinggalkan semakin berkurang bahkan hilang. Dari sinilah kemudian terpilihlah rute terpendek antara sarang dan sumber makanan.

2.5 *Ant System (AS)*

Algoritma *Ant System* (AS) adalah algoritma ACO pertama yang dirumuskan dan diuji untuk menyelesaikan kasus TSP (Dorigo, M., Maniezzo, V., dan Colorni, A., 1996). Algoritma ini tersusun atas sejumlah m semut yang bekerjasama dan berkomunikasi secara tidak langsung melalui komunikasi *Pheromone*.

Secara informal, AS bekerja sebagai berikut : Setiap semut memulai tournya melalui sebuah titik yang dipilih secara acak (setiap semut memiliki titik awal yang berbeda). Secara berulang kali, satu-persatu titik yang ada dikunjungi oleh semut dengan tujuan untuk menghasilkan sebuah tour. Pemilihan titik-titik yang akan dilaluinya didasarkan pada suatu fungsi probabilitas, dinamai aturan transisi status (*state transition rule*), dengan mempertimbangkan *visibility* (invers dari jarak) titik tersebut dan jumlah *Pheromone* yang terdapat pada ruas yang menghubungkan titik tersebut. Semut lebih suka untuk bergerak menuju ke titik-titik yang dihubungkan dengan ruas yang pendek dan memiliki tingkat *Pheromone* yang tinggi (Dorigo, M., dan Gambardella, L. M., 1997). Setiap semut memiliki sebuah memori, dinamai *tabulist*, yang berisi semua titik yang telah dikunjunginya pada setiap tour. *Tabulist* ini mencegah semut untuk mengunjungi titik-titik yang sebelumnya telah dikunjungi selama tour tersebut berlangsung, yang membuat solusinya mendekati optimal.

Setelah semua semut menyelesaikan tour mereka dan *tabulist* mereka menjadi penuh, sebuah aturan pembaruan *Pheromone* global (*global Pheromone updating rule*) diterapkan pada setiap semut. Penguapan *Pheromone* pada semua ruas dilakukan, kemudian setiap semut menghitung panjang tour yang telah mereka lakukan lalu meninggalkan sejumlah *Pheromone* pada edge-edge yang merupakan bagian dari tour mereka yang sebanding dengan kualitas dari solusi yang mereka hasilkan. Semakin pendek sebuah tour yang dihasilkan oleh setiap semut, jumlah *Pheromone* yang ditinggalkan pada edge-edge yang dilaluinya pun semakin besar. Dengan kata lain, edge-edge yang merupakan bagian dari tour-tour terpendek adalah edge-edge yang menerima jumlah *Pheromone* yang lebih besar. Hal ini menyebabkan edge-edge yang diberi *Pheromone* lebih banyak akan lebih diminati pada tour-tour selanjutnya, dan sebaliknya edge-edge yang tidak diberi *Pheromone* menjadi kurang diminati. Dan juga, rute terpendek yang ditemukan oleh semut disimpan dan semua *tabu list* yang ada dikosongkan kembali.

Peranan utama dari penguapan *Pheromone* adalah untuk mencegah stagnasi, yaitu situasi dimana semua semut berakhir dengan melakukan tour yang sama. Proses di atas kemudian diulangi sampai tour-tour yang dilakukan mencapai jumlah maksimum atau sistem ini menghasilkan perilaku stagnasi dimana sistem ini berhenti untuk mencari solusi alternatif.

2.5.1 Aturan Transisi Status

Aturan transisi status yang digunakan oleh AS dinamai *random-proportional rule* (Dorigo, M., Maniezzo, V., dan Colorni, A., 1996), yang ditunjukkan oleh persamaan (2.1). P_{rs}^{k} merupakan probabilitas dari semut *k* pada titik r yang memilih untuk menuju ke titik s.

$$P_{rs}^{k} = \begin{cases} \dfrac{[\tau_{rs}]^{\alpha}.[\eta_{rs}]^{\beta}}{\sum_{u \in J_r^k}[\tau_{ru}]^{\alpha}.[\eta_{ru}]^{\beta}} & untuk\ \ s \in J_r^k \\ 0 & untuk\ \ s\ \ \ lainnya \end{cases} \quad \text{.....(2.1)}$$

Dimana τ_{rs} adalah jumlah *Pheromone* yang terdapat pada edge antara titik r dan titik s, $(\eta_{rs}) = \dfrac{1}{d_{rs}}$ adalah *visibility* (invers dari jarak d_{rs}) dimana $d_{rs} = \sqrt{(x_r - x_s)^2 + (y_r - y_s)^2}$ (jika hanya diketahui koordinat titiknya saja). α adalah sebuah parameter yang mengontrol bobot (*weight*) relatif dari *Pheromone* dan β adalah parameter pengendali jarak ($\alpha > 0 \ \ dan \ \ \beta > 0$).

J_r^k adalah himpunan titik yang akan dikunjungi oleh semut *k* yang sedang berada pada titik r (untuk membuat solusinya mendekati optimal), dan pada persamaan (2.1) kita mengalikan *Pheromone* pada edge (r,s) dengan nilai *visibility* yang sesuai (η_{rs}). Dengan cara ini kita memilih edge yang lebih pendek dan memiliki jumlah *Pheromone* yang lebih besar.

2.5.2 *Update Pheromone Trail*

Setelah semua semut menyelesaikan tour-nya masing – masing maka *Pheromone* di-*update*. Dalam *Ant System*, aturan pembaruan *Pheromone global* (Dorigo, M., Maniezzo, V., dan Colorni, A., 1996) diimplementasikan menurut persamaan 2.2 sebagai berikut :

$$\tau_{rs} \leftarrow (1-\rho).\tau_{rs} + \sum_{k=1}^{m} \Delta\tau_{rs}^{k} \qquad \text{.....(2.2)}$$

$$\text{dengan } \Delta\tau_{rs}^{k} = \begin{cases} \dfrac{1}{C^{k}} & \text{jika } (r,s) \in \text{tour yang dilakukan oleh semut k} \\ 0 & \text{sebaliknya} \end{cases} \qquad \text{.....(2.3)}$$

Dimana C^k, panjang tour yang dilalui oleh semut k. $0 < \rho \leq 1$ adalah sebuah parameter tingkat *evaporasi Pheromone*, dan *m* adalah jumlah dari semut.

Untuk memastikan bahwa semut mengunjungi n titik yang berbeda, diberikan *tabu list* pada masing – masing semut, yaitu sebuah struktur data yang menyimpan titik – titik yang telah dikunjungi semut dan melarang semut mengunjungi kembali titik – titik tersebut sebelum mereka menyelesaikan sebuah tour. Ketika sebuah tour selesai, *tabulist* digunakan untuk menghitung solusi yang ditemukan semut pada tour tersebut. *Tabulist* kemudian dikosongkan dan semut kembali bebas memilih titik tujuannya pada tour berikutnya. $Tabu_k$ adalah *tabu list* untuk semut k. $Tabu_k$ (r) adalah elemen ke-r dari $Tabu_k$, yaitu titik ke-r yang dikunjungi semut k pada suatu tour.

2.6 *Elitist Ant System (EAS)*

Pengembangan pertama dari AS adalah *elitist strategy for Ant System* (EAS), seperti yang dikemukakan Dorigo, M., Maniezzo, V., dan Colorni, A. (1991a), (1991b), dan (1996). Ide ini berawal ketika adanya penguatan *Pheromone* pada edge – edge yang merupakan tour terbaik yang ditemukan sejak awal algoritma. Tour terbaik ini dinotasikan sebagai T^{bs} (*best-so-far tour*).

2.6.1 *Update Pheromone Trail*

Penambahan intensitas *Pheromone* dari tour T^{bs} adalah dengan memberi penambahan quantity $\frac{e}{C^{bs}}$ untuk setiap edge, dimana *e* parameter yang diberikan untuk mendefinisikan nilai tour terbaik (T^{bs}) dan C^{bs} adalah panjang tour terbaik. Perubahan *Pheromone* didefinisikan sebagai berikut :

$$\tau_{rs} \leftarrow (1-\rho).\tau_{rs} + \sum_{k=1}^{m} \Delta\tau_{rs}^{k} + e\Delta\tau_{rs}^{bs} \qquad \text{..... (2.4)}$$

Dimana $\Delta\tau_{rs}^{k}$ didefinisikan pada pers. (2.3) dan $\Delta\tau_{rs}^{bs}$ didefinisikan sebagai berikut :

$$\Delta\tau_{rs}^{bs} = \begin{cases} \frac{1}{C^{bs}}, & jika \quad edge(r,s) \quad terdapat \quad pada \quad T^{bs} \\ 0, & yang \quad lainnya \end{cases} \qquad \text{..... (2.5)}$$

Sebagai catatan untuk EAS, bagian dari algoritma yang lain sama seperti pada AS, yang dibahas pada bagian sebelumnya.

2.7 *Rank-Based Ant System (AS_{Rank})*

Rank-based version of Ant System (AS_{Rank}) ((Bullnheimer, B., Hartl, R. F., dan Strauss, C., 1997) dan (1999)) merupakan pengembangan dari AS dan menerapkan *elitist strategy.* Pada setiap iterasi, metode ini lebih dahulu mengurutkan semut berdasarkan tingkat fluktuasi solusi (panjang/pendeknya tour) yang telah mereka temukan sebelumnya.

2.7.1 *Update Pheromone Trail*

Saat melakukan *update Pheromone* hanya (w-1) semut terbaik dan semut yang memiliki solusi *best-so-far* yang diperbolehkan meninggalkan *Pheromone.* Semut yang ke-*z* terbaik memberikan kontribusi *Pheromone* sebesar max{0, w-*z}* sementara jalur *best-so-far tour* memberikan kontribusi *Pheromone* paling besar yaitu sebanyak w, dimana w adalah parameter yang menyatakan adanya tour terbaik dan z adalah peringkat semut. Dalam AS_{rank} aturan *Pheromone update*-nya diberikan sebagai berikut :

$$\tau_{rs} = (1-\rho)\tau_{rs} + \sum_{z=1}^{w-1}(w-z)\Delta\tau_{rs}^{z} + w\Delta\tau_{rs}^{bs}, \qquad \text{..... (2.6)}$$

Dimana $\Delta\tau_{rs} = \frac{1}{C^{z}}$ dan $\Delta\tau_{rs}^{bs} = \frac{1}{C^{bs}}$. C^{z} adalah panjang tour yang dilewati semut ke-z, C^{bs} adalah panjang tour terbaik. Hasil dari evaluasi eksperimen oleh Bullnheimer, B., Hartl, R. F., dan Strauss, C., (1999). menunjukkan AS_{rank} mempunyai hasil yang lebih baik daripada EAS dan lebih signifikan daripada AS.

2.8 *MAX – MIN Ant System (MMAS)*

MAX-MIN Ant System (*MMAS*) merupakan pengembangan dari algoritma AS selanjutnya, dengan beberapa perubahan utama. Berikut empat perubahan utama didalam *MMAS* (Stu¨ tzle, T., dan Hoos, H. H., 1997) terhadap AS :

- Pertama, penambahan *Pheromone* bisa dilakukan pada edge – edge yang merupakan bagian dari tour terbaik yang ditemukan sejak awal algoritma (*best so-far*-tour) atau pada tour terbaik yang ditemukan pada iterasi tersebut (*iteration best*-tour). Bisa juga penambahan *Pheromone* pada keduanya, *best so-far*-tour dan *iteration best*-tour sekaligus. Tetapi, strategi ini memungkinkan terjadinya stagnasi yang menyebabkan semua semut melalui jalur yang sama, karena pemberian *Pheromone* yang berlebihan pada edge, meskipun bagian dari tour yang terbaik.
- Kedua, untuk mengatasi masalah pada perubahan pertama, maka *MMAS* memberikan batasan dalam pemberian nilai *Pheromone* dengan interval $[\tau_{\min}, \tau_{\max}]$.
- Ketiga, menginisialisasi *Pheromone* dengan batas atas nilai *Pheromone*, yang mana bersama dengan tingkat *evaporasi Pheromone* yang kecil, meningkatkan eksplorasi tour sejak dimulainya pencarian.
- Keempat, *Pheromone* di inisialisasi kembali pada saat terjadi stagnasi atau ketika tidak ditemukan tour yang sesuai dengan iterasi yang diinginkan.

2.8.1 *Update Pheromone*

Setelah semua semut membangun tournya, *Pheromone* di-*update* menurut persamaan sebagai berikut :

$$\tau_{rs} \leftarrow (1-\rho).\tau_{rs} + \Delta\tau_{rs}^{best} \qquad \text{.....(2.7)}$$

dimana $\Delta\tau_{rs}^{best} = \begin{cases} 1/C^{best}, & \text{jjika semut menemukan best so - far - tour} \\ 1/C^{ib}, & \text{jika semut menemukan iteration best - tour} \end{cases}$

Dengan C^{best} merupakan panjang tour terbaik dan C^{ib} adalah panjang iterasi terbaik sebuah tour. Pada umumnya, *MMAS* mengimplementasikan keduanya baik iterasi terbaik maupun panjang tour terbaiknya.

2.9 *Ant Colony System (ACS)*

Algoritma *Ant Colony System* (ACS) merupakan pengembangan dari AS selanjutnya, setelah beberapa algoritma diatas. Algoritma ini tersusun atas sejumlah m semut yang bekerjasama dan berkomunikasi secara tidak langsung melalui komunikasi *Pheromone*.

Secara informal, ACS bekerja sebagai berikut: pertama kali, sejumlah *m* semut ditempatkan pada sejumlah *n* titik berdasarkan beberapa aturan inisialisasi (misalnya, secara acak). Setiap semut membuat sebuah tour (yaitu, sebuah solusi TSP yang mungkin) dengan menerapkan sebuah aturan transisi status secara berulang kali. Selagi membangun tournya, setiap semut juga memodifikasi jumlah *Pheromone* pada edge-edge yang

dikunjunginya dengan menerapkan aturan pembaruan *Pheromone* lokal yang telah disebutkan tadi. Setelah semua semut mengakhiri tour mereka, jumlah *Pheromone* yang ada pada edge-edge dimodifikasi kembali (dengan menerapkan aturan pembaruan *Pheromone* global). Seperti yang terjadi pada *Ant system*, dalam membuat tour, semut 'dipandu' oleh informasi *heuristic* (mereka lebih memilih edge-edge yang pendek) dan oleh informasi *Pheromone*. Sebuah edge dengan jumlah *Pheromone* yang tinggi merupakan pilihan yang sangat diinginkan. Kedua aturan pembaruan *Pheromone* itu dirancang agar semut cenderung untuk memberi lebih banyak *Pheromone* pada edge-edge yang harus mereka lewati. Berikutnya akan dibahas mengenai tiga karakteristik utama dari ACS, yaitu aturan transisi status, aturan pembaharuan *Pheromone* global, dan aturan pembaharuan *Pheromone* lokal.

2.9.1 Aturan Transisi Status

Aturan transisi status yang berlaku pada ACS (Dorigo, M., dan Gambardella, L., 1996) ditunjukkan pada persamaan 2.8. Semut k yang berada di titik r, akan memilih titik berikutnya s, menurut persamaan berikut :

$$s = \begin{cases} \arg\max_{u \in J_r^k} \left\{ \tau_{ru} \left[\eta_{ru} \right]^{\beta} \right\} & jika\ q \le q_0\ (ekploitasi) \\ J, & jika\ tidak\ (eksplorasi) \end{cases} \quad \text{.....(2.8)}$$

Dimana q adalah bilangan random dalam [0,1], $q_0 \left(0 \le q_0 \le 1\right)$ adalah sebuah parameter pembanding bilangan random, dan J = P_{rs}^k probabilitas

dari semut *k* pada titik r yang memilih untuk menuju ke titik s (persamaan 2.1).

Dengan kata lain, Jika $q \le q_0$ maka semut tersebut akan memanfaatkan pengetahuan heuristik tentang jarak antara titik tersebut dengan titik-titik lainnya dan juga pengetahuan yang telah didapat dan disimpan dalam bentuk *Pheromone*. Hal ini mengakibatkan edge terbaik (berdasarkan persamaan (2.8)) dipilih. Jika sebaliknya maka sebuah edge dipilih berdasarkan persamaan (2.1).

2.9.2 *Global Pheromone Update*

Setelah semua semut menyelesaikan sebuah tour, tingkat *Pheromone* di-*update* dengan mengaplikasikan *global updating rule* (Dorigo, M., dan Gambardella, L., 1996) menurut persamaan berikut :

$$\tau_{rs} \leftarrow (1-\rho).\tau_{rs} + \rho.\Delta\tau_{rs}^{bs} \qquad \text{.....(2.9)}$$

$$\text{dengan } \Delta\tau_{rs}^{bs} = \begin{cases} \dfrac{1}{C^{bs}} & \text{jika } (r,s) \in \text{lintasan terbaik keseluruhan} \\ 0 & \text{jika tidak} \end{cases} \qquad \text{.....(2.10)}$$

Dimana ρ adalah parameter *evaporasi global*, yang mempunyai nilai $0 < \rho < 1$. $\Delta\tau_{rs}^{bs}$ adalah $\dfrac{1}{(\text{panjang lintasan terbaik keseluruhan})}$, jika (i,j) merupakan bagian panjang lintasan terbaik keseluruhan (C^{bs}), dan 0 jika tidak.

Persamaan *update* jejak *Pheromone* secara *offline* ini, dilakukan pada akhir sebuah iterasi algoritma, saat semua semut telah menyelesaikan

sebuah tour. Persamaan diaplikasikan ke edge yang digunakan semut menemukan lintasan keseluruhan yang terbaik sejak awal percobaan. Tujuan pemberian nilai ini adalah memberi sejumlah jejak *Pheromone* pada lintasan terpendek, dimana tour terbaik (lintasan dengan panjang terpendek) mendapat penguatan. Bersama dengan *pseudo-random-proportional rule* dimaksudkan untuk membuat pencarian lebih terarah.

2.9.3 *Local Pheromone Update*

Ketika membangun solusi (tour) dari TSP, semut mengaplikasikan *local updating rule* (Dorigo, M., dan Gambardella, L., 1996) menurut persamaan berikut :

$$\tau_{rs} \leftarrow (1-\xi).\tau_{rs} + \xi.\tau_0 \qquad \text{……..……..….(2.11)}$$

ξ adalah parameter *evaporasi* lokal $0 < \xi < 1$. τ_0 adalah nilai awal jejak *Pheromone*, $\tau_0 = \frac{1}{nC^{nn}}$ dimana n adalah jumlah titik dan C^{nn} adalah panjang sebuah tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic* [Rosenkrantz, D.J, Stearns, R.E, dan Lewis, P.M. (1977)].

Persamaan *update Pheromone online* ini diaplikasikan saat semut membangun tour TSP, yaitu ketika melewati edge dan mengubah tingkat *Pheromone* pada edge (r,s). Tujuannya untuk membantu melewati sebuah edge, edge ini menjadi kurang diinginkan (karena berkurangnya jejak *Pheromone* pada edge yang bersesuaian).

BAB III

PEMBAHASAN

3.1 *Ant Colony Optimization (ACO)* untuk *Traveling Salesman Problem* (TSP)

ACO diaplikasikan dalam TSP dengan cara sebagai berikut :

- *Pertama* adalah dengan mengkonstruksikan masalah ke dalam sebuah graph G = (V, E) dengan V himpunan vertek yang merepresentasikan himpunan titik – titik, dan E adalah himpunan dari edge yang merepresentasikan jarak antara dua titik.
- *Kedua*, kendala yang terdapat pada TSP yaitu mengunjungi n titik dengan titik-titik yang ada hanya dikunjungi sekali dimana titik awal sama dengan titik akhir. Tujuan dari TSP yaitu mencari tour terpendek terhadap *n* titik.
- *Ketiga*, pemberian nilai intensitas jejak semut (*Pheromone*) dan informasi heuristik. Pemberian nilai *Pheromone*(τ_{rs}) dalam TSP dilakukan saat semut mengunjungi titik s setelah mengunjungi titik r. Informasi heuristik (η_{rs}) merupakan informasi yang merepresentasikan kualitas suatu edge antara titik r dan titik s, informasi ini dihitung sebelum algoritma dijalankan. Dengan $\eta_{rs} = \frac{1}{d_{rs}}$, d_{rs} adalah jarak antara titik r dan titik s.

- *Keempat,* (*tour construction*). Sebuah tour dibangun dengan mengaplikasikan prosedur sederhana sebagai berikut : Inisialisasi, ditempatkan m semut di n titik menurut aturan tertentu, kemudian semut mengaplikasikan *state transition rule* secara iteratif. Semut membangun lintasan sebagai berikut. Pada titik r, semut memilih secara probabilistik titik s yang belum dikunjungi menurut intensitas *Pheromone* (τ_{rs}) pada egde antara titik r ke titik s, serta informasi heuristik lokal yang ada, yaitu panjang sisi (egde). Semut secara probabilistik lebih menyukai titik yang dekat dan terhubung dengan tingkat *Pheromone* yang tinggi. Untuk membangun jalur terpendek yang mungkin, setiap semut mempunyai suatu bentuk memori yang disebut *tabu list. Tabu list* digunakan untuk menentukan himpunan titik yang masih harus dikunjungi pada setiap langkah dan untuk menjamin terbentuknya jalur terpendek yang mungkin. Selain itu semut bisa melacak kembali lintasannya, ketika sebuah lintasan itu diselesaikan. Setelah semua semut membangun sebuah tour, *Pheromone* di-*update* dengan cara mengurangi tingkat *Pheromone* oleh suatu faktor konstanta dan kemudian semut meletakkan *Pheromone* pada edge yang dilewati. *Update* dilakukan sedemikian rupa sehingga edge dari lintasan yang lebih pendek dan dilewati banyak semut menerima jumlah *Pheromone* yang lebih banyak. Karena itu pada iterasi algoritma yang berikutnya akan mempunyai probabilitas yang lebih tinggi untuk dipilih.

Secara umum algoritma ACO untuk TSP mengikuti skema algoritma (Dorigo, M. dan Socha, K., 2007) sebagai berikut :

procedure ACO Algorithm for TSPs

Penetapan parameters, initialize *Pheromone* trails

while (termination condition not met) **do**

ConstructAntSolutions

ApplyLocalSearch (optional)

UpdatePheromones

Endwhile

End procedure ACO Algorithm for TSPs

3.2 *Ant System (AS)* untuk *Traveling Salesman Problem* (TSP)

3.2.1 Langkah – langkah dalam menyelesaikan AS untuk TSP

Dalam menyelesaikan *Ant System* untuk TSP maka dilakukan langkah – langkah sebagai berikut :

a) Penetapan Parameter dan Nilai *Pheromone* Awal

- Inisialisasi harga parameter – parameter algoritma.

1) Intensitas jejak semut antar titik dan perubahannya (τ_{rs}). Nilai (τ_{rs}) akan selalu diperbaharui pada setiap iterasi algoritma, mulai dari iterasi pertama sampai iterasi maksimum yang ditentukan atau telah mencapai hasil yang optimal.

2) Banyaknya titik (n) dan juga koordinat (x,y) atau jarak antar titik (d_{rs}), dalam TSP simetrik nilai $d_{rs} = d_{sr}$.

3) Titik berangkat (awal) dan titik tujuan, dalam kasus TSP titik awal dan titik tujuan adalah sama.

4) Tetapan iterasi semut (Q).

5) Tetapan pengendali intensitas jejak semut (α). Nilai $(\alpha) > 0$.

6) Tetapan pengendali visibilitas (β). Nilai $(\beta) > 0$.

7) Visibilitas antar titik $\left(\eta_{rs} = \frac{1}{d_{rs}}\right)$.

8) Banyaknya semut (m).

9) Tetapan penguapan *Pheromone* (ρ). Nilainya $0 < \rho \leq 1$, hal ini untuk mencegah jumlah *Pheromone* yang tak terhingga.

10) Jumlah iterasi maksimum (NCmax), bersifat tetap selama algoritma berjalan.

- Inisialisasi titik pertama setiap semut.

  Setelah inisialisasi (τ_{rs}) dilakukan, kemudian m semut ditempatkan pada titik pertama tertentu secara acak pada sejumlah n titik. Hasil inisialisasi titik pertama setiap semut, diisikan sebagai elemen pertama *tabu list*. Hasil dari langkah ini adalah terisinya elemen pertama *tabu list* setiap semut dengan indeks titik tertentu, yang berarti bahwa setiap $tabu_k(i)$ bisa berisi indeks titik antara *1* sampai *n*.

b) Pemilihan Titik Berikutnya

Setelah setiap semut menempati masing – masing titik yang ditentukan, maka semut akan mulai melakukan perjalanan dari titik pertama masing-masing sebagai titik asal dan menuju salah satu titik - titik lainnya sebagai tujuan. Kemudian dari titik kedua masing-masing, semut akan melanjutkan perjalanan dengan memilih salah satu dari titik – titik yang tidak terdapat pada $tabu_k$ sebagai titik tujuan selanjutnya. Perjalanan semut berlangsung terus menerus sampai semua titik dikunjungi satu persatu atau telah menempati $tabu_k$. Jika s menyatakan indeks urutan kunjungan, titik asal dinyatakan sebagai $tabu_k(s)$ dan titik-titik lainnya dinyatakan sebagai $\{N\text{-}tabu_k\}$, maka untuk menentukan titik tujuan selanjutnya digunakan persamaan 2.1.

c) Perhitungan Panjang Tour dan Pencarian Jalur Terpendek

- Perhitungan panjang tour setiap semut.

  Perhitungan panjang tour setiap semut k (C^k) dilakukan setelah setiap semut menyelesaikan tournya masing - masing. Perhitungan ini dilakukan berdasarkan $tabu_k$ masing – masing dengan persamaan berikut :

$$C^k = d_{tabu_k(n),tabu_k(1)} + \sum_{s=1}^{n-1} d_{tabu_k(s),tabu_k(s+1)} \qquad \text{.....(3.1)}$$

  Dimana $d_{tabu_k(s),tabu_k(s+1)}$ merupakan jarak edge dari titik s sampai titik s+1 pada *tabu list* yang ditempati oleh semut k, dan $d_{tabu_k(n),tabu_k(1)}$

adalah jarak antara titik n (akhir) dan titik pertama (awal) pada *tabu list* yang ditempati oleh semut k

- Pencarian tour terpendek.

  Setelah C^k setiap semut dihitung, akan diperoleh tour terpendek pada setiap iterasi. Panjang tour terpendek terbaik secara keseluruhan dinyatakan dengan C^{bs}.

d) Perhitungan Perubahan Harga Intensitas *Pheromone*

- Perhitungan perubahan harga intensitas *Pheromone* antar titik.

  Koloni semut akan meninggalkan *Pheromone* pada lintasan antar titik yang dilaluinya. Adanya penguapan dan perbedaan jumlah semut yang lewat, menyebabkan kemungkinan terjadinya perubahan harga intensitas *Pheromone* antar titik. Persamaan perubahan ini adalah :

$$\Delta\tau_{rs} = \sum_{k=1}^{m} \Delta\tau_{rs}^{k} \quad \text{.....(3.2)}$$

  dengan $\Delta\tau_{rs}^{k}$ adalah perubahan harga intensitas *Pheromone* antara titik r dan s untuk semut k. Dimana $\Delta\tau_{rs}^{k}$ dihitung sebagai berikut :

$$\Delta\tau_{rs}^{k} = \begin{cases} \dfrac{Q}{C^k} & ,untuk(r,s) \in kota\ \ asal\ \ dan\ \ kota\ \ tujuan\ \ dalam\ \ tabu_k \\ 0 & ,untuk(r,s)\ \ lainnya \end{cases}$$

- Perhitungan harga intensitas *Pheromone* antar titik untuk iterasi selanjutnya.

Harga intensitas *Pheromone* semut antar titik pada semua lintasan antar titik ada kemungkinan berubah karena adanya penguapan dan perbedaan jumlah semut yang melewati. Untuk iterasi selanjutnya, semut yang akan melewati lintasan tersebut harga intensitasnya telah berubah. Harga intensitas *Pheromone* antar titik untuk iterasi selanjutnya dihitung dengan persamaan 2.3.

e) Proses pemberhentian (Pengosongan *tabu list* dan ulangi langkah b)

Tabu list dikosongkan untuk diisi kembali dengan urutan titik yang baru pada iterasi selanjutnya, jika NCmax belum tercapai atau belum terjadi konvergensi (semua semut hanya menemukan satu tour yang sama dengan jarak yang sama pula). Algoritma diulang lagi dari langkah b dengan harga parameter intensitas *Pheromone* antar titik yang sudah diperbaharui.

Perhitungan akan dilanjutkan hingga semut telah menyelesaikan perjalanannya mengunjungi tiap-tiap titik. Hal ini akan berulang hingga sesuai dengan NC_{max} yang telah ditentukan atau telah mencapai konvergensi. Kemudian akan ditentukan jarak terpendek dari masing-masing iterasi. Jarak terpendek inilah yang merupakan solusi terbaik dari algoritma *Ant System*.

3.2.2 Penetapan Parameter Algoritma AS

Pada eksperimen yang dilakukan oleh Dorigo, M., Maniezzo, V., dan Colorni, A, (1996) yang dilakukan dengan algoritma AS untuk menyelesaikan TSP, parameter yang digunakan untuk mencapai solusi yang mendekati optimal adalah masing-masing mempunyai nilai sebagai berikut : $\alpha = 1$, $\beta = 2$ *sampai* $\beta = 5$, dan $\rho = 0{,}5$. Selain ketiga parameter diatas, untuk mendapatkan hasil yang lebih mendekati optimal maka masih terdapat parameter yang lain yaitu τ_0 (Nilai *Pheromone* awal), dengan nilai τ_0 dihitung sebagai berikut :

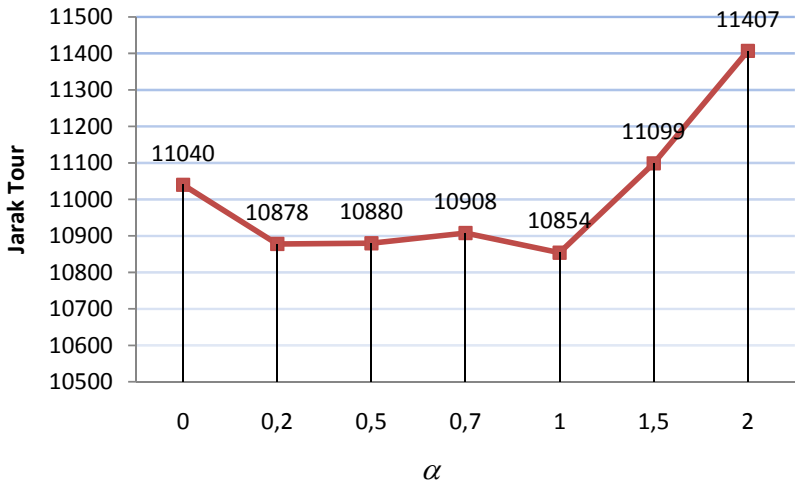
$$\tau_0 = \frac{m}{C^{nn}} \qquad (3.3)$$

Dimana C^{nn} adalah panjang sebuah tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic*. Banyaknya semut sama dengan jumlah titik pada masalah (m = n). Hal ini untuk menghindari jumlah semut yang berlebih sehingga akan menimbulkan ketidak optimalan dalam penyelesaian. Penempatan semut pada awal algoritma yaitu dengan menempatkan satu semut pada satu titik saja. Hal ini untuk menghindari penumpukan semut pada satu jalur yang sama yang akan menimbulkan stagnasi.

3.2.3 Pengaruh α, β dan ρ terhadap Performa Algoritma AS

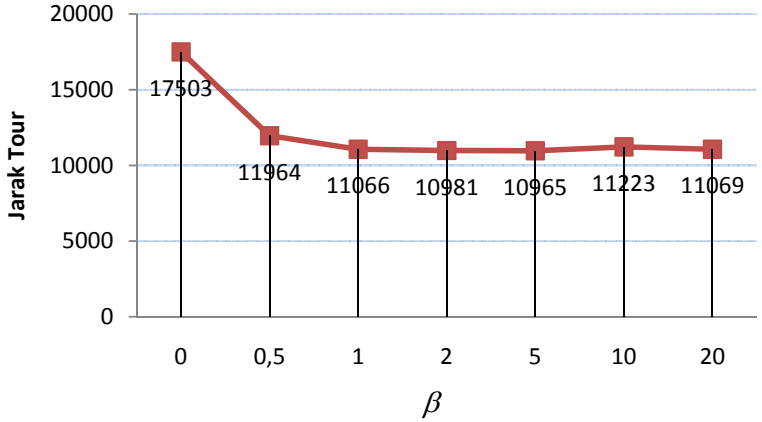
Parameter – parameter α, β, dan ρ mempunyai pengaruh terhadap performa hasil yang diperoleh pada AS, sehingga untuk mengetahui

pengaruh dari parameter – parameter tersebut, berikut diberikan hasil simulasi dengan ACOTSP, versi 1.0 yang dikembangkan oleh Dorigo dan *Stützle*. Permasalahan yang diujikan yaitu att48 dengan $\alpha, \beta,$ dan ρ masing – masing dengan nilai : $\alpha \in \{0; 0{,}2; 0{,}5; 0{,}7; 1; 1{,}5; 2\}$, $\beta \in \{0; 0{,}5; 1; 2; 5; 10; 20\}$ dan $\rho \in \{0{,}1; 0{,}3; 0{,}5; 0{,}7; 0{,}9; 1\}$. Hasil eksperimen yang diperoleh ditunjukkan pada gambar 3.1 untuk nilai α, gambar 3.2 untuk nilai β, dan gambar 3.3 untuk nilai ρ .



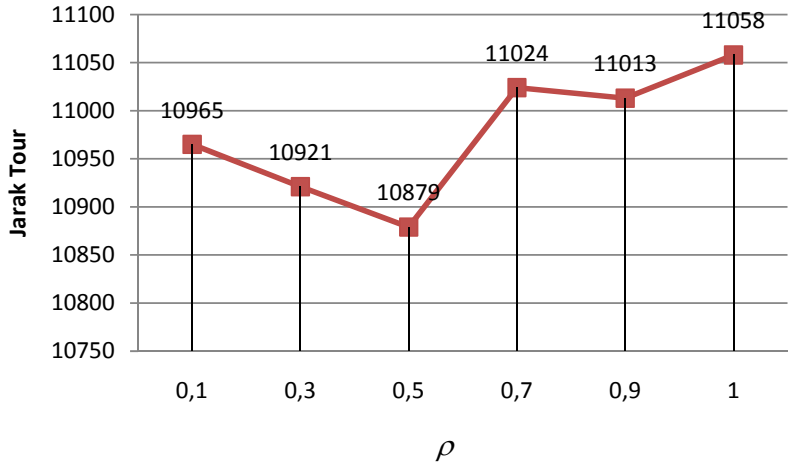
Gambar 3.1 Pengaruh Nilai α terhadap Performa AS

Gambar 3.1 diatas menunjukkan dengan nilai $\alpha = 1$ diperoleh performa hasil yang terbaik dengan panjang tour terbaik 10854 dan performa paling jelek diperoleh dengan nilai $\alpha = 2$ dengan panjang tour terbaiknya 11407.



Gambar 3.2 Pengaruh Nilai β terhadap Performa AS

Gambar 3.2 diatas menunjukkan dengan nilai $\beta = 5$ diperoleh performa yang paling baik dengan panjang tour terbaik 10965 dan hasil yang paling jelek diperoleh dengan nilai $\beta = 0$ dengan panjang tour terbaiknya 17503. Meskipun performa terbaik dihasilkan dengan nilai $\beta = 5$, pada kenyataannya nilai $\beta = 2$ juga sangat mendekati hasil yang optimal dengan panjang tour terbaiknya 10981. Jadi dengan kata lain performa terbaik pada algoritma AS bisa diperoleh dengan nilai $\beta = 2$ *sampai* $\beta = 5$.



Gambar 3.3 Pengaruh Nilai ρ terhadap Performa AS

Gambar 3.3 diatas menunjukkan dengan nilai $\rho = 0{,}5$ diperoleh performa yang paling baik, dengan panjang tour terbaik 10878 dan hasil yang paling jelek diperoleh dengan nilai $\rho = 1$ dengan panjang tour terbaiknya 11058.

Dari ketiga parameter tersebut diatas maka, terdapat kombinasi yang paling bagus untuk memperoleh hasil yang mendekati optimal. Masing – masing nilai ketiga parameter tersebut adalah sebagai berikut : $\alpha = 1$, $\beta = 2$ *sampai* $\beta = 5$, dan $\rho = 0{,}5$. Tetapi pada beberapa kasus yang ada, kombinasi ini tidak selalu menghasilkan nilai yang mendekati optimal. Hal ini tergantung pada besarnya masalah yang diselesaikan.

3.3 *Elitist Ant System (EAS)* untuk *Traveling Salesman Problem* (TSP)

3.3.1 Langkah – langkah dalam menyelesaikan EAS

Elitist Ant System (EAS) merupakan algoritma pengembangan pertama dari algoritma *Ant System*. Algoritma EAS pada dasarnya memiliki penyelesaian yang serupa dengan AS dalam menyelesaikan permasalahan TSP, tetapi pada sistem *update Pheromone*-nya berbeda. Pada AS, *Pheromone* di-*update* pada semua edge yang dilewati semut (persamaan 2.3). Sehingga yang mendapatkan penambahan *Pheromone* hanya pada edge – edge yang dilewati semut saja dan penambahan *Pheromone* sebesar sejumlah semut yang melewati edge – edge tersebut.

Pada EAS, *update Pheromone* dilakukan pada semua edge yang dilewati semut dan ada penambahan khusus pada edge – edge yang

merupakan bagian dari tour terpendek. Penambahan ini sebesar $\frac{e}{C^{bs}}$, dimana e adalah parameter yang menunjukkan adanya tour terbaik (T^{bs}) dan C^{bs} adalah panjang tour terbaik. *Update Pheromone* pada EAS ini dihitung berdasarkan persamaan 2.4. Penambahan ini dimaksudkan untuk mempersempit ruang pencarian semut pada iterasi selanjutnya, sehingga pada iterasi selanjutnya semut diharapkan akan lebih terkonsentrasi pada edge – edge yang mempunyai jarak pendek dan jumlah *Pheromone* yang terbanyak. Dengan demikian, semut akan lebih mudah menemukan tour terbaik karena hanya mempunyai ruang pencarian yang lebih sempit.

Setelah selesai melakukan *update*, semut akan berhenti melakukan pencarian apabila semua semut hanya menemukan satu tour yang sama dengan jarak yang sama pula (konvergensi) atau telah memenuhi NCmax yang telah ditentukan sebelumnya. Jika sebaliknya, maka proses pencarian akan dilanjutkan dan *tabulist* dari masing-masing semut dikosongkan kembali dan semut memulai pencarian dengan *Pheromone* awal yang telah diperbaharui.

Seperti yang dikatakan diatas, bahwa EAS pada dasarnya hampir sama dengan AS, maka parameter yang digunakan pun hampir sama dengan AS. Perbedaan yang ada hanya pada parameter e, yang hanya ada pada EAS dan nilai dari τ_0 -nya (intensitas *Pheromone* awal).

Secara jelasnya EAS mempunyai langkah-langkah sebagai berikut :

Langkah 1 : Penetapan parameter dan Nilai *Pheromone* Awal

Sebagai langkah awal, ditentukan parameter-parameter yang akan dimasukkan pada algoritma EAS, parameter yang dimaksud sama dengan parameter yang terdapat pada algoritma AS. Selain itu terdapat parameter *e* yaitu parameter yang didefinisikan sebagai nilai pada tour terbaik. Nilai *e* disini biasanya sama dengan jumlah n titik (masalah) yang ada (e = n).

Langkah 2 : Setiap semut membangun solusi dengan aturan transisi

Setelah semua semut ditempatkan pada setiap titik awal, dengan satu titik hanya ditempati satu semut saja. Selanjutnya, semut akan memilih titik selanjutnya dengan aturan probabilitas transisi, pada persamaan 2.1. setelah titik dipilih maka tidak akan dipilih lagi sampai semut menyelesaikan sebuah tour.

Langkah 3 : Mengecek tour terbaik

Setelah semua semut menyelesaikan tournya, maka panjang dari tour masing- masing semut dihitung. Setelah itu diketahui tour yang terbaik (jarak tour terpendek), kemudian di-*update*.

Langkah 4 : *Update Pheromone*

Setelah semut menyelesaikan satu perjalanan (tour), maka intensitas *Pheromone* pada iterasi selanjutnya akan berubah sesuai dengan aturan pembaharuan *Pheromone* pada *EAS.*

Perhitungan perubahan intensitas *Pheromone* pada EAS ini sama dengan *update Pheromone* pada AS. Tetapi, pada EAS ada penambahan *Pheromone* pada edge-edge yang merupakan tour terbaik pada iterasi yang telah dilewati. Pada iterasi selanjutnya, semut yang akan melewati lintasan tersebut intensitas *Pheromone*-nya telah berubah. Harga intensitas *Pheromone* untuk edge-edge pada tour terbaik untuk iterasi selanjutnya dihitung berdasarkan persamaan 2.4.

Langkah 5 : Pemberhentian

Tabu list dikosongkan kembali untuk diisi dengan urutan titik baru pada iterasi selanjutnya, jika NCmax belum tercapai atau belum terjadi konvergensi (semua semut hanya menemukan satu tour yang sama dengan jarak yang sama pula). Algoritma diulang lagi dari langkah 2 dengan harga parameter intensitas *Pheromone* antar titik yang sudah diperbaharui. Perhitungan akan dilanjutkan hingga semut telah menyelesaikan perjalanannya mengunjungi tiap-tiap titik. Hal ini akan berulang hingga sesuai dengan NC_{max} yang telah ditentukan atau telah mencapai konvergensi. Kemudian akan ditentukan jarak terpendek dari masing-masing iterasi. Jarak terpendek inilah yang merupakan solusi terbaik dari algoritma *Elitist Ant System*.

3.3.2 Penetapan Parameter Algoritma EAS

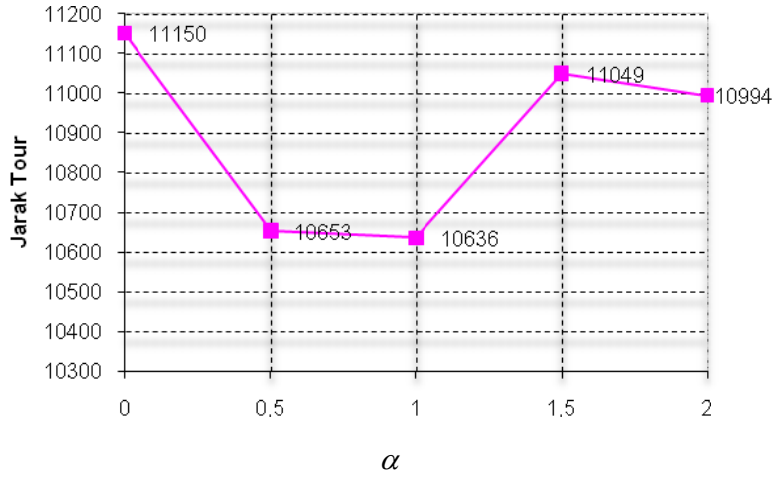
Dalam eksperimen yang dilakukan oleh Bullnheimer, B., Hartl, R. F., dan Strauss, C, (1997) dengan algoritma EAS untuk menyelesaikan TSP, parameter yang digunakan untuk mencapai solusi yang lebih mendekati optimal adalah masing – masing mempunyai nilai sebagai berikut : $\beta = 2$ *sampai* 5, $\rho = 0{,}5$, $\alpha = 1$, e = n, dan τ_0 yang dihitung sebagai berikut :

$$\tau_0 = \frac{(e+m)}{\rho C^{nn}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.4)$$

Dimana C^{nn} adalah panjang sebuah tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic,* m jumlah semut, dan n adalah jumlah titik yang ada pada masalah. Penetapan parameter diatas bisa berubah sesuai dengan besarnya masalah yang dihadapi kecuali untuk nilai τ_0, yaitu dengan nilai seperti persamaan 3.4. Jumlah semut yang digunakan untukmendapatkan solusi yang lebih mendekati optimal dalam EAS yaitu m = n. Hal ini untuk menghindari jumlah semut yang berlebih sehingga akan menimbulkan ketidakoptimalan dalam penyelesaian. Penempatan semut pada awal algoritma yaitu dengan menempatkan satu semut pada satu titik saja. Hal ini untuk menghindari penumpukan semut pada satu jalur yang sama sehingga akan menimbulkan stagnasi.

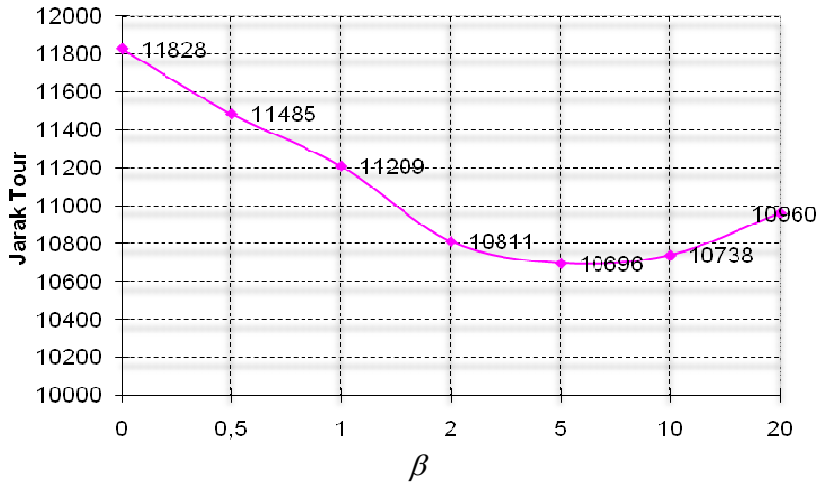
3.3.3 Pengaruh α, β dan ρ terhadap Performa Algoritma EAS

Parameter – parameter $\alpha, \beta,$ dan ρ mempunyai pengaruh terhadap performa hasil yang diperoleh pada EAS, sehingga untuk mengetahui pengaruh dari parameter – parameter tersebut, berikut diberikan hasil simulasi dengan ACOTSP, versi 1.0 yang dikembangkan oleh Dorigo dan *Stützle*. Permasalahan yang diujikan yaitu att48 dengan $\alpha, \beta,$ dan ρ masing – masing dengan nilai : $\alpha \in \{0; 0{,}5; 1; 1{,}5; 2\}, \beta \in \{0; 0{,}5; 1; 2; 5; 10; 20\}$ dan $\rho \in \{0{,}1; 0{,}3; 0{,}5; 0{,}7; 0{,}9; 1\}$. Hasil eksperimen yang diperoleh ditunjukkan pada gambar 3.4 untuk nilai α, gambar 3.5 untuk nilai β, dan gambar 3.6 untuk nilai ρ .



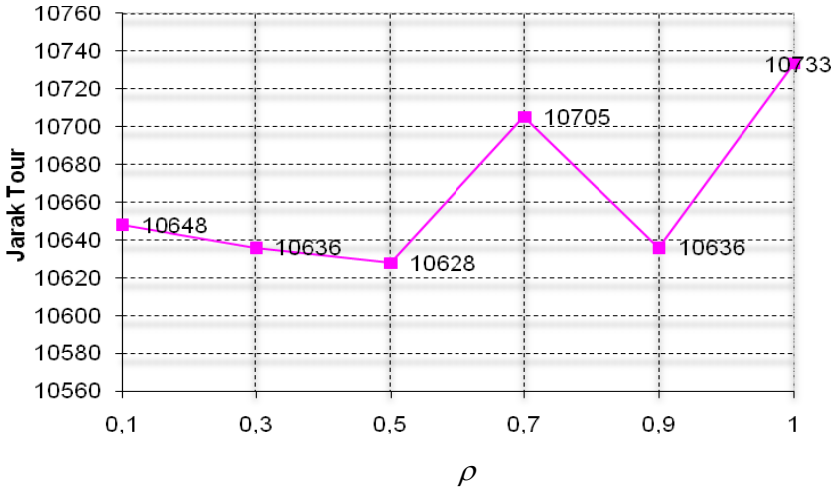
Gambar 3.4 Pengaruh Nilai α terhadap Performa EAS

Gambar 3.4 diatas menunjukkan dengan nilai $\alpha = 1$ diperoleh hasil tour yang paling baik dengan panjang tour terbaik 10636 dan hasil yang paling jelek diperoleh dengan nilai $\alpha = 0$ dengan panjang tour terbaiknya 11150.



Gambar 3.5 Pengaruh Nilai β terhadap Performa EAS

Gambar 3.5 diatas menunjukkan dengan nilai $\beta = 5$ diperoleh hasil tour yang terbaik dengan panjang tour terbaiknya 10696 dan hasil yang paling jelek diperoleh dengan nilai $\beta = 0$ dengan panjang tour terbaiknya 11828.



Gambar 3.6 Pengaruh Nilai ρ terhadap Performa EAS

Gambar 3.6 diatas menunjukkan dengan nilai $\rho = 0{,}5$ diperoleh hasil tour yang paling bagus dengan panjang tour terbaik 10628 dan hasil

yang paling jelek diperoleh dengan nilai $\rho = 1$ dengan panjang tour terbaiknya 10733.

Dari ketiga parameter tersebut diatas maka, terdapat kombinasi yang paling bagus untuk memperoleh hasil yang mendekati optimal. Masing – masing nilai ketiga parameter tersebut adalah sebagai berikut : $\alpha = 1$, $\beta = 5$, dan $\rho = 0{,}5$. Tetapi pada beberapa kasus yang ada, kombinasi ini tidak selalu menghasilkan nilai yang optimal. Hal ini tergantung pada besarnya masalah yang diselesaikan dan pangujian yang dilakukan.

3.4 *Rank-Based Ant System (AS_{Rank})* untuk *Traveling Salesman Problem*

3.4.1 Langkah – langkah dalam menyelesaikan AS_{Rank}

Algoritma AS_{Rank} merupakan pengembangan dari *Ant System* (AS) dengan menerapkan EAS, secara keseluruhan algoritma AS_{Rank} untuk TSP penyelesaiannya juga hampir sama dengan AS. Akan tetapi, dalam system *update Pheromone* (perubahan nilai *Pheromone*-nya) mempunyai perbedaan baik dengan AS maupun EAS. Setelah semua semut menyelesaikan tournya, semut akan mengelompokkan diri berdasarkan peringkat (panjang pendeknya tour yang mereka temukan). Kemudian, semut meng – *update* edge – edge yang telah mereka lalui dengan jumlah *Pheromone* yang berbeda, sesuai dengan tingkatannya. *Update Pheromone* disini hanya dilakukan pada (w-1) semut terbaik dan semut yang memiliki solusi *best-so-far tour* yang diperbolehkan menambahkan *Pheromone.*

Semut yang ke-*z* terbaik memberikan kontribusi *Pheromone* sebesar max {0, w-*z}* sementara jalur *best-so-far tour* memberikan kontribusi *Pheromone* paling besar yaitu sebesar w. Dalam *update Pheromone*-nya AS_{Rank} menerapkan persamaan 2.6. Penambahan beberapa parameter membuat hasil penyelesaian lebih optimal daripada AS maupun EAS.

Setelah selesai melakukan *update*, semut akan berhenti melakukan pencarian apabila semua semut telah menemukan tour yang sama dengan jarak yang sama pula (konvergensi) atau telah memenuhi NCmax yang telah ditentukan sbelumnya. Jika sebaliknya, maka proses pencarian akan diulangi dari langkah ke-2 dan *tabu list* dari masing-masing semut dikosongkan kembali. Semut memulai pencarian dengan *Pheromone* awal yang telah diperbaharui pada iterasi sebelumnya.

Secara jelasnya AS_{Rank} mempunyai langkah-langkah sebagai berikut :

Langkah 1 : Penetapan parameter dan Nilai *Pheromone* Awal

Seperti pada AS dan EAS sebagai langkah awal, kita menentukan parameter-parameter yang akan dimasukkan pada algoritma AS_{Rank}, parameter yang dimaksud sama dengan parameter yang terdapat pada algoritma AS. Selain itu terdapat parameter w dan z, dimana w adalah parameter yang menyatakan adanya tour terbaik dan z adalah peringkat semut. C^z adalah panjang tour yang dilewati semut ke-z dan C^{bs} adalah panjang tour terbaik.

Langkah 2 : Setiap semut membangun solusi dengan aturan transisi

Setelah semut ditempatkan dititik awalnya masing - masing maka, selanjutnya semut akan memilih titik selanjutnya dengan aturan probabilitas transisi, pada persamaan 2.1.

Langkah 3 : Mengecek tour terbaik

Setelah semua semut menyelesaikan iterasinya maka, panjang dari tour masing – masing semut dihitung. Setelah itu diketahui tour yang terbaik (jarak tour terpendek), kemudian di-*update*.

Langkah 4 : *Update Pheromone*

Setelah m semut menyelesaikan tournya dalam satu iterasi, mereka mengelompokkan diri berdasarkan tingkat fluktuasi solusi (panjang/pendeknya tour) yang telah mereka temukan sebelumnya. Kemudian, semut – semut melakukan *update Pheromone*. Dalam AS_{Rank} aturan *Pheromone update*- nya diberikan pada persamaan 2.6.

Langkah 5 : Pemberhentian

Semut akan berhenti melakukan pencarian tour terbaik, jika semua semut telah menemukan satu tour yang sama dengan jarak yang sama pula (konvergensi) atau sudah mencapai NCmax yang ditentukan sejak awal algoritma. Jika belum, maka semut akan kembali lagi mengulangi proses dari langkah 2.

3.4.2 Penetapan Parameter Algoritma AS_{Rank}

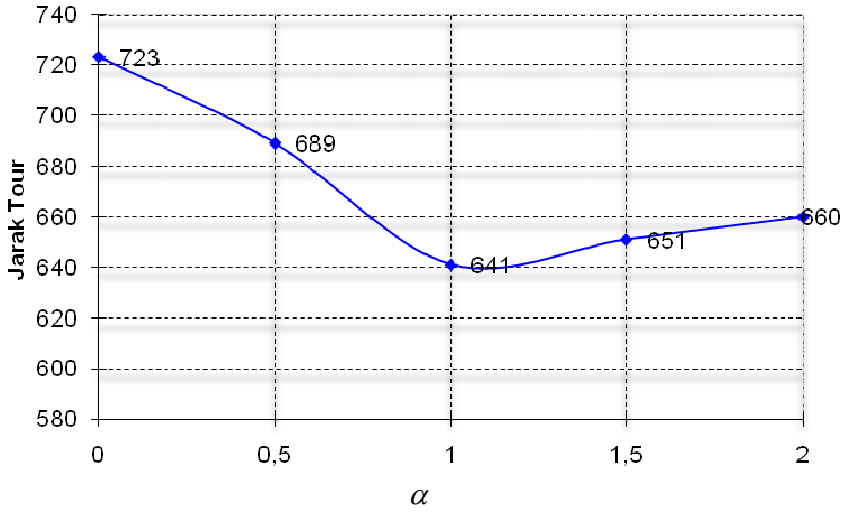
Dalam eksperimen yang dilakukan oleh Bullnheimer, B., Hartl, R. F., dan Strauss, C. (1997) dengan algoritma AS_{Rank} untuk menyelesaikan TSP, parameter yang digunakan untuk mencapai hasil yang lebih mendekati optimal adalah masing-masing mempunyai nilai sebagai berikut : $\beta = 2$ *sampai* 5, $\rho = 0{,}01$, $\alpha = 1$, dan τ_0 yang dihitung sebagai berikut :

$$\tau_0 = \frac{0{,}5z(z-1)}{\rho C^{nn}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3.5)$$

Dimana C^{nn} adalah panjang tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic,* z adalah peringkat dari semut, n adalah jumlah titik yang ada pada masalah, dan parameter w = 6. Penetapan parameter diatas bisa berubah sesuai dengan besarnya masalah yang dihadapi kecuali untuk nilai τ_0 yang dihitung seperti persamaan 3.5. Jumlah semut yang digunakan untuk memdapatkan solusi yang lebih mendekati optimal dalam AS_{Rank} yaitu sama dengan jumlah titik pada masalah yang diselesaikan (m = n). Hal ini untuk menghindari jumlah semut yang berlebih sehingga akan menimbulkan ketidakoptimalan dalam penyelesaian. Penempatan semut pada awal algoritma yaitu dengan setiap titik hanya boleh ditempati satu semut saja, hal ini untuk menghindari penumpukan semut pada satu jalur yang sama.

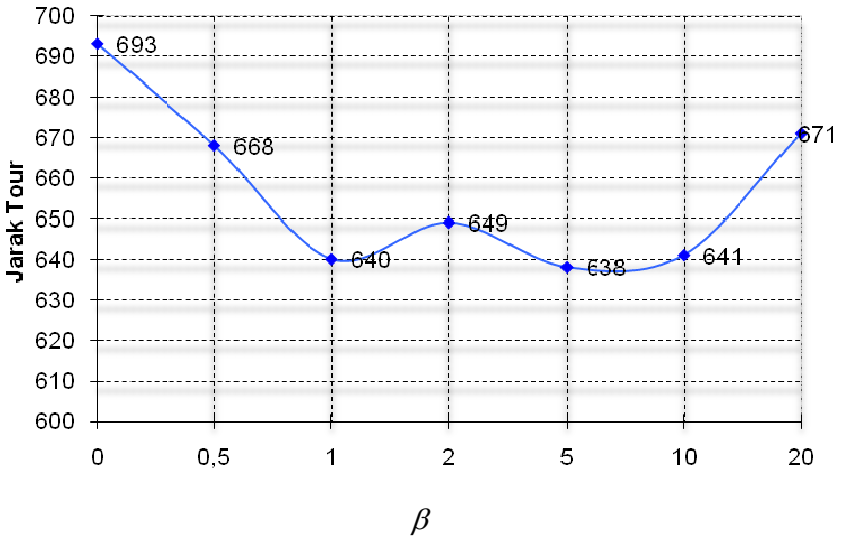
3.4.3 Pengaruh α, β dan ρ terhadap Performa Algoritma AS_{Rank}

Parameter – parameter $\alpha, \beta,$ dan ρ mempunyai pengaruh terhadap performa hasil yang diperoleh pada AS_{Rank}, sehingga untuk mengetahui pengaruh dari parameter – parameter tersebut berikut diberikan hasil simulasi dengan ACOTSP, versi 1.0 yang dikembangkan oleh Dorigo dan *Stützle*. Permasalahan yang diujikan yaitu eil101 dengan $\alpha, \beta,$ dan ρ masing – masing dengan nilai : $\alpha \in \{0; 0{,}5; 1; 1{,}5; 2\}, \beta \in \{0; 0{,}5; 1; 2; 5; 10; 20\}$ dan $\rho \in \{0{,}01; 0{,}05; 0{,}1; 0{,}5; 0{,}9; 1\}$. Hasil eksperimen yang diperoleh ditunjukkan pada gambar 3.7 untuk nilai α, gambar 3.8 untuk nilai β, dan gambar 3.9 untuk nilai ρ .



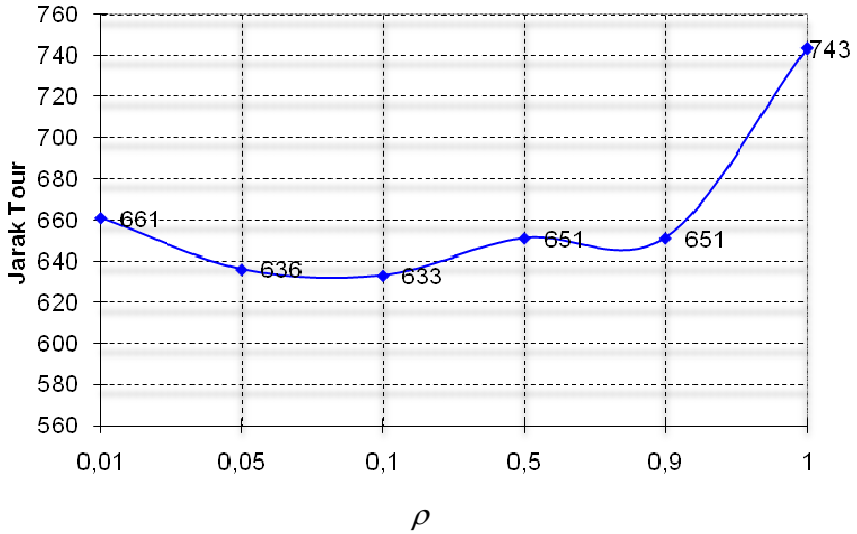
Gambar 3.7 Pengaruh Nilai α terhadap Performa AS_{Rank}

Gambar 3.7 diatas menunjukkan dengan nilai $\alpha = 1$ diperoleh hasil tour yang paling baik dengan panjang tour terbaik 641 dan hasil yang paling jelek diperoleh dengan nilai $\alpha = 0$ dengan panjang tour terbaiknya 723.



Gambar 3.8 Pengaruh Nilai β terhadap Performa AS_{Rank}

Gambar 3.8 diatas menunjukkan dengan nilai $\beta = 5$ diperoleh hasil tour yang paling baik dengan panjang tour terbaiknya 638 dan hasil yang paling jelek diperoleh dengan nilai $\beta = 0$ dengan panjang tour terbaiknya 693.



Gambar 3.9 Pengaruh Nilai ρ terhadap Performa AS_{Rank}

Gambar 3.9 diatas menunjukkan hasil tour yang paling mendekati optimal dengan panjang tour terbaik 633 diperoleh pada saat nilai $\rho = 0{,}1$ dan hasil yang paling jelek diperoleh dengan nilai $\rho = 1$ dengan panjang tour terbaiknya 743.

Dari ketiga parameter tersebut diatas maka, terdapat kombinasi yang paling bagus untuk memperoleh hasil yang mendekati optimal. Masing – masing nilai ketiga parameter tersebut adalah sebagai berikut : $\alpha = 1$, $\beta = 5$, dan $\rho = 0{,}1$. Tetapi pada beberapa kasus yang ada, kombinasi ini tidak selalu menghasilkan nilai yang optimal. Hal ini tergantung pada besarnya masalah yang diselesaikan dan pangujian yang dilakukan.

3. 5 *MAX – MIN Ant System (MMAS)* untuk *Traveling Salesman Problem*

3.5.1 Langkah – langkah dalam menyelesaikan *MMAS*

Algoritma *MMAS* merupakan pengembangan selanjutnya dari algoritma *Ant System* (AS) setelah EAS dan AS_{Rank}. Algoritma *MMAS* pada dasarnya juga memiliki penyelesaian yang hampir sama dengan AS dalam menyelesaikan permasalahan TSP, tetapi pada sistem *update Pheromone*-nya berbeda.

Algoritma *MMAS* mempunyai langkah – langkah penyelesaian yang mirip dengan AS, secara jelasnya berikut langkah – langkah pada Algoritma *MMAS* :

- Pertama, menginisialisasi parameter dan penempatan semut awal pada sejumlah n titik.
- Kedua, semut memilih titik selanjutnya berdasarkan persamaan 2.1.
- Ketiga, semua semut menyelesaikan tournya masing – masing, panjang tour semut dihitung dan diperoleh tour terbaik.
- Keempat, dilakukan *update Pheromone*, pada *update* ini penambahan *Pheromone* bisa dilakukan pada tour terbaik yang ditemukan sejak awal algoritma (*best so-far tour*) atau bisa juga pada tour terbaik yang ditemukan pada iterasi tersebut (*iteration best-tour*), bisa juga ditambahkan pada keduanya baik *best so-far tour* dan *iteration best-tour* secara bersamaan. Aturan *update Pheromone* pada *MMAS* menerapkan persamaan 2.7.
- Kelima, dilakukan test terhadap kondisi akhir yang menandakan pemberhentian proses pencarian tour terpendek. Jika semua semut telah menemukan satu tour yang sama dengan jarak yang sama pula (konvergensi) atau NCmax terpenuhi maka pencarian dihentikan, apabila belum maka proses pencarian akan dilanjutkan sampai terjadi konvergensi atau NCmax terpenuhi.

Pada algoritma *MMAS* terdapat beberapa hal penting yang perlu dicatat, yaitu :

1. Proses pembaharuan atau *update Pheromone* dilakukan terhadap edge – edge dengan *best so-far tour* atau *iteration best-tour,* penambahan ini bisa pada salah satu atau keduanya sekaligus. Hal ini tergantung

pada pengguna, sesuai dengan penetapan parameter yang dipilih pada awal algoritma.

2. Untuk mengantisipasi terjadinya stagnasi (semut terkonsentrasi pada satu jalur yang sama) pada algoritma karena penambahan *Pheromone*, maka diberikan batasan dalam pemberian nilai *Pheromone* dengan interval $[\tau_{\min}, \tau_{\max}]$.
3. Menginisialisasi nilai *Pheromone* dengan batas atas nilai *Pheromone*, yang mana bersama dengan tingkat evaporasi *Pheromone* yang kecil dapat meningkatkan eksplorasi tour sejak dimulainya pencarian.
4. *Pheromone* di inisialisasi kembali pada saat terjadi stagnasi atau ketika tidak ditemukan tour yang sesuai dengan iterasi yang diinginkan.

3.5.2 Penetapan Parameter Algoritma *MMAS*

Dalam eksperimen yang dilakukan dengan algoritma *MMAS* untuk menyelesaikan TSP (Stu¨ tzle, T., dan Hoos, H. H., 1997), parameter yang digunakan untuk mencapai solusi yang lebih mendekati optimal adalah masing – masing mempunyai nilai sebagai berikut : $\beta = 2$ *sampai* 5, $\rho = 0{,}98$, $\alpha = 1$, $\rho_{best} = 0{,}05$ dan τ_0 dihitung menurut persamaan berikut :

$$\tau_0 = \frac{1}{\rho C^{nn}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3.6)$$

Dimana C^{nn} adalah panjang sebuah tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic* dan n adalah jumlah titik yang

ada pada masalah. Batas atas nilai *Pheromone* ($\tau_{\max}$) dan batas bawah ($\tau_{\min}$ **)** dihitung sesuasi persamaan 3.7 dan 3.8 sebagai berikut :

$$\tau_{\max} = \frac{1}{\rho C^{bs}} \quad \text{. . . . .}$$

(3.7)

dimana C^{bs} adalah panjang tour terbaik yang ditemukan sejak awal algoritma berjalan.

$$\tau_{\min} = \frac{\tau_{\max}\left(1 - \sqrt[n]{\rho_{best}}\right)}{\left((avg - 1)\sqrt[n]{\rho_{best}}\right)} \quad \text{. . . . .} \quad (3.8)$$

Dimana *avg* adalah rata – rata jumlah dari pilihan yang berbeda pada semut dalam setiap langkah untuk menemukan tour terbaiknya, dengan nilai $\rho_{best} = 0{,}05$. Penetapan parameter diatas bisa berubah sesuai dengan besarnya masalah yang dihadapi kecuali untuk nilai τ_0 , yang dihitung sesuai dengan persamaan 3.6**,** nilai $\tau_{\max}$ yang dihitung sesuai persamaan 3.7, dan nilai $\tau_{\min}$ yang dihitung sesuai persamaan 3.8. Jumlah semut yang digunakan untuk memperoleh solusi lebih mendekati optimal dalam *MMAS* adalah m = n. Hal ini untuk menghindari jumlah semut yang berlebih sehingga akan menimbulkan ketidakoptimalan dalam penyelesaian. Penempatan semut pada awal algoritma yaitu dengan menempatkan satu semut pada satu titik saja. Hal ini untuk menghindari penumpukan semut pada satu jalur yang sama yang akan menimbulkan stagnasi.

3.5.3 Inisialisasi Nilai *Pheromone*

Saat algoritma mulai dijalankan, nilai *Pheromone* diinisialisasi dengan batas atas nilai *Pheromone*. Inisialisasi nilai *Pheromone* dikombinasikan dengan parameter evaporasi *Pheromone* akan menyebabkan peningkatan secara perlahan perbedaan relatif pada level *Pheromone*. Oleh karena itu, phase pencarian dalam *MMAS* menjadi sangat *explorative*.

Pada *MMAS* nilai *Pheromone* kadang – kadang di reinisialisasi kembali, ini berarti peningkatan eksplorasi hanya mempunyai probabilitas yang kecil dari pemilihan jalur. *Pheromone* di reinisialisasi pada saat pencarian tour terbaik mendekati stagnasi atau setelah beberapa iterasi algoritma tidak menemukan tour yang lebih baik.

Pada algoritma *MMAS*, inisialisasi hanya dilakukan pada batas atas nilai *Pheromone* saja. Untuk menunjukkan manfaat dari inisialisasi pada batas atas nilai *Pheromone*, diberikan hasil eksperimen (Stu¨ tzle, T., dan Hoos, H. H., 1997) yang dapat dilihat pada tabel 3.1 yang dibandingkan dengan inisialisasi pada batas bawah *Pheromone*. Hasil ini menunjukkan bahwa semua masalah yang diinisialisasi pada batas atas *Pheromone* diperoleh hasil yang lebih baik, kecuali pada satu masalah dengan kategori terkecil yaitu pada eil51. Dengan semakin membesarnya masalah maka inisialisasi batas atas *Pheromone* perlu dilakukan untuk memperoleh hasil yang lebih mendekati optimal.

Tabel 3.1. Hasil eksperimen inisialisasi *Pheromone* untuk batas atas $(\tau(0) = \tau_{\max})$ dan batas bawah $(\tau(0) = \tau_{\min})$.

Masalah	$\tau(0) = \tau_{\max}$	$\tau(0) = \tau_{\min}$
eil51	427,8	**427,7**
kro100	**21336,9**	21362,3
dl98	**15952,3**	16051,8
lin318	**42346,6**	42737,6

3.5.4 Pengaruh Batas Bawah *Pheromone* Terhadap Performa *MMAS*

Pengaruh keefektifan batas bawah *Pheromone* terhadap performa *MMAS* dapat dilihat dari hasil eksperimen (Stu¨ tzle, T., dan Hoos, H. H., 1997) pada tabel 3.2. Dalam eksperimen ini digunakan iterasi maksimum 2500.n, Penetapan parameter standar sama seperti pada tabel 3.5 dengan $\rho = 5$ dan nilai $\rho_{best} = \{0{,}0005;\ 0{,}005;\ 0{,}05;\ 0{,}5\}$, serta digunakan $\tau_{\min} = 0$ untuk eksperimen tanpa nilai batas bawah.

Secara keseluruhan hasil yang diperoleh dengan menggunakan batas bawah *Pheromone* lebih baik dibandingkan dengan tanpa menggunakan batas bawah *Pheromone*. Untuk nilai ρ_{best} rata – rata menghasilkan nilai yang lebih mendekati optimal pada setiap masalah, kecuali untuk $\rho_{best} = 0{,}0005$. Dibandingkan nilai ρ_{best} yang lain, nilai $\rho_{best} = 0{,}0005$ paling jelek hasilnya, tetapi dibandingkan dengan tanpa nilai batas bawah masih lebih baik. Untuk $\rho_{best} = 0{,}005$ mempunyai hasil

terbaik hanya pada masalah yang sangat besar yaitu, pada lin318. Sedangkan untuk $\rho_{best} = 0{,}05$ mempunyai hasil terbaik pada masalah yang cukup besar yaitu pada dl98 dan kro100, dan untuk $\rho_{best} = 0{,}5$ mempunyai hasil yang mendekati optimal pada masalah berkategori kecil yaitu pada eil51.

Tabel 3.2. Hasil eksperimen nilai batas bawah *Pheromone* dan tanpa batas bawah *Pheromone*.

Masalah	$\rho_{best} = 0{,}0005$	$\rho_{best} = 0{,}005$	$\rho_{best} = 0{,}05$	$\rho_{best} = 0{,}5$	$\tau_{\min} = 0$
eil51	428,5	428,0	427,8	**427,7**	427,8
kro100	21344,8	21352,8	**21336,9**	21353,9	21373,2
d198	16024,9	15973,2	**15952,3**	16002,3	16047,6
lin318	42363,4	**42295,7**	42346,6	42423,0	42631,8

3.5.5 *Best so-far tour* (C^{best}) *Vs Iteration best-tour* (C^{ib})

Performa *MMAS* juga dipengaruhi oleh C^{ib} atau C^{best} yang dipilih dalam melakukan *update Pheromone*. Pangaruh dari masing – masing nilai tersebut terhadap performa *MMAS* dapat dilihat pada tabel 3.3, dalam eksperimen (Stu¨ tzle, T., dan Hoos, H. H., 1997) digunakan nilai batas bawah (+ limit) dan tanpa batas bawah *Pheromone* (- no – limit).

Dari hasil yang diperoleh terlihat jelas bahwa dengan mengunakan C^{ib} + limit diperoleh hasil yang lebih baik dibandingkan dengan C^{best} + limit. Begitu pula pada C^{ib} – no – limit juga lebih baik dibandingkan dengan C^{best} – no – limit. Performa *MMAS* dengan C^{ib} + limit merupakan yang terbaik dibandingkan dengan yang lainnya. Dari sini dapat dikatakan

bahwa *MMAS* akan mempunyai performa yang lebih baik bila *update Pheromone* mengunakan C^{ib} daripada menggunakan C^{best}.

Tabel 3.3. Hasil eksperimen *Best so-far tour* (C^{ib}) *Vs Iteration best-tour* (C^{best}) dengan dan tanpa menggunakan batas bawah *Pheromone*.

Masalah	C^{ib} + limit	C^{best} + limit	C^{ib} – no – limit	C^{best} – no – limit
eil51	**427,8**	429,2	**427,8**	434,1
kro100	**21336,9**	21417,1	21373,2	21814,7
dl98	**15952,3**	16136,1	16047,6	16473,7
lin318	**42346,6**	42901,0	42631,8	44558,5

3.6 *Ant Colony System (ACS)* untuk *Traveling Salesman Problem* (TSP)

3.6.1 *Pseudocode* Algoritma ACS Untuk TSP

Berikut ini adalah *Pseudocode* algoritma *Ant Colony System* untuk menyelesaikan *Travelling Salesman Problem* (TSP) :

1. {Fase Inisialisasi}

For r = 1 **to** n **do** {n adalah jumlah titik}

For s = 1 **to** n **do**

$\tau(r,s) := \tau_0$

End-for

End-for

For k : = 1 **to** m **do** {m adalah jumlah semut}

Ambil r_{k_1} menjadi titik awal untuk semut k

$J_k(r_{k_1}) := \{1,\ldots,n\} - r_{k_1}$

{ J_k (r_{k_1}) adalah himpunan titik – titik yang belum dan akan dikunjungi oleh semut k yang berada di titik r_{k_1} }

$r_k := r_{k_1}$ { r_k adalah titik dimana semut k berada}

End-for

2. {Fase saat semut membangun lintasannya. Lintasan semut k disimpan di tour k}

For i : = 1 **to** n **do**

If i < n

Then

For k : = 1 **to** m **do**

Pilih titik berikutnya s_k menurut persamaan (2.8) dan (2.1)

$J_k (s_k) := J_k (r_k) - s_k$

{ J_k (s_k) adalah himpunan titik – titik yang belum dan akan dikunjungi oleh semut k yang berada di titik s_k }

$Tour_k$ (i) : = (r_k , s_k)

End-for

Else

For k : = 1 **to** m **do**

{pada tour ini semua semut kembali ketitik awal r_{k_1} }

$s_k := r_{k_1}$

Tour_k (i) : = (r_k, s_k)

End-for

End if

{ pada fase ini *local updating* berlangsung dan *Pheromone di-update* menurut persamaan (2.11)}

For k : = 1 **to** m **do**

$\tau(r_k, s_k) := (1-\xi).\tau(r_k, s_k) + \xi.\tau_0$

$r_k := s_k$ {titik yang baru untuk semut k}

End-for

End-for

3. {Pada fase ini *global updating* berlangsung dan *Pheromone* di-*update*}

For k : = 1 **to** m **do**

Hitung C^k { C^k adalah panjang lintasan yang dihasilkan semut k}

End-for

Tetapkan C^{bs} {C^{bs} adalah panjang tour terbaik }

{*update* edge-edge yang menjadi bagian dari C^{bs} menurut pers. (2.9)}

For setiap edge (r,s)

$\tau(r_k, s_k) := (1-\rho).\tau(r_k, s_k) + \rho(C^{bs})^{-1}$

End-for

4. **If** (End_Condition = True)

Then print C^k yang terpendek

Else go to fase 2

3.6.3 Penjelasan Algoritma ACS

Penyelesaian algoritma ACO yang terdapat pada ACS dalam usahanya untuk mencari jalur terpendek dari sebuah permasalahan yang direpresentasikan dalam sebuah graph adalah sebagai berikut :

- Pertama, ACS menggunakan beberapa *agent* yang dinamakan semut. Algoritma ini dimulai dengan menempatkan setiap semut pada titik awalnya masing – masing. Untuk mendapatkan hasil yang lebih beragam, sebaiknya setiap semut memiliki titik awal yang berlainan, dan titik awal untuk setiap semut tidak akan berubah selama algoritma berjalan. Tour yang dilakukan oleh setiap semut dimulai dari sebuah titik awal dan melewati edge-edge yang menghubungkan n titik yang ada dengan setiap titik hanya dikunjungi sekali saja, kemudian kembali lagi ke titik awal tersebut.
- Kedua, setelah ditempatkan pada titik awalnya masing-masing, setiap semut memulai tournya dengan memilih titik berikutnya yang akan dikunjungi berdasarkan persamaan (2.8) dan (2.1). Pemilihan titik berikutnya ini dipengaruhi oleh panjangnya edge yang menghubungkan antara titik dimana semut berada saat ini dengan titik yang akan ditujunya, dan jumlah *Pheromone* yang ada pada edge tersebut.
- Ketiga, *update Pheromone*.

  Pheromone, merupakan informasi yang ditinggalkan oleh semut yang telah lebih dahulu melewati edge tersebut. Edge yang lebih pendek

dengan jumlah *Pheromone* yang lebih besar akan mendapat prioritas yang lebih tinggi untuk dipilih. Setelah menentukan titik berikutnya yang akan dituju, semut melewati edge yang menghubungkan kedua titik dan setelah sampai, semut memperbarui jumlah *Pheromone* yang terdapat pada edge yang dilewatinya itu berdasarkan persamaan (2.11). Kemudian semut memasukkan edge dan titik yang dilewatinya itu ke dalam *tabu list*-nya untuk menandakan bahwa edge dan titik tersebut merupakan bagian dari tour mereka. Pada saat ini juga, posisi semut telah berubah dari titik awal menjadi titik yang telah dituju dan semut kembali memilih titik berikutnya yang akan dikunjungi. Pada saat semua semut telah mengunjungi *n-1* titik, titik berikutnya yang akan dituju adalah titik awal dari masing-masing semut. Setiap semut akan memilih edge yang menghubungkan antara titik yang merupakan posisinya saat ini dengan titik awal dari masing-masing semut, lalu memperbarui jumlah *Pheromone* pada edge tersebut dan memasukkan edge dan titik awal tersebut ke dalam *tabu list*nya.

- Kelima, setelah semua semut menyelesaikan tour mereka, panjang tour dari setiap semut dihitung dan dipilih yang paling pendek. Tour terpendek yang dihasilkan ini dijadikan sebagai tour terbaik pada saat ini lalu dibandingkan dengan tour terbaik yang telah dihasilkan sebelumnya. Jika panjang tour terbaik saat ini lebih pendek daripada panjang tour terbaik yang sebelumnya maka panjang tour dari semut yang menghasilkan tour terbaik saat ini dijadikan sebagai tour terbaik

yang baru dan *tabu list* semut tersebut dimasukkan ke dalam *tabu list* dari tour terbaik, kemudian dilakukan pembaruan jumlah *Pheromone* pada edge-edge yang merupakan bagian dari tour terbaik tersebut berdasarkan persamaan (2.9).

- Keenam, dilakukan tes terhadap kondisi akhir yang menandakan proses penghentian pencarian tour terbaik. Jika benar maka tour terbaik yang telah dihasilkan akan dicetak dan algoritma dihentikan. Jika sebaliknya maka proses pencarian akan dilanjutkan dan *tabu list* dari masing-masing semut dikosongkan kembali.

Pada algoritma ACS , ada beberapa hal penting yang perlu dicatat, yaitu:

1. Untuk melakukan perbandingan antara tour terbaik saat ini dengan tour terbaik sebelumnya, pertama kali yang diperlukan adalah sebuah nilai tour terbaik awal. Nilai tour terbaik awal (C^{nn}) ini dapat diperoleh dengan metode *nearest neigberhoud heuristic*. Hal ini bertujuan agar pada proses pencarian tour terbaik yang pertama akan diperoleh nilai tour terbaik yang baru.
2. Pada proses pemilihan titik berikutnya, yang didasarkan pada persamaan (2.8), diperlukan sebuah nilai parameter q_0 yang merupakan sebuah bilangan pecahan dimana $0 \le q_0 \le 1$.
3. Setiap semut memiliki *tabu list* untuk menyimpan hasil tournya masing - masing. *Tabu list* ini berupa panjang tour, dan koleksi edge-edge dan titik - titik yang merupakan bagian dari tour mereka. Nilai

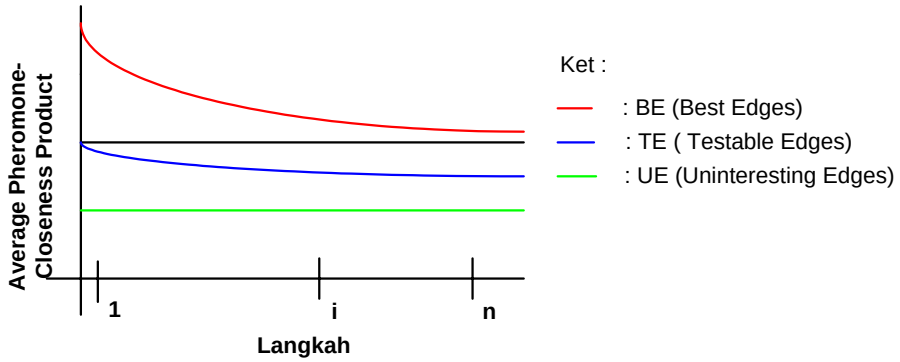
dari masing-masing *tabu list* ini akan dikosongkan kembali setiap kali semut akan memulai tournya.

4. Proses pembaruan *Pheromone* yang didasarkan pada persamaan (2.11) (aturan pembaruan *Pheromone* lokal) dipengaruhi oleh dua parameter, yaitu ξ dan $\Delta\tau$. Nilai ξ sama dengan nilai parameter ρ (*evaporasi Pheromon*) pada persamaan (2.9), sedangkan nilai $\Delta\tau$ didapatkan dari invers hasil perkalian antara panjang tour yang diperoleh melalui penelusuran titik-titik terdekat dengan jumlah titik yang ada pada graph tersebut. Tour yang dilakukan dengan menelusuri titik-titik terdekat ini mengikuti aturan yang diterapkan oleh Dorigo dan Gambardella (1997) dimana aturan ini telah terbukti dapat menunjukkan hasil yang bagus.
5. Proses pembaruan *Pheromone* yang didasarkan pada persamaan (2.9) (aturan pembaruan *Pheromone* global) dipengaruhi oleh dua parameter, yaitu ρ dan $\Delta\tau$. ρ adalah parameter *evaporasi Pheromone* dimana $0<\rho<1$ dan nilai $\Delta\tau$ diperoleh dari invers terhadap panjang tour terbaik yang paling akhir yang ditemukan oleh semut sejak dimulainya pencarian.

3.6.4 Perilaku *Pheromone* dan Hubungannya dengan Performanya

Untuk memahami mekanisme ACS digunakan secara langsung dalam perubahan *Pheromone-closeness product* $\left([\tau_{rs}]*[\eta_{rs}]^{\beta}\right)$ saat perjalanan mencari tour terbaik, ditunjukkan pada gambar 3.10. Perubahan

nilai *Pheromone* ini terjadi pada setiap langkah ketika semut membangun sebuah tour.



Gambar 3.10. Perubahan nilai *Pheromone* saat perjalanan

Pada gambar 3.10 diatas terdapat tiga kelompok edge yaitu :

(i) Edge dengan tour terbaik (BE, *Best Edges*).

(ii) Edge dengan tour bukan yang terbaik, tetapi masih memungkinkan untuk menjadi edge dengan tour terbaik pada dua iterasi berikutnya (TE, *Testable Edges*).

(iii) Bukan edge dengan tour terbaik ataupun yang masih memiliki peluang untuk menjadi tour terbaik pada dua iterasi berikutnya (UE, *Unintereresting Edges*).

Gambar 3.10 menunjukkan bahwa eksploitasi terbaik ACS terdapat pada BE (BE pemilihan edge dengan probabilitas $q_0 = 0{,}9$) dan ekplorasi terjadi pada edge dengan TE (pers. (2.1) dan pers. (2.8), edge dengan nilai Pheromone yang terbanyak mempunyai kesempatan yang lebih besar untuk terjadinya eksplorasi) (Dorigo & Gambardella,1996).

Aspek menarik terjadi saat semut melewati sebuah edge, ketika *local updating rule* (pers. 2.11) diaplikasikan, membuat *Pheromone* pada edge berkurang dan kehilangan daya tarik, sehingga edge yang telah dilewati tidak akan dilewati lagi oleh semut yang lainnya. Dengan kata lain, pengaruh dari pembaruan lokal ini adalah untuk membuat tingkat ketertarikan edge – edge yang ada berubah secara dinamis. Setiap kali seekor semut melewati sebuah edge maka edge ini dengan segera akan berkurang tingkat ketertarikannya (karena edge tersebut kehilangan sejumlah *Pheromone*-nya), secara tidak langsung semut yang lain akan memilih edge-edge lain yang belum dikunjungi. Konsekuensinya, semut tidak akan memiliki kecenderungan untuk berkumpul pada jalur yang sama. Fakta ini, yang telah diamati dengan melakukan percobaan Dorigo dan Gambardella (1997), merupakan sifat yang diharapkan bahwa jika semut membuat tour - tour yang berbeda maka akan terdapat kemungkinan yang lebih tinggi dimana salah satu dari mereka akan menemukan solusi yang lebih baik daripada jika mereka semua berkumpul dalam tour yang sama. Dengan cara ini, semut akan membuat penggunaan informasi *Pheromone* menjadi lebih baik. Tanpa pembaruan lokal, semua semut akan mencari pada lingkungan yang sempit dari tour terbaik yang telah ditemukan sebelumnya.

Pada observasi eksperimen yang dilakukan oleh Dorigo dan Gambardella (1997) pada saat ACS mencapai performa terbaiknya, *Pheromone* pada edge BE akan mengalami penurunan mendekati jumlah

Pheromone pada edge TE setelah iterasi algoritma berlangsung (lihat gambar 3.10). Begitu pula edge pada TE juga akan mengalami penurunan jumlah *Pheromone* mendekati UE. *Pheromone* pada keduanya yaitu BE dan TE akan hilang sepanjang perjalanan menemukan tour terpendek baru.

3.6.5 Penetapan Parameter Algoritma ACS

Dalam setiap eksperimen yang dilakukan dengan algoritma ACS untuk menyelesaikan TSP, parameter yang digunakan untuk mencapai solusi yang lebih mendekati optimal adalah masing-masing mempunyai nilai sebagai berikut : $\beta = 2$ *sampai* 5, $q_0 = 0{,}9$, $\xi = \rho = 0{,}1$, dan τ_0 yang dihitung sebagai berikut :

$$\tau_0 = \frac{1}{\left(nC^{nn}\right)} \quad \ldots \ldots (3.9)$$

Dimana C^{nn} adalah panjang sebuah tour terbaik yang diperoleh dari metode *nearest neighbourhood heuristic* dan n adalah jumlah titik yang ada pada masalah. Penetapan parameter diatas bisa berubah sesuai dengan besarnya masalah yang dihadapi kecuali untuk nilai τ_0, yang dihitung seperti pada persamaan 3.9. Jumlah semut yang digunakan untuk mendapatkan hasil terbaik pada ACS yaitu m = 10. Penempatan semut pertama pada awal algoritma yaitu dengan menempatkan semut secara acak pada setiap titik dan setiap titik hanya ditempati satu semut saja.

3.7 Perbandingan Algoritma – Algoritma ACO

3.7.1 Kelebihan Algoritma ACO berdasarkan kinerjanya

Beberapa kelebihan yang dimiliki masing – masing algoritma ACO dalam menemukan solusi berdasarkan kinerjanya ditunjukkan sebagai berikut :

1) Kelebihan – kelebihan AS adalah sebagai berikut :

- Di dalam cakupan parameter keoptimalan, algoritma ini selalu menemukan solusi yang mendekati optimal untuk semua permasalahan yang mempunyai jumlah titik sedikit.
- Algoritma ini dengan cepat menemukan solusi yang bagus, meskipun demikian ia tidak memperlihatkan perilaku stagnasi, maksudnya semut terus mencari kemungkinan adanya tour baru yang lebih baik.

2) Kelebihan – kelebihan EAS adalah sebagai berikut :

- Jumlah iterasi yang dilakukan lebih sedikit, hal ini karena edge – edge pada tour terbaik mendapatkan penambahan *Pheromone* lebih banyak dan menyebabkan ruang pencarian lebih sempit.
- Ruang pencarian solusi lebih sempit, sehingga lebih cepat dalam menemukan solusi. Walaupun ruang pencarian lebih sempit tetapi EAS tidak memperlihatkan terjadinya stagnasi, maksudnya semut terus mencari kemungkinan adanya tour baru yang lebih baik.

3) Kelebihan – kelebihan AS_{Rank} adalah sebagai berikut :

- Algoritma ini menggunakan sistem peringkat dalam *update Pheromone*-nya, sehingga akan mempermudah semut dalam menentukan sebuah tournya.
- Semut lebih cepat menemukan solusi karena edge – edge yang merupakan bagian tour terpendek memiliki jumlah *Pheromone* lebih banyak, hal ini karena semut lebih menyukai edge yang pendek dan mempunyai jumlah *Pheromone* yang lebih banyak untuk dipilih.

4) Kelebihan – kelebihan $\mathcal{MMAS}$ adalah sebagai berikut :

- Adanya batas nilai Pheromone membuat pencarian tidak mengalami stagnasi dan bisa lebih fokus.
- Penginisialisasian kembali membuat performa algoritma menjadi lebih baik. Pada jumlah titik masalah yang besar akan membuat pencarian solusi lebih mudah dan tidak menimbulkan stagnasi. Karena, pada waktu terjadi stagnasi maka akan dilakukan inisialisasi kembali.

5) Kelebihan – kelebihan ACS adalah sebagai berikut :

- Aturan transisi status pada sistem ini memberikan suatu cara langsung untuk menyeimbangkan antara penjelajahan

(*exploration*) edge-edge yang baru dengan eksploitasi (*exploitation*) (pers. 2.8 dan 2.1).

- Aturan pembaruan *Pheromone* global hanya dilakukan pada edge-edge yang merupakan bagian dari tour terbaik, sehingga semut akan lebih mudah dalam menentukan titik selanjutnya.
- Disaat *semut-semut* membangun sebuah tour, diterapkan suatu aturan pembaruan *Pheromone* lokal (*local Pheromone updating rule*). Sehingga membuat *Pheromone* pada edge berkurang dan kehilangan daya tarik, dengan demikian edge yang telah dilewati tidak akan dilewati lagi oleh semut yang lainnya. Hal ini menyebabkan semut mencari edge baru yang lebih pendek atau edge yang banyak dikunjungi semut dengan jumlah *Pheromone* yang banyak.

3.7.2 Perbedaan Algoritma-algoritma ACO Berdasarkan Cara Kerjanya

Berdasarkan pembahasan dari beberapa algoritma ACO, maka diperoleh perbedaan dari cara kerjanya. Berikut diberikan perbedaan cara kerja yang ada pada masing – masing algoritma ACO, yang ditunjukkan pada tabel 3.8 dibawah ini :

Tabel 3.4 Perbedaan cara kerja pada algoritma-algoritma ACO

Algoritma	Konstruksi Tour	Evaporasi	*Update Pheromone*
AS	*Random Proportional Rule*	Semua edge dengan faktor konstan ρ terendah.	Menambahkan *Pheromone* pada semua edge yang dikunjungi semut.
EAS	*Random Proportional Rule*	Semua edge dengan faktor konstan ρ terendah.	Menambahkan *Pheromone* pada semua edge yang dikunjungi semut, menambahkan *Pheromone* pada edge-edge yang merupakan bagian dari tour terbaik (jarak terpendek).
AS $_{Rank}$	*Random Proportional Rule*	Semua edge dengan faktor konstan ρ terendah.	Mengelompokkan semut berdasarkan panjang pendek tournya (peringkat); menambahkan *Pheromone* pada edge-edge yang dikunjungi semut dan sesuai dengan rank dari setiap semut.
MMAS	*Random Proportional Rule*	Semua edge dengan faktor konstan ρ terendah.	*Pheromone* bisa ditambahkan pada edge – edge yang merupakan bagian dari best tour yang ditemukan sejak awal algoritma (*best so-far tour*) atau pada egde-edge yang merupakan bagian best tour yang ditemukan pada setiap iterasi(*iteration best-tour*). Penambahan *Pheromone* boleh juga ditambahkan pada keduanya. Terdapat interval penambahan intensitas *Pheromone* sebesar $[\tau_{\min}, \tau_{\max}]$.
ACS	*Pseudorandom Proportional Rule*	Hanya pada edge – edge yang merupakan bagian dari *best-so-far tour* yang terlemah	Hanya menambahkan *Pheromone* pada edge – edge yang merupakan bagian dari tour terbaik yang ditemukan sejak awal algoritma dijalankan.

3.7.3 Parameter Terbaik pada Setiap Algoritma ACO

Menurut eksperimen yang telah dilakukan pada algoritma – algoritma ACO untuk TSP, maka diperoleh penetapan parameter yang bisa menghasilkan performa terbaik dari tiap-tiap algoritma tersebut. Pada tabel 3.5 berikut adalah Penetapan parameter terbaik yang dihasilkan dari algoritma-algoritma ACO.

Tabel 3.5. Penetapan parameter terbaik algoritma ACO untuk TSP

Algoritma ACO	α	β	ρ	**m**	τ_0
AS	1	2 sampai 5	0,5	n	$\frac{m}{C^{nn}}$
EAS	1	2 sampai 5	0,5	n	$\frac{(e+m)}{\rho C^{nn}}$
AS_{Rank}	1	2 sampai 5	0,1	n	$\frac{0{,}5z(z-1)}{\rho C^{nn}}$
$\mathcal{MMAS}$	1	2 sampai 5	0,02	n	$\frac{1}{\rho C^{nn}}$
ACS	-	2 sampai 5	0,1	10	$\frac{1}{nC^{nn}}$

Dimana n adalah jumlah titik dalam TSP, m adalah jumlah semut dalam algoritma. Untuk semua algoritma kecuali algoritma AS ada penambahan parameter. Berikut nilai terbaik untuk parameter pada algoritma variasi AS :

EAS : parameter e = n.

AS_{Rank} : Parameter yang menyatakan adanya tour terbaik, w = 6.

MMAS : Batas nilai *Pheromone* adalah $\tau_{\max} = \frac{1}{\rho C^{bs}}$ dan

$\tau_{\min} = \frac{\tau_{\max}\left(1-\sqrt[n]{\rho_{best}}\right)}{\left((avg-1)\sqrt[n]{\rho_{best}}\right)}$ dengan nilai $\rho_{best} = 0{,}05$.

ACS : Dalam *Local Pheromone Update* nilai $\xi = 0{,}1$. Dan didalam *pseudorandom proportional rule* nilai $q_0 = 0{,}9$.

Penetapan parameter diatas mungkin saja tidak berlaku pada beberapa kasus dan eksperimen yang berbeda, bisa saja diperoleh penetapan parameter berbeda yang bisa menghasilkan performa yang lebih baik.

3.7.4 Perbandingan hasil algoritma ACO pada beberapa kasus

- **Perbandingan AS, EAS dan AS_{Rank} pada beberapa kasus**

Untuk mengetahui hasil terbaik yang diperoleh pada masing – masing algoritma (AS, EAS dan AS_{Rank}), maka diberikan hasil percobaan Bullnheimer, B., Hartl, R. F., dan Strauss, C., (1997) pada tabel 3.6. Dengan parameter – parameter yang digunakan seperti pada tabel 3.5, kecuali pada AS_{Rank} dengan nilai $\rho = 0{,}5$ dan w = 6 semut terbaik yang boleh meng-*update Pheromone*-nya. Dalam percobaan ini dilakukan pada kasus dengan jumlah titik yang bervariasi yaitu 30 titik, 57 titik, 80 titik, 96 titik, dan 132 titik. Untuk nilai error (selisih relatif antara solusi terbaik yang dihasilkan algoritma dengan solusi optimal yang diperoleh dari metode *nearest neighbourhood heuristic*) dihitung sebagai berikut :

$$error = \frac{solusi \quad terbaik - optimal}{optimal} x100\% \quad \ldots\ldots\ldots\ldots\ldots\ldots (3.10).$$

Tabel 3.6. Perbandingan hasil perhitungan AS, EAS dan AS_{Rank}

Algoritma	Solusi rata-rata	Error (%)	Solusi terbaik	Error (%)	Solusi terjelek	Error (%)
n = 30			Optimal = 423,74		t = 30dtk	
AS	426,24	0,59	423,91	0,04	431,29	1,78
EAS	426,08	0,55	423,74	0,00	438,438,38	3,46
AS_{Rank}	425,72	0,47	423,74	0,00	431,29	1,78
n = 57			Optimal = 920,08		t = 50dtk	
AS	930,86	1,17	924,20	0,45	932,85	1,39
EAS	928,00	0,86	920,08	0,00	934,68	1,59
AS_{Rank}	926,91	0,74	920,08	0,00	934,70	1,59
n = 80			Optimal = 370,97		t = 60dtk	
AS	380,19	2,49	373,36	0,64	383,03	3,25
EAS	374,44	0,94	370,97	0,00	381,85	2,93
AS_{Rank}	373,74	0,75	370,97	0,00	376,84	1,58
n = 96			optimal = 1041,67		t = 80dtk	
AS	1068,85	2,61	1053,26	1,11	1080,66	3,74
EAS	1055,14	1,29	1045,98	0,41	1067,93	2,52
AS_{Rank}	1055,00	1,28	1043,70	0,19	1065,20	2,26
n = 132			Optimal = 1528,78		t = 120dtk	
AS	1568,02	2,57	1544,30	1,02	1587,63	3,85
EAS	1558,15	1,92	1537,73	0,59	1587,27	3,83
AS_{Rank}	1556,65	1,82	1533,54	0,31	1587,67	3,85

Dari tabel 3.6 dapat diketahui bahwa performa algoritma EAS dan AS_{Rank} lebih baik dibandingkan dengan AS. Pada kasus 30 titik hanya AS yang memperoleh hasil kurang optimal dengan error 0,04 %, begitu pula

pada kasus 57 titik dan 80 titik masing – masing memiliki error 0,45 % dan 0,64 %. Sedangkan EAS dan AS_{Rank} mampu memperoleh hasil yang optimal.

Pada kasus yang lebih besar (96 titik dan 132 titik) dari ketiga algoritma tersebut tidak ada yang mencapai optimal. Tetapi, performa EAS dan AS_{Rank} jauh lebih baik dibandingkan dengan AS. Pada kedua kasus (96 titik dan 132 titik) tersebut, EAS dan AS_{Rank} memiliki error kurang dari 1 % sedangkan AS lebih dari 1 %. Perfoma terbaik ditunjukkan oleh AS_{Rank} dengan error paling sedikit yaitu 0,19 % untuk kasus 96 titik dan 0,31 % untuk 132 titik.

Secara keseluruhan algoritma AS mempunyai performa yang lebih jelek dibandingkan dengan EAS dan $AS_{Rank.}$ Untuk performa terbaik ditunjukkan oleh AS_{Rank} pada semua kasus, sedangkan EAS lebih baik dari AS pada semua kasus.

- **Perbandingan *MMAS,* dan ACS pada beberapa kasus**

Untuk mengetahui hasil optimal yang diperoleh pada masing – masing algoritma (*MMAS,* dan ACS), maka diberikan hasil percobaan Stu¨ tzle, T., dan Hoos, H. H. (1997) untuk algoritma *MMAS* sedangkan percobaan Dorigo, M., dan Gambardella, L. M. (1997) untuk algoritma ACS pada tabel 3.7. Dengan parameter – parameter yang digunakan seperti pada tabel 3.5. Dalam percobaan ini dilakukan pada kasus dengan jumlah titik yang bervariasi yaitu eil51 (51 titik), kroA100 (100 titik),

d198 (198 titik), att532 (532 titik), dan rat783 (783 titik), dengan nilai error-nya dihitung menggunakan persamaan 3.10.

Pada tabel 3.7 dapat ditunjukkan bahwa ACS dan *MMAS* mempunyai hasil yang optimal pada dua kasus, yaitu eil51 dan kroA100. Sedangkan pada kasus d198 ACS mempunyai hasil yang lebih baik daripada *MMAS* dengan error 0,68 %, sedangkan pada *MMAS* mempunyai error 1,01 % dari nilai optimal yang diketahui. Untuk dua kasus dengan tipe besar (att532 dan rat783) menunjukkan hasil yang sebaliknya, yaitu *MMAS* mempunyai hasil yang lebih baik daripada ACS. Masing – masing dengan error 1,03 % dan 1,29 % untuk *MMAS,* sedangkan pada ACS mempunyai error 1,67 % dan 2,37 %.

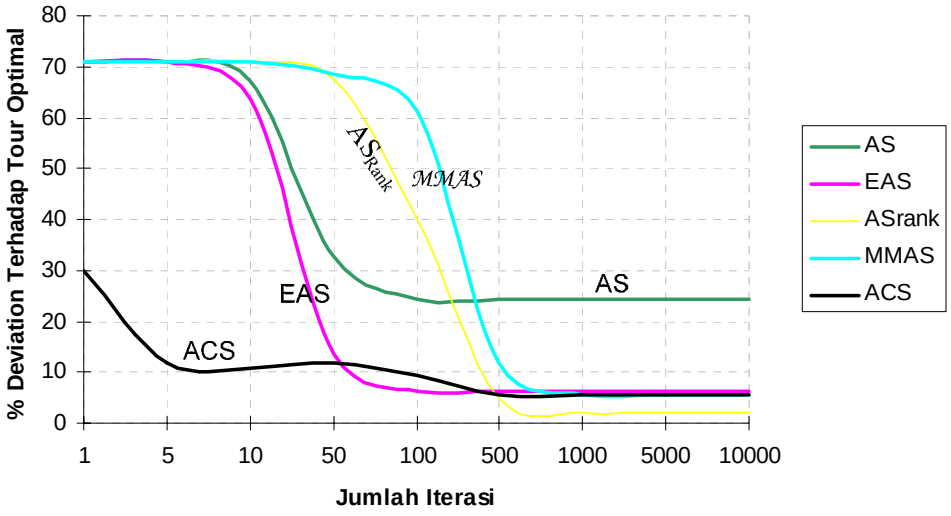
Tabel 3.7. Perbandingan hasil perhitungan *MMAS* dan ACS

Kasus	*MMAS*		ACS		Optimal
	Hasil terbaik	Error (%)	Hasil terbaik	Error (%)	
eil51	426	0.00	426	0.00	426
kroA100	21282	0.00	21282	0.00	21282
d198	15940	1.01	15888	0.68	15780
att532	27971	1.03	28147	1.67	27686
rat783	8920	1.29	9015	2.37	8,806

Dari hasil yang diperoleh diatas, maka dapat disimpulkan bahwa ACS dan *MMAS* sama – sama mempunyai hasil yang optimal untuk masalah bertipe kecil. Sedangkan untuk masalah bertipe sedang ACS lebih baik daripada *MMAS.* Untuk masalah bertipe besar algoritma *MMAS* menghasilkan penyelesaian lebih baik daripada ACS.

3.7.5 Perbandingan Algoritma ACO Berdasarkan Jumlah Iterasi

Untuk mengetahui algoritma yang mempunyai jumlah iterasi paling efektif dibandingkan algoritma yang lainnya, maka diberikan hasil percobaan Dorigo, M., dan Stu¨tzle, T. (2004) pada gambar 3.5. Pada gambar 3.5 kasus yang digunakan adalah kasus TSP simetri kroA100 (100 titik), dengan parameter – parameter yang digunakan seperti pada tabel 3.5, kecuali untuk nilai β, yaitu dengan nilai $\beta = 2$ untuk semua algoritma. Untuk data selengkapnya dapat dilihat pada lampiran 1.



Gambar 3.11. Grafik performa algoritma ACO berdasarkan jumlah iterasi dan deviasi (error) terhadap tour optimal pada masalah kroA100 (100 titik).

Gambar 3.11 dapat ditunjukkan bahwa algoritma ACS mempunyai hasil yang lebih baik dibandingkan algoritma yang lain dengan jumlah iterasi yang sedikit (sampai iterasi ke-50an). Pada iterasi ke 500 dan seterusnya algoritma ACS juga mempunyai performa yang lebih baik

dibandingkan algoritma yang lain, kecuali dengan algoritma AS_{Rank}. Algoritma AS_{Rank} pada awal iterasi memiliki kecendrungan hasil yang kurang bagus dibandingkan algoritma yang lain. Tetapi, mulai pada iterasi ke-75 algoritma AS_{Rank} lebih baik dibandingkan dengan *MMAS*, begitu pula pada iterasi ke-250 algoritma AS_{Rank} lebih baik daripada AS. Pada iterasi ke-500 algoritma AS_{Rank} jauh lebih baik dibandingkan algoritma yang lain. Dengan kata lain, semakin besar jumlah iterasi yang diberikan pada algoritma AS_{Rank} akan memberikan hasil yang lebih baik dibandingkan keempat algoritma yang lain.

Algoritma AS juga memiliki kecendrungan hasil yang kurang bagus dengan jumlah iterasi yang sedikit. Algoritma AS memiliki hasil yang lebih baik dibandingkan dengan AS dan AS_{Rank} pada iterasi ke-10 sampai iterasi ke-250. Selanjutnya, dengan iterasi yang lebih besar dari 250, algoritma AS mempunyai hasil yang paling jelek dibandingkan dengan keempat algoritma yang lain. Seperti halnya AS dan AS_{Rank}, algoritma *MMAS* juga memiliki kecendrungan hasil yang kurang bagus dengan jumlah iterasi yang sedikit, *MMAS* sedikit lebih baik dibandingkan dengan AS_{Rank} hanya sampai dengan iterasi ke-60. setelah itu, sampai sekitar iterasi ke-400an *MMAS* mempunyai hasil yang paling jelek dibandingkan keempat algoritma yang lainnya. Mulai iterasi ke-450, *MMAS* lebih baik daripada AS dan sedikit lebih baik daripada EAS mulai iterasi ke-850an.

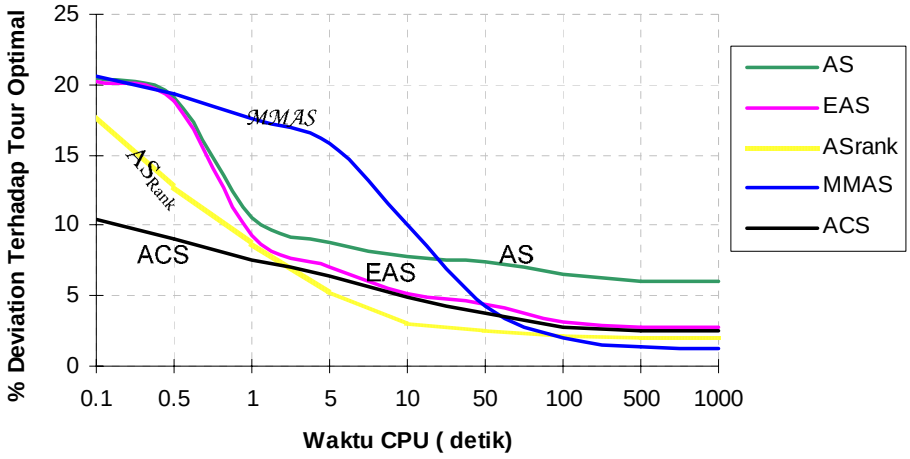
Sama halnya dengan ketiga algoritma yang lainnya (AS, $AS_{Rank,}$ *MMAS*), algoritma EAS juga memiliki kecendrungan hasil yang kurang bagus pada awal – awal iterasi. Tetapi, mulai iterasi ke-10 algoritma EAS mempunyai hasil penyelesaian yang lebih baik dibandingkan ketiga algoritma tersebut (AS, $AS_{Rank,}$ *MMAS*). Bahkan, algoritma EAS mempunyai hasil yang paling bagus dibandingkan keempat algoritma yang lainnya pada iterasi ke-50 sampai iterasi ke-350an. Setelah itu, algoritma EAS hanya lebih baik daripada algoritma AS saja. Tetapi, algoritma EAS hanya kurang bagus dibandingkan algoritma ACS dan *MMAS* pada jumlah iterasi yang besar.

Secara umum dapat dikatakan, hanya algoritma AS yang memiliki performa kurang bagus dibadingkan algoritma yang lainnya untuk jumlah iterasi berapapun. Sedangkan, hasil yang paling optimal diperoleh algoritma AS_{Rank} dengan jumlah iterasi yang besar dan pada saat jumlah iterasi sedikit hanya algoritma ACS yang mempunyai hasil paling bagus.

3.7.6 Perbandingan Algoritma ACO Berdasarkan Waktu CPU

Untuk mengetahui algoritma yang mempunyai waktu paling efektif dalam menyelesaikan suatu kasus dibandingkan algoritma yang lainnya maka, diberikan hasil percobaan Dorigo, M., dan Stu¨tzle, T. (2004) pada gambar 3.12 dan gambar 3.13. Pada gambar 3.12 kasus yang digunakan adalah kasus TSP simetri d198 (198 titik), dengan parameter – parameter yang digunakan seperti pada tabel 3.5, kecuali untuk nilai β, yaitu dengan

nilai $\beta = 5$ untuk semua algoritma. Dengan batas waktu maksimum yang ditentukan adalah 1000 detik. Untuk data selengkapnya dapat dilihat pada lampiran 2.



Gambar 3.12. Grafik performa algoritma ACO berdasarkan waktu CPU pada kasus d198 (198 titik).

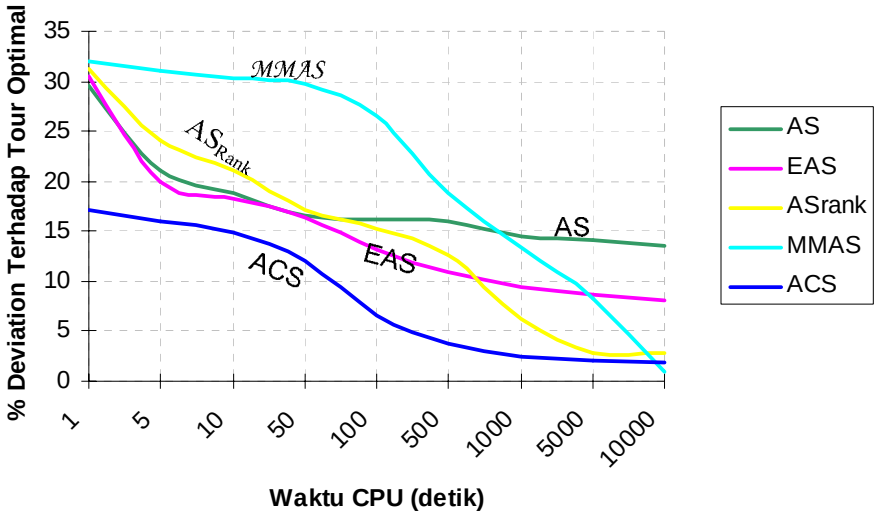
Dari gambar 3.12 diatas dapat ditunjukkan bahwa performa ACS lebih baik dibandingkan keempat algoritma yang lainnya pada waktu – waktu awal, hal ini terjadi sampai sekitar detik ke-2. Setelah itu, sampai sekitar detik ke-50 algoritma ACS hanya kurang bagus dari algoritma AS_{Rank} . Pada sekitar detik ke-60-an dan seterusnya Algoritma ACS hanya lebih baik dari algoritma AS dan EAS saja, sedangkan *MMAS* dan AS_{Rank} lebih baik daripada ACS. Algoritma AS_{Rank} mempunyai performa yang lebih baik dibandingkan algoritma yang lainnya kecuali dari algoritma ACS, hal ini terjadi hanya pada sekitar 2 detik awal saja. Algoritma AS_{Rank} juga lebih baik daripada algoritma yang lainnya pada sekitar detik ke-95 sampai waktu maksimal yang ditentukan, kecuali dari algoritma *MMAS*.

Algoritma EAS mempunyai performa yang lebih baik hanya dengan algoritma AS saja. Sedangkan dengan algoritma *MMAS*, performa EAS lebih baik hanya sampai sekitar mendekati detik ke-50. setelah itu performa algoritma *MMAS* lebih baik dibandingkan dengan EAS. Sementara algoritma AS mempunyai performa yang lebih baik dibandingkan dengan algoritma *MMAS* sekitar 20 detik awal, setelah itu performa algoritma AS tidak lebih baik dari keempat algoritma yang lainnya. Algoritma *MMAS* mempunyai performa yang kurang baik dibandingkan dengan algoritma lainnya hanya pada sekitar 20 detik awal saja. Algoritma *MMAS* mempunyai performa yang paling baik dibandingkan dengan keempat algoritma yang lainnya pada sekitar detik ke-100 sampai waktu maksimum yang ditentukan.

Pada kasus d198 ini, hanya algoritma AS yang menunjukkan performa kurang bagus sejak detik pertama sampai batas waktu maksimum yang ditentukan. Sedangkan performa yang stabil ditunjukkan oleh algoritma ACS, walau pada akhir waktu algoritma ACS mempunyai hasil yang kurang bagus dibandingkan dengan algoritma *MMAS* dan $AS_{Rank.}$ Sedangkan performa kurang stabil ditunjukkan oleh algoritma *MMAS*, pada detik – detik awal performanya kurang baik dibandingkan yang lainnya, tetapi pada akhir waktu performanya menjadi yang terbaik.

Pada gambar 3.13 kasus yang digunakan adalah kasus TSP simetri rat783 (783 titik), dengan parameter – parameter yang digunakan seperti pada tabel 3.5, kecuali untuk nilai β, yaitu dengan nilai $\beta = 5$ untuk

semua algoritma. Dengan batas waktu maksimum yang ditentukan adalah 10000 detik. Untuk data selengkapnya dapat dilihat pada lampiran 3.



Gambar 3.13. Grafik performa algoritma ACO berdasarkan waktu CPU pada kasus rat783 (783 titik)

Pada gambar 3.13 diatas, sebenarnya performa dari kelima algoritma hampir sama dengan kasus pada gambar 3.12 (kasus d198). Tetapi, pada kasus rat783 ini performa algoritma ACS menjadi paling bagus dari awal waktu sampai waktu maksimum yang ditentukan, hanya pada waktu terakhir performa ACS kurang bagus dibandingkan dengan performa *MMAS*. Performa kurang stabil tetap diperlihatkan oleh algoritma *MMAS*, pada awal waktu memiliki performa kurang baik tetapi pada akhir waktu mempunyai performa yang terbaik dibandingkan dengan algoritma yang lainnya. Pada algoritma AS, pada awal waktu mempunyai performa kurang bagus hanya dibandingkan dengan algoritma ACS saja.

Tetapi, semakin lama algoritma ini mempunyai performa kurang baik dibandingkan berturut – turut dengan algoritma EAS, AS_{Rank}, dan pada akhirnya menjadi algoritma yang mempunyai performa yang kurang bagus dibandingkan dengan keempat algoritma lainnya.

Pada algoritma EAS, pada awalnya mempunyai performa yang lebih baik daripada *MMAS* dan AS_{Rank}, tetapi pada akhir waktu hanya lebih baik dibandingkan dengan AS. Sedangkan pada algoritma $AS_{Rank,}$ pada awalnya performanya hanya lebih baik daripada *MMAS* saja, tetapi pada akhir waktu mempunyai performa yang lebih baik dibandingkan dengan AS dan EAS. Algoritma $AS_{Rank,}$ mempunyai performa yang kurang baik dibandingkan dengan ACS dan *MMAS* pada akhir waktu yang ditentukan.

Dari kedua kasus yang ada, dapat dikatakan bahwa tidak ada algoritma yang mempunyai performa sama baik untuk dua kasus yang berukuran berbeda. Untuk masing – masing algoritma mempunyai waktu yang tidak sama untuk mencapai hasil terbaiknya. Tetapi, dari kedua kasus yang telah dibahas dapat dikatakan bahwa waktu sangat berpengaruh terhadap performa yang dihasilkan oleh setiap algoritma. Dengan kata lain, semakin lama waktu untuk menyelesaikan masalah tersebut, maka performa yang dihasilkan oleh setiap algoritma juga meningkat dengan sendirinya.

3.7.7 Hasil Simulasi Algoritma ACO

Untuk menyelesaikan permasalahan TSP simetri dengan menerapkan algoritma ACO (AS, EAS, AS_{Rank}, $\mathcal{MMAS}$, dan ACS), digunakan software package ACOTSP version 1.0. yang dikembangkan oleh Thomas St¨utzle (http://www.aco-metaheuristic.org/aco-code). Software ini dibuat dengan program ANSI C dalam system operasi Linux, menggunakan GNU 2.95.3 gcc compiler. Dalam tugas akhir ini, system operasi linux dijalankan dalam system operasi windows.

Dalam software ini, penempatan semut dilakukan secara acak dan bilangan random yang dibangkitkan tidak sama sehingga hasil yang diperoleh untuk setiap percobaan tidak sama, walaupun dalam kasus dan parameter yang sama. Untuk mendapatkan hasil yang terbaik, maka eksperimen dilakukan dalam 10 kali percobaan dengan setiap percobaan waktu yang digunakan yaitu 120 detik. Hasil terbaik dari 10 kali percobaan inilah yang diambil sebagai hasil yang terbaik. Dalam tugas akhir ini kasus yang diselesaikan bersumber dari TSPLIB (*Traveling Salesman Problem Library*) (http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html) dan kasus yang dibuat sendiri dengan input pada program berupa titik koordinat (lampiran 4). Parameter – parameter yang digunakan yaitu parameter pada tabel 3.5 untuk semua algoritma kecuali $\mathcal{MMAS}$ untuk nilai $\rho = 0{,}5$ dan nilai $\beta = 5$ untuk semua algoritma.

- **Hasil simulasi Algoritma ACO dengan kasus bersumber dari TSPLIB**

  Kasus yang diselesaikan dengan sumber dari TSPLIB yaitu : ulysses22, eil51, eil76, kroA100, d198, lin318, d493, p654, u724, dan rat783. Berikut hasil yang diperoleh dari penyelesaian kasus – kasus diatas dengan algoritma ACO yang ditunjukkan dalam tabel 3.8 :

Tabel 3.8. Perbandingan hasil perhitungan AS, EAS, AS_{Rank}, *MMAS* dan ACS

		AS		**EAS**		**AS_{Rank}**		***MMAS***		**ACS**	
Kasus	N	*Best Tour* (mil)	Waktu (dtk)	*Best Tour* (mil)	Waktu (dtk)	*Best Tour* (mil)	Waktu (dtk)	*Best Tour* (mil)	Waktu (dtk)	*Best Tour* (mil)	Waktu (dtk)
ulysses22	22	7.046	25,95	**7.013**	26,28	**7.013**	25,25	**7.013**	2,00	**7.013**	35,83
eil51	51	438	20,86	427	0,83	**426**	0,83	**426**	0,14	**426**	4,52
eil76	76	550	51,41	541	7,17	539	4,36	**538**	78,74	**538**	15,25
kroA100	100	22.821	8,6	21.655	78,30	21.417	10,14	**21.282**	76,55	21.308	67,53
d198	198	17.076	15,51	16.470	47,94	16.140	38,30	16.135	107,93	**16.032**	71,35
lim318	318	45.879	119,04	43.678	107,64	42.908	109,09	42.783	115,73	**42.694**	98,69
d493	493	39.553	32,16	38.667	104,09	38.272	120,22	**37.138**	119,66	37.666	110,09
p654	654	39.957	25,78	38.432	76,37	38.368	99,54	37.902	121,66	**36.548**	111,57
u724	724	48.289	55,15	47.567	117,49	46.844	122,52	46.115	115,69	**43.831**	119,73
rat783	783	10.312	54,49	10.098	105,40	10.262	10,262	9.955	122,86	**9.705**	116,15

Keterangan tabel :

N : menyatakan jumlah titik pada setiap kasus.

Best Tour : menyatakan total jarak minimal yang dilalui semut dalam satuan mil.

Waktu : waktu CPU (CPU time) yang diperlukan untuk proses perhitungan setiap kasus dalam satuan detik.

Cetak tebal : hasil tour yang terbaik diantara kelima algoritma diatas.

Hasil simulasi yang ditunjukkan oleh tabel 3.8 diatas, dapat diketahui bahwa performa hasil yang paling jelek untuk semua kasus dimiliki oleh AS. Performa hasil yang terbaik dimiliki oleh ACS, walaupun pada kasus kroA100 dan d493 performanya kurang bagus dibandingkan dengan *MMAS*. Untuk algoritma EAS memiliki performa yang lebih bagus hanya bila dibandingkan dengan AS saja, tetapi pada kasus rat783 performa EAS lebih baik dibandingkan dengan AS_{Rank}. Algoritma AS_{Rank} memiliki performa yang kurang bagus dibandingkan dengan *MMAS* dan ACS, tetapi lebih baik daripada AS dan EAS. AS_{Rank} memiliki performa yang kurang bagus pada kasus terbesar rat783 bila dibandingkan dengan EAS. Algoritma *MMAS* memiliki performa lebih baik bila dibandingkan dengan AS, EAS, dan AS_{Rank}, tetapi kurang bagus bila dibandingkan dengan ACS.

Pada tipe kasus terkecil yaitu kasus ulysses22 hampir semua algoritma memiliki penyelesaian yang sama bagus, kecuali algoritma AS yang paling jelek dengan jarak tour yang dihasilkan yaitu 7.046 mil. Untuk kasus eil51 hanya algoritma AS dan EAS yang hasilnya kurang bagus, sedangkan ketiga algoritma yang lain memiliki penyelesaian yang sama bagus yaitu 426 mil. Kasus eil76 hanya *MMAS* dan ACS yang memiliki penyelesaian paling baik yaitu 538 mil, sedangakan untuk kasus kroA100 dan d493 hanya *MMAS* yang memiliki penyelesaian yang terbaik masing – masing sebesar 21.282 mil dan 37.138 mil. Untuk kasus yang lainnya hanya algoritma ACS yang memiliki penyelesaian terbaik.

Dari hasil simulasi diatas tidak berbeda jauh dengan hasil yang dibahas pada subbab 3.7.4, sehingga dapat dikatakan bahwa hanya algoritma AS yang memiliki performa yang paling jelek sedangkan yang terbaik algoritma ACS. Tetapi, pada beberapa kasus ACS belum tentu memiliki performa yang terbaik begitu pula dengan algoritma yang lainnya belum tentu jadi yang terjelek. Setiap algoritma dipengaruhi oleh beberapa faktor tertentu sehingga hasil yang diperoleh akan berbeda pada tiap kasus yang diselesaikan.

- **Hasil Simulasi Algoritma ACO dengan Kasus lain (dibuat sendiri)**

  Untuk lebih mendalami algoritma ACO, maka diberikan hasil penghitungan kasus yang dibuat sendiri mulai dari jumlah titik terendah yaitu 20 titik sampai 115 titik (data selengkapnya lihat lampiran 4). Percobaan dilakukan dalam waktu 10 detik untuk tiap algoritma yang dilakukan berulang kali hingga menemukan hasil yang terbaik. Berikut hasil penghitungan algoritma ACO yang ditunjukkan pada tabel 3.9

Tabel 3.9. Perbandingan hasil perhitungan Algoritma ACO

	AS			**EAS**			**AS_{Rank}**			***MMAS***			**ACS**		
N	*Best* (mil)	Wkt (dtk)	Itr	*Best* (mil)	Wkt (dtk)	Itr	*Best* (mil)	Wkt (dtk)	Itr	*Best* (mil)	Wkt (dtk)	Itr	*Best* (mil)	Wkt (dtk)	Itr
20	**251**	0.010	134	**251**	0.000	425	**251**	0.000	15	**251**	0.000	10	**251**	0.000	15
25	**301**	0.010	644	**301**	0.090	3953	**301**	0.000	26	**301**	0.000	50	**301**	0.000	9
30	**322**	0.370	7033	**322**	0.000	3	**322**	0.000	28	**322**	0.000	27	**322**	0.010	1366
35	**362**	4.840	88919	**362**	0.010	28	**362**	0.010	47	**362**	0.090	341	**362**	0.000	181
40	**413**	1.730	18989	**413**	0.800	7573	**413**	0.010	46	**413**	0.020	102	**413**	0.020	277
45	435	8.120	80813	434	0.010	308	433	0.000	40	**426**	1.330	3806	**426**	1.160	23495
50	458	1.560	8468	453	0.120	1521	453	0.030	61	**451**	0.420	1566	**451**	0.160	1138
55	483	2.910	18871	**476**	0.790	4114	**476**	0.020	58	**476**	0.310	759	**476**	2.910	53498
60	501	9.910	46685	491	7.340	65018	490	0.110	194	**485**	0.430	1447	**485**	7.640	54369
65	550	1.890	9733	537	0.770	4559	542	0.070	102	**535**	1.840	2184	**535**	4.700	31615
70	583	2.960	12961	564	6.790	43233	563	0.150	103	**562**	8.760	9413	**562**	5.710	95052
75	599	1.520	4946	582	0.510	1991	580	0.110	132	**571**	0.750	1553	**571**	2.330	17357
80	607	3.880	7836	595	4.570	11053	588	0.200	200	**583**	3.230	2699	**583**	0.420	1989
85	619	1.660	4259	596	3.020	11312	596	0.170	138	**592**	5.040	2925	**592**	2.260	14665
90	640	7.080	22355	623	8.530	22623	619	0.060	63	614	1.810	1397	**611**	0.830	4097

95	652	4.470	3631	625	3.430	8127	621	0.410	144	**620**	4.400	2479	**620**	0.710	3845
100	668	5.980	3599	631	3.260	11265	631	0.090	80	**628**	9.860	10318	**628**	9.780	65785
105	675	4.890	2713	654	9.950	25266	640	0.120	96	636	7.130	7837	**635**	0.110	2364
110	705	8.500	4416	670	0.770	1674	669	0.130	117	664	1.360	696	**663**	2.010	8717
115	742	8.890	5957	709	0.880	913	709	0.240	148	**704**	0.900	462	**704**	2.920	10381

Keterangan tabel :

N : menyatakan jumlah titik pada setiap kasus.

Best : menyatakan total jarak minimal yang dilalui semut dalam satuan mil.

Wkt : waktu CPU (CPU time) yang diperlukan untuk proses perhitungan setiap kasus dalam satuan detik.

Itr : jumlah iterasi yang diperlukan untuk proses penghitungan setiap kasus untuk mencapai tour terbaik.

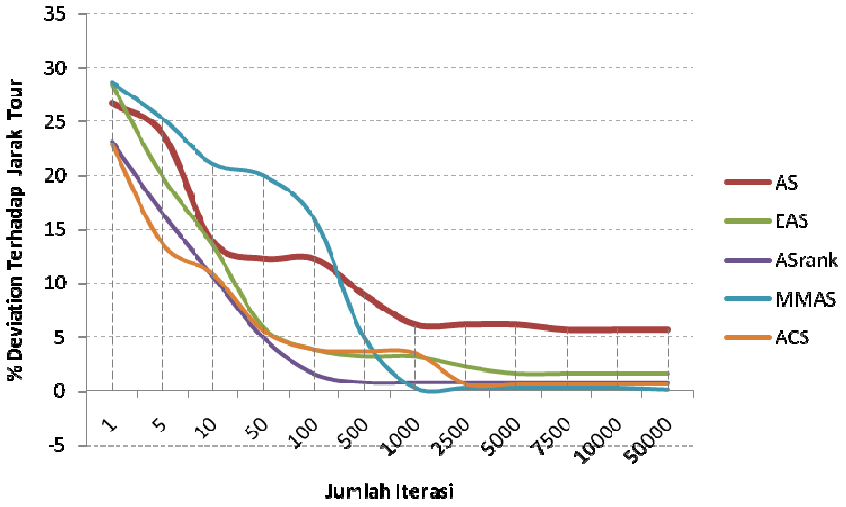
Cetak tebal : hasil tour yang terbaik diantara kelima algoritma diatas.

Hasil simulasi yang ditunjukkan oleh tabel 3.9 diatas, dapat diketahui bahwa performa terbaik dimiliki oleh Algoritma ACS dan yang terjelek algoritma AS. Pada kasus – kasus yang mempunyai jumlah titik kecil mulai dari 20 titik sampai 40 titik semua algoritma mempunyai penyelesaian yang sama baiknya. Untuk kasus 55 titik hanya algoritma AS yang mempunyai hasil terjelek dengan tour terbaiknya 483 mil sedangkan algoritma lainnya mempunyai penyelesaian yang sama bagus yaitu 476 mil. Sedangkan kasus lainnya hanya algoritma *MMAS* dan ACS yang mempunyai penyelesaian terbaik. Secara umum algoritma *MMAS* mempunyai peyelesaian sama bagus dengan algoritma ACS, kecuali untuk kasus 90 titik, 105 titik, dan 110 titik. Performa algoritma EAS lebih baik dari AS, tetapi lebih jelek daripada algoritma AS_{Rank}, *MMAS* dan ACS. Kecuali untuk kasus 50 titik algoritma EAS lebih baik daripada AS_{Rank}. Algoritma AS_{Rank} mempunyai performa yang lebih baik daripada AS dan EAS tetapi lebih jelek daripada *MMAS* dan ACS.

Hasil yang diperoleh pada penghitungan kasus diatas secara umum hampir sama dengan hasil yang diperoleh pada kasus TSPLIB. Algoritma ACS mempunyai performa paling bagus dibandingkan dengan algoritma lainnya, sedangkan algoritma AS mempunyai performa paling jelek dibandingkan dengan yang lainnya.

Dari hasil simulasi yang diperoleh pada tabel 3.8 dan tabel 3.9, dapat diambil kesimpulan bahwa tidak ada hubungan antara waktu dan banyaknya titik yang diselesaikan. Hal ini disebabkan setiap algoritma

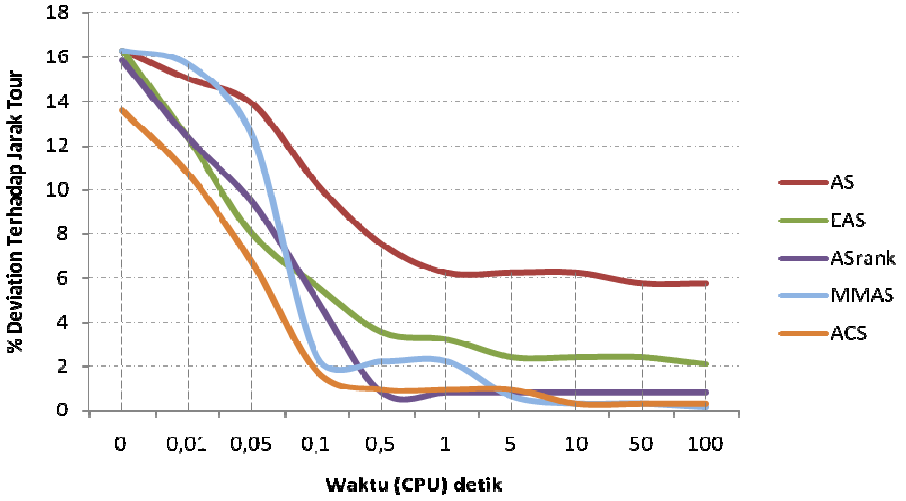
dalam mencari tour terbaiknya menggunakan jumlah semut yang sama dengan jumlah titik masalah (kecuali algoritma ACS dengan 10 semut saja), sehingga ada semut yang bisa menemukan tour tebaik dalam waktu yang cepat dan ada yang memerlukan waktu yang lama. Begitu pula jumlah iterasi yang diperlukan dalam mencapai hasil terbaik tidak tergantung pada jumlah titik yang diselesaikan, tetapi tergantung dari waktu yang diperlukan untuk mencapai hasil tour terbaik. Semakin cepat waktu yang diperlukan untuk mencapai tour terpendek maka semakin sedikit jumlah iterasi yang diperlukan begitu pula sebaliknya.



Gambar 3.14. Grafik performa algoritma ACO berdasarkan jumlah iterasi pada kasus 100 titik.

Gambar 3.14 merperlihatkan performa algoritma ACO berdasarkan jumlah iterasi, hasil simulasi yang dilakukan pada kasus 100 titik. Secara umum hasil yang diperoleh pada simulasi hampir sama dengan eksperimen yang dilakukan oleh Dorigo, M.,.& Stu¨tzle, T. (2004) pada gambar 3.11, algoritma ACS memiliki performa terbaik pada jumlah iterasi yang sedikit,

sedangkan algoritma AS mempunyai performa yang kurang bagus pada jumlah iterasi berapapun. Untuk jumlah iterasi yang besar algoritma *MMAS* yang terbaik pada hasil simulasi sedangkan pada percobaan Dorigo, M.,.& Stu¨tzle, T. (2004) algoritma AS_{Rank} yang terbaik pada jumlah iterasi yang besar.



Gambar 3.15. Grafik performa algoritma ACO berdasarkan waktu perhitungan CPU pada kasus 100 titik.

Gambar 3.15 merperlihatkan performa algoritma ACO berdasarkan waktu perhitungan CPU, hasil simulasi dilakukan pada kasus 100 titik. Secara umum hasil yang diperoleh pada simulasi hampir sama dengan eksperimen yang dilakukan oleh Dorigo, M.,.& Stu¨tzle, T. (2004) pada gambar 3.12 maupun gambar 3.13, algoritma ACS memiliki performa terbaik pada waktu – waktu awal, sedangkan algoritma AS mempunyai performa yang paling jelek pada waktu berapapun. Untuk waktu yang paling lama algoritma *MMAS* yang terbaik.

BAB IV

PENUTUP

4.1 Kesimpulan

Dari pembahasan yang telah diuraikan pada bab sebelumnya, dapat diambil suatu kesimpulan tentang algoritma *Ant Colony Optimization* untuk menyelesaikan masalah *Traveling Salesman* sebagai berikut :

1. Algoritma *Ant Colony Optimization* (*Ant system, Elitist Ant System, Rank Based Ant System, Max-Min Ant System, dan Ant Colony System*) dapat digunakan untuk menyelesaikan masalah meminimalkan jarak tempuh salesman.
2. Secara umum dari kelima algoritma ACO, algoritma ACS dapat memberikan solusi yang lebih mendekati optimal untuk masalah traveling salesman jika dibandingkan dengan keempat algoritma ACO yang lainnya. Sedangkan, solusi yang kurang bagus dihasilkan oleh algoritma AS jika dibandingkan dengan yang lainnya.
3. Durasi waktu proses perhitungan pada setiap algoritma sangat berpengaruh besar terhadap solusi yang dihasilkan. Semakin lama waktu yang dipergunakan, maka solusi yang dihasilkan algoritma ACO lebih mendekati optimal.
4. Waktu yang diperlukan oleh setiap algoritma untuk mencapai tour terbaiknya tidak berkorelasi positif dengan jumlah titik yang

diselesaikan, karena dalam mencari tour terbaiknya algoritma menggunakan semut lebih dari satu.

5. Jumlah iterasi yang dipergunakan memiliki pengaruh yang cukup signifikan, semakin banyak jumlah iterasi yang dipergunakan maka solusi yang dihasilkan lebih mendekati optimal.
6. Jumlah iterasi yang diperlukan dalam mencapai hasil terbaik tidak mempunyai korelasi positif dengan jumlah titik yang diselesaikan, tetapi tergantung dari waktu yang diperlukan untuk mencapai hasil tour terbaik.

4.2 Saran

Masalah optimasi yang dapat diselesaikan dengan algoritma *Ant Colony Optimization* selain masalah *traveling salesman* masih banyak lagi dan algoritma ini dapat dimodifikasi sesuai dengan aplikasi masalah yang diselesaikan. Karena keterbatasan penulis dalam mengerjakan tugas akhir ini maka masih banyak hal yang perlu dibenahi dan disempurnakan lagi. Oleh karena itu, diharapkan untuk melanjutkan penelitian tentang penerapan algoritma *Ant Colony Optimization* yang telah dimodifikasi untuk masalah – masalah optimasi lainnya.

DAFTAR PUSTAKA

ACOTSP, Version 1.0 : http://www.aco-metaheuristic.org/aco-code. Diakses tanggal 17 januari 2009. pukul 06.14 wib.

Bullnheimer, B., Hartl, R. F., dan Strauss, C. (1997). ***A new rank based version of the Ant System—A computational study***. Technical report, Institute of Management Science, University of Vienna, Austria.

Bullnheimer, B., Hartl, R. F., dan Strauss, C. (1999). ***An improved ant system algorithm for the vehicle routing problem***. Technical report, Institute of Management Science, University of Vienna, Austria.

Dorigo, M., dan Gambardella, L. M. (1997). ***Ant colonies for the traveling salesman problem***. Tech.Rep/IRIDIA/1996-003, Université Libre de Bruxelles, Belgium.

Dorigo, M., dan Gambardella, L., (1996). ***Ant Colony System: A Cooperative learning Approach to the Traveling Salesman Problem.*** Tech.Rep/IRIDIA/1996-005, Université Libre de Bruxelles, Belgium.

Dorigo, M., Maniezzo, V., dan Colorni, A. (1991a). ***Positive feedback as a search strategy***. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan.

Dorigo, M., Maniezzo, V., dan Colorni, A. (1991b). ***The Ant System: An autocatalytic optimizing process***. Technical report 91-016 revised, Dipartimento di Elettronica, Politecnico di Milano, Milan.

Dorigo, M., Maniezzo, V., dan Colorni, A. (1996). ***The Ant System: Optimization by a colony of cooperating agents***. IEEE Transactions on Systems, Man, and Cybernetics—Part B, 26(1), pp.1-13.

Dorigo, M., dan Socha K. (2007), ***An Introduction to Ant Colony Optimization***, Tech.Rep/IRIDIA/2006-010, Université Libre de Bruxelles, Belgium.

Dorigo, M., dan Stu¨tzle, T. (2004). ***Ant colony optimization***. A Bradford book. The MIT Press Cambridge, Massachusetts London, England.

Irawanto, B., Sarwadi, dan Surarso, B. (2004), ***Buku Ajar Program Linear***, Jurusan Matematika Undip. Semarang.

Mutakhiroh, I., Saptono, F., Hasanah, N., dan Wiryadinata, R,. (2007). ***Pemanfaatan Metode Heuristik Dalam Pencarian Jalur Terpendek Dengan Algoritma Semut dan Algoritma Genetik***. Seminar Nasional Aplikasi Teknologi Informasi. ISSN: 1907-5022. Yogyakarta.

Rosenkrantz, D.J, Stearns, R.E, dan Lewis, P.M. (1977). "***An analysis of several heuristics for the traveling salesman problem***," *SIAM Journal on Computing*, vol. 6, pp. 563–581.

Stu¨ tzle, T., dan Hoos, H. H. (1997). ***The MAX-MIN Ant System and local search for the traveling salesman problem***. In T. Ba¨ck, Z. Michalewicz, & X. Yao (Eds.), Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97) (pp. 309–314). Piscataway, NJ, IEEE Press.

TSPLIB : http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html. Diakses tanggal 27 desember 2008. pukul 05.45 wib.

Wardy, I. S. (2007). ***Penggunaan graph dalam algoritma semut untuk melakukan optimisasi,*** Program studi Teknik Informatika, ITB, Bandung.

Wilson, R. J., dan Watkhins, J. J. (1990). ***Graph An Introductionary Approach, A First Course in Discrete Mathematics***. John Willey and Sons, New York.

Lampiran 1.a

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan jumlah iterasi pada kasus kroA100 (error dalam %)

Iterasi ke -	1	5	10	50	100	500	1000	5000	10000
AS	70.8	70.8	67.2	32.6	24.5	24.5	24.5	24.5	24.5
EAS	70.8	70.8	63.6	13.5	6.3	6.3	6.3	6.3	6.3
ASrank	70.8	70.8	70.8	67.65	40	5	2	2	2
MMAS	70.8	70.8	70.8	68.55	61.35	11.7	5.6	5.6	5.6
ACS	30	11.7	10.8	11.7	9.45	5.4	5.4	5.4	5.4

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan jumlah iterasi pada kasus kroA100 (panjang tour)

Iterasi ke -	1	5	10	50	100	500	1000	5000	10000
AS	36349.656	36349.656	35583.504	28219.932	26496.09	26496.09	26496.09	26496.09	26496.09
EAS	36349.656	36349.656	34817.352	24155.07	22622.766	22622.766	22622.766	22622.766	22622.766
ASrank	36349.656	36349.656	36349.656	35679.273	29794.8	22346.1	21707.64	21707.64	21707.64
MMAS	36349.656	36349.656	36349.656	35870.811	34338.507	23771.994	22473.792	22473.792	22473.792
ACS	27666.6	23771.994	23580.456	23771.994	23293.149	22431.228	22431.228	22431.228	22431.228

Lampiran 2.

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan waktu CPU pada kasus d198 (error dalam %))

Waktu (detik)	0.1	0.5	1	5	10	50	100	500	1000
AS	20.53	19.06	10.53	8.8	7.73	7.46	6.53	6	6
EAS	20.26	18.8	9.3	7	5.2	4.4	3.2	2.8	2.8
ASrank	17.86	12.8	8.8	5.3	3	2.48	2.13	2	2
MMAS	20.6	19.3	17.6	15.86	10.1	4.26	2	1.33	1.2
ACS	10.4	9.06	7.6	6.46	4.93	3.73	2.8	2.53	2.53

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan waktu CPU pada kasus d198 (panjang tour)

Waktu (detik)	0.1	0.5	1	5	10	50	100	500	1000
AS	19019.634	18787.668	17441.634	17168.64	16999.794	16957.188	16810.434	16726.8	16726.8
EAS	18977.028	18746.64	17247.54	16884.6	16600.56	16474.32	16284.96	16221.84	16221.84
ASrank	18598.308	17799.84	17168.64	16616.34	16253.4	16171.344	16116.114	16095.6	16095.6
MMAS	19030.68	18825.54	18557.28	18282.708	17373.78	16452.228	16095.6	15989.874	15969.36
ACS	17421.12	17209.668	16979.28	16799.388	16557.954	16368.594	16221.84	16179.234	16179.234

Lampiran 3.

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan waktu CPU pada kasus rat783 (error dalam %)

Waktu (detik)	1	5	10	50	100	500	1000	5000	10000
AS	29.58	21	18.75	16.6	16.13	16	14.5	14.16	13.6
EAS	30.4	20	18.3	16.45	13.13	11	9.5	8.75	8.13
ASrank	31.25	24.1	21.08	17.08	15.25	12.7	6.25	2.9	2.9
MMAS	32.08	31	30.3	29.8	26.45	18.75	13.3	8.3	1
ACS	17.08	16	14.8	11.95	6.6	3.75	2.5	2.08	1.87

Hasil eksperiment (Dorigo, M.,.& Stu¨tzle, T. (2004)) perbandingan algoritma ACO berdasarkan waktu CPU pada kasus rat783 (panjang tour)

Waktu (detik)	1	5	10	50	100	500	1000	5000	10000
AS	11410.8148	10655.26	10457.125	10267.796	10226.4078	10214.96	10082.87	10052.9296	10003.616
EAS	11483.024	10567.2	10417.498	10254.587	9962.2278	9774.66	9642.57	9576.525	9521.9278
ASrank	11557.875	10928.246	10662.3048	10310.0648	10148.915	9924.362	9356.375	9061.374	9061.374
MMAS	11630.9648	11535.86	11474.218	11430.188	11135.187	10457.125	9977.198	9536.898	8894.06
ACS	10310.0648	10214.96	10109.288	9858.317	9387.196	9136.225	9026.15	8989.1648	8970.6722

Lampiran 4

Koordinat Masalah

No	X	Y	No	X	Y	No	X	Y
1.	41	49	**41.**	42	7	**81.**	55	54
2.	35	17	**42.**	24	12	**82.**	15	47
3.	55	45	**43.**	23	3	**83.**	14	37
4.	55	20	**44.**	11	14	**84.**	11	31
5.	15	30	**45.**	6	38	**85.**	16	22
6.	25	30	**46.**	2	48	**86.**	4	18
7.	20	50	**47.**	8	56	**87.**	28	18
8.	10	43	**48.**	13	52	**88.**	26	52
9.	55	60	**49.**	6	68	**89.**	26	35
10.	30	60	**50.**	47	47	**90.**	31	67
11.	20	65	**51.**	49	58	**91.**	15	19
12.	50	35	**52.**	27	43	**92.**	22	22
13.	30	25	**53.**	37	31	**93.**	18	24
14.	15	10	**54.**	57	29	**94.**	26	27
15.	30	5	**55.**	63	23	**95.**	25	24
16.	10	20	**56.**	53	12	**96.**	22	27
17.	5	30	**57.**	32	12	**97.**	25	21
18.	20	40	**58.**	36	26	**98.**	19	21
19.	15	60	**59.**	21	24	**99.**	20	56
20.	45	65	**60.**	17	34	**100.**	18	18
21.	45	20	**61.**	12	24	**101.**	30	12
22.	45	10	**62.**	24	58	**102.**	23	45
23.	55	5	**63.**	27	69	**103.**	56	28
24.	65	35	**64.**	15	77	**104.**	35	37
25.	65	20	**65.**	62	77	**105.**	47	19
26.	45	30	**66.**	49	73	**106.**	56	28
27.	35	40	**67.**	67	5	**107.**	67	45
28.	41	37	**68.**	56	39	**108.**	76	33
29.	64	42	**69.**	37	47	**109.**	12	79
30.	40	60	**70.**	37	56	**110.**	65	54
31.	31	52	**71.**	57	68	**111.**	76	45
32.	35	69	**72.**	47	16	**112.**	34	78
33.	53	52	**73.**	44	17	**113.**	56	78
34.	65	55	**74.**	46	13	**114.**	36	27
35.	63	65	**75.**	49	11	**115.**	88	33
36.	2	60	**76.**	49	42			
37.	20	20	**77.**	53	43			
38.	5	5	**78.**	61	52			
39.	60	12	**79.**	57	48			
40.	40	25	**80.**	56	37			

Lampiran 5
Contoh Hasil perhitungan algoritma ACO untuk beberapa kasus

```
ACO algorithms for the TSP, V1.0
Parameter-settings

Max – tries : 1
Max - tours : 100
Max - time  : 10 /* seconds */
optimum     : 1
ants        : 20
nnants      : 20
alpha       : 1
beta        : 5
rho         : 0.5
q0          : 0.0
nnls        : 20
localsearch : 0
dlb         : 1
as.flag     : 1
eas.flag    : 0
ras.flag    : 0
mmas.flag   : 0
bwas.flag   : 0
acs.flag    : 0

begin problem 20.tsp
seed 1243277326
begin try 0,

best 290          iteration 1       tour 22     time 0.000
best 257          iteration 2       tour 42     time 0.000
best 256          iteration 18      tour 362    time 0.000
best 252          iteration 47      tour 942    time 0.010
best 251          iteration 134     tour 2682   time 0.010

end try 0
end problem 20


ACO algorithms for the TSP, V1.0
Parameter-settings

Max – tries : 1
Max - tours : 100
Max - time  : 10 /* seconds */
optimum     : 1
ants        : 25
nnants      : 20
alpha       : 1
beta        : 5
rho         : 0.5
q0          : 0.0
elitistants : 25
nnls        : 20
```

```
localsearch : 0
dlb         : 1
as.flag     : 0
eas.flag    : 1
ras.flag    : 0
mmas.flag   : 0
bwas.flag   : 0
acs.flag    : 0

begin problem 25.tsp
seed 1243277959
begin try 0,

best 324          iteration 1       tour 27     time 0.000
best 313          iteration 2       tour 52     time 0.000
best 312          iteration 5       tour 127    time 0.000
best 311          iteration 6       tour 152    time 0.000
best 308          iteration 8       tour 202    time 0.000
best 304          iteration 51      tour 1277   time 0.000
best 303          iteration 173     tour 4327   time 0.010
best 301          iteration 3953    tour 98827  time 0.090

end try 0
end problem 25

ACO algorithms for the TSP, V1.0
Parameter-settings

Max – tries : 1
Max - tours : 100
Max - time  : 10 /* seconds */
optimum     : 1
ants        : 30
nnants      : 20
alpha       : 1
beta        : 5
rho         : 0.1
q0          : 0.0
rasranks    : 6
nnls        : 20
localsearch : 0
dlb         : 1
as.flag     : 0
eas.flag    : 0
ras.flag    : 1
mmas.flag   : 0
bwas.flag   : 0
acs.flag    : 0

begin problem 30.tsp
seed 1243287282
begin try 0,

best 363          iteration 1       tour 32     time 0.000
best 358          iteration 2       tour 62     time 0.000
best 335          iteration 3       tour 92     time 0.000
```

```
best 333          iteration 4       tour 122    time 0.000
best 324          iteration 7       tour 212    time 0.000
best 323          iteration 15      tour 452    time 0.000
best 322          iteration 28      tour 842    time 0.000

end try 0
end problem 30


ACO algorithms for the TSP, V1.0
Parameter-settings

Max – tries : 1
Max - tours : 100
Max - time  : 10 /* seconds */
optimum     : 1
ants        : 35
nnants      : 20
alpha       : 1
beta        : 5
rho         : 0.5
q0          : 0.0
nnls        : 20
localsearch : 0
dlb         : 1
as.flag     : 0
eas.flag    : 0
ras.flag    : 0
mmas.flag   : 1
bwas.flag   : 0
acs.flag    : 0

begin problem 35.tsp
seed 1243289220
begin try 0,

best 417          iteration 1       tour 37     time 0.000
best 401          iteration 2       tour 72     time 0.000
best 385          iteration 4       tour 142    time 0.000
best 377          iteration 5       tour 177    time 0.010
best 370          iteration 7       tour 247    time 0.010
best 369          iteration 311     tour 10887  time 0.080
best 367          iteration 322     tour 11272  time 0.080
best 362          iteration 341     tour 11937  time 0.090

end try 0
end problem 35
```

```
ACO algorithms for the TSP, V1.0
Parameter-settings

Max – tries : 1
Max - tours : 100
Max - time  : 10 /* seconds */
optimum     : 1
ants        : 80
nnants      : 20
alpha       : 1
beta        : 5
rho         : 0.1
q0          : 0.9
nnls        : 20
localsearch : 0
dlb         : 1
as.flag     : 0
eas.flag    : 0
ras.flag    : 0
mmas.flag   : 0
bwas.flag   : 0
acs.flag    : 1

begin problem 80.tsp
seed 1243361799
begin try 0,

best 665          iteration 1       tour 12     time 0.000
best 658          iteration 2       tour 22     time 0.000
best 651          iteration 11      tour 112    time 0.010
best 644          iteration 14      tour 142    time 0.010
best 643          iteration 17      tour 172    time 0.010
best 636          iteration 18      tour 182    time 0.010
best 625          iteration 28      tour 282    time 0.020
best 618          iteration 36      tour 362    time 0.020
best 615          iteration 42      tour 422    time 0.030
best 603          iteration 70      tour 702    time 0.040
best 602          iteration 123     tour 1232   time 0.040
best 600          iteration 316     tour 3162   time 0.070
best 597          iteration 319     tour 4712   time 0.070
best 595          iteration 471     tour 8512   time 0.100
best 590          iteration 851     tour 9182   time 0.100
best 589          iteration 918     tour 19752  time 0.180
best 584          iteration 1975    tour 19892  time 0.190
best 583          iteration 1989    tour 22356  time 0.420

end try 0
end problem 80
```

Lampiran 6

Hasil simulasi algoritma ACO berdasarkan jumlah iterasi pada kasus 100 titik (Jarak Tour (mil))

Iterasi ke-	1	5	10	50	100	500	1000	2500	5000	7500	10000	50000
AS	793	775	713	703	703	682	665	665	665	662	662	662
EAS	804	750	711	663	650	646	646	640	636	636	636	636
ASrank	771	729	693	657	636	631	631	631	631	631	631	631
MMAS	805	784	758	751	726	657	628	628	628	628	628	627
ACS	769	711	694	661	650	649	648	630	630	630	630	630

Hasil simulasi algoritma ACO berdasarkan jumlah iterasi pada kasus 100 titik (Deviasi (%))

Iterasi ke-	1	5	10	50	100	500	1000	2500	5000	7500	10000	50000
AS	26,67732	23,80192	13,89776	12,30032	12,30032	8,945687	6,230032	6,230032	6,230032	5,750799	5,750799	5,750799
EAS	28,4345	19,80831	13,57827	5,910543	3,833866	3,194888	3,194888	2,236422	1,597444	1,597444	1,597444	1,597444
ASrank	23,16294	16,45367	10,70288	4,952077	1,597444	0,798722	0,798722	0,798722	0,798722	0,798722	0,798722	0,798722
MMAS	28,59425	25,23962	21,08626	19,96805	15,97444	4,952077	0,319489	0,319489	0,319489	0,319489	0,319489	0,159744
ACS	22,84345	13,57827	10,86262	5,591054	3,833866	3,674121	3,514377	0,638978	0,638978	0,638978	0,638978	0,638978

Lampiran 7

Hasil simulasi algoritma ACO berdasarkan waktu perhitungan CPU pada kasus 100 titik (Jarak Tour (mil))

Waktu (dtk)	0	0,01	0,05	0,1	0,5	1	5	10	50	100
AS	728	720	713	690	673	665	665	665	662	662
EAS	728	703	676	661	648	646	641	641	641	639
ASrank	725	703	685	657	631	631	631	631	631	631
MMAS	728	724	704	641	640	640	630	628	628	627
ACS	711	693	668	637	632	632	632	628	628	628

Hasil simulasi algoritma ACO berdasarkan waktu perhitungan CPU kasus 100 titik (Deviasi (%))

Waktu (dtk)	0	0,01	0,05	0,1	0,5	1	5	10	50	100
AS	16,29393	15,01597	13,89776	10,22364	7,507987	6,230032	6,230032	6,230032	5,750799	5,750799
EAS	16,29393	12,30032	7,98722	5,591054	3,514377	3,194888	2,396166	2,396166	2,396166	2,076677
ASrank	15,8147	12,30032	9,42492	4,952077	0,798722	0,798722	0,798722	0,798722	0,798722	0,798722
MMAS	16,29393	15,65495	12,46006	2,396166	2,236422	2,236422	0,638978	0,319489	0,319489	0,159744
ACS	13,57827	10,70288	6,709265	1,757188	0,958466	0,958466	0,958466	0,319489	0,319489	0,319489