Studi Perbandingan Algoritma *Ant Colony System* dan Algoritma *Ant System* pada Permasalahan *Traveling Salesman Problem*

Leonardo Z Tomarere

Laboratorium Ilmu dan Rekayasa Komputasi
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Jl. Ganesa 10, Bandung

e-mail: if12028@students.if.itb.ac.id

ABSTRAK

Pada tugas akhir ini akan dibandingkan dua buah algoritma yang terinspirasi dari kelakuan koloni semut pada saat mencari makan, yaitu Algoritma *Ant System* (AS) serta Algoritma *Ant Colont System* (ACS) yang merupakan pengembangan dari Algoritma AS. Performa keduanya akan dibandingkan melalui permasalahan *Traveling Salesman Problem* (TSP) yang didefenisikan sebagai permasalahan untuk menemukan Sirkuit Hamilton dengan bobot minimum pada sebuah graf terhubung. Perangkat lunak untuk pengujian dikembangkan menggunakan Java 6 dengan kakas pengembangan Java NetBeans 6.5. Seluruh TSP yang digunakan diambil dari TSPLIB95. Sedangkan fokus pengujian terletak pada kualitas solusi yang dihasilkan serta waktu pencarian yang dibutuhkan. Hasil pengujian menunjukkan bahwa untuk jumlah semut dan jumlah iterasi yang sama, Algoritma ACS memberikan solusi yang lebih baik serta waktu pencarian yang lebih cepat daripada Algoritma AS. Dengan kata lain, Algoritma ACS terbukti lebih baik daripada Algoritma AS.

Kata kunci: *Ant System, Ant Colony System,* TSP, *Pheromone,* Sirkuit Hamilton, TSPLIB95, Java.

1. Pendahuluan

Alam merupakan keajaiban. Salah satu keajaiban yang dapat diamati adalah koloni serangga. Koloni serangga digolongkan sebagai serangga sosial, karena tingkah lakunya lebih diarahkan untuk kepentingan koloni secara keseluruhan dibandingkan dengan kebutuhannya sendiri sebagai individu[6]. Koloni melakukan koordinasi melalui interaksi yang relatif sederhana. Namun, melalui interaksi tersebut koloni diketahui mampu memecahkan permasalahan yang sulit[2]. Hal inilah yang mendasari ilmuwan untuk mengadakan riset terhadap kecerdasan yang dimiliki koloni serangga (*Swarm Intelligence*).

Salah satu riset yang diadakan mengarah pada koloni semut. Riset menunjukkan bahwa pada saat mencari makanan (foraging), koloni semut mampu menemukan rute terpendek dari sarang menuju sumber makanan[6]. Hal ini dapat dilakukan karena semut berkomunikasi secara tidak langsung dengan semut yang lain menggunakan "jejak *pheromone*" (untuk selanjutnya akan disebut dengan *pheromone*), dimana semut cenderung untuk memilih rute dengan konsentrasi *pheromone* yang tinggi[6].

Secara singkat, perilaku ini dapat dijelaskan sebagai berikut: pada mulanya, semut memilih rute secara acak sambil meletakkan *pheromone*. Melalui observasi, jumlah semut per satuan waktu yang berhasil melewati rute yang pendek akan lebih banyak daripada rute yang panjang. Dengan kata lain, konsentrasi *pheromone* pada rute yang pendek akan lebih tinggi dibandingkan rute yang panjang. Mulai saat ini, semut cenderung memilih rute yang pendek karena memiliki konsentrasi *pheromone* yang lebih tinggi daripada yang lain. Pada akhirnya, rute terpendek yang ditemukan koloni merupakan rute dengan konsentrasi *pheromone* paling tinggi[5].

Terinspirasi dari perilaku tersebut, pada tahun 1991 Marco Dorigo mengemukakan metode pencarian heuristik bernama Algoritma Ant System (AS)[6]. Metode ini meniru perilaku koloni semut ketika mencari makan dimana sekumpulan semut buatan yang relatif sederhana akan bekerja secara kolektif sambil berkomunikasi menggunakan *pheromone* pada permasalahan optimasi diskrit yang sulit[6]. Karena kesederhanaan serta kemiripan rumusannya, Algoritma AS pertama kali dikembangkan pada TSP (Traveling Salesman Problem)[5].

TSP sendiri dapat dipandang sebagai masalah mencari rute terpendek yang harus ditempuh oleh seseorang yang berangkat dari kota asal untuk mengunjungi setiap kota tepat satu kali lalu kembali lagi ke kota asal keberangkatannya. Dengan kata lain, TSP adalah

permasalahan untuk mencari Sirkuit Hamilton dengan bobot minimum pada sebuah graf terhubung[8].

Walaupun formulasinya sederhana dan mudah dimengerti, TSP merupakan salah satu permasalahan optimasi yang sulit dipandang dari sudut komputasinya[8]. Menggunakan algoritma *exhauIstive search* (pada graf lengkap), TSP dapat dipecahkan dengan mengenumerasi $\frac{(n-1)!}{2}$ Sirkuit Hamilton. Akibatnya, untuk *n* yang besar, waktu komputasinya akan meningkat secara eksponesial[9].

Algoritma AS memberikan hasil yang menjanjikan pada TSP yang kecil ($n \leq 30$). Namun seiring bertambahnya ukuran TSP, kualitas waktu serta solusi yang dihasilkan akan menurun. Oleh karena itu Algoritma AS dimodifikasi menjadi Algoritma *Ant Colony System* (ACS) dengan tujuan untuk meningkatkan performanya pada TSP yang lebih besar[5].

Prinsip utama yang dipakai pada Algoritma AS masih dipakai pada Algoritma ACS yaitu pemakaian umpan balik positif (*positive feedback*) melalui pemakaian *pheromone*. *Pheromone* yang diletakkan disepanjang rute dimaksudkan agar semut lebih tertarik untuk mengambil rute tersebut (*reinforcement*), sehingga solusi terbaik nantinya akan memiliki konsentrasi *pheromone* yang tinggi. Agar tidak terjebak kedalam optimal lokal (*local optima*) maka digunakan umpan balik negatif (*negative feedback*) berupa penguapan *pheromone*[12].

Sedangkan perbedaan utama antara Algoritma AS dan ACS adalah: aturan transisi status yang berbeda, aturan pembaharuan *pheromone* global yang berbeda, serta penambahan aturan pembaharuan *pheromone* lokal[15]. Dengan modifikasi ini diharapkan hasil optimasi pada TSP yang diperoleh akan menjadi lebih baik (mendapatkan rute terpendek dalam waktu seminimal mungkin).

1.1 Ruang Lingkup

Yang menjadi pokok bahasan dalam makalah ini adalah:

1. Mempelajari Algoritma AS dan ACS serta proses penerapannya pada TSP untuk mendapatkan solusi yang lebih baik / mendekati optimal.
2. Merancang dan membangun perangkat lunak yang melakukan optimasi pada TSP menggunakan Algoritma AS dan ACS.
3. Melakukan analisis perbandingan terhadap hasil penerapan Algoritma AS dan Algoritma ACS pada TSP untuk membuktikan bahwa Algoritma ACS memberikan hasil yang lebih baik daripada Algoritma AS.

2. Dasar Teori

2.1 Graf

Graf didefinisikan sebagai struktur diskrit yang terdiri dari sekumpulan simpul (*vertices*) dan sisi (*edges*) yang secara matematis dituliskan sebagai *G(V,E)*[10]. Secara geometri, graf dapat digambarkan sebagai sekumpulan noktah (simpul) pada bidang dwimatra yang dihubungkan dengan sekumpulan garis (sisi)[8].

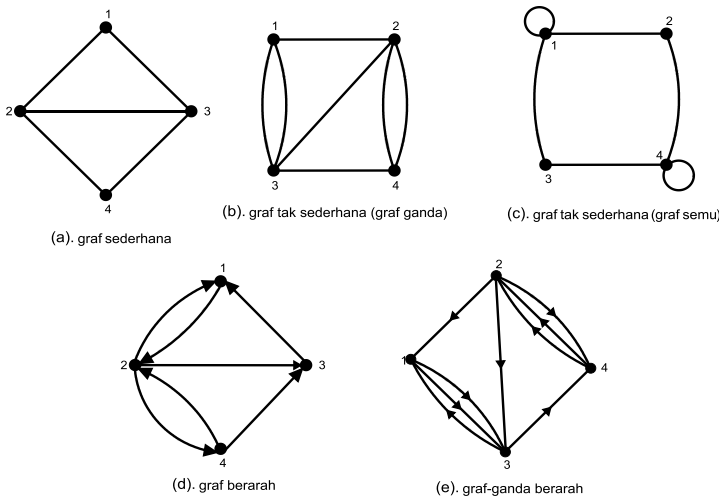
2.1.1 Jenis-jenis graf

Berdasarkan ada tidaknya sisi gelang (loop) atau sisi ganda (multiple / pararel edges) pada suatu graf, maka graf dapat dikelompokkan menjadi dua[8], yaitu:

1. Graf sederhana (*simple graph*)
   Merupakan graf yang tidak memiliki sisi gelang maupun sisi ganda, seperti pada gambar 1-a. Sisi gelang merupakan sisi yang berawal dan berakhir pada simpul yang sama, sedangkan sisi ganda merupakan sisi-sisi yang menghubungkan dua buah simpul yang sama.
2. Graf tak-sederhana (*unsimple graph*)
   Merupakan graf yang memiliki gelang atau sisi ganda. Ada 2 macam graf tak-sederhana, yaitu graf semu (*pseudograph*) (gambar 1-b) dan graf ganda (*multigraph*) dan (gambar 1-c). Graf semu lebih umum daripada graf ganda, karena sisi pada graf semu dapat terhubung ke dirinya sendiri (membentuk gelang).

Sedangkan berdasarkan orientasi arah pada sisi, maka graf dapat dibedakan atas dua jenis[8], yaitu:

1. Graf tak berarah (*undirected graf*)
   Merupakan graf yang tidak mempunyai orientasi arah (gambar 1 (a, b, c)), sehingga urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan, jadi $(v_j,v_k) = (v_k,v_j)$ adalah sisi yang sama.
2. Graf berarah (*directed graf*/*digraph*)
   Merupakan graf yang setiap sisinya diberikan orientasi arah (gambar 1-d). Pada graf berarah, sisi (v_j,v_k) dan sisi (v_k,v_j) menyatakan dua sisi berbeda. Graf berarah sering dipakai untuk menggambarkan aliran proses, peta lalu lintas (jalan searah / dua arah), dsb. Pada graf berarah, gelang diperbolehkan tetapi sisi ganda tidak.



Gambar 1. Jenis-jenis graf[8]

Definisi graf diatas dapat diperluas hingga mencakup graf ganda berarah (gambar 1-e)[8]. Pada graf ganda berarah, gelang dan sisi ganda diperbolehkan. Ringkasan perluasan definisi graf dapat dilihat pada tabel 1.

Tabel 1. Ringkasan jenis-jenis graf[10]

Jenis	Sisi	Sisi ganda dibolehkan ?	Sisi gelang dibolehkan ?
Graf sederhana	Tak-berarah	Tidak	Tidak
Graf ganda	Tak-berarah	Ya	Tidak
Graf semu	Tak-berarah	Ya	Ya
Grah berarah	Berarah	Tidak	Ya
Graf-ganda berarah	Berarah	Ya	ya

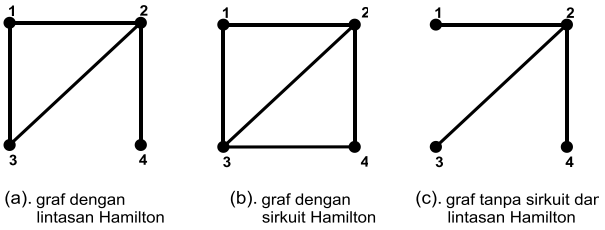
2.1.2 Lintasan dan Sirkuit Hamilton

Lintasan Hamilton didefinisikan sebagai lintasan yang melalui tiap simpul didalam graf tepat satu kali. Bila lintasan itu kembali ke simpul asal sehingga membentuk lintasan tertutup, maka lintasan tertutup tersebut dinamakan Sirkuit Hamilton. Dengan kata lain, Sirkuit Hamilton ialah sirkuit yang melalui tiap simpul didalam graf tepat satu kali, kecuali simpul awal (sekaligus simpul akhir) yang dilalui 2 kali[8].

Graf yang memiliki Sirkuit Hamilton dinamakan Graf Hamilton, sedangkan graf yang hanya memiliki Lintasan Hamilton disebut Graf semi-Hamilton (gambar 2). Setiap graf lengkap adalah Graf Hamilton[8].

2.2 Permasalahan *NP-Complete*

Permasalahan optimasi kombinatorial yang "NP-Complete" biasanya memiliki rumusan masalah yang sederhana. Untuk masukan yang kecil, permasalahan "NP-Complete" dapat diselesaikan dengan cepat. Namun jika masukannya semakin besar, maka waktu yang dibutuhkan akan meningkat luar biasa (biasanya meningkat secara eksponensial).



Gambar 2. Graf dengan Lintasan dan Sirkuit Hamilton [MUN03]

Beberapa sifat yang dimiliki oleh permasalahan "NP-*Complete*" adalah:

1. Sangat sulit diselesaikan dilihat dari sisi komputasi. Belum ada algoritma yang dapat menyelesaikannya dalam orde waktu polinomial[8].
2. Belum ada bukti bahwa permasalahan yang "NP-*Complete*" tidak mungkin diselesaikan dalam orde waktu polinomial[4].
3. Jika ada sembarang masalah "NP-*Complete*" yang dapat diselesaikan dalam orde waktu polinomial, maka semua masalah yang termasuk kedalam "NP-*Complete*" dapat diselesaikan pula dalam orde waktu polinomial[14].
4. Sebaliknya, jika sembarang permasalahan "NP-*Complete*" dapat dibuktikan tidak dapat diselesaikan dalam orde waktu polinomial, maka semua masalah yang termasuk kedalam "NP-*Complete*" juga terbukti tidak dapat diselesaikan dalam orde waktu polinomial[14].

Jika solusi optimal pada permasalahan "NP-*Complete*" tidak bisa diperoleh secara efisien, maka pada prakteknya keoptimalan seringkali dikorbankan demi efisiensi[7]. Dengan kata lain, daripada mencari solusi optimal lebih baik mencari solusi yang mendekati optimal asalkan selesai dalam orde waktu polinomial. Hal ini dilakukan dengan menggunakan metode heuristik.

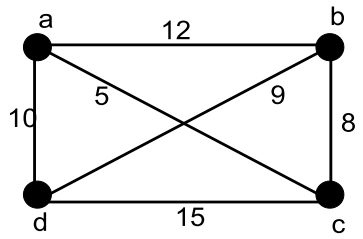
Metode heuristik bertujuan untuk mencari solusi yang amat baik (mendekati optimal) dari suatu permasalahan optimasi kombinatorial[11]. Metode ini tidak menjamin keoptimalan solusi yang diberikan[7]. Namun karena biaya komputasi yang lebih rendah, sebagian besar permasalahan "NP-*Complete*" diselesaikan menggunakan metode heuristik.

2.3 *Traveling Salesman Problem* (TSP)

TSP merupakan permasalahan optimasi kombinatorial yang sangat terkenal dalam teori graf. TSP dikategorikan sebagai permasalahan yang sulit ditinjau dari sudut komputasinya[8]. TSP juga termasuk permasalahan "NP-*Complete*" yang klasik karena telah dipelajari selama beberapa dekade. TSP dapat dipandang sebagai masalah mencari rute terpendek yang harus ditempuh oleh seseorang yang berangkat dari kota asal untuk mengunjungi setiap kota tepat satu kali lalu kembali lagi ke kota asal keberangkatannya.

Dalam teori graf, TSP dinyatakan sebagai graf G_{TSP} = *(N,A)* dimana *N* menyatakan himpunan simpul kota, sedangkan *A* himpunan sisi yang menghubungkan simpul-simpul kota pada graf. Bobot pada sisi menyatakan jarak antar dua buah kota. Dengan kata lain, TSP tidak lain adalah permasalahan untuk menemukan Sirkuit Hamilton yang memiliki bobot minimum pada sebuah graf terhubung[8].

Untuk TSP pada sembarang graf lengkap, sangat mudah menghitung jumlah Sirkuit Hamilton yang ada. Dari simpul pertama ada $n-1$ pilihan, dari simpul kedua ada $n-2$ pilihan, dari simpul ketiga ada $n-3$ pilihan, dst. Sehingga untuk n buah kota, ada $(n-1)!$ pilihan yang mungkin. Namun karena Sirkuit Hamilton terhitung 2 kali maka pilihan yang ada harus dibagi dua menjadi $\frac{(n-1)!}{2}$ buah Sirkuit Hamilton. Sebagai contoh dapat dilihat pada gambar 3.



Gambar 3. Contoh graf lengkap dengan 4 simpul[8]

Untuk TSP pada graf lengkap dengan $n=4$ seperti pada gambar 3, akan terdapat $\frac{(4-1)!}{2}=3$ Sirkuit Hamilton, yaitu:

- $S_1 = (a,b,c,d,a)$ atau (a,d,c,b,a) dengan panjang rute = 12 + 8 + 15 + 10 = 45
- $S_2 = (a,c,d,b,a)$ atau (a,b,d,c,a) dengan panjang rute = 5 + 15 + 9 + 12 = 41
- $S_3 = (a,c,b,d,a)$ atau (a,d,b,c,a) dengan panjang rute = 5 + 8 + 9 + 10 = 32

Jadi, Sirkuit Hamilton terpendek adalah $S_3 = (a,c,b,d,a)$ atau (a,d,b,c,a) dengan panjang rute sebesar 32.

Secara teoritis TSP dapat dipecahkan dengan melakukan *exhaustive search* pada $\frac{(n-1)!}{2}$ Sirkuit Hamilton, menghitung panjang rute masing-masing lalu menentukan rute terpendek. Namun untuk *n* yang besar, maka jumlah Sirkuit Hamilton yang harus diperiksa satu persatu akan sangat banyak. Sebagai contoh, untuk *n* = 20 akan ada sekitar 6×10^{16} penyelesaian. Akibatnya, waktu komputasi yang diperlukan juga akan meningkat secara eksponensial (kompleksitasnya *O(n!)*). Dengan kata lain, TSP termasuk kategori permasalahan yang NP-Complete[9].

2.4 Perilaku koloni semut pada saat mencari makanan

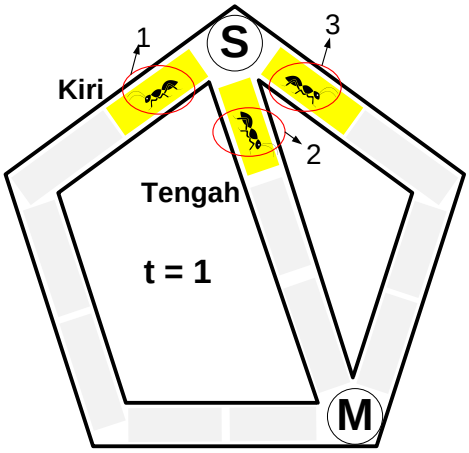
Ketika mencari makan, pengetahuan individu semut terhadap lingkungannya amat terbatas. Secara individu, semut akan kesulitan untuk menemukan rute menuju sumber makanan jika pencarian dilakukan pada area yang luas. Untuk mengatasi keterbatasan tersebut, semut akan bekerja secara kolektif didalam koloni.

Karena semut memiliki penglihatan yang hampir buta, komunikasi tidak dapat dilakukan secara visual. Teknik yang digunakan semut adalah dengan berkomunikasi secara tidak langsung (*stigmergy*) menggunakan zat kimia yang disebut dengan *pheromone*. Komunikasi dapat dilakukan karena semut cenderung memilih rute dengan konsentrasi *pheromone* yang tinggi[6].

Ketika berjalan, setiap semut akan meninggalkan *pheromone* pada suatu rute. Semut berikutnya dapat mendeteksi perbedaan konsentrasi *pheromone* dan cenderung memilih rute dengan konsentrasi *pheromone* tertinggi sambil menambahkan *pheromone*-nya sendiri. Karena semut lain menunjukkan perilaku yang sama, akibatnya rute yang sering dipilih oleh setiap semut akan memiliki konsentrasi *pheromone* paling tinggi. Untuk lebih jelasnya dapat melihat contoh pada gambar 4 dan gambar 5.

Gambar 4 menunjukkan contoh kondisi awal koloni semut pada saat mencari makanan. Koloni pada gambar 4 terdiri dari tiga ekor semut. Pada mulanya (*t* = 1) setiap semut akan memilih rute yang berbeda. "S" Menyatakan sarang sedangkan "M" menyatakan lokasi makanan.

Panjang rute kiri enam langkah, panjang rute tengah tiga langkah, dan panjang rute kanan empat langkah. Warna pada rute menunjukkan tingkat konsentrasi *pheromone* saat ini. *Pheromone* akan mengalami penguapan setelah enam langkah. Pada dasarnya semut memilih rute secara acak. Namun secara probabilitas, pilihannya cenderung jatuh pada rute dengan *pheromone* yang tinggi.



Gambar 4. Contoh kondisi koloni semut di awal pencarian

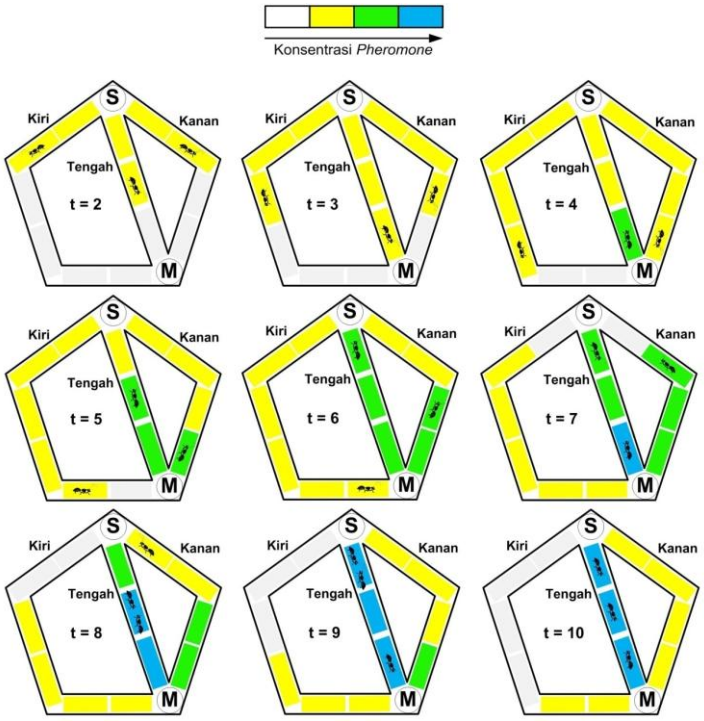
Langkah-langkah koloni melakukan pencarian dapat dijelaskan sebagai berikut (gambar 5):

1. Untuk *t* = 2. Semut 1,2,3 bergerak menuju lokasi makanan sesuai rute masing-masing.
2. Untuk *t* = 3. Karena rute tengah lebih pendek, semut 2 tiba terlebih dahulu di lokasi makanan.
3. Untuk *t* = 4. Karena konsentrasi *pheromone* pada rute tengah lebih tinggi, maka probabilitas rute tersebut untuk dipilih acak menjadi lebih besar. Akibatnya semut 2 memilih rute tengah untuk kembali ke sarang sambil menambahkan *pheromone*-nya. Pada saat yang sama semut 3 tiba di lokasi makanan.
4. Untuk *t* = 5. Walaupun probabilitas rute tengah lebih tinggi, semut 3 memilih rute kanan untuk kembali ke sarang. Hal ini terjadi karena pada dasarnya semut memilih secara acak.
5. Untuk *t* = 6. Semut 1 tiba di lokasi makanan. Semut 2 tiba di sarang.
6. Untuk *t* = 7. Secara acak semut 1 memilih rute tengah untuk kembali ke sarang (saat ini probabilitas rute tengah dan kanan adalah sama). Saat ini telah terjadi penguapan *pheromone* di seluruh rute. *Pheromone* di rute tengah tetap lebih tinggi karena semut 2 menambahkan *pheromone* kembali sesaat setelah penguapan.
7. Untuk *t* = 4. Karena konsentrasi *pheromone* pada rute tengah lebih tinggi, maka probabilitas rute tersebut untuk dipilih acak menjadi lebih besar. Akibatnya semut 2 memilih rute tengah untuk kembali ke sarang sambil menambahkan *pheromone*-nya. Pada saat yang sama semut 3 tiba di lokasi makanan.
8. Untuk *t* = 5. Walaupun probabilitas rute tengah lebih tinggi, semut 3 memilih rute kanan untuk kembali ke sarang. Hal ini terjadi karena pada dasarnya semut memilih secara acak.
9. Untuk *t* = 6. Semut 1 tiba di lokasi makanan. Semut 2 tiba di sarang.
10. Untuk *t* = 7. Secara acak semut 1 memilih rute tengah untuk kembali ke sarang (saat ini probabilitas rute tengah dan kanan adalah sama). Saat ini telah terjadi penguapan *pheromone* di seluruh rute. *Pheromone* di rute tengah tetap lebih tinggi karena semut 2 menambahkan *pheromone* kembali sesaat setelah penguapan.
11. Untuk *t* = 8. Semut 3 tiba di sarang.
12. Untuk *t* = 9. Semut 3 akan kembali ke lokasi makanan. Karena probabilitas rute tengah lebih besar, maka pilihan acak semut 3 jatuh pada rute tengah.
13. Pada *t* = 10, seluruh semut telah memilih rute tengah. Semut cenderung selalu memilih rute tengah karena probabilitasnya sangat tinggi. Sedangkan probabilitas rute kanan dan kiri terus menurun karena penguapan *pheromone*.

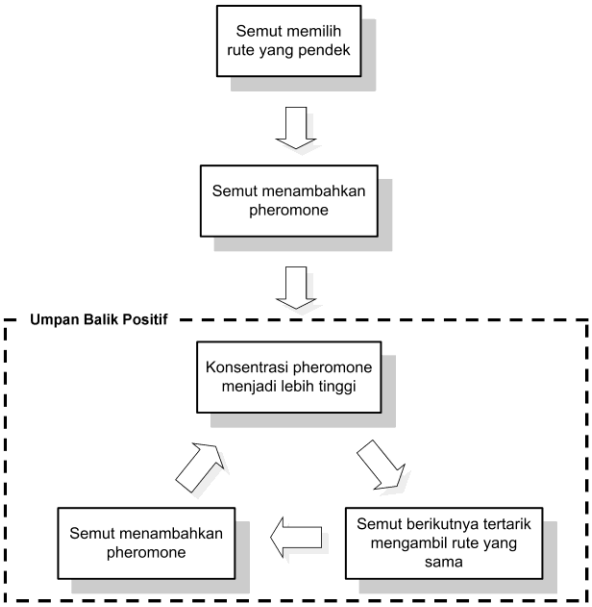
Dari keterangan diatas dapat disimpulkan bahwa:

1. Pada mulanya konsentrasi *pheromone* mengalami kenaikan karena rute yang dilewati adalah rute yang pendek (proses bolak-balik lebih cepat dilakukan).
2. Konsentrasi *pheromone* yang tinggi akan menarik perhatian semut lain.
3. Akibatnya, setelah beberapa saat, konsentrasi *pheromone* menjadi semakin tinggi karena semut yang melewati rute tersebut lebih banyak dibandingkan rute yang lain.

Perilaku ini lebih dikenal sebagai perilaku umpan-balik positif (*positive feedback*) dengan menggunakan *pheromone*. Semakin tinggi akumulasi *pheromone* pada suatu rute, maka rute tersebut semakin menarik untuk diikuti. Dengan kata lain, probabilitas suatu rute untuk dipilih berbanding lurus dengan konsentrasi *pheromone* pada rute tersebut[3]. Perilaku inilah yang mendasari proses optimasi pada algoritma semut. Ilustrasinya dapat dilihat pada gambar 6.



Gambar 5. Contoh proses pencarian makanan oleh koloni semut



Gambar 6. Siklus umpan balik positif pada koloni semut

2.4 Algoritma Semut

Algoritma Semut dikemukakan oleh Marco Dorigo pada tahun 1991 yang terinspirasi langsung dari observasi terhadap koloni semut[6]. Pada algoritma ini, koloni "semut buatan" akan bekerjasama untuk menemukan solusi permasalahan optimasi diskrit sambil berkomunikasi secara tidak langsung dengan menggunakan *pheromone*[6]. Solusi akhir yang diberikan merupakan hasil kerja kolektif didalam koloni[7].

2.4.1 Karakteristik Algoritma Semut

Algoritma semut tergolong algoritma konstruktif karena setiap "semut buatan" (untuk selanjutnya akan disebut dengan "semut") akan membangun solusi secara bertahap[3]. Sesuai dengan analoginya, koloni semut akan menelusuri graf eksplorasi GS = (C,L) dimana:

1. C merupakan himpunan simpul kandidat solusi.
2. L merupakan himpunan sisi yang menyatakan relasi diantara pasangan simpul C. Pasangan (c_i,c_j) dengan $c_i,c_j \in C$ dapat menjadi indikasi bahwa pasangan (c_i,c_j) merupakan solusi yang layak (*feasible*).
3. Bobot pada sisi (c_i,c_j) menyatakan biaya (*cost*) yang diperlukan untuk menjadikan pasangan (c_i,c_j) sebagai solusi yang layak.

Solusi optimal dari suatu permasalahan optimasi kombinatorial diasosiasikan dengan rute terpendek dari graf G_S.

Setiap semut k pada algoritma semut akan memiliki karakteristik sebagai berikut[7]:

1. Semut akan mengeksplorasi graf G_S berusaha mencari solusi optimal.
2. Semut memiliki memori $\mathcal{M}^k$ yang menyimpan informasi mengenai rute yang ditempuh sejauh ini. Informasi ini dapat digunakan untuk membangun serta mengevaluasi solusi.
3. Sebelum memulai pencarian, setiap semut memiliki kondisi awal tertentu. Umumnya kondisi awal setiap semut diekspresikan oleh himpunan solusi kosong.
4. Setiap semut akan mengunjungi setiap simpul sesuai batasan masalah sambil membangun solusi secara bertahap. Pencarian berhenti ketika tidak ada lagi simpul layak yang dapat ditambahkan atau telah mencapai kondisi berhenti tertentu.
5. Semut bergerak secara probabilitas menggunakan aturan transisi status tertentu (*State transition rule*). Aturan ini merupakan fungsi dari konsentrasi *pheromone*, informasi heuristik, dan memori semut bersangkutan.
6. Di akhir tiap iterasi dilakukan proses pembaharuan dalam bentuk penambahan *pheromone* oleh semut serta penguapan *pheromone* pada sisi graf G_S.

Secara informal algoritma semut terdiri dari tiga bagian utama: Inisialisasi, pembangunan solusi (*solution construction*), dan pembaharuan *pheromone* (*pheromone update*). Garis besar algoritma semut untuk permasalahan optimasi kombinatorial dapat dilihat pada gambar 7.

```
Procedure AlgoritmaSemut
   Inisialisasi
   While (kondisi berhenti belum dipenuhi) do
      BangunSeluruhSolusi
      PerbaharuiPheromone
   End
End
```

Gambar 7. Garis besar algoritma semut[7]

Dari gambar 7 dapat dilihat bahwa proses pembangunan solusi dan pembaharuan *pheromone* akan terus diulang hingga kondisi berhenti yang telah ditentukan terpenuhi.

2.4.2 Inisialisasi

Inisialisasi dilakukan agar tiap semut siap melakukan eksplorasi pada graf. Secara umum, langkah-langkah yang dilakukan pada saat inisialisasi adalah:

1. Memilih jumlah semut yang akan digunakan.
2. Meletakkan setiap semut pada simpulnya masing-masing secara acak.
3. Mengosongkan / menginsialisasi memori tiap semut.
4. Menginisialisasi *pheromone* dan informasi heuristik.

2.4.3 Pembangunan Solusi

Koloni semut akan membangun solusi secara bertahap dengan mengunjungi seluruh simpul solusi sesuai dengan batasan masalah. Simpul yang akan dikunjungi dipilih berdasarkan aturan transisi status tertentu dengan mempertimbangkan faktor *pheromone*, informasi heuristik, serta memori semut.

Pheromone dapat diumpamakan sebagai bentuk lain dari pengalaman koloni selama proses pencarian. Pada mulanya semut belum memiliki pengalaman karena seluruh sisi graf memiliki *pheromone* yang sama. Beberapa saat kemudian, pengalaman semut akan bertambah seiring perbedaan *pheromone* yang timbul akibat aktivitas koloni selama pencarian.

Informasi heuristik adalah sebuah nilai diluar *pheromone* yang dapat memberikan gambaran bagi semut mengenai graf yang sedang dieksplorasi. Secara umum informasi ini sangat bermanfaat diawal pencarian ketika semut masih belum berpengalaman (belum ada perbedaan *pheromone*). Karena informasi heuristik telah diinisialisasi sejak awal, maka informasi ini dapat membimbing semut agar tidak sepenuhnya "buta" ketika mengeksplorasi area baru. Seiring bertambahnya pengalaman semut terhadap area eksplorasi, maka pengaruh informasi ini akan berkurang[3].

Setelah setiap semut selesai membangun solusi, maka langkah selanjutnya adalah melakukan evaluasi menggunakan aturan pembaharuan *pheromone* (*pheromone update rule*) untuk menentukan besar konsentrasi *pheromone* yang akan ditambahkan (atau dikurangi)[7].

2.4.4 Pembaharuan *Pheromone*

Proses pembaharuan *pheromone* merupakan proses modifikasi terhadap *pheromone* yang ada pada graf eksplorasi. *Pheromone* dapat meningkat karena penambahan oleh semut atau berkurang karena penguapan. Secara sederhana, umpan balik positif melalui proses penambahan *pheromone* akan meningkatkan peluang suatu sisi untuk dipilih kembali di masa depan. Sebaliknya, penguapan *pheromone* merupakan bentuk umpan balik negatif yang bermanfaat agar semut dapat "melupakan" pilihan sebelumnya dan mencoba mencari rute alternatif sehingga tidak terjebak kedalam optimal lokal (*local optima*)[7].

3. Analisis

3.1 Algoritma Semut pada TSP

Algoritma semut dikembangkan agar dapat menyelesaikan beragam permasalahan optimasi diskrit[6]. Pada tugas akhir ini, algoritma semut akan diterapkan pada permasalahan TSP. Permasalahan ini dipilih karena:

1. Perilaku koloni semut pada saat mencari makan serupa dengan rumusan masalah TSP[1].
2. TSP termasuk permasalahan optimasi diskrit yang NP-*Complete*[1].
3. TSP merupakan permasalahan optimasi diskrit yang paling banyak dipelajari sehingga sering disebut sebagai *bechmark problem*[12].
4. Kesederhaan rumusan TSP[7].
5. TSP dapat dikembangkan kedalam beragam variasi serta aplikasi[12].

Sedangkan TSP yang digunakan pada pengujian nanti merupakan graf yang terhubung lengkap, tidak berarah dengan jumlah simpul tidak lebih dari 550 buah.

3.1.1 Graf Eksplorasi

Graf yang digunakan semut untuk mengeksplorasi identik dengan graf permasalahan pada TSP. Komponen graf eksplorasi $G_S = (C,L)$ berkorespondensi dengan graf TSP $G_{TSP} = (N,A)$. Komponen himpunan simpul C pada G_S berkorespondensi dengan himpunan kota N pada G_{TSP}. Komponen himpunan sisi L pada G_S berkorespondensi dengan himpunan sisi A pada G_{TSP}. Bobot pada himpunan sisi L berkorespondensi dengan d_{ij} yang menyatakan jarak antara kota i ke kota j.

3.1.1 *Pheromone* dan Informasi Heuristik

Pheromone yang diletakkan pada TSP menyatakan ketertarikan semut untuk mengunjungi kota j dari kota i. *Pheromone* yang dinyatakan dengan τ_{ij} akan diletakkan disepanjang sisi jalan antara kota i dan kota j.

Nilai awal *pheromone* (τ_0) diatur sedemikian supaya sedikit lebih besar dari yang dapat diletakkan seekor semut dalam sekali iterasi[7]. Hal ini penting karena jika nilai τ_0 terlalu kecil, maka pencarian akan berat sebelah karena semut cenderung selalu memilih rute yang telah ditemukan sebelumnya. Sedangkan jika nilai τ_0 terlalu besar, maka *pheromone* yang ditambahkan tidak akan memiliki arti selama proses pencarian.

Informasi heuristik dinyatakan dengan η_{ij}. Untuk TSP, informasi heuristik akan menyatakan visibilitas semut terhadap kota-kota disekitarnya. Nilai visibilitas η_{ij}, dihitung secara heuristik berdasarkan persamaan 1.

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

Persamaan 1 menyatakan bahwa nilai visibiltas η_{ij} merupakan nilai yang konstan karena menyatakan invers jarak kota i ke kota j (d_{ij}). Informasi mengenai visibilitas akan sangat berguna di awal pencarian ketika semut belum memiliki pengalaman (karena pada TSP belum ada perbedaan konsentrasi *pheromone*).

3.1.2 Pembangunan Solusi

Satu-satunya batasan masalah pada TSP adalah setiap kota harus dikunjungi satu kali (kecuali kota awal). Batasan ini diimplementasikan dengan mengharuskan setiap semut untuk memastikan bahwa kota yang akan dikunjungi belum pernah dikunjungi sebelumnya. Hal ini dilakukan lewat bantuan memori $\mathcal{M}^k$ yang dimiliki semut.

Setiap semut didalam koloni akan membangun solusinya masing-masing. Setiap semut memiliki sembarang kota awal yang berbeda satu sama lain. Pada setiap langkah, semut secara iteratif akan menambahkan kota yang dikunjungi kedalam daftar turnya.

Proses pembangunan solusi pada suatu iterasi dianggap selesai jika seluruh semut telah menyelesaikan turnya, yaitu seluruh kota telah dikunjungi oleh setiap semut dan kembali ke kota asal keberangkatan.

3.2 Algoritma Ant System (AS)

Algoritma *Ant System* (AS) merupakan usaha pertama untuk menerapkan perilaku koloni semut pada TSP[6]. Ada tiga versi Algoritma AS yang dikembangkan: *ant-density*, *ant-quantity*, dan *ant-cycle*. Perbedaan ketiganya terletak pada cara menghitung *pheromone* yang hendak ditambahkan. Pada versi a*nt-density* dan *ant-quantity*, *pheromone* yang akan ditambahkan dihitung di setiap langkah perpindahan semut menuju kota yang baru. Sedangkan pada versi *ant-cycle*, *pheromone* yang hendak ditambahkan dihitung setelah semut menyelesaikan turnya masing-masing[3]. Karena performa *ant-cycle* lebih baik daripada *ant-density* maupun *ant quantity*, maka ketika menyinggung Algoritma AS yang dimaksud merupakan versi *ant-cycle*[7]. Kompleksitas Algoritma AS adalah $O(n^3)$. *Pseudo-code* Algoritma AS dapat dilihat pada gambar 8.

```
inisialisasi
while (not kondisi berhenti) do
      pembangunan solusi
      pembaharuan pheromone global
end while
```

Gambar 8. *Pseudo Code* Algoritma AS

3.2.1 Inisialisasi

Pada mulanya *pheromone* (τ_0) diinisialisasi pada setiap sisi pada TSP menggunakan pendekatan heuristik. Aturan inisialisasi *pheromone* pada Algoritma AS dapat dilihat pada persamaan 2.

$$\tau_{ij} = \tau_0 = \frac{m}{C_{greedy}} \quad (2)$$

Dimana m menyatakan jumlah semut dan C menyatakan panjang tur yang dihasilkan melalui algoritma *greedy* biasa[7]. Setelah itu dilakukan inisialisasi informasi heuristik berupa visibilitas (η_{ij}) yang dihitung berdasarkan persamaan 1.

3.2.1 Pembangunan Solusi

Jumlah semut paling banyak yang digunakan pada Algoritma AS sebaiknya sama dengan jumlah kota pada TSP[3]. Setelah setiap semut diletakkan secara acak pada setiap kota, semut *k* yang berada di kota *i* akan memilih kota *j* menurut aturan transisi status *random proportional*. Pada aturan ini, semut akan memilih kota secara acak namun cenderung memilih kota dengan probabilitas paling besar. Aturan ini dinyatakan pada persamaan 3.

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{j \in N_i^k} [\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}, & \text{jika } j \in N_i^k \\ 0, & \text{sebaliknya} \end{cases} \quad (3)$$

Dimana p_{ij}^{k} menyatakan peluang semut *k* untuk mengunjungi kota *j* dari kota *i*. τ_{ij} menyatakan konsentrasi *pheromone* antara kota *i* dan kota *j*, η_{ij} menyatakan visibilitas semut terhadap kota-kota disekitar *i*. N_k^i merupakan kumpulan tetangga kota *i* yang mungkin dikunjungi oleh semut *k*. Sedangkan α dan β merupakan konstanta yang menyatakan kuat pengaruh *pheromone* (τ_{ij}) dan visibilitas (η_{ij}) terhadap pilihan semut. Jika konstanta $\alpha = 0$, kecendrungan kota yang dipilih adalah kota yang terdekat atau serupa dengan proses pencarian *greedy*. Jika $\beta = 0$, maka pencarian hanya mengandalkan *pheromone*, yang biasanya akan berujung pada hasil pencarian yang buruk [DOR04].

Nantinya semua kota yang dikunjungi oleh semut akan dicatat pada memori $\mathcal{M}^k$. Selain menjamin agar semut tidak mengunjungi kota yang sama, memori ini juga digunakan untuk menghitung jarak yang sudah ditempuh.

Pada Algoritma AS, proses pembangunan solusi dapat diimplementasikan secara paralel atau sekuensial. Pada pencarian paralel, semua semut bergerak dari satu kota ke kota yang lain (membangun solusi) secara konkuren. Sedangkan pada pencarian sekuensial, sebuah solusi harus dibangun terlebih dahulu, sebelum solusi berikutnya dapat dibangun oleh semut yang lain[7].

3.2.2 Pembaharuan *Pheromone* Global

Setelah semua semut menyelesaikan turnya, *pheromone* akan diperbaharui secara menyeluruh. Hal ini dilakukan dengan terlebih dahulu menguapkan *pheromone* pada seluruh sisi TSP menggunakan persamaan 4.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i, j) \in TSP \quad (4)$$

Dimana ρ ($0 < \rho \leq 1$) adalah konstanta sedemikian sehingga nilai (1 - ρ) akan menggambarkan laju penguapan *pheromone*. Setelah penguapan, setiap sisi TSP akan menerima sejumlah *pheromone* berdasarkan persamaan 5.

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i, j) \in TSP \quad (5)$$

Dimana τ_{ij} menyatakan kadar *pheromone* yang baru dan $\Delta\tau_{ij}^{k}$ menyatakan jumlah *pheromone* yang akan ditambahkan oleh semut *k*. Nilai $\Delta\tau_{ij}^{k}$ dihitung berdasarkan persamaan 6

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^{k}, & \text{jika pasangan } (i, j) \in T^{k} \\ 0, & \text{sebaliknya} \end{cases} \quad (6)$$

Dimana C^k menyatakan panjang tur T^k. Sedangkan tur T^k berisi sisi TSP yang diambil oleh semut *k*.

Berdasarkan persamaan 6 dapat disimpulkan bahwa semakin pendek rute yang ditempuh, semakin besar jumlah *pheromone* yang dapat ditambahkan. Semakin sering sebuah rute dilewati juga akan menyebabkan rute tersebut menerima *pheromone* yang lebih banyak. Jika kadar *pheromone* semakin tinggi, maka rute tersebut kemungkinan besar akan dipilih kembali pada iterasi berikutnya.

Melalui pengujian, Algoritma AS memberikan hasil yang menjanjikan pada TSP yang kecil ($n \leq 30$). Namun, performa Algoritma AS cenderung menurun seiring dengan bertambahnya ukuran masukan. Oleh karena itu Algoritma AS dimodifikasi menjadi Algoritma *Ant Colony System* (ACS)[5].

3.3 Algoritma Ant Colony System (ACS)

Algoritma *Ant Colony System* (ACS) merupakan pengembangan yang dilakukan terhadap AS[5]. Secara garis besar, Algoritma ACS sama dengan Algoritma AS. Namun Algoritma ACS memiliki tiga perbedaan utama, yaitu:

1. Perubahan aturan transisi status yang mencoba untuk menyeimbangkan kecendrungan mengeksploitasi rute yang lama dengan keinginan untuk mengeksplorasi rute yang baru[5].
2. Pembaharuan *pheromone* global (penguapan serta penambahan) hanya dilakukan pada rute terbaik saat ini[5].

3. Penambahan aturan pembaharuan *pheromone* lokal setiap kali seekor semut berpindah ke kota berikutnya[5].

Algoritma ACS memiliki kompleksitas algoritma $O(n^3)$. Sedangkan *pseudo-code* Algoritma ACS dapat dilihat pada gambar 9.

```
inisialisasi
while (not kondisi berhenti) do
    while (pembangunan solusi) do
        kunjungi kota berikutnya
        pembaharuan pheromone lokal
    end while
    pembaharuan pheromone global
end while
```

Gambar 9. Pseudo Code Algoritma ACS

3.3.1 Inisialisasi

Inisialiasi awal *pheromone* (τ_0) pada Algoritma ACS ditentukan secara heuristik melalui persamaan 7.

$$\tau_{ij} = \tau_0 = \frac{1}{nC_{greedy}} \quad (7)$$

Dengan n adalah jumlah kota pada TSP sedangkan C_{greedy} menyatakan rute terbaik berdasarkan algoritma greedy[7]. Sedangkan untuk inisialisasi visibilitas (η_{ij}), aturan yang digunakan masih sama seperti persamaan 1.

3.3.2 Pembangunan Solusi

Aturan transisi status yang digunakan pada Algoritma ACS memiliki perbedaan dengan Algoritma AS. Aturan ini dinamakan aturan *pseudorandom proportional* yang dinyatakan melalui persamaan 8.

$$j = \begin{cases} \arg\max_{l \in N_i^k} \{\tau_{il}[\eta_{il}]^\beta\}, & \text{jika } q \le q_0 \text{ (eksploitasi)} \\ J, & \text{jika } q > q_0 \text{ (eksplorasi)} \end{cases} \quad (8)$$

Dimana q merupakan bilangan acak dengan nilai antara 0 sampai 1, q_0 merupakan konstanta pembanding yang telah ditentukan dari awal, β merupakan konstanta yang menyatakan kuat pengaruh visibilitas (η_{ij}), N_i^k merupakan kumpulan kota disekitar i yang dapat dipilih oleh semut k. Sedangkan J merupakan kota yang dipilih berdasarkan persamaan III-9.

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}][\eta_{ij}]^\beta}{\sum_{j \in N_i^k}[\tau_{ij}][\eta_{ij}]^\beta}, & \text{jika } j \in N_i^k \\ 0, & \text{sebaliknya} \end{cases} \quad (9)$$

Dapat dilihat bahwa persamaan 9 sama dengan persamaan 3 pada Algoritma AS jika menggunakan konstanta $\alpha = 1$.

Aturan *pseudorandom proportional* bekerja sebagai berikut :

1. Mula-mula dibangkitkan bilangan acak q.
2. Jika $q \le q_0$, maka semut akan memilih kota dengan nilai terbaik berdasarkan informasi *pheromone* (τ_{ij}) dan visibilitas (η_{ij}) dari seluruh kota yang mungkin dikunjungi. Aturan yang digunakan adalah persamaan 8 (eksploitasi).
3. Jika $q > q_0$, maka semut akan memilih kota berdasarkan persamaan 9 (eksplorasi).

Dari keterangan diatas dapat disimpulkan, pada probabilitas q_0, semut akan memilih sisi jalan dengan nilai *pheromone* dan visibilitas yang baik (mengeksploitasi informasi yang sudah dianggap baik). Sedangkan pada probabilitas (1 - q_0) semut akan mengeksplorasi rute yang baru. Sehingga dengan mengatur parameter q_0 diharapkan dapat menyeimbangkan kecendrungan semut untuk mengambil rute yang lama (dianggap baik) dengan keinginan untuk mengeksplorasi rute alternatif[7].

3.3.3 Pembaharuan *Pheromone* Global

Proses pembaharuan *pheromone* global pada Algoritma ACS hanya dilakukan pada rute terbaik saat ini. Sama seperti Algoritma AS, proses ini dilakukan diakhir tiap iterasi setelah seluruh semut menyelesaikan turnya. Aturan pembaharuan *pheromone* yang pada Algoritma ACS diberikan pada persamaan 10.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{best}, \quad \forall(i,j) \in T^{best} \quad (10)$$

Dimana $\Delta\tau_{ij}^{best}$ menyatakan kadar *pheromone* yang akan ditambahkan. Nilai $\Delta\tau_{ij}^{best}$ dihitung berdasarkan persamaan 11.

$$\Delta\tau_{ij}^{best} = 1/C^{best} \quad (11)$$

Persamaan 11 menyatakan bahwa penguapan serta penambahan *pheromone* hanya dilakukan pada rute terbaik (T^{best}), tidak pada seluruh sisi TSP[5]. Sama seperti Algoritma AS, nilai $(1 - \rho)$ juga menyatakan penguapan *pheromone*.

3.3.4 Pembaharuan *Pheromone* Lokal

Pembaharuan *pheromone* lokal, merupakan aturan baru pada Algoritma ACS. Aturan ini menyatakan bahwa seekor semut yang bergerak dari kota *i* ke kota *j* dapat segera melakukan pembaharuan *pheromone* pada sisi (*i*,*j*) sebelum menyelesaikan turnya[7]. Pembaharuan *pheromone* lokal dinyatakan pada persamaan 12.

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0 \qquad (12)$$

Dimana ξ $(0 \le \xi \le 1)$ merupakan konstanta yang melalui eskperimen disarankan nilainya adalah 0.1[5].

Efek dari pembaharuan *pheromone* lokal adalah setiap kali seekor semut melewati sebuah sisi *(i,j)*, konsentrasi *pheromone* pada sisi tersebut akan berkurang. Akibatnya pada iterasi berikutnya, sisi *(i,j)* menjadi kurang menarik untuk dikunjungi. Dengan kata lain, aturan ini dapat membuat daya tarik sebuah sisi berubah secara dinamis. Ini merupakan hal yang baik, karena akan meningkatkan keinginan semut untuk mengeksplorasi solusi alternatif [DOR04]. Tanpa adanya aturan ini, semut akan cenderung mengekplorasi rute yang sebelumnya telah ditemukan (karena sudah dianggap baik)[5].

Sebelumnya telah disebutkan bahwa pencarian pada Algoritma AS dapat dilakukan dalam dua cara: pararel maupun sekuensial. Berbeda dengan Algoritma AS, karena pada Algoritma ACS terdapat aturan pembaharuan *pheromone* lokal, maka pencarian pada Algoritma ACS sebaiknya dilakukan secara pararel[7].

4. Perangkat Lunak

4.1 Spesifikasi Perangkat Lunak

Spesifikasi kebutuhan perangkat lunak adalah sebagai berikut:

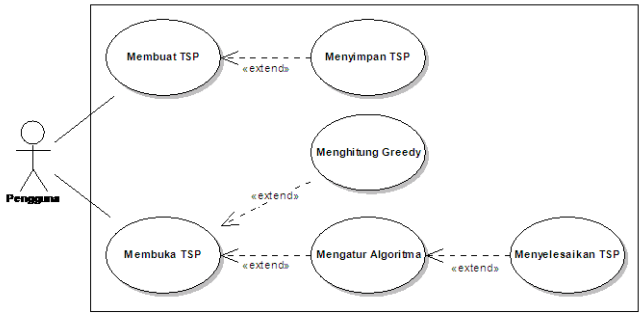
1. Perangkat lunak mampu membuka arsip yang berisi deskripsi TSP.
2. Perangkat lunak mampu mencari solusi pada TSP dengan menggunakan Algoritma AS, ACS, atau *Greedy*.
3. Perangkat lunak menyediakan fasilitas untuk mengatur algoritma serta konstanta yang hendak dipergunakan.
4. Perangkat lunak menyediakan fasilitas untuk membuat TSP.
5. Perangkat lunak menyediakan fasilitas untuk menyimpan TSP yang baru dibuat.
6. Perangkat lunak mampu mencari solusi dari TSP menggunakan algoritma AS atau ACS serta menampilkan prosesnya secara visual.

4.2 Diagram *Use Case*

Berdasarkan spesifikasi kebutuhan perangkat lunak yang dibuat, maka *use case* yang teridentifikasi adalah sebagai berikut:

1. *Use Case* "Membuka TSP", merupakan fungsionalitas yang digunakan oleh pengguna untuk membuka arsip TSP yang akan diselesaikan. Arsip TSP berisi nama permasalahan, komentar, jumlah kota, jenis permasalahan, posisi/koordinat kota, solusi optimal (jika ada), dan tur optimal (jika ada).
2. *Use Case* Membuat TSP, merupakan fungsionalitas yang digunakan oleh pengguna untuk membuat TSP sendiri.
3. *Use Case* Menyimpan TSP, merupakan fungsionalitas yang dapat digunakan untuk menyimpan TSP yang baru saja dibuat / dibangkitkan kedalam arsip eksternal.
4. *Use Case* Menyelesaikan TSP, merupakan fungsionalitas untuk menyelesaikan TSP menggunakan Algoritma AS atau ACS.
5. *Use Case* Mengatur Algoritma, merupakan fungsionalitas untuk memilih algoritma, konstanta, serta kondisi berhenti yang diinginkan.
6. *Use Case* Menghitung *Greedy*, merupakan fungsionalitas untuk mencari solusi baru bagi TSP berdasarkan algoritma *greedy*.

Diagram Use Case dari perangkat lunak dapat dilihat pada gambar 10.



Gambar 10. Diagram *Use Case* dari Perangkat Lunak

4.3 Implementasi Kelas

Kelas-kelas yang diperlukan diimplementasikan menggunakan bahasa pemrograman Java 6 dengan kakas pengembangan Netbeans IDE 6.5. Kelas yang diimplementasikan dapat dilihat pada tabel 2.

Tabel 2. Implementasi kelas

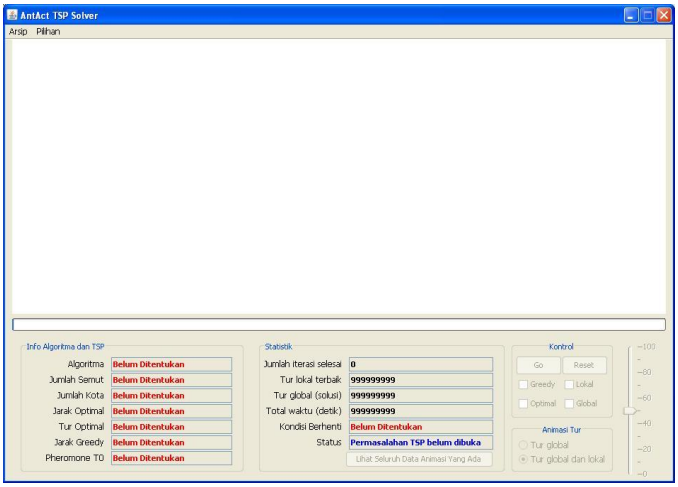
No	Nama Kelas	Nama File
1.	AntActGUI	AntActGUI.java
2.	DialogAturAlgoritma	DialogAturAlgoritma.java
3.	DialogBuatTSP	DialogBuatTSP.java
4.	DialogBukaTSP	DialogBukaTSP.java
5.	DialogDataAnimasi	DialogDataAnimasi.java
6.	TSP	TSP.java
7.	Algo	Algo.java
8.	ParserTSP	ParserTSP.java
9.	DrawHandler	DrawHandler.java
10.	C_BuatTSP	C_BuatTSP.java
11.	C_Main	C_Main.java
12.	Ant	Ant.java
13.	Bound	Bound.java
14.	Matriks	Matriks.java
15.	Coordinate	Coordinate.java
16.	Tabel	Tabel.java
17.	Tour	Tour.java
18.	Utilities	Utilities.java

4.3 Implementasi Antarmuka

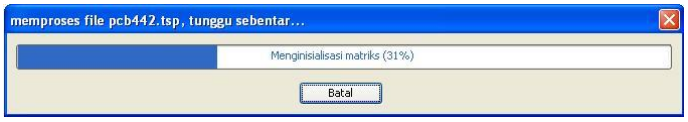
Perangkat lunak yang dikembangkan memiliki lima antarmuka yang diimplementasikan kedalam lima kelas. Impelementasi seluruh kelas antarmuka tersebut dapat dilihat pada tabel 3. Sedangkan tampilan keseluruhan antarmuka tersebut dapat dilihat pada gambar 11 hingga gambar 15.

Tabel 3. Implementasi kelas antarmuka

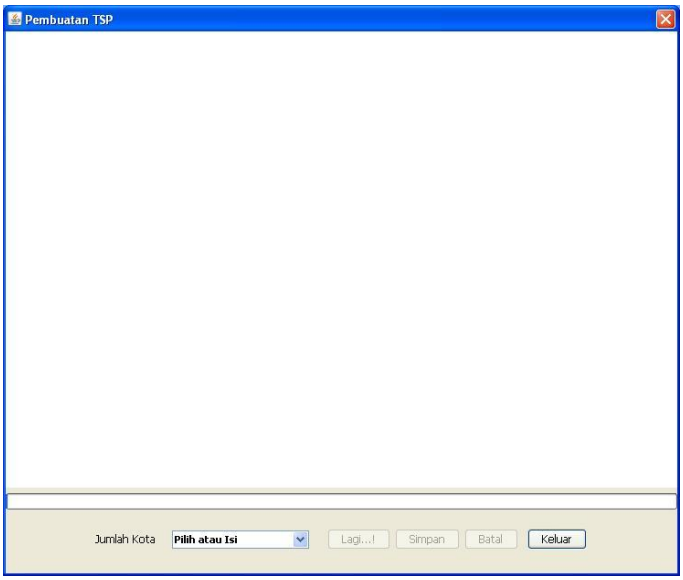
No	Antarmuka	Nama File
1.	Antarmuka utama	AntActGUI.java
2.	Antarmuka membuka arsip TSP	DialogBukaTSP.java
3.	Antarmuka membuat TSP	DialogBuatTSP.java
4.	Antarmuka untuk mengatur algoritma	DialogAturAlgoritma.java
5.	Antarmuka data animasi	DialogDataAnimasi.java



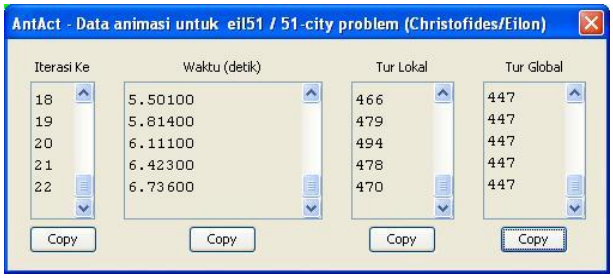
Gambar 11. Implementasi antarmuka utama



Gambar 12. Implementasi antarmuka ketika membuka arsip TSP



Gambar 13. Implementasi antarmuka pembuatan TSP



Gambar 14. Implementasi antarmuka data animasi

Gambar 15. Implementasi antarmuka mengatur algoritma

5. Pengujian Terhadap Algoritma

Skenario pengujian yang direncanakan adalah sebagai berikut:

1. Menguji Algoritma AS dan ACS menggunakan sebuah TSP dengan jumlah semut dan jumlah iterasi yang bervariasi.
2. Menguji Algoritma AS dan ACS menggunakan berbagai TSP dengan jumlah semut serta jumlah iterasi yang sama untuk masing-masing algoritma.

Seluruh TSP yang digunakan dalam pengujian diambil dari TSPLIB95[13]. TSPLIB95 merupakan *library* TSP yang sering digunakan dalam penelitian maupun riset. TSPLIB95 memiliki TSP berukuran kecil (14 kota) hingga yang berukuran besar (13.000 kota). Selain itu, TSPLIB menyertakan solusi optimalnya, bahkan beberapa diantaranya menyertakan tur optimal. Ukuran TSP yang digunakan selama pengujian berukuran dari 14 hingga 532 kota.

Parameter konstanta yang digunakan pada Algoritma AS dan ACS mengikuti saran[7], yaitu:

1. Algoritma AS: $\alpha = 1$, $\beta = 2$, $\rho = 0,5$
2. Algoritma ACS: $\alpha = 1 (selalu)$, $\beta = 2$, $\rho = 0,1$, $\xi = 0,9$, $q0 = 0,1$

5.1 Pengujian Pertama

Tujuan dari pengujian ini adalah untuk melihat performansi Algoritma AS dan Algoritma ACS terhadap perubahan jumlah semut serta jumlah iterasi. TSP yang digunakan adalah kroB200. TSP ini dipilih dengan pertimbangan ukurannya tidak terlalu besar (waktu komputasinya tidak terlalu lama) maupun terlalu kecil (dapat menunjukkan perbedaan waktu pencarian pada kedua algoritma).

5.1.1 Analisis Pengaruh Jumlah Semut terhadap Solusi yang dihasilkan

Solusi optimal pada kroB200 adalah 29437 sedangkan solusi greedy yang ditemukan adalah 36003. Jumlah semut yang digunakan antara 20 hingga 200 ekor dengan kelipatan 20. Sedangkan iterasi yang digunakan adalah 1000 iterasi.

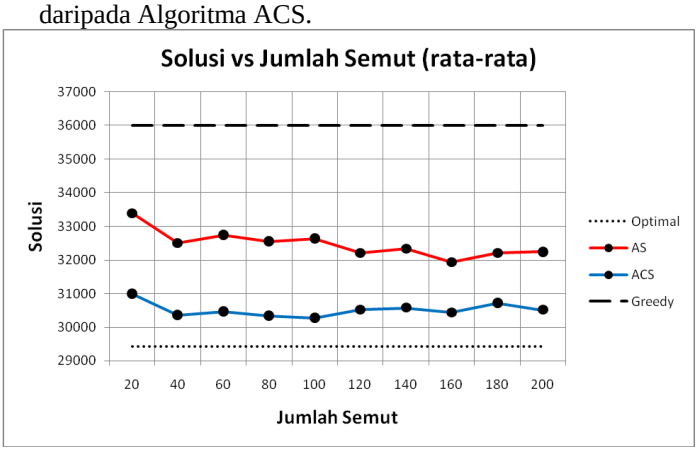
Berdasarkan grafik-1 dapat dilihat bahwa pada Algoritma AS, kenaikan jumlah semut akan memperbaiki solusi yang dihasilkan. Solusi terbaik diperoleh ketika menggunakan 160 semut dan terburuk ketika menggunakan 20 semut.

Berdasarkan grafik-1 dapat dilihat bahwa pada Algoritma ACS kenaikan jumlah semut juga mampu memperbaiki solusi (solusi paling buruk diperoleh ketika menggunakan 20 semut dan terbaik ketika menggunakan 100 semut). Namun pengaruh kenaikan jumlah semut pada ACS tidak sekuat pada Algoritma AS.

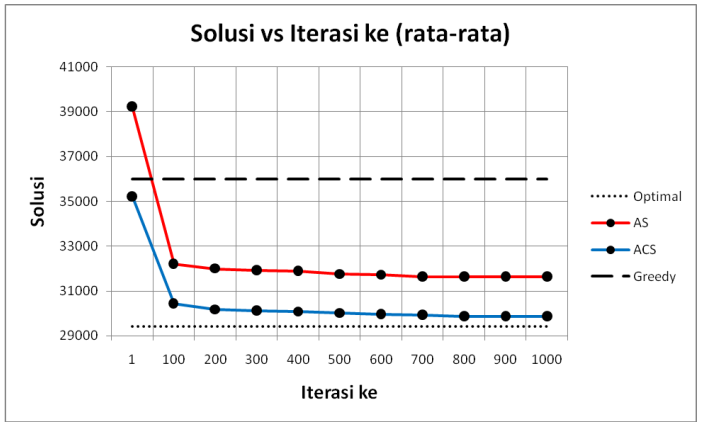
5.1.2 Analisis Pengaruh Jumlah Iterasi terhadap Solusi yang dihasilkan

Berdasarkan grafik 2, dapat dilihat bahwa kenaikan jumlah iterasi mampu memperbaiki kualitas solusi (rata-rata) dari Algoritma AS dan ACS. Hal ini disebabkan semut memiliki kesempatan yang lebih banyak untuk menelusuri TSP, sehingga kemungkinan untuk menemukan solusi yang lebih baik lebih besar.

Berdasarkan grafik 1 dan grafik 2 dapat pula dilihat bahwa untuk jumlah semut serta jumlah iterasi yang sama, Algoritma ACS memberikan solusi yang lebih baik daripada Algoritma ACS.



Grafik 1. Grafik Solusi - Jumlah Semut pada kroB200



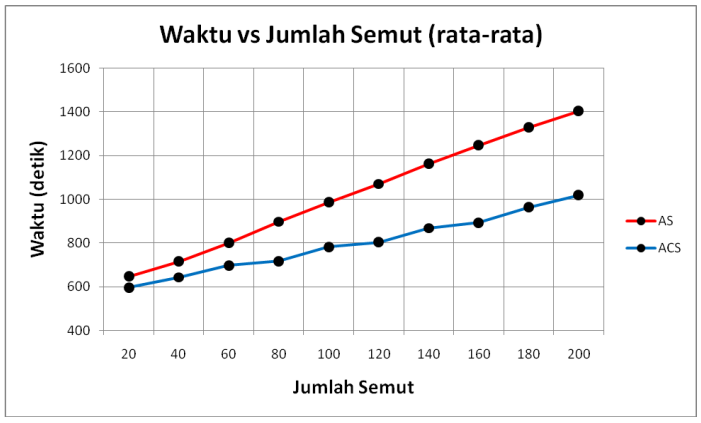
Grafik 2. Grafik Solusi - Jumlah Iterasi pada kroB200

5.1.3 Analisis Pengaruh Jumlah Semut terhadap Waktu Pencarian

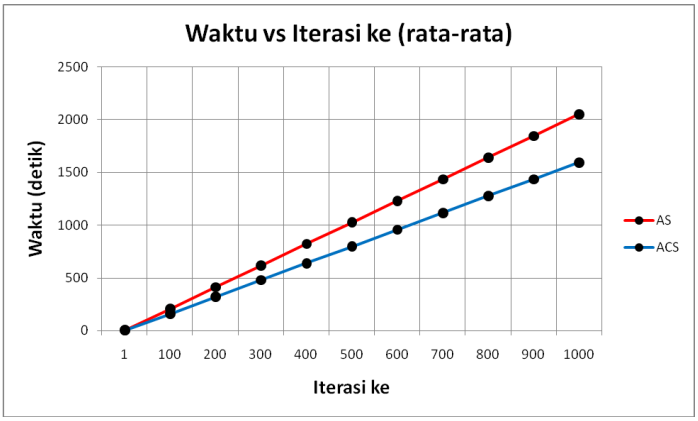
Berdasarkan grafik 3, dapat dilihat bahwa kenaikan jumlah semut akan meningkatkan waktu pencarian yang dibutuhkan oleh Algoritma AS maupun Algoritma ACS. Hal ini disebabkan waktu yang diperlukan untuk memobilisasi koloni dalam satu iterasi menjadi bertambah.

Berdasarkan grafik 4, dapat dilihat bahwa kenaikan jumlah iterasi akan meningkatkan waktu pencarian yang dibutuhkan oleh Algoritma AS dan Algoritma ACS. Hal ini disebabkan karena jumlah iterasi yang dikerjakan semakin banyak.

Berdasarkan grafik 3 dan grafik 4 dapat disimpulkan bahwa Algoritma ACS membutuhkan waktu pencarian yang lebih singkat dibandingkan Algoritma AS. Dimana perbedaan diantara keduanya semakin terlihat jika jumlah semut atau iterasi yang digunakan bertambah besar.



Grafik 3. Grafik Waktu - Jumlah Semut pada kroB200



Grafik 4. Grafik Waktu - Jumlah Iterasi pada kroB200

5.2 Pengujian ke-2

TSP yang digunakan berukuran dari 14 kota hingga 532 kota. Jumlah semut yang digunakan adalah setengah dari jumlah kota. Setiap TSP diuji dengan 500 iterasi. Sedangkan jumlah pengujian pada TSP, adalah sebagai berikut:

1. TSP berukuran 14 hingga 100 kota diuji sebanyak 5 kali
2. TSP berukuran 101 hingga 300 kota diuji sebanyak 3 kali
3. TSP berukuran diatas 300 kota diuji sebanyak 1 kali

Solusi rata-rata akan dicatat, lalu simpangannya dihitung dengan menggunakan persamaan 13.

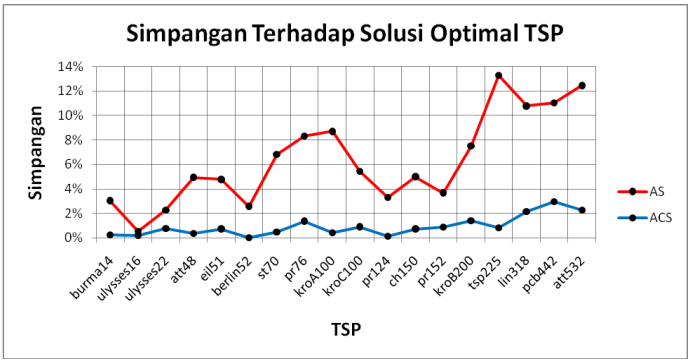
$$\text{Simpangan} = \frac{\text{Solusi Algoritma}}{\text{Solusi Optimal}} \times \text{Solusi Optimal} \qquad (13)$$

5.2.1 Analisis Simpangan Pada Berbagai TSP

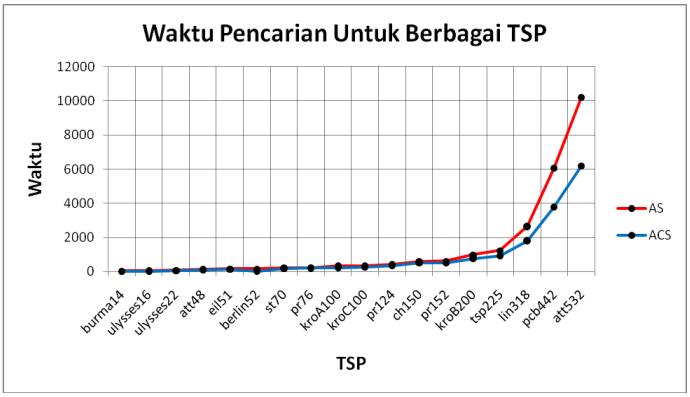
Dari grafik 5 dapat dilihat bahwa seiring bertambahnya ukuran TSP, perbedaan simpangan diantara keduanya semakin besar. Namun, simpangan Algoritma ACS tidak sebesar simpangan Algoritma AS. Dimana pada Algoritma AS, simpangan terbesar ada pada tsp225 (13,270%) sedangkan pada Algoritma ACS, simpangannya pada pcb442 hanya 2,932%.

4.2.2 Analisis Waktu Pencarian Terhadap Berbagai TSP

Berdasarkan grafik 5, dapat dilihat bahwa semakin besar ukuran TSP, maka waktu pencarian yang dibutuhkan juga semakin besar.



Grafik 5. Grafik simpangan pada berbagai TSP terhadap solusi optimal



Grafik 6. Grafik waktu pencarian terhadap berbagai TSP

Berdasarkan tabel 4, untuk TSP yang kecil (dibawah 100 kota) Algoritma ACS memiliki waktu pencarian rata-rata yang lebih cepat (kecuali eil51) daripada Algoritma AS walaupun selisihnya tidak terlalu besar. Namun untuk permasalahan yang lebih besar (diatas 100 kota), Algoritma ACS tetap memiliki waktu pencarian yang lebih cepat dibandingkan Algoritma AS dengan selisih waktu yang cukup besar.

Tabel 4. Waktu Pencarian Algoritma pada Berbagai TSP

No	TSP	Semut	Optimal	Waktu Pencarian	
				AS	ACS
1	burma14	7	3323	43.958	33.670
2	ulysses16	8	6859	50.047	45.122
3	ulysses22	11	7013	67.887	63.611
4	att48	24	10628	143.812	123.277
5	eil51	26	426	152.530	152.722
6	berlin52	26	7542	154.865	37.448
7	st70	35	675	208.859	208.788
8	pr76	38	108159	226.971	226.352

Tabel 4. Waktu Pencarian Algoritma pada Berbagai TSP (lanjutan)

No	TSP	Semut	Optimal	Waktu Pencarian	
				AS	ACS
9	kroA100	50	21282	346.067	255.160
10	kroC100	50	20749	346.432	272.119
11	pr124	62	59030	432.187	380.297
12	ch150	75	6528	595.275	519.909

5. Kesimpulan dan Saran

Kesimpulan yang diperoleh adalah sebagai berikut:

1. Algoritma semut, baik Algoritma *Ant System* (AS) maupun *Ant Colony System* (ACS) dapat diterapkan kepada TSP.
2. Perangkat lunak yang dibangun sanggup menyelesaikan TSP dengan menggunakan Algoritma AS dan ACS. Perangkat lunak juga mampu menampilkan TSP serta solusinya secara visual sehingga memudahkan pengguna untuk melakukan pengamatan
3. Penggujian menggunakan TSPLIB95 menunjukkan bahwa kenaikan jumlah semut pada Algoritma AS dan ACS mampu memperbaiki solusi yang dihasilkan, namun pada Algoritma ACS, pengaruh kenaikan jumlah semut tidak sekuat pada Algoritma AS.
4. Pengujian menggunakan TSPLIB95 menunjukkan bahwa kenaikan jumlah iterasi pada Algoritma AS dan ACS dapat memperbaiki kualitas solusi karena memberikan kesempatan yang lebih besar bagi semut untuk menemukan solusi yang lebih baik.
5. Pengujian terhadap berbagai TSP dari TSPLIB95 menunjukkan bahwa dengan jumlah semut dan jumlah iterasi yang sama, Algoritma ACS memberikan solusi yang lebih baik serta waktu pencarian yang lebih cepat dibandingkan dengan Algoritma AS.

Beberapa saran yang dapat diberikan adalah sebagai berikut:

1. Mengkaji ulang pengaturan konstanta yang digunakan pada tugas akhir ini untuk membuktikan bahwa pengaturan konstanta yang disarankan merupakan kombinasi yang terbaik.
2. Menerapkan Algoritma AS maupun ACS pada permasalahan optimasi yang lain.
3. Membandingkan kinerja Algoritma ACS terhadap metode heuristik yang lain.
4. Mengembangkan perangkat lunak supaya pengguna dapat menggambar TSP secara manual (tidak hanya dibangkitkan otomatis secara acak).

5. Mengembangkan perangkat lunak agar dapat melaporkan kesimpulan pencarian dengan menggunakan grafik agar analis dapat dilakukan dengan lebih cepat dan lebih mudah.

5. Referensi dan Daftar Pustaka

[1] Bonabeau, Eric; Dorigo, Marco; Theraulaz, Guy. (1999). *Swarm Intelligence From Natural to Artificial Systems*. Oxford University Press.

[2] Bonabeau, Eric; Théraulaz, Guy. (2000). *Swarm Smarts*. Scientific American, Maret 2000, pp. 72-79.

[3] Colorni, Alberto; Dorigo, Marco; Maniezzo, Vittorio. (1991). *Positive feedback as a search strategy*. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, IT.

[4] Cormen, Thomas H., et al. (2001). *Introduction to Algorithms*. McGraw-Hill.

[5] Dorigo, Marco; Gambardella, Luca. M. (1997). *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. TR/IRIDIA/1996-5, Université Libre de Bruxelles, Belgium.

[6] Dorigo, Marco; Di Caro, Giani; Gambardella, Luca. M. (1999). *Ant Algorithm for Discrete Optimization*. IRIDIA, Universite Libré de Bruxelles, Belgium.

[7] Dorigo, Marco; Stützle, Thomas. (2004). *Ant Colony Optimization*. A Bradford Book, The MIT Press.

[8] Munir, Rinaldi. (2003). Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[9] Munir, Rinaldi. (2006). Strategi Algoritmik. Departemen Teknik Informatika, Institut Teknologi Bandung.

[10] Rosen, Kenneth H. (1999). *Discrete Mathematics and Its Applications*. McGraw-Hill.

[11] Rosen, Kenneth H., et al. (2000). *Handbook of Discrete and Combinatorial Mathematics*. CRC Press.

[12] Pintea, Camelia-Mihaela; Serban, Gabriela. (2004). *Heuristics and Learning Approaches for Solving The Traveling Salesman Problem*. Studia University, Babeş-Bolyai, Informatica, Vol. XLIX, No. 2.

[13] TSPLIB95, (2009): Ruprecht-Karls-Universität Heidelberg, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, Tanggal akses: 13 November 2008, pkl 12:56.

[14] Wilf, Herbert S., (2002). *Algorithms and Complexity*. AK Peters, Ltd.