BAB II

LANDASAN TEORI

2.1 Graph

Graph merupakan bahasan yang sudah tua usianya namun memiliki banyak terapan sampai saat ini. Graph digunakan untuk merepresentasikan objek – objek diskrit dan hubungan antara objek – objek tersebut. Representasi visual dari graph adalah dengan menyatakan objek sebagai noktah, bulatan, atau titik. Sedangkan hubungan antar objek dinyatakan dengan garis.

2.1.1 Definisi Graph

Graph merupakan representasi dari suatu masalah yang digambarkan sebagai sekumpulan noktah atau simpul (*vertex*) yang dihubungkan dengan sekumpulan garis atau sisi (*edge*). Secara singkat suatu graph dapat ditulis sebagai G = (V,E) yang dalam hal ini:

V = Himpunan berhingga dan tidak kosong dari simpul-simpul (*vertex*).

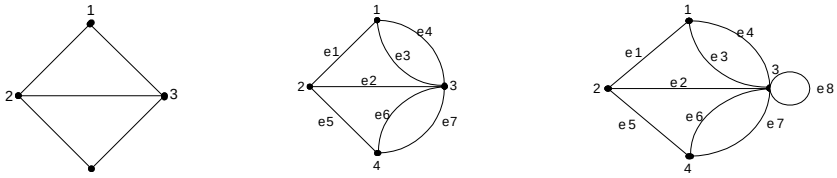
= { v_1, v_2,…v_n }

E = Himpunan sisi yang menghubungkan sepasang simpul.

= { e_1, e_2,…e_n }

Simpul pada graph dapat dinomori dengan huruf, seperti v,w,…, dengan bilangan asli 1, 2, 3,…, atau gabungan keduanya. Sisi (*edge*) yang menghubungkan dua simpul (*vertex*) v_i dan v_j dinyatakan dengan pasangan (v_i , v_j)

atau dengan lambang e_1, e_2 , dengan kata lain, jika e adalah sisi yang menghubungkan simpul v_i dan v_j, maka e dapat ditulis sebagai $e = (v_i , v_j)$.



(a) (b) (c)

Gambar 2.1 Graph Sederhana (a), Graph Ganda (b), Graph Semu (c)

2.1.2 Jenis-jenis Graph

Graph dapat dikelompokkan menjadi beberapa jenis bergantung pada sudut pandang pengelompokkannya. Pengelompokkan graph dapat dipandang dari ada tidaknya sisi ganda atau sisi gelang (*loop*), dan berdasarkan pada orientasi arah pada sisi.

Berdasarkan ada atau tidak adanya sisi ganda atau sisi gelang (*loop*) pada suatu graph, maka secara umum graph dapat dibedakan menjadi dua jenis, yaitu:

1. Graph sederhana

   Graph sederhana adalah graph yang tidak mengandung gelang maupun sisi ganda. Pada gambar 2.1. gambar (a) merupakan contoh graph sederhana.

2. Graph tak-sederhana

Graph tak-sederhana adalah graph yang mengandung gelang atau sisi ganda. Ada dua macam graph tak-sederhana, yaitu graph ganda dan graph semu. Graph ganda yaitu graph yang apabila pada graph tersebut terdapat sisi ganda, pada gambar 2.1 gambar (b) merupakan contoh dari graph ganda. Graph semu adalah

graph yang apabila pada graph tersebut terdapat gelang, pada gambar 2.1 gambar (c) merupakan contoh dari graph semu.

Sisi (*edge*) pada graph dapat mempunyai orientasi arah. Berdasarkan orientasi arah pada sisi (*edge*), secara umum graph dapat dibedakan menjadi dua jenis, yaitu:

1. Graph tak berarah

   Graph tak berarah adalah graph yang sisinya (*edge*-nya) tidak mempunyai orientasi arah. Pada graph tak berarah, urutan pasangan simpul (*vertex*) yang dihubungkan oleh sisi (*edge*) tidak diperhatikan, jadi (vj , vk) = (vk , vj) adalah sisi (*edge*) yang sama.

2. Graph berarah

   Graph berarah adalah graph yang setiap sisinya (*edge*) diberikan orientasi arah dan urutan pasangan simpul pada graph berarah diperhatikan, sehingga pada graph berarah (vj , vk) ? (vk , vj).

2.1.3 Terminologi Graph

Terminologi (istilah) pada pembahasan mengenai graph cukup banyak terminologi yang sering dipakai, meliputi:

1. Ketetanggaan (*Adjacent*).

Dua buah simpul (*vertex*) dikatakan bertetangga jika keduanya terhubung langsung. Pada gambar 2.1 (a), simpul (*vertex*) 1 bertetangga dengan simpul (*vertex*) 2 dan 3, tetapi tidak bertetangga dengan simpul (*vertex*) 4.

2. Derajat (*Degree*).

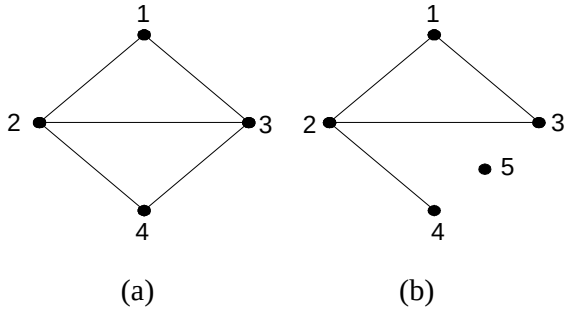
Derajat (*Degree*) suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut dan jika terdapat gelang (*loop*), gelang tersebut dihitung dua (dua buah sisi). Notasinya: d(v). Pada gambar 2.1 (a) d(1) = d(4) = 2, d(2) = d(3) = 3

3. Sirkuit (*Circuit*).

Sirkuit adalah lintasan dengan simpul (*vertex*) pertama sampai dengan simpul yang terakhir, dimana semua simpul yang dilalui hanya muncul sekali. Contoh sirkuit, pada gambar 2.1 (a), 1, 2, 3, 1 adalah merupakan sirkuit.

4. Terhubung (*Connected*).

Graph terhubung (*Connected Graph*) merupakan graph tak berarah dimana pada setiap pasang simpul (*vertex*) v_i dan v_j terdapat lintasan dari v_i ke v_j yang juga berarti ada lintasan v_j ke v_I, jika tidak maka graph tersebut disebut graph tak terhubung (*disconnected graph*). Gambar 2.2 (a) menunjukan contoh graph terhubung (*connected graph*), dan gambar 2.2 (b) adalah contoh dari graph tak terhubung (*disconnected graph*).

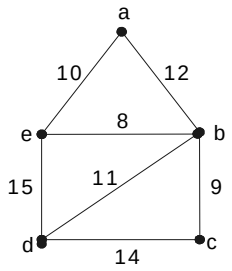


(a) (b)

Gambar 2.2 Graph Terhubung (a) dan Graph Tak Terhubung (b)

5. Graph Berbobot (*Weighted Graph*).

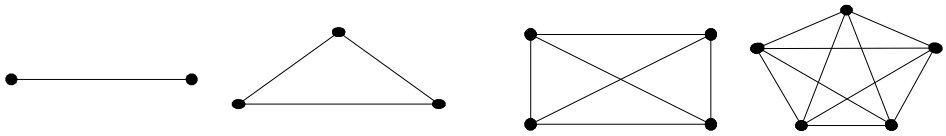
Graph berbobot (w*eighted graph*) adalah graph yang setiap sisinya diberi sebuah nilai (bobot). Bobot tiap sisi dapat menyatakan jarak antara dua buah kota atau panjang suatu jalan pada sebuah kota, biaya perjalanan antara dua buah kota, ongkos produksi, dan sebagainya. Gambar 2.3 adalah contoh graph berbobot.



Gambar 2.3 Graph Berbobot

6. Graph Lengkap (*Complete Graph*).

Graph lengkap (*complete graph*) adalah graph sederhana yang setiap simpulnya (*vertex*-nya) terhubung dengan semua simpul (*vertex*) yang lainnya. Derajat tiap simpul dari graph lengkap dengan n buah simpul adalah n-1, karena setiap simpul terhubung dengan semua simpul lainnya melalui satu sisi.



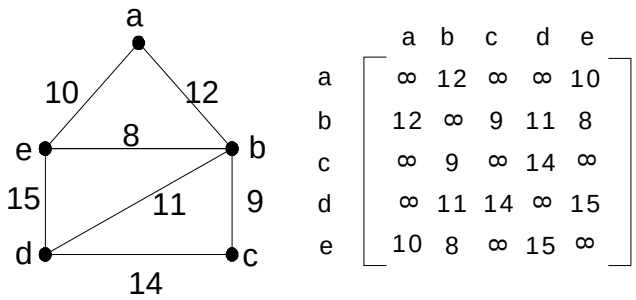
Gambar 2.4 Graph Lengkap Dari Dua, Tiga, Empat Dan Lima Simpul

2.1.4 Representasi Graph

Pemrosesan graph dengan program komputer, memerlukan representasi graph dalam memori. Ada beberapa representasi graph yang mungkin untuk dilakukan, salah satunya adalah dengan matrik ketetanggaan (*adjacency matrix*).

Matrik ketetanggaan adalah representasi graph yang paling umum digunakan. Misalkan G = (V,E) adalah graph dengan n simpul, n ≥ 1. Matrik ketetanggaan G adalah matrik yang berukuran n x n. bila matrik tersebut dinamakan matrik A = [a_{ij}] maka bernilai 1 jika simpul *i* dan simpul *j* bertetangga dan bernilai 0 jika simpul *i* dan simpul *j* tidak bertetangga. Matrik ketetanggan untuk graph sederhana dan tidak berarah selalu simetri dan diagonal utamanya selalu bernilai 0 (nol) karena tidak ada sisi gelang (*loop*). Jumlah elemen matrik ketetanggaan untuk graph dengan n simpul adalah n^2, jika tiap elemen membutuhkan ruang memori sebesar p maka ruang memori yang dibutuhkan seluruhnya adalah pn^2. Pada matrik ketetanggaan untuk graph tak berarah sederhana simetri, cukup dengan menyimpan elemen segitiga atas saja, karena matriknya simetri, sehingga ruang memori akan dapat dihemat sebesar $pn^2/2$. Pada graph berbobot, a_{ij} menyatakan tiap sisi yang menghubungkan simpul i dengan simpul j.

Gambar 2.5 merupakan contoh graph berbobot beserta matrik ketetanggaannya.



Gambar 2.5 Matrik Ketetanggaan Untuk Graph Berbobot

2.2 Traveling Salesman Problem

Travelling salesman problem termasuk ke dalam persoalan yang sangat terkenal dalam teori graf. Nama persoalan ini diilhami oleh masalah seorang pedagang yang akan mengunjungi sejumlah kota. Uraian persoalannya adalah sebagai berikut: 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 kebererangkatan. Kota dapat dinyatakan sebagai simpul graf, sedangkan sisi menyatakan jalan yang menghubungkan antar dua buah kota. Bobot pada sisi menyatakan jarak antara dua buah daerah. Persoalan pedagang tidak lain adalah menentukan sirkuit Hamilton yang memiliki bobot minimum pada sebuah graf terhubung.

2.3 Algoritma Genetika

Algoritama genetika adalah algoritma pencarian yang didasarkan atas mekanisme evolusi biologis. Keberagaman pada evolusi biologis adalah variasi dari kromosom antar individu organisme. Variasi kromosom ini akan mempengaruhi laju reproduksi dan tingkat kemampuan organisme untuk tetap hidup. Pada dasarnya ada empat kondisi yang sangat mempengaruhi proses evolusi, yaitu :

a. Kemampuan organisme untuk melakukan reproduksi.
b. Keberadaan populasi organisme yang bisa melakukan reproduksi.
c. Keberagaman organisme dalam suatu populasi.

d. Perbedaan kemampuan untuk *survive*.

Individu yang lebih kuat (*fit)* akan memiliki tingkat *survival* dan tingkat reproduksi yang lebih tinggi jika dibandingkan dengan individu yang kurang kuat. Pada kurun waku tertentu (sering dikenal dengan istilah generasi), populasi secara keseluruhan akan lebih banyak memuat organisme yang *fit.*

Algoritma genetika pertama kali dikembangkan oleh John Holland dari Univeristas Michigan (1975). John Holland mengatakan bahwa setiap masalah yang berbentuk adaptasi (alami maupun buatan) dapat diformulasikan dalam terminologi genetika. Algoritma genetika adalah simulasi dari proses evolusi Darwin dan operasi genetika atas kromosom.

2.3.1 Struktur Umum Algoritma Genetika

Pada algoritma ini, teknik pencarian dilakukan sekaligus atas sejumlah solusi yang mungkin yang dikenal dengan istilah populasi. Individu yang terdapat dalam suatu populasi disebut dengan kromosom. Kromosom ini merupakan suatu solusi yang masih berbentuk simbol. Populasi awal dibangun secara acak, sedangkan populasi berikutnya merupakan hasil evolusi kromosom-kromosom melalui iterasi yang disebut dengan dengan istilah generasi. Pada setiap generasi, kromosom akan melalui proses evaluasi dengan menggunakan alat ukur yang dengan fungsi *fitness*. Nilai *fitness* suatu kromosom akan menunjukkan kualitas kromosom dalam populasi tersebut. Generasi berikutnya dikenal dengan istilah anak (*offspring*) yang terbentuk dari gabungan dua kromosom generasi sekarang yang bertindak sebagai induk (*parent*) dengan menggunakan operator penyilangan

(*crossover*). Selain operator penyilangan, suatu kromosom dapat juga dimodifikasi dengan menggunakan operator mutasi. Populasi generasi yang baru dibentuk dengan cara menyeleksi nilai *fitness* dari kromosom induk (*parent*) dan nilai *fitness* dari kromosom anak (*offspring*), serta menolak kromosom-kromosom yang lainnya sehingga ukuran populasi (jumlah kromosom dalam suatu populasi) konstan. Setelah melalui beberapa generasi, maka algoritma ini akan konvergen ke kromosom terbaik.

2.3.2 Operator Genetika

Pada algoritma genetika terdapat tiga operator genetika, yaitu :

a. Seleksi
b. Pindah Silang (*crossover*)
c. Mutasi

2.3.3 Seleksi

Seleksi akan menentukan individu-individu mana saja yang akan dipilih untuk dilakukan rekombinasi dan bagaiman *offspring* terbentuk dari individu-individu terpilih tersebut. Langkah pertama yang dilakukan dalam seleksi ini adalah pencarian nilai *fitness*. Masing-masing individu dalam suatu wadah seleksi akan menerima probabilitas reproduksi yang tergantung pada nilai objektif dirinya sendiri terhadap nilai objektif dari semua individu dalam wadah seleksi tersebut. Nilai *fitness* inilah yang nantinya akan digunakan pada tahap-tahap seleksi berikutnya.

2.3.3.1 Seleksi Roda Roultte (*Roulettte Wheel Selection*)

Metode seleksi roda *roulette* ini merupakan metode yang paling sederhana dan sering juga dikenal dengan nama *stocastic sampling with replacement*. Pada Metode ini, individu-individu dipetakan dalam suatu segmen garis secara berurutan sedemikian sehingga tiap segmen individu memiliki ukuran yang sama dengan ukuran *fitness*-nya. Sebuah bilangan random dibangkitkan dan individu yang memiliki segmen dalam kawasan bilangan random tersebut akan terseleksi. Proses ini diulang hingga diperoleh sejumlah individu yang diharapkan.

2.3.3.2 Seleksi Lokal (*Local Selection*)

Pada seleksi lokal,setiap individu yang berada di dalam konstrain tertentu disebut dengan nama lingkungan lokal. Interaksi antar individu hanya dilakukan de dalam wilayah tersebut. Lingkungan tersebut ditetapkan sebagai struktur dimana populasi tersebut terdistribusi. Lingkungan tersebut juga dapat dipandang sebagai kelompok pasangan-pasangan yang potensial.

2.3.3.3 Seleksi dengan Pemotongan (*Truncation Selection*)

Seleksi ini biasanya digunakan untuk populasi yang jumlahnya sangat besar. Pada metode ini, individu-individu diurutkan berdasar nilai *fitness*-nya. Hanya individu yang terbaik saja yang akan dipilih sebagai induk. Parameter yang digunakan dalam metode ini adalah suatu nilai ambang *trunk* yang mengindikasikan ukuran populasi yang akan diseleksi sebagai induk yang berkisar

antara 50%-10%. Individu-individu yang ada di bawah nilai ambang ini tidak akan menghasilkan keturuan.

2.3.3.4 Seleksi dengan Turnamen (*Tournament Selection*)

Pada metode ini, akan ditetapkan suatu nilai *tour* untuk individu-individu yang dipilih secara random dari suatu populasi. Individu-individu yang terbaik dalam kelompok ini akan diseleksi sebagai induk. Parameter yang digunakan adalah *tour* yang bernilai dua sampai `N` (jumlah individu dalam suatu populasi).

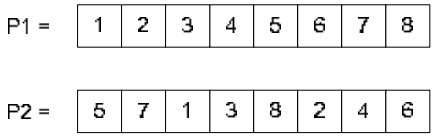
2.3.4 Pindah Silang *(Crossover)*

Operator pindah silang adalah operator yang sangat penting didalam algoritma genetik. Pindah silang dilakukan atas 2 (dua) kromosom dan sebuah kromosom mempunyai dua bagian yang berbeda sifatnya sehingga persilangan harus bisa menaganinya. Tujuan dari pindah silang adalah menambah individu pada populasi untuk menghasilkan kromosom anak *(offspring)*. Kromosom anak yang terbentuk akan mewarisi sebagian sifat kromosom induknya.

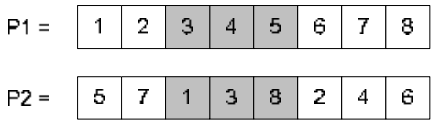
2.3.4.1 Metode Order (OX)

Pada metode ini, *offspring* dibentuk dengan cara memilih sebagian jalur dari suatu induk, kemudian menata ulang jalur berdasarkan urutan tertentu dari induk yang lainya.

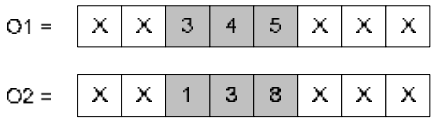
Misalkan ada 2 induk:



Pertama-tama kita pilih 2 bilangan bulat positif pada range 1 sampai jumlah fasilitas secara acak untuk menetukan posisi persilangan. Misalkan : r_1=2 dan r_2=5.



Kemudian kita copy-kan bagian tengah (antara kedua titik persilangan) ke setiap kromosom anak (0_1 dan 0_2).



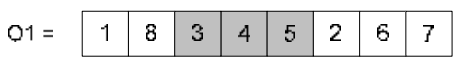
Kemudian dimulai dari titik persilangan kedua pada suatu induk, copy-kan fasilitas-fasilitas-fasilitas dengan urutan yang sama dari induk yang lain. Apabila sudah sampai pada fasilitas terakhir, lanjutkan mulai dari fsilitas pertama. Sehingga urutan fasiltas untuk induk kedua adalah:

2 – 4 – 6 – 5 – 7 – 1 – 3 – 8

Fasilitas ke- 3,4,dan 5 sudah ada pada anak pertama (0_1), sehingga hapus fasilitas ini dari urutan induk kedua, hasilnya:

2 – 6 –7 –1 –8

Letakan urutan ini ke anak pertama (0_1), dimulai dari titik persilangan kedua, sehingga anak pertama (0_1), menjadi:



Dengan cara yang sama, urutan fasilitas untuk induk pertama adalah:

6 –7 –8 – 1 – 2 –3 – 4 – 5

Fasilitas ke-1,3, dan 8 sudah ada pada anak kedua (0_2), sehingga hapus fasilitas ini dari urutan induk pertama, hasilnya:

6 –7 –2 – 4 –5

Letakan urutan ini ke anak kedua (0_2) dimulai dari titik persilangan kedua, sehingga anak kedua (0_2) menjadi:

O2 =	4	5	1	3	8	6	7	2

2.3.5 Mutasi

Setelah mengalami proses rekombinasi, pada *offspring* dapt dilakukan mutasi. Variabel *offspring* dimutasi dengan menambahkan nilai random yang sangat kecil (ukuran langkah mutasi), dengan probabilitas yang rendah. Peluang mutasi (p_m) didefinisikan sebagai persentasi dari jumlah total gen pada populasi yang mengalami mutasi. Peluang mutasi mengendalikan banyaknya gen baru yang akan dimunculkan untuk dievaluasi.

2.3.5.1 Mutasi Bilangan *Real*

Pada mutasi bilangan *real*, ukuran langkah mutasi biasanya sangat sulit ditentukan. Ukuran yang kecil biasanya sering mengalami kesuksesan, namun ada kalanya ukuran yang lebih besar akan berjalan lebih cepat. Operator mutasi untuk bilangan *real* ini dapat ditetapkan sebagai berikut :

- Variabel yang dimutasi = variabel ± *range**delta; (+ atau – memiliki probabilitas yang sama).
- *Range* = `0.5 * domain variabel`; (interval pencarian).
- Delta = $?(a_i * 2^{-i})$; `ai=1` dengan probabilitas `1/m`, selain itu `ai = 0`, dengan `m = 20`.

2.3.5.2 Mutasi Biner

Cara sederhana untuk mendapatkan mutasi biner adalah dengan mengganti satu atau beberapa nilai gen dari kromosom. Langkah-langkah mutasi ini adalah :

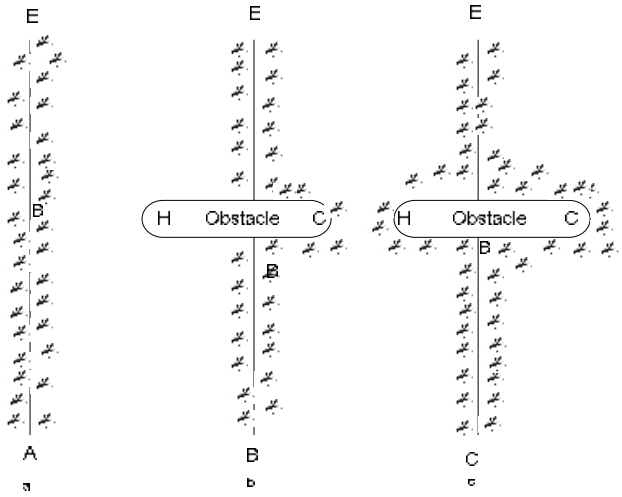
- Hitung jumlah gen pada populasi (panjang kromosom dikalikan dengan ukuran populasi).
- Pilih secara acak gen yang akan dimutasi.
- Tentukan kromosom dari gen yang akan dimutasi.
- Ganti nilai gen (0 ke 1, atau 1 ke 0) dari kromosom yang akan dimutasi tersebut.

2.4 Ant Colony System (ACS)

Ant Colony System (ACS) adalah algoritma yang didasarkan pada cara kerja koloni semut untuk menemukan jarak terpendek tanpa menggunakan ketajaman penglihatan mereka tetapi dengan memamfaatkan jejak *pheromone* yang dimanfaatkan sebagai komunikasi tidak langsung antar semut. *Pheromone* berasal dari kata “fer” (membawa) dan “hormon” (hormon) yang artinya “pembawa hormon”, yang berarti suatu zat/hormon yang diproduksi oleh kelenjar endokrin

yang memungkinkan korespondensi antar semut yang memberikan isyarat kimiawi dan ketika disekresikan sebagai isyarat oleh seekor semut maka semut yang lain bisa mengenalinya. Ketika semut berjalan, ia menyimpan *pheromone* dalam jumlah tertentu sehingga membentuk jejak *pheromone*. Semut dapat mencium *pheromone* dan ketika memilih jalur mereka cenderung untuk memilih jalur dengan konsentrasi *pheromone* yang lebih kuat. Pada ACS *update pheromone* dilakukan dua kali yaitu setiap semut melewati node (*update pheromone* lokal) dan setiap satu koloni semut melewati node (*update pheromone* global). Dengan adanya pembaharuan tingkat *pheromone* dimaksudkan agar semut-semut cenderung memberikan kadar *pheromone* lebih pekat pada setiap node sehingga dapat ditentukan node mana yang harus dikunjungi oleh semut. Jalan-jalan yang tidak dilalui oleh semut akan mengalami penguapan nilai *pheromone* lebih besar daripada jalan yang dilalui semut.

Pada gambar berikut ini dapat dilihat bagaimana cara semut tersebut menemukan jalur terpendek:



Gambar 2.6 Semut mencari jalur terpendek

Pada bagian (a) dari gambar di atas. Semut berjalan pada alur yang menghubungkan sumber maka (A) dengan sarangnya (E) disepanjang jalur ini terdapat jejak *pheromone* yang dibentuk oleh semut-semut.

Pada bagian (b) dapat dilihat bahwa jalur yang dilaui oleh sebuah rintangan. Jadi pada posisi B, semut berjalan dari A ke E harus menetukan apakah ia bergerak kekiri atau kekanan. Hal yang sama akan terjadi ketika pada semut yang berada pada posisi D. pilihan semut untuk bergerak ke kiri atau ke kanan dipengaruhi oleh intensitas jejak *pheromone* yang ditinggalkan oleh semut sebelumnya. Semakin besar jumlah *pheromone* pada sebuah jalur maka kemungkinan terpilihnya akan semakin besar.

Semut pertama yang mencapai titik B atau titik D memiliki probabilitas yang sama untuk bergerak ke kiri atau ke kanan, hal ini dikarenakan tidak ada jumlah *pheromone* pada jalur kiri dan jalur kanan. Pada bagian (C) dapat dilihat bahwa semut pertama yang melalui jalur BCD akan mencapai D sebelum semut pertama yang melalui jalur BHD karena jalur BCD lebih pendek daripada jalur BHD. Semut-semut yang berjalan dari E ke D akan menemukan jumlah pheromone yang lebih banyak dari jalur DCB, sehingga kemungkinan terpilihnya jalur DCB akan lebih besar daripada jalur DHB. Lama-lama, jumlah semut yang melalui jalur BCD akan lebih banyak dari pada jumlah semut yang melalui jalur BHD. Hal ini mengakibatkan jumlah *pheromone* pada jalur yang lebih pendek akan bertambah lebih cepat dari pada jalur yang lebih panjang sehingga kemungkinan semut untuk memilih jalur lebih pendek akan lebih besar. Akhirnya, semua semut akan memilih jalur yang lebih pendek.

Dalam algoritma ACS terdapat jumlah semut yang masing-masing akan melakukan perjalanan dari satu kota ke kota yang lainnya. Banyaknya semut yang akan melakukan pejalanan ditentukan oleh pengguna ACS. Semakin banyak jumlah semut pada ACS, maka solusi yang akan diperoleh akan lebih baik dibandingkan dengan jumlah semut yang lebih sedikit pada jumlah iterasi yang sama, jumlah semut yang semakin banyak akan mengakibatkan waktu komputasi untuk memperoleh solusi menjadi lebih lama. Pada saat semut melakukan perjalanan untuk mencari solusi dari TSP ia akan dipengaruhi oleh jejak pheromone dan jarak antar kota

Ketika algoritma ACS diterapkan pada masalah TSP. Jumlah *pheromone* antara kota i dengan kota j akan dilambangkan dengan $\tau(i, j)$, nilai $\tau(i, j)$ akan selalu dimodifikasi selama algoritma ACS dijalankan. Jika algoritma ACS diterapkan pada TSP simetris, maka $\tau(i, j) = \tau(j, i)$. Tetapi untuk TSP asimetris akan terdapat $\tau(i, j) \neq \tau(j, i)$

Ada tiga buah aturan yang akan digunakan dalam ACS yaitu :

1. Aturan *State Transition.*

Aturan ini digunakan oleh semut untuk memutuskan ke kota mana ia akan pergi, dengan menerapkan aturan ini, semut dapat memilih untuk pergi kesisi yang baru (sisi yang belum dilewati oleh semut) atau sisi yang terbaik (sisi yang memiliki jumlah *pheromone* terbanyak dan jarak terpendek) secara probabilitas.

2. Aturan *Local Updating*

Setelah semut melalui sisi tertentu, maka jumlah *pheromone* pada sisi tersebut akan berkurang. Jumlah *pheromone* pada sisi yang baru saja dikunjungi oleh semut dapat diperoleh dengan menerapkan *local updating.*

3. Aturan *Global Updating*

Aturan ini digunakan setelah semua semut membentuk jalur perjalanan. Pada aturan *global updating* akan dilakukan pengurangan jumlah *pheromone* pada semua sisi, kemudian dilakukan penambahan jumlah *pheromone* pada sisi-sisi yang termasuk dalam perjalanan dengan jarak yang terpendek.

Dalam algoritma ACS akan dilakukan proses iterasi (pengulangan dalam mencari jalur terpendek dan jarak perjalanan yang terpendek). Banyak iterasi yang dilakukan sesuai dengan banyaknya iterasi yang diinginkan oleh pengguna. Dalam sebuah proses iterasi, semua semut akan melaukan perjalanan sampai semua kota dikunjungi, ketika semut melakukan perjalannya, semut akan melukan *state transition* untuk memilih kota yang akan dituju dan aturan *local updating* untuk memodifikasi jumlah *pheromone* pada sisi yang telah dikunjungi oleh semut. Setelah semua kota dikunjungi oleh semut, maka digunakan aturan *global updating* untuk mengurangi jumlah *pheromone* pada semua sisi dan menambah jumlah *pheromone* pada sisi-sisi yagn termasuk dalam perjalanan dengan jarak terpendek. Cara kerja ACS secara umum dapat dilihat pada algoritma berikut :

```
Inisialisasi semua parameter dan variable yang akan digunakan
Ulangi ( pengulangan pada bagian ini disebut iterasi)
      Masing-masing semut diletakan secara acak pada sebuah kota yang  diperoleh
      dari hasil inisialisasi
         Ulangi
            Masing-masing semut menerapkan aturan state transition dan local
            updating sampai semua semut selesai mengunjungi semua kota
      Cari panjang perjalan untuk semua semut
      Cari semut yang memiliki jarak perjalanan terpendek
      Terapkan aturan global updating
Sampai jumlah iterasi yang dikehendaki
Tampilkan jalur dan jarak yang terpendek
```

Gambar 2.7 Struktur Algoritma ACS

Berdasarkan algoritma diatas cara kerja ACS adalah sebagai berikut : masing-masing semut mula-mula ditempatkan pada sebuah kota yang diperoleh dari hasil inisialisasi. Setiap semut melakukan perjalanan dengan dengan menerapkan aturan *state transition* untuk mencari kota yang ditujunya. Ketika semut sudah menggunakan aturan *state transition*, ia langsung menerapkan aturan *local updating* untuk mengurangi jumlah *pheromone* pada sisi yang telah dikunjunginya. Penggunaan aturan *state transition* dan *local updating* ini terus dilakukan berulang-ulang sampai semua semut mengunjungi semua kota yang ada pada permasalahan TSP. Setelah semua semut menyelesaikan perjalananya maka dilakukan perhitungan panjang perjalanan untuk semua semut dan dilakukan pencarian semut memiliki panjang perjalanan terpendek, kemudian jumlah *pheromone* pada semua sisi dimodifikasi lagi denga menerapkan aturan *global updating*. Langkah-langkah tersebut dilakukan berulang kali sampai jumlah iterasi yang diinginkan oleh pengguna program.

2.4.1 Aturan *State Transition*

Aturan *state transition* dalam ACS adalah sebagai berikut : sebuah semut yang berada dikota i memilih untuk pergi ke kota *s* dengan menerapkan persamaan berikut :

$$s = \begin{cases} \arg\max_{i \in Jk(i)} \left\{ [\tau(i,l)].[\eta(i,l)]^{\beta} \right\} & \text{jika } q \leq q0 \text{ (}exploitasi\text{)} \\ Pilihmenurutpersamaanberikutnya & \text{jika } q \geq q0 \text{ (}explorasi\text{)} \end{cases}$$

dimana :

q_0 = Parameter yang menunjukan hubungan antara exploitasi dan eksplorasi ($0<q_0<1$)

q = Parameter bernilai acak antara 0 sampai 1 ($0<q<1$)

β = Mengatur bobot intensitas relative jejak pheromone. Jika $\beta = 0$ jarak antar kota tidak berpengaruh

$\tau(i,l)$ = Jumlah *pheromone* pada sisi yang menghubungkan kota i dengan kota *l*

$\eta(i,l)$ =Untuk memilih kota terdekat yang mempunyai probabilitas tinggi dan merupakan fungsi heuristic yang digunakan sebagai invers jarak $d(i,l) = 1/dil$

$J_k(i)$ = Daftar sekumpulan kota yang masih harus dikunjungi oleh semut k ketika semut masih di kota i, lengkap dengan daftar semua kota yang akan dikunjungi, $J_k(i)$ disebut juga sebagai tempat penyimpanan kota-kota yang akan dikunjungi semut

Jika $q \leq q_0$ (*exploitasi*) yaitu semut akan memilih jalur yang terbaik menurut nilai *pheromone* dan dari heuristic η.

Jika $q \geq q_0$ (*explorasi*) yaitu semut akan memilih kota berikutnya menurut persamaan probabilitas berikut ini :

$$Pk(i,j) = \begin{cases} \dfrac{[\tau(i,j)].[\eta(i,j)]^{\beta}}{\sum_{j \in Jk(i)} [\tau(i,l)].[\eta(i,l)]^{\beta}} & \text{jika } J \in Jk(i) \\ 0 & \text{jika } J \notin Jk(i) \end{cases}$$

dimana :

$Pk(i,j)$ = Probabilitas semut k memilih untuk berpindah dari kota i ke kota j

2.4.2 Aturan *Local Updating*

Ketika semut sedang mencari solusi dari Travelling Salesman Problem. Semut mengunjungi sisi yang menghubungkan satu kota dengan kota yang lainnya dan langsung memodifikasi jumlah *pheromone* pada sisi yang baru saja dikunjunginya dengan menerapkan persamaan berikut :

$$\tau(i,j)_{baru} \leftarrow (1-\rho).\tau(i,j)_{lama} + \rho.\tau_0$$

dimana :

ρ = parameter yang merepresentasikan *evaporation pheromone local* yang bernilai antara 0 sampai 1 ($0 < \rho < 1$)

τ_0 = (jumlah *pheromone* awal) = $\dfrac{1}{jarakawal * jumlahkota}$

Parameter ρ akan menentukan besarnya pengurangan *pheromone* pada sisi yang baru saja dikunjunginya oleh semut. Semakin besar nilai ρ maka pengurangan jumlah *pheromone* akan semakin banyak, sehingga perjalanan semut akan cepat konvergen pada jalur tertentu.

Dengan adanya *local updating*, maka jumlah *pheromone* pada sisi yang baru saja dikunjungi menjadi berkurang sehingga kemungkinan terpilihnya sisi tersebut oleh semut yang lain pada langkah iterasi yang sama menjadi berkurang. Akibatnya semut tidak pernah konvergen pada sisi tertentu saja, tetapi semut lebih suka memilih sisi-sisi yang belum pernah dikunjungi.

2.4.2 Aturan *Global Updating*

Aturan *global updating* dilakukan setelah semua semut selesai melakukan perjalanannya. Jumlah *pheromone* pada setiap sisi dimodifikasi dengan menerapkan persamaan berikut :

$$\tau(i, j) \leftarrow (1-\rho).\tau(i, j) + \rho.\frac{1}{L_{gb}}$$

dimana :

ρ = Parameter yang merepresentasikan *evaporation pheromone* global yang bernilai antara 0 dan 1 ($0< \rho <1$)

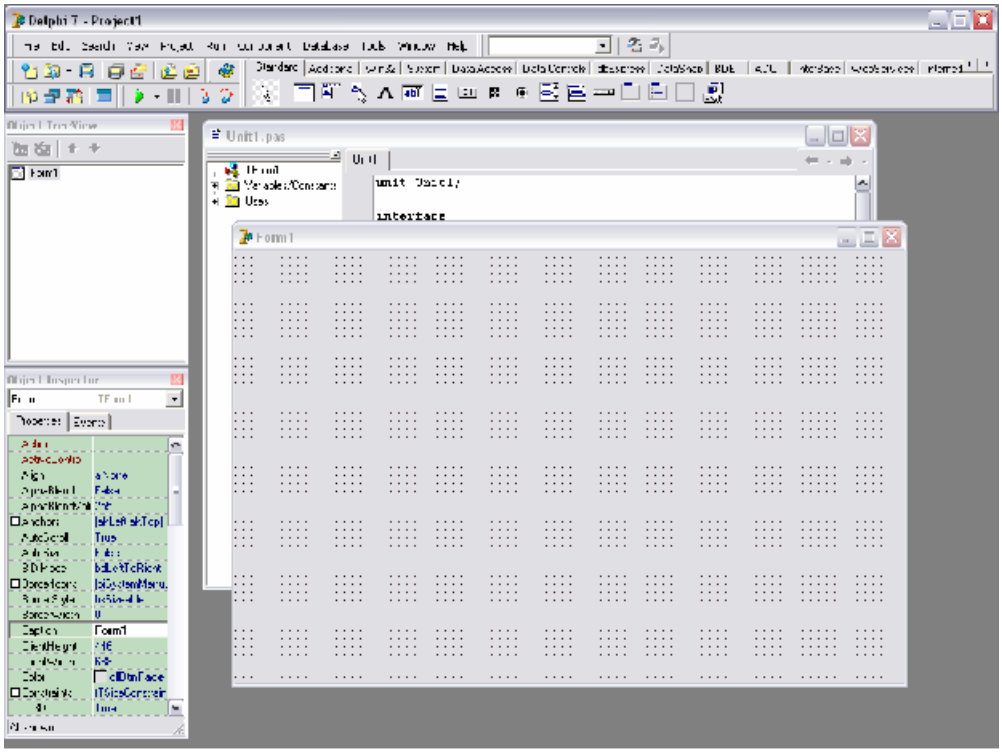
L_{gb} = Jarak tempuh terpendek yang di peroleh sejak awal iterasi sampai dengan iterasi yang sedang berlangsung.

Global updating dimaksudkan mengurangi jumlah *pheromone* pada semua sisi sebesar ρ kemudian dilakukan penambahan jumlah *pheromone* pada sisi yang termasuk dalam perjalanan dengan jarak yang paling pendek.

Parameter ρ pada persamaan diatas untuk menetukan besarnya penguapan *pheromone*. Semakin besar nilai ρ maka penguapan *pheromone* akan semakin banyak. Penguapan *pheromone* yang terlalu banyak akan menyebabkan perjalan semut menjadi cepat konvergen pada jalur tertentu.

2.5 Borland Delphi 7.0

Perangkat lunak yang digunakan pada penyusunan tugas akhir ini adalah program Delphi 7. Borland Delphi 7 adalah suatu bahasa pemrograman yang memberikan berbagai fasilitas pembuatan aplikasi visual. Keunggulan bahasa pemograman ini terletak pada produktivitas, kualitas, pengembangan perangkat lunak, kecepatan kompilasi, pola desain yang menarik serta diperkuat dengan pemogramannya yang tersruktur. Keunggulan lain dari delphi adalah dapat digunakan untuk merancang program aplikasi yang memiliki tampilan aplikasi seperti program aplikasi lain yang berbasis windows.



Gambar 2.8 Tampilan dasar delphi.