BAB II
TINJAUAN PUSTAKA

Pada bab ini dibahas definisi yang digunakan pada pembahasan *hybrid* algoritma genetika (GA) dengan algoritma *Ant Colony Optimization* (ACO) dalam menyelesaikan *Dynamic Travelling Salesman Problem* (DTSP).

2.1 *Dynamic Traveling Salesman Problem* (DTSP)

Soleimanian, dkk. (2012) menyatakan bahwa DTSP pertama kali dikenalkan oleh Psaraftis pada tahun 1988. DTSP adalah pengembangan dari *Travelling Salesman Problem* (TSP). DTSP merupakan suatu permasalahan yang bertujuan untuk mencari rute dengan jarak tempuh terpendek dalam mengunjungi sejumlah *n* kota tujuan. *Salesman* menjalankan rute yang dimulai dari kota awal dan kemudian mendapat *update* kota tujuan ketika rute sedang berlangsung. Rute dikatakan selesai jika tidak ada lagi *update* kota tujuan dan *salesman* sudah kembali ke kota awal. Terdapat dua tipe dari DTSP, yaitu :

1. Penambahan atau penghapusan sejumlah *m* kota tujuan.

   Saat *salesman* melakukan perjalanan, terdapat perubahan kota tujuan dengan membatalkan kota tujuan atau dengan menambah kota tujuan sejumlah *m* kota sehingga solusi berubah (*dynamic*).

2. Perubahan *cost* (biaya) tiap kota namun kota tujuan tetap.

   Saat *salesman* melakukan perjalanan, terdapat perubahan biaya di setiap kota yang akan dikunjungi *salesman* dengan pengurangan biaya atau

penambahan biaya sehingga solusi berubah (*dynamic*).

Pengkajian *Dynamic Travelling Salesman Problem* tipe pertama disebabkan karena tipe ini lebih sering diterapkan dalam kehidupan nyata, contohnya ketika terdapat penambahan kota tujuan saat terdapat pelanggan yang baru dan pengurangan kota tujuan sebab pelanggan membatalkan pembelian. Guntsch, dkk (2001) mengemukakan bahwa DTSP tipe pertama ini melibatkan seorang *salesman* (petugas) yang harus melakukan kunjungan ke-n kota tujuan dengan kemungkinan adanya penambahan atau pengurangan sejumlah m kota tujuan ketika rute kunjungan sedang berjalan. Rangkaian kunjungan yang dimulai dari kota awal ke semua kota tujuan harus membentuk suatu jalur dengan ketentuan setiap kota tujuan tersebut hanya boleh dikunjungi tepat satu kali dan kembali lagi ke kota awal. Penyelesaian akhir terhadap permasalahan DTSP ini adalah untuk memperoleh rute kunjungan terpendek.

Untuk memudahkan proses penyelesaian *Dynamic Travelling Salesman Problem* (DTSP) dapat dilihat pada contoh ilustrasi Gambar 2.1, Gambar 2.2 dan Gambar 2.3 berikut ini :

D	**C**	**G**	**A**	**E**	**J**	**I**	**F**	**H**	**B**	**D**

Gambar 2.1 Kota Awal

D	**C**	**G**	**A**	**E**	**I**	**F**	**H**	**B**	**D**

Gambar 2.2 Update Hapus Kota

D	**C**	**G**	**A**	**E**	**I**	**K**	**F**	**H**	**B**	**D**

Gambar 2.3 Update Tambah Kota

Berikut beberapa definisi yang terkait dengan DTSP :

Definisi 2.1 *Graph G* didefinisikan sebagai himpunan berhingga *V(G)* yang tak kosong dengan elemen-elemennya disebut titik (*vertice*) dan himpunan *E(G)* (mungkin kosong) yang elemen-elemennya merupakan pasangan tak terurut 2 elemen berbeda dari *V(G)* dan disebut garis (*edge*).

(Chartrand dan Oellermann, 1993)

Definisi 2.2 Perjalanan *(walk)* pada *graph G* adalah urutan secara bergantian titik-titik elemen $V(G)$ dan *edge* elemen $E(G)$,misalnya berbentuk:

$$W = v_0, e_1, v_1, e_2, v_0, \dots, e_{n-1}, v_n \quad (n \geq 0)$$

yang dimulai dan diakhiri titik sehingga $e_i = (v_{i-1}, v_i)$ untuk $i = 1,2, \dots, n$.

(Chartrand dan Oellermann, 1993)

Definisi 2.3 *Digraph (directed graph)D* adalah himpunan berhingga yang tak kosong $V(D)$ yang anggotanya disebut titik dan himpunan $E(D)$ dari pasangan garis titik terurut. *Arc* (garis berarah) adalah elemen-elemen dari $E(D)$.

(Chartrand dan Oellermann, 1993)

Definisi 2.4 *Graph* berbobot adalah graph yang setiap garisnya mempunyai bobot.

(Chartrand dan Oellermann, 1993)

Mengacu pada **Garfinkel dan Nemhauser (1972)**, permasalahan DTSP dapat digambarkan dalam *graph* $D = (V, E)$. Misal V adalah himpunan dari kota dan E himpunan garis berarah penghubung kota. Garis (i_1, i_2) menunjukkan bahwa ada hubungan dari kota i_1 ke kota i_2.

Titik dari *tour* DTSP dengan *n* kota tujuan adalah sebagai berikut:

$$t = (i_1, i_2, ..., i_{n-1}, i_n)$$

dengan $(i_1, i_2, ..., i_{n-1}, i_n)$ adalah permutasi (urutan indeks) dari bilangan bulat $(1,2,\ldots,n)$ dan paling banyak kemungkinan jumlah total dari *tour*nya adalah $n!$.

Garis dari *tour* pada DTSP dengan *n* kota tujuan adalah sebagai berikut:

$$g = \{(i_1, i_2), (i_2, i_3), ..., (i_{n-1}, i_n), (i_n, i_1)\}$$

Variabel x_{ij} didefinisikan sebagai berikut:

$$x_{ij} = \begin{cases} 1, & \text{jika ada garis } (i, j) \text{ dalam } tour \\ 0, & \text{yang lain} \end{cases} \qquad (2.1)$$

dengan $i = 1,2,3,\ldots,n$ dan $j = 1,2,3,\ldots,n$

Untuk setiap titik i, tepat satu garis (i, j) harus ada pada setiap *tour*, jadi:

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,3,\ldots,n.$$

Dan juga untuk setiap titik j, tepat satu garis (i, j) harus ada pada setiap *tour*, dengan demikian

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,3,\ldots,n.$$

Misalkan jarak dari titik (i, j) adalah c_{ij}, jika $i = j$, maka $c_{ij} = 0$ dengan $i = 1,2,3,\ldots,n.$ Dengan demikian fungsi tujuan TSP dapat ditulis sebagai berikut : Meminimalkan : $z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$ (2.2)

Dengan batasan :

$\sum_{j=1}^{n} x_{ij} = 1, i = 1,2,3,\ldots,n.$

$\sum_{i=1}^{n} x_{ij} = 1, j = 1,2,3,\ldots,n.$

$x_{ij} \in \{0,1\}, i = 1,2,3,\ldots,n.$ dan $j = 1,2,3,\ldots,n.$

2.2 Optimasi

Definisi 2.5 Optimasi adalah suatu proses untuk mencapai hasil yang optimal (nilai paling baik yang dapat dicapai).

Dalam disiplin ilmu matematika optimasi 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, 2007)

Definisi 2.6 Algoritma adalah suatu himpunan aturan atau instruksi yang telah dirumuskan dengan baik (*well-defined*) untuk memperoleh keluaran

khusus (*spesific output*) dari suatu masukan khusus (*spesific input*) dalam langkah-langkah yang jumlahnya berhingga.

(Chartrand dan Oellerman, 1993)

2.3 Algoritma Ant Colony Optimization

Definisi 2.7 ACO adalah algoritma optimasi yang meniru kinerja alami semut ketika mencari makanan dan mencari jalan terpendek antara sarang dan sumber makanan berkat pertukaran pesan lokal atau *pheromone*.

(Dorigo dkk., 1996)

Tingkah laku semut dalam pemilihan jalan terpendek ini merupakan hasil timbal balik positif (*autokatalitis*). Hal ini diperoleh melalui bentuk komunikasi tidak langsung (*stimergy*) seperti digambarkan berikut ini :

Stimergy didefinisikan sebagai perubahan lokal dari lingkungan yang disebabkan oleh semut ketika pergi, semut-semut itu meninggalkan zat kimia yang disebut *pheromone*. *Pheromone* adalah zat kimia yang berasal dari kelenjar endokrin dan digunakan makhluk hidup untuk mengenali sesama jenis dalam satu spesies. Setiap pemilihan titik, semut membuat keputusan yang diwujudkan dalam bentuk banyak sedikitnya *pheromone*. Proses ini merupakan proses *autokatalis* karena dalam kenyataannya seekor semut yang memilih jalan dalam gilirannya akan menambah *pheromone* seperti yang dilakukan oleh semut lainnya pada saat kemudian.

Hal yang membedakan antara semut sebenarnya dengan semut pada Algoritma ACO adalah sebagai berikut :

a. Semut pada algoritma ini akan memiliki ingatan (*memory*).

b. Semut pada algoritma ini tidak sepenuhnya buta.

c. Semut pada algoritma ini hidup pada lingkungan dengan waktu yang diskrit.

Langkah – langkah Algoritma ACO adalah sebagai berikut :

1. Inisialisasi parameter, yaitu:

   Alpha (α) = Tetapan pengendali intensitas *pheromone* ($\alpha > 0$).

   Betha (β) = Tetapan pengendali *visibility* ($\beta > 0$).

   ρ = Koefisien penguapan intensitas *pheromone*, $\rho \in [0,1]$ untuk mencegah jumlah *pheromone* yang tak terhingga.

   ant_max = Jumlah semut.

   jml_kota = Jumlah kota.

   max_ite untuk iterasi maksimal, dan konstanta ($Q > 0$).

2. Membuat matriks pheromone awal dengan persamaan $\frac{1}{jml\_kota}$ jika $kota_i \neq kota_j$ dan 0 untuk yang lain.

3. Menempatkan sejumlah semut pada *node* awal secara acak.

4. Menghitung nilai probabilitas pilih kota dari *node* awal ke *node* yang akan dikunjungi. *Tabu list* adalah tempat yang disediakan untuk penyimpanan solusi sementara yang dihasilkan pada tiap perhitungan probabilitas pilih kota. Persamaan untuk menentukan nilai probabilitas ditunjukkan pada persamaan **(2.3)**.

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{j\in\Gamma}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}} & , i = 1,2,\dots,n; j = 1,2,\dots,n \\ 0, \quad j \notin \Gamma \end{cases} \qquad (2.3)$$

keterangan :

$p_{ij}(t)$ = Probabilitas *node* i untuk memilih *node* j sebagai *node* berikutnya disaat t.

t_w = Iterasi *ke-w* , dengan *w* = 1,2,3,...,*ite.*

$\tau_{ij}(t)$ = Nilai *pheromone* antara *node* i dan *node* j disaat t.

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

$\eta_{ij}(t)$ = Fungsi visibility (lebih memilih *node* yang berdekatan dengan probabilitas yang tinggi).

d_{ij} = Jarak antara *node* i dan *node* j.

Γ = *Tabu list*.

(Dorigo dkk., 1996)

5. Menghitung panjang perjalanan.

Setelah semua semut menyelesaikan satu siklus selanjutnya dihitung panjang perjalanan. Indeks s menyatakan indeks urutan perjalanan, node asal dinyatakan $tabu_k\,(s)$ dan *node-node* lainnya dinyatakan sebagai $\{N - tabu_k\}$. Menghitung panjang perjalanan dengan persamaan **(2.4)** :

$$L_k = d_{tabu_k(n),\,tabu_k(1)} + \sum_{s=1}^{n-1} d_{tabu_k(s),\,tabu_k(s+1)} \qquad (2.4)$$

(Dorigo dkk., 1996)

6. Memperbarui matrik *pheromone* dengan persamaan **(2.5)** dan mengosongkan *tabu list*.

$$\tau_{ij}(t+1) = \rho \,.\, \tau_{ij}(t) \; + \; \sum_{k=1}^{m} \Delta\tau_{ij}^{\,k} \qquad (2.5)$$

dengan,

$\tau_{ij}(t+1)$ = Nilai *pheromone* antara *node* i dan *node* j disaat $(t+1)$.

$\Delta\tau_{ij}^{\,k}$ = Perubahan harga *pheromone* antar *node* setiap semut

yang dihitung berdasarkan persamaan :

$$\Delta\tau_{ij}^{\,k} = \begin{cases} \dfrac{Q}{L_k}, & \text{untuk } (i,j) \in \text{node dalam tabu}_k \\ 0, & \text{untuk } (i,j) \text{ lainnya} \end{cases}$$

keterangan :

Q = Konstanta positif yang menyatakan banyaknya *pheromone* yang ditinggalkan oleh semut (tetapan siklus semut).

L_k = Total jarak yang dilalui semut ke-*k,* dengan $k = 1,2,..,m$

(Dorigo dkk., 1996)

2.4 Algoritma Genetika

Definisi 2.8 Algoritma genetika adalah algoritma pencarian yang meniru mekanisme seleksi dan evolusi alam. Algoritma ini akan mengkombinasikan daya tahan (*survival*) dari suatu struktur data yang paling baik (*fittest*).

(Goldberg, 1989)

Beberapa istilah yang digunakan dalam algoritma genetika dan definisinya menurut **Gen dan Cheng (1997)** dan **Obitko (1998)** adalah sebagai berikut:

Definisi 2.9 Populasi adalah sebuah kumpulan solusi yang direpresentasikan dengan kromosom.

Definisi 2.10 Kromosom adalah individu dalam populasi yang terdiri dari kumpulan simbol merepresentasikan solusi dari permasalahan.

Definisi 2.11 Anak adalah kromosom baru untuk membentuk populasi baru yang terbentuk dari menggabungkan dua kromosom dari populasi sekarang dengan menggunakan operator *crossover* atau memodifikasi kromosom dengan menggunakan operator mutasi.

Definisi 2.12 *Fitness* adalah nilai yang menunjukkan keandalan suatu individu untuk bertahan dalam populasi.

Pencarian algoritma genetika dilakukan pada sekumpulan solusi yang dinamakan populasi. Pencarian ini dimulai dengan pembentukan populasi awal kromosom, dimana masing-masing kromosom merupakan solusi tunggal dari masalah ini. Dalam setiap iterasi (generasi), sejumlah populasi dimodifikasi dengan menerapkan dua operator genetika yaitu c*rossover* dan mutasi.

Berikut adalah komponen utama algoritma genetika yaitu pengkodean, fungsi *fitness*, seleksi, *crossover*, mutasi, dan *max_ite*.

2.4.1. Pengkodean

Definisi 2.13 Pengkodean adalah cara menyajikan suatu individu yang berisi informasi dari solusi atau individu.

(Obitko, 1998)

Langkah awal dari algoritma genetika adalah pengkodean. Menurut **Gen dan Cheng (1997)** dan **Obitko (1998)** terdapat beberapa jenis pengkodean, diantaranya :

1. Pengkodean biner adalah pengkodean yang setiap kromosomnya direpresentasikan dengan dua bilangan, yaitu 0 dan 1.

   Contoh :

0	0	1	1	0	1	1	0	1

2. Pengkodean permutasi adalah pengkodean yang setiap kromosomnya direpresentasikan sebagai sebuah urutan bilangan.

   Contoh :

2	1	5	4	8	7	9	3	6

3. Pengkodean nilai adalah pengkodean yang setiap kromosomnya direpresentasikan sebagai nilai (nilai dapat berupa angka bilangan real, atau karakter).

   Contoh :

0.12	0.32	0.41	0.44	0.91	0.05	0.37	0.93	0.11

2.4.2. Seleksi

Dalam **Obitko (1998)**, seleksi merupakan proses pemilihan individu, menutut teori evolusi bahwa individu yang terbaik yang bertahan dan dapat membuat keturunan baru.

Ada beberapa contoh seleksi, antara lain:

a. Seleksi *rank*

Seleksi ini dilakukan dengan cara membuat ranking diantara individu, kemudian menghitung nilai *fitness* berdasarkan rangking tersebut dan berikutnya dilakukan seperti seleksi *roulette wheel*.

b. Seleksi *elitism*

Seleksi ini dilakukan dengan cara mengambil individu terbaik (atau beberapa individu terbaik) untuk populasi baru, sisanya dilakukan menggunakan seleksi lain.

c. Seleksi *roulette wheel* (roda rollet)

Dalam **Gen dan Cheng (1997)**, sebuah *roulette wheel* yang ditempati semua individu dalam populasi. Masing – masing individu menempati luasan menurut nilai *fitness*nya. Selanjutnya *roulette wheel* diputar untuk menyeleksi individu. Individu dengan *fitness* yang lebih besar akan lebih besar kesempatannya untuk terpilih.

i. Menghitung nilai *fitness eval* (v_k) masing – masing individu.

$$eval(v_k) = f(x), \quad k = 1,2, \dots \dots , n$$

n : ukuran populasi

$f(x)$: fungsi tujuan

ii. Menghitung fitness total populasi (F)

$$F = \sum_{k=1}^{n} eval(v_k)$$

Menghitung fitness relatif (p_k) masing – masing individu v_k.

$$p_k = \frac{eval(v_k)}{F}, \quad k = 1,2, \dots, n$$

iii. Menghitung fitness kumulatif (q_k) masing – masing individu.

$$q_k = \sum_{j=1}^{k} p_j, \quad k = 1,2, \dots, n$$

Proses pemilihan individu pada *roulette wheel* sebagai berikut:

1. Memunculkan bilangan secara acak r_i, dengan range $[0,1]$; dengan $i = 1,2, \dots, n$.
2. Jika $r \leq q_1$ maka individu pertama yang terpilih.
3. Jika $q_{k-1} < r \leq q_k$, dengan $2 \leq k \leq n$, maka individu ke-k yang terpilih.

2.4.3. *Crossover*

Definisi 2.14 *Crossover* adalah persilangan dua individu yang menghasilkan anak (*offspring*) yang masih mewarisi sebagian sifat induknya.

(**Gen dan Cheng, 1997)**

Pada **Obitko (1998)** dijelaskan bahwa tipe dan bentuk *crossover* bergantung pada pengkodean dan jenis permasalahannya. Ada bermacam-macam teknik *crossover* yang bisa digunakan, yaitu :

1. *Single point crossover*

Satu titik *crossover c* terpilih. Kromosom anak dibentuk dari gabungan penggandaan untaian induk 1 dari gen awal sampai ke titik *c* dan sisanya untaian dari induk 2 setelah titik *c*.

2. *Two point crossover*

Dua titik *crossover* terpilih, rangkaian gen dari awal sampai pada titik pertama kromosom diperoleh dari induk pertama, bagian antara titik pertama dan titik kedua diperoleh dari induk kedua dan sisanya diperoleh dari induk pertama.

3. *Uniform Crossover*

i. Memilih bilangan 0 atau 1 secara acak untuk setiap lokus.

ii. Apabila pada suatu lokus didapatkan bilangan acak 1, maka gen pada lokus tersebut saling dipertukarkan diantara kedua induk untuk mendapatkan anak.

4. *PMX (Partial Mapped Crossover)*

i. Memilih dua lokus secara acak dari untaian. Subuntaian didefinisikan melalui dua lokus tersebut yang namakan bagian *mapping*.

ii. Menukar dua subuntaian diantara induk untuk menghasilkan calon-anak.

iii. Menentukan relasi mapping diantara dua bagian *mapping*.

iv. Membentuk anak berdasarkan relasi *mapping*.

Namun dalam proposal skripsi ini, metode *crossover* yang akan digunakan adalah *Single Point Crossover.*

Untuk mengetahui berapa banyak individu yang akan dihasilkan dari proses *crossover* diberikan laju *crossover* (p_c) yaitu rasio banyaknya keturunan yang dihasilkan dari tiap-tiap generasi terhadap ukuran populasi **(Gen dan Cheng, 1997)**. Nilai laju *crossover* (p_c) direkomendasikan antara 0,8 sampai 0,95. Namun untuk beberapa persoalan *crossover*, $p_c = 0{,}6$ memiliki hasil yang paling baik **(Obitko, 1998)**.

2.4.4. Mutasi

Definisi 2.15 Mutasi adalah proses perubahan sebagian sifat individu sehingga menghasilkan struktur genetika baru.

(**Obitko, 1998)**

Menurut **Gen dan Cheng (1997)**, beberapa jenis mutasi, diantaranya adalah sebagai berikut :

1. *Inversion Mutation*

   Memilih dua posisi dalam kromosom secara acak dan kemudian membalikkan subuntaian antara dua posisi tersebut.

2. *Insertion Mutation*

   Memilih satu gen secara acak, kemudian menyisipkan gen tersebut pada lokus yang dipilih secara acak.

3. *Displacement Mutation*

Memilih subuntaian secara acak, kemudian menyisipkan subuntaian tersebut pada lokus yang dipilih secara acak.

4. *Resiprocal Exchange Mutation*

Memilih dua posisi secara acak dan kemudian menukarkan gen pada kedua posisi tersebut.

5. *Heuristic Mutation*

Mutasi ini dirancang dengan teknik tetangga untuk menghasikan anak yang lebih baik. Tetangga diperoleh dari perubahan kromosom berdasarkan semua kemungkinan permutasi dengan menukarkan gen-gen yang yang terpilih. Solusi yang terbaik di antara semua tetangga digunakan sebagai anak yang dihasilkan oleh mutasi.

Namun dalam proposal skripsi ini, mutasi yang akan digunakan adalah *Resiprocal Exchange Mutation.*

Untuk mengetahui berapa banyak individu yang akan dihasilkan dari proses mutasi diberikan laju mutasi (p_m) yaitu rasio jumlah gen yang dimutasi dari total gen pada suatu populasi **(Gen dan Cheng, 1997)**. Rekomendasi nilai p_m sebesar 0,05 sampai 0,01 memiliki hasil yang paling baik **(Obitko, 1998)**.

2.4.5. Maksimal Iterasi

Dalam penelitian ini kriteria yang digunakan adalah maksimal iterasi (*max_ite*) artinya proses pencarian algoritma genetika dibatasi oleh jumlah generasi/iterasi yang telah ditentukan.

Secara garis besar, algoritma genetika dapat dijabarkan sebagai berikut :

1. [mulai] membangkitkan populasi secara random sebanyak *n* individu.
2. [*fitness*] menilai keandalan setiap individu dalam populasi.
3. [populasi baru] menciptakan populasi baru lewat pengulangan pengoperasian operator genetika berikut sampai populasi baru lengkap.
   a. [seleksi] memilih induk dari populasi sesuai dengan nilai keandalannya (keandalan yang lebih baik, lebih berpeluang untuk terpilih)
   b. [*crossover*] dengan suatu konstanta laju *crossover* (p_c), *crossover* induk untuk membentuk anak (individu baru). Jika tidak ada *crossover* yang dilaksanakan, anak merupakan kopian yang sama dengan induknya.
   c. [mutasi] menggunakan suatu konstanta laju mutasi (p_c), mutasi *lokus* pada masing-masing induk. (*lokus* = posisi gen dalam individu).
   d. [*accepting*] tempat anak pada populasi baru.
4. [mengganti] menggunakan populasi yang baru dibentuk untuk menjalankan algoritma lebih lanjut.
5. [menguji] jika sudah mencapai *Max_ite* atau optimal, berhenti dan diperoleh solusi terbaik dari populasi ini. Jika tidak kembali ke langkah 2.

(Obitko, 1998)

2.5 Hybrid Genetic Algorithms (GA) dan Ant Colony Optimization (ACO)

Algoritma genetika (GA) memiliki kemampuan dalam menghasilkan solusi permasalahan DTSP yang lebih baik dalam setiap generasinya. Sebagai akibat dari penggunaan solusi efektif oleh algoritma semut (ACO), menyelidiki setiap rute yang efisien dalam ruang lingkup permasalahan sampai terdapat solusi yang lebih baik lagi. GA dan ACO memiliki kecepatan dalam mencapai nilai kekonvergenan solusi menuju jarak yang lebih pendek, akan tetapi mereka tidak dapat menyelesaikan masalah nilai konvergen nilai lokal optimal yang belum waktunya sendiri-sendiri, oleh karena itu digunakanlah *hybrid* ACO dengan GA. Untuk mengetahui lebih lanjut tentang *hybrid* ACO dan GA dapat dilihat pada **Lampiran 1**.

2.6 Pemrograman Java

Java banyak digunakan untuk membangun program, dirilis pertama kali pasa tahun 1995 oleh Sun Microsystems. Penciptanya adalah James Gosling. Java merupakan bahasa pemrograman yang didasari oleh OOP (Oriented Object Programing) yaitu merupakan teknik membuat suatu program berdasarkan objek. Salah satu fitur dalam OOP adalah pewarisan. Fitur inilah yang membuat suatu kode yang telah ditulis dalam bentuk kelas sangat mudah untuk diwariskan ke kelas lain guna mendukung sifat *reusable*.

(Kadir, 2012)

Java memiliki JVM (*Java Virtual Machine*) yaitu lingkungan tempat eksekusi program java berlangsung dimana setiap objek saling berinteraksi satu dengan yang lainnya. *Virtual machine* inilah yang menyebabkan java mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar. Namun demikian java tidak terikat oleh lisensi karena java bersifat *open-source* sehingga java merupaka bahasa pemrograman *portable* yang bisa digunakan secara *muti-platform* (Sistem Operasi) dan multi-arsitektur dimana arsitektur java terbagi menjadi tiga bagian,

yaitu:

1. *Java 2 Enterprise Edition* (J2EE) untuk aplikasi berbasis web, aplikasi sistem tersebar dengan beraneka ragam *klien* dengan kompleksitas yang tinggi.
2. *Java 2 Standard Edition* (J2SE) untuk aplikasi standar berbasis dekstop.
3. *Java 2 Mobile Edition* (J2ME) untuk aplikasi *mobile* seperti handphone.

(Utama, 2002)