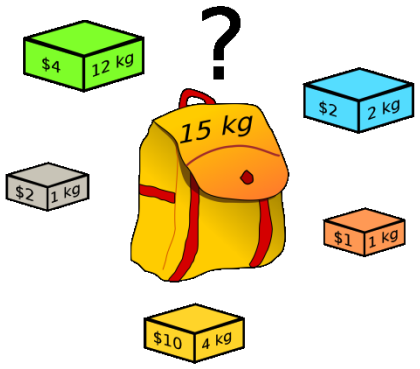
BAB II. TINJAUAN PUSTAKA

2.1 Persoalan *Knapsack*



Gambar 1 Ilustrasi persoalan Knapsack (Wikipedia, 2007)

Secara umum, Persoalan *Knapsack* dikenal sebagai persoalan pemilihan kumpulan optimum dari berat yang terbatas dari beberapa elemen yang memiliki berat dan nilai keuntungan di dalam *knapsack* dengan kapasitas. Pada penelitian ini, penulis menggunakan persoalan *Knapsack* 0-1, yaitu memilih sekumpulan n item, yang memiliki nilai dan berat, dengan jumlah maksimum tanpa melebihi kapasitas. Model matematis persoalan *Knapsack* 0-1 adalah sebagai berikut:

$$max: \sum_{i=1}^{n} p_i x_i \qquad 2.1$$

$$s.t: \sum_{i=1}^{n} w_i x_i \leq c; x_i \in \{0,1\} \qquad 2.2$$

merupakan variabel binari, 1 untuk item *i* terpilih ke dalam knapsak dan 0 sebaliknya (Pisinger, 1995).

Persoalan *Knapsack* pertama kali diteliti oleh Dantzig pada akhir tahun 1950an. Persoalan *Knapsack* telah diteliti sebagai persoalan optimisasi kombinatorial. Selain itu, persoalan *Knapsack* juga termasuk ke dalam persoalan *NP-Hard*, dimana algoritma penyelesaiannya memiliki kompleksitas nonpolinomial dengan skala ruang solusi yang sangat besar dan peningkatan koefisien secara eksponensial (Pisinger, 1995, Martello dkk, 2000).

Persoalan *Knapsack* biasa diaplikasikan pada dunia industri, seperti, *cargo loading*, pemotongan stok, dan lainnya. Persoalan *Knapsack* juga sering terjadi pada kehidupan sehari hari seperti sersoalan bagaimana seorang pendaki dapat mengisi tasnya dengan kebutuhan pendakiannya dengan berat yang minimum dan fungsi yang maksimal, bagaimana seorang pencuri dapat memasukan barang curian ke karungnya dengan berat yang miminal tetapi meminili nilai yang besar, atau bagaimana seorang pengusaha yang memiliki modal sekian rupiah mendapatkan keuntungan dengan investasinya, dan sebagainya.

Pada penelitian ini, penulis menggunakan persoalan *Knapsack* 0-1 sebagai studi kasus. *Knapsack* 0-1 merupakan persoalan pemilihan memilih n item untuk dikumpulkan dengan kapasistas *c* untuk mendapatkan nilai keuntungan yang maksimum tanpa melebihi kapasitas *c* .

Beberapa teknik atau metode telah digunakan untuk menyelesaikan persoalan *Knapsak*, diantaranya adalah *Branch and Bound*, *Dinamic Programming*, state *space relaxations*, *preprosecessing*, GA dan sebagainya.

2.2 Implementasi Heuristik pada Persoalan *Knapsack*

Penelitian persoalan *Knapsack* telah banyak dilakukan untuk menemukan solusi optimal. Beberapa metode telah diaplikasikan. Beberapa penelitian persoalan *Knapsak* yang telah dilakukan diantaranya adalah sebagai berikut:

a. *New a Trends in exact algorithm for the 0-1 Knapsack probelm* (Martello.S., Pisinger.D., dan Toth.P, 2000)

Penelitian ini memberikan gambaran tentang teknik terkini untuk menyelesaikan persoalan *Knapsack*. Hasil komputasi disajikan. Algoritma yang digunakan merupakan algoritma baru kombinasi dari beberapa algoritma lama, seperti *branch and bound, core algorithm* dan *dynamic* programming. Algoritma baru tersebut telah dibangun pada penelitian sebelumnya, yaitu

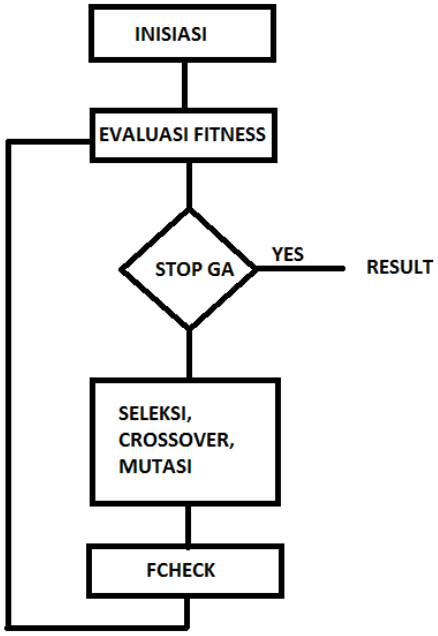
- mt2 (Martello dan Toth, 1990),
- mth, algoritma yang menggunakan *brach and bound* dengan batas atas yang diperoleh dari model matematis dengan menambahkan pertidaksamaan pada kardinalitas solusi optimum dan relaksasi dengan Lagrange (Martello dan Toth 1997),

- minknap, algoritma yang menggunakan *core algorithm* dengan minimalisasi *core* dengan basis pendekatan *dynamic programming* (Pisinger 1997),
- *combo*, yaitu kombinasi algoritma antara *Dinamic Programming* dan *Strong Bounds* (Martello dan Toth 1999).

b. *A Fast and Efficient Genetic Algorithm to Solve 0-1 Knapsack Problem* (Gupta.M, 2013)

Penelitian ini mencoba untuk menyelesaikan persoalan 0-1 *Knapsack* dengan menggunakan algortima genetika. Persoalan *Knapsack* memiliki fungsi batasan sehingga dalam penyelesaian dengan GA digunakan dua fungsi, yaitu fungsi batasan dan fungsi *fitness*. Namun, pada tahapan *crossover* dan mutasi, *offspring* yang dihasilkan dapat memungkinkan solusi yang tidak layak. Untuk mengatasi kendala tersebut, Gupta menyisipkan operator *fcheck* setelah operasi genetik untuk memeriksa apakah *offspring* hasil operasi genetik menghasilkan solusi layak atau tidak (Gupta, 2013)..

Operator *fcheck* dilakulan setelah operator genetika dilakukan, seperti yang ditunjukan pada Gambar 2.



Gambar 2 Flowchart yang digunakan Gupta (Gupta, 2013).

Pseudocode operasi *fcheck* sebagai berikut:

```
For i = 1 to Ukuran Populasi
  Cek solusi;
  If ( tidak layak)
   Ganti dengan solusi layak random
  else
   Lanjut ke populasi selanjutnya
  end
end
```

Penelitian ini membandingkan hasil dari algortima genetika standar dan algoritma dengan fungsi (*Proposed GA*). *Data sample* yang digunakan adalah sebagai berikut.

- Kapasitas knapsak adalah 10.
- Berat dan nilai masing-masing item pada Tabel 1.

Tabel 1 Berat dan nilai item

Item	Value	Weight
A	42	7
B	12	3
C	40	4
D	25	5

Setelah dilakukan percobaan, Gupta mendapatkan hasil perbandingan pada Tabel 2.

Tabel 2 Hasil Penelitian

Poin perbandingan	SGA	PGA
Set	[0 0 1 1]	[0 0 1 1]
Nilai set	65	65
Berat set	9	9
Waktu komputasi	2.91 detik	0.20 detik
iterasi	11	2

c. *Genetic Algorithm based on Greedy Strategy in the 0-1 Knapsack Problem* (Zhao.J, Pang.F, Huang.T, dan Liu.Y, 2009)

Penelitian ini mencoba untuk menggabungkan GA dengan *greedy algorithm*. Strategy greedy pada algoritma greedy digunakan pada tahap inisiasi populasi untuk perkiraan solusi pada kode binari. Algoritma ini baik untuk mengurangi banyak iterasi pada GA.

Penelitian ini menggunakan *Knapsack* dengan berat 50, item sebanyak 50 item dengan berat dan nilai yang berbeda. Probabiliti *crossover* yang dipakai adalah 0.15 dan probaliti mutasi adalah 0.05. Generasi pada algoritma adalah sebanyak 100 generasi.

Hasil penelitian pada GA standar didapatkan rata-rata solusi optimum 3382,24 dengan rata-rata 38 aljabar sedangkan algoritma dengan *greedy strategy* mendapat rata rata solusi 1412,19 dengan rata-rata 27 aljabar (Zhao dkk, 2009).

2.3 *Genetic Algorithm* (GA)

2.3.1 Pengertian

GA merupakan suatu metode pencarian heuristik yang dikembangkan berdasarkan gagasan evolusi seleksi alamiah pada Teori Darwin (Sutojo dkk, 2011 dan Zukhri, 2014). GA ini pertama kali dikenalkan oleh John Holland (1975) pada bukunya yang berjudul "*Adaptation in Natural and Artificial Systems*" (Purnomo, 2014) dan dikembangkan oleh muridnya David Goldberg. Prinsip GA diinspirasi oleh prinsip yang ada pada teori genetika pada ilmu biologi sehingga proses yang terjadi pada GA sama dengan proses yang terjadi pada evolusi biologis (Suyanto, 2005).

GA dibangun oleh dua hal yang penting. Pertama adalah algoritma stokastik. *Random* atau ketidakberaturan merupakan peran utama dalam GA. Pada proses *crossover* dan mutasi memerlukan suatu parameter yang dibangun secara acak. Hal kedua adalah GA selalu memilih atau mempertimbangkan populasi solusi. Pada tiap tahap GA, populasi solusi selalu direkombinasi dan dievaluasi untuk mendapatkan solusi yang lebih baik (Sivanan dan Deepa, 2008).

GA dapat digunakan untuk persoalan optimasi. Sebenarnya, persoalan optimasi tidak hanya dapat diselesaikan dengan GA. Ciri-ciri permasalahan yang dapat diselesaikan dengan GA adalah sebagai berikut:

a. ruang pencarian yang tidak dapat dipahami, sangat besar dan kompleks,

b. pengetahuan yang belum memadai untuk menyederhadakan ruang pencarian,
c. belum adanya analisis matematika yang dapat menyelesaikan persoalan,
d. solusi yang diperlukan tidak harus optimum,
e. solusi diperlukan secepat mungkin (Sutojo dkk, 2011).

Menurut Gen dan Cheng (1997), kelebihan dari GA adalah sebagai berikut:

a. algoritma ini tidak menggunakan perhitungan matematis yang rumit,
b. operasi genetika cocok untuk pencarian global,
c. fleksibelitas GA sangat tinggi untuk digabungkan dengan metode lainnya untuk mengoptimalkan algoritma (Zukhri, 2014 dan Syarif, 2014).

Menurut Goldberg (1989), GA memiliki perbedaan dengan algoritma pencarian yang lain, yaitu:

a. algoritma bekerja pada ruang solusi,
b. algoritma bekerja pada beberapa titik solusi,
c. tidak hanya untuk satu titik solusi,
d. algoritma bekerja berdasarkan fungsi objektif dan algoritma bekerja berdasarkan aturan probalistik (Zukhri, 2014).

Beberapa istilah yang digunakan pada GA adalah sebagi berikut:

a. **gen**, adalah bagian dari komponen representasi kromosom,

b. **kromosom**, adalah representasi kandidat solusi persoalan,

c. **individu**, sama dengan satu kromosom,

d. ***fitness***, adalah nilai kualitas kromosom,

e. **populasi**, adalah himpunan kromosom,

f. **parent atau orang tua**, adalah kromosom terpilih yang melakukan proses reproduksi,

g. **crossover atau persilangan**, adalah proses perkawinan dua kromosom parent,

h. **mutasi**, adalah proses modifikasi gen,

i. dan, ***offspring*** **atau anak**, adalah kromosom yang dihasilkan dari proses reproduksi.

2.3.2 Struktur GA

Secara garis besar, struktur GA adalah sebagai berikut:

```
begin
t  0;
inisiasi P(t);
evalusasi P(t);
 while (bukan kondisi berhenti) do
 begin
  rekombinasi P(t) menjadi C(t);
  evaluasi C(t);
  seleksi P(t);
  t = t + 1;
 end
end
```

(Gen dan Chen, 2000 dan Syarif, 2014).

Struktur Algoritma tersebut dipengaruhi oleh lima komponen dasar algoritma gentika, diantaranya adalah sebagai berikut:

a. Representasi solusi,
b. Cara untuk membuat inisiasi populasi solusi,
c. Fungsi evaluasi untuk mengukur ke-baik-an (*fitmess*) suatu solusi,
d. Operator genetika uang mengganti komposisi gen,
e. dan Nilai parameter (Gen dan Cheng, 2008).

Parameter yang digunakan pada GA diantaranya adalah:

a. ukuran populasi, merupakan banyak kromosom yang ada pada suatu generasi,
b. generasi maksimum, merupakan banyak generasi atau banyak iterasi algoritma yang akan terjadi,
c. probabilitas persilangan, merupakan parameter yang menandakan seberapa sering terjadinya persilangan kromosom, dan
d. probabilitas mutasi, merupakan parameter yang menandakan seberapa sering terjadinya proses mutasi.

a) Pengkodean

Konsep utama yang diambil oleh GA adalah hereditas. Hereditas merupakan penurunan atau pewarisan sifat individu suatu generasi ke generasi berikutnya melalui suatu kode tertentu (DNA). Setiap individu memiliki gen yang berisi kromosom yang memiliki DNA dan memiliki kode dengan urutan dasar tertentu (A, C, G, dan T). Pengkodean seperti ini diperlukan untuk membangun dan menjaga sifat suatu individu. Pada

GA, individu tersebut merupakan suatu solusi suatu fungsi permasalahan. Solusi akan disimpan dalam kromosom yang direpresentasikan atau dikodekan oleh kode tertentu (Suyanto, 2008).

Representasi kromosom diklasifikasikan menjadi tiga klasifikasi, diantaranya:

a. Berdasarkan jenisnya, representasi kromosom dapat diklasifikasi menjadi empat macam, diantaranya adalah
    1) kode binari, yang telah digunakan oleh Holland untuk menyelesaikan permasalahan fungsi optimisasi,
    2) kode bilangan riil, lebih baik dari pada pengkodean Gray,
    3) kode integral atau permutasi, pengkodean ini baik digunakan pada permasalaha persoalan optimasi kombinatorial,
    4) kode struktur data, digunakan untuk mengungkap permasalahan alam.

b. Berdasarkan struktur pengkodean, representasi kromosom dapat dikelompokan menjadi dua tipe, yaitu pengkodean satu dimensi dan multidimensi. Pengkodean satu dimensi sering digunakan. Namun, penggunaan pengkodean secara multidimensial juga sering digunakan, seperti Vignaus dan Michalewicz yang menggunakan matrik pada persoalan transportasi.

c. Berdasarkan isi kode, pengkodean kromosom dibagi menjadi dua macam, yaitu solusi dan solusi bersama dengan parameternya (Gen dan Cheng, 2008).

b) Nilai *Fitness* dan Seleksi

Teori Evolusi Darwin mengatakan bahwa asal mula spesies berdasar pada terpeliharanya variasi baik dan tertolaknya variasi yang tidak baik. Variasi terlihat pada beragamnya beberapa keturunan dengan orang tua yang sama. Individu yang memiliki keuntungan tertentu akan memiliki kesempatan bertahan yang lebih besar (Sivanan dan Deepa, 2008). Pada GA, nilai keuntungan dan proses bertahan tersebut adalah nilai *fitness* dan proses seleksi.

Nilai *fitness* merupakan ukuran nilai performansi dari suatu fungsi. Pada permasalahan optimasi, jika solusi yang diinginkan adalah nilai maksimum, maka nilai *fitness*-nya adalah nilai fungsi tersebut. Namun, apabila solusi yang diinginkan adalah nilai minimum, maka nilai fitnesnya adalah invers dari nilai fungsi itu sendiri. Untuk proses selanjutnya, nilai fitnes akan digunakan untuk proses seleksi. Apabila individu memiliki nilai *fitness* tinggi maka individu tersebut akan bertahan dari proses seleksi. Proses seleksi akan memilih kromosom yang pantas untuk proses selanjtunya, baik itu proses reproduksi maupun proses awal pada generasi yang selanjutnya (Suyanto, 2008 dan Sutojo dkk, 2011).

Proses seleksi dalam GA dapat dilakukan dalam beberapa metode, diantaranya adalah

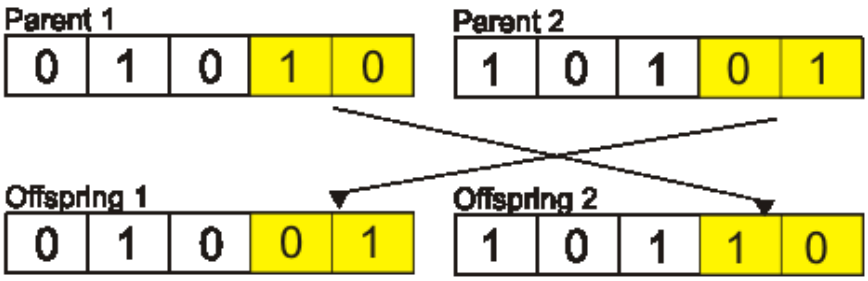
a. seleksi *roulette wheel*, sama seperti *roulette wheel* yang sebenarnya. Namun, luas daerah ditentukan oleh besar nilai firnes. Nilai *fitness* besar akan memiliki luas daerah yang besar
b. seleksi ($\mu + \lambda$) , yang dilakukan dengan memilih μ kromosom terbaik pada populasi μ orang tua dan λ *offspring*,
c. seleksi turnamen,
d. *steady state reproduction*,
e. perangkingan dan penskalaan, seleksi yang dilakukan untuk menghindari kendala pada metode *roulette wheel*,
f. dan sharing (Gen dan Cheng, 2000).

c) Operasi Rekombinasi Genetika

a. Crossover

Pada proses evolusi, individu akan bereproduksi untuk melestarikan sifatnya. Salah satu proses reproduksi adalah reproduksi secara seksual. Reproduksi seksual dilakukan dengan cara mengkombinasi dua kromosom individu untuk menghasilkan individu baru. Proses pengkombinasian ini disebut juga sebagai proses crossover. GA juga melakukan proses *crossover*. Sama dengan teori evolusi, tujuan crossover pada GA adalah untuk meneruskan sifat baik orang tua ke generasi selanjutnya. Metode crossover diantaranya adalah *N Point Crossover*, *Partial Mapped Crossover*, *Ordered-based Crossover*,

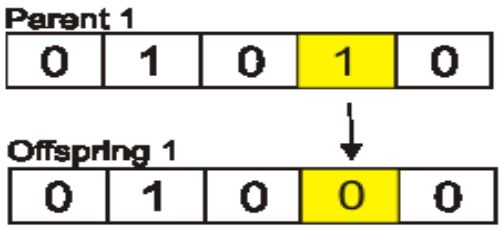
Position-based Crossover, *Cycle Crossover*, *Weight Mapping Crossover* (Suyanto, 2005 dan Syarif, 2014).



Gambar 3 Ilustrasi One-Point Crossover

b. Mutasi

Dalam proses perkembangan suatu individu akan terjadi mutasi, yaitu perubahan rangkaian DNA yang disebabkan oleh mutagen sehingga variasi individu semakin beragam. Pada GA, mutasi dilakukan dengan beberapa metode diantaranya adalah pembalikan, penyisipan, pemindahan, penukaran, dan penggantian (Syarif, 2014).

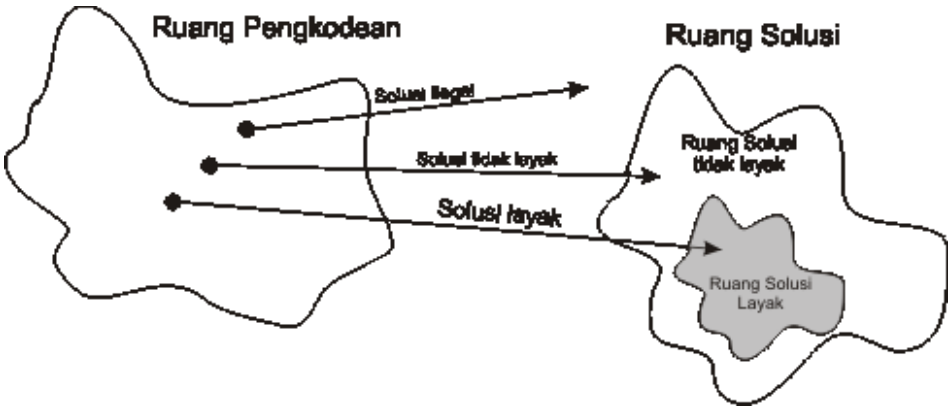


Gambar 4 Ilustrasi Flip-Mutation

d) Evaluasi

GA bekerja pada dua ruang, yaitu, ruang pengkodean dan ruang solusi. Dalam teori evolusi, kedua ruang ini disetarakan dengan istilah genotip dan fenotip. Genotip mencakup operator genetika, yaitu crossover dan mutasi, dan fenotip mencakup evaluasi dan seleksi. Pada tahap seleksi, nilai *fitness* digunakan untuk mengukur kelayakan suatu kromosom atau individu.

Namun, beberapa persoalan optimasi memiliki fungsi pembatas. Fungsi pembatas tersebut akan membagi kromosom menjadi beberapa bagian, yaitu kromosom layak, kromosom tidak layak, dan kromosom ilegal. Kromosom layak, seperti keterangan sebelumnya, merupakan kromosom yang memenuhi fungsi persoalan. Kromosom tidak layak merupakan kromosom yang berada di luar ruang kelayakan.



Gambar 5 Ruang Kelayakan Solusi

Kromosom yang tidak layak biasanya ditemukan pada persoalan optimasi berkendala yang direpresentasikan dengan persamaan atau tidak persamaan. Kendala pada persoalan membuat ruang tidak layak. Pada persoalan optimisasi berkendala, solusi maksimum biasanya berada diantara ruang solusi layak dan tidak layak. Kromosom ilegal merupakan kromosom yang tidak merepresentasikan solusi persoalan. Kromosom ilegal biasanya bermula dari tenik operasi gentika (crossover atau mutasi) yang digunakan. Operasi genetika terkadang akan menghasilkan *offspring* yang ilegal (Gen dan Cheng, 2000). Dengan adanya kromosom yang tidak layak dan ilegal, populasi kromosom perlu dilakukan evaluasi dan perbaikan.

Ada beberapa metode yang digunakan untuk menangani fungsi pembatas tersebut. Metode tersebut diantaranya adalah sebagai berikut:

a. penolakan kromosom (*Rejecting*),

   metode ini akan membuang kromosom yang tidak layak. Kelemahan dari metode ini adalah saat individu yang dihasilkan pada generasi pertama tidak layak semua, maka semua individu akan tertolak semua (Zukri, 2014),

b. perbaikan kromosom (*Repairing*),

   metode perbaikan akan membuat prosedur perbaikan untuk kromosom yang tidak layak atau ilegal. Metode perbaikan

kromosom diklasifikasikan menjadi dua, yaitu prosedur yang hanya mengevaluasi kromosomnya tanpa merubah dan prosedur yang akan merubah kromosom menjadi kromosom layak. Letak kesulitan metode ini adalah membuat prosedur perbaikannya itu sendiri. Metode perbaikan biasa digunakan pada persoalan kombinatorial yang menghasilkan *offspring* ilegal. Metode akan mempertahankan kromosom yang ilegal dan merubah kromosom tersebut tersebut menjadi kromosom legal. Orvosh dan Davis (1994) telah membuktikan banyak penyelesaian persoalan kombinatorik relatif lebih mudah dengan perbaikan kromosom tidak layak atau ilegal dan metode perbaikan kromosom ini lebih baik dari metode penolakan dan pemberian penalti (Syarif, 2014 dan Zukri, 2014),

c. modifikasi operator genetika.

   Metode ini akan menyediakan representasi kromosom khusus atau operasi genetika yang mampu menghasilkan kromosom yang layak. Metode ini akan selalu menjaga kromosom pada ruang solusi layak (Syarif, 2014),

d. pemberian penalti (*Penalty strategy*),

   metode dengan pemberian nilai penalti menganggap semua kromosom yang telah dibangkitkan akan memberi solusi layak. Metode ini biasanya diaplikasikan pada persoalan optimasi berkendala. Beberapa persoalan optimasi berkendala akam

menghasilkan ruang tidak layak yang besar sehingga apabila pencarian hanya pada ruang layak, maka ruang pencarian akan menjadi terbatas. Oleh karena itu, ruang pencarian metode ini berada pada ruang tidak layak. Ada dua teknik untuk menghitung nilai fungsi evaluasi untuk metode pinalti, yaitu, teknik penambahan nilai pinalti pada fungsi tujuan atau objektif dan teknik pengalian fungsi tujuan dengan fungsi penalti (Syarif, 2014).

2.4 Bahasa Pemrograman Matlab

MATLAB (*Matrix Laboratory)* merupaka sebuah bahasa pemrograman generasi empat. Bahas pembrograman ini sering digunakan dalam perhitungan matematika berbasis matrik. Bahasa pemrograman ini dikembangkan oleh Cleve Moler bersama dengan beberapa temannya seperti Jack Little, Steve Bangert. Awalnya apada athun 1980an, Matlab dibuat oleh Cleve hanya untuk mempermudah mahasiswanya untuk mengakses LINPACK dan EISPACK tanpa harus mempelajari fortran terlebih dahulu. Karyanya tersebut banyak digunakan oleh matematikawan terapan. Oleh karena itu, dalam perkembangannya, MATLAB sangatlah cepat. Tahun 1983, MATLAB ditulis ulang dalam bahasa pemrograman C. Selama perkembangannya, Mereka menambahkan beberapa fungsi ke dalam MATLAB, seperti Simulink, *color grapghic*, *sparse matrices.* Selain itu, MATLAB juga dikembangkan untuk berbagai *platform* sistem operasi,

seperti UNIX, Windows, Linux. Sesuai dengan namanya, MATLAB bekerja menggunakan konsep matrik sebagai variabel utamanya. Kelebihan utama MATLAB adalah pengguna dapat melakukan analisis data dengan fasilitas Simulink.

Pemrograman MATLAB dapat dilakukan dengan dua cara, yaitu *Command Window* dan editor. Pada *command* window, perintah akan dieksekusi secara langsung baris perbaris sedangkan pada editor, pemrograman akan dilakukan sama seperti bahasa pemrograman lainnya. Urutan kode program akan dieksekusi dalam sebuah .file kode. Ektensi file bahasa pemrograman MATLAB adalah **.m**.