BAB II

TINJAUAN PUSTAKA

2.1. *Knapsack Problem*

Knapsack Problem merupakan permasalahan optimasi kombinatorik dengan memaksimalkan *profit* dari *item* didalam *knapsack* (karung) tanpa melebihi kapasitasnya. *Knapsack problem* dapat diilustrasikan sebagai berikut. Diberikan beberapa barang yang masing-masing memiliki berat (*weight*) dan keuntungan (*profit*), dengan ketentuan barang yang dimasukkan kedalam *Knapsack* memiliki total berat barang yang jika dimasukkan tidak boleh melebihi kapasitas *Knapsack*, maka dapat dikatakan telah mencapai *optimal packing*.

Sebagai contoh, terdapat 8 macam *item*. Masing-masing *weight* dan *profit* dapat dilihat pada Tabel 2.1., kapasitas = 10.

Tabel 2.1. *Weight* dan *Profit* untuk *Knapsack Problem* (Singh, 2011).

Item	1	2	3	4
p_j	42	12	40	25
w_j	7	3	4	5

Terdapat beberapa macam tipe *knapsack problem* (Martello, 1990), diantaranya adalah:

2.1.1. *0/1 Knapsack Problem* (0/1 KP)

0/1 atau binary, *Knapsack problem* adalah diberikan satu set *n item* dan satu *Knapsack* dengan:

p_i = *profit* dari *item* ke-*i*

w_i = *weight* dari *item* ke-*i*

c = kapasitas *Knapsack*

Pilih sebagian dari *item* tersebut sehingga memenuhi persamaan berikut:

$$\textbf{max} : \quad z = \sum_{i=1}^{n} p_i x_i \tag{2.1}$$

$$\textbf{s.t} \quad : \quad \sum_{i=1}^{n} w_i x_i \le c \tag{2.2}$$

$$x_i = \ 0 \text{ or } 1, \quad i \in N = (1,...,n) \tag{2.3}$$

$$x_i = \begin{cases} 1; \text{ jika item ke-}i \text{ terpilih} \\ 0; \text{ lainnya} \end{cases}$$

2.1.2. *Bounded Knapsack Problem (BKP)*

Diberikan tipe-tipe *n item* dan *Knapsack*, dengan:

p_i = *profit* dari *item i*

w_i = *weight* dari *item i*

b_i = batas atas yang tersedia dari *item* tipe *i*

c = kapasitas dari *Knapsack*

Pilih sejumlah $x_i(i=1,...,n)$ dari masing-masing tipe *item* sehingga:

$$\textbf{max}: \quad z=\sum_{i=1}^{n} p_i x_i \tag{2.4}$$

$$\textbf{s.t} \quad : \quad \sum_{i=1}^{n} w_i x_i \le c \tag{2.5}$$

$$0 \le x_i \le b_i \quad \text{dan integer,} \quad i \in N=(1,...,n) \tag{2.6}$$

2.1.3. 0-1 *Multiple Knapsack Problem* (MKP)

Diketahui *item* sejumlah *n* dan *Knapsack* sejumlah m ($m \le n$), dengan

p_i = *profit* dari *item i*

w_i = *weight* dari *item i*

c = kapasitas *Knapsack*

$$\textbf{max}: \quad z=\sum_{i=1}^{n}\sum_{j=1}^{m} p_i x_{ij} \tag{2.7}$$

$$\textbf{s.t} \quad : \quad \sum_{i=1}^{n} w_i x_{ij} \le c_i, \quad j \in M=\{1,...,m\} \tag{2.8}$$

$$\sum_{j=1}^{m} x_{ij} \le 1, \quad i \in N=\{1,...,n\}$$
$$x_{ij} = \begin{cases} 1; \text{ jika item ke-}i \text{ terpilih ke Knapsack } j \\ 0; \text{ lainnya} \end{cases} \tag{2.9}$$

2.2. Implementasi Heuristik untuk *Knapsack*

Penyelesaian *Knapsack Problem* secara umum dikelompokkan menjadi dua, yaitu solusi eksak dan solusi heuristik. Solusi eksak dikembangkan menggunakan algoritma yang berbasis pada matematika yang sangat kompleks atau bersifat matematis. Algoritma ini dapat bekerja pada data yang berskala kecil. Contohnya

G*reedy Algorithm*, D*ynamic Programming, Branch and Bound,* dll. Namun solusi heuristik menunjukkan beberapa kemungkinan solusi terbaik . Contohnya GA.

Saat ini telah banyak penelitian-penelitian terkait *Knapsack problem*, tidak hanya menggunakan algoritma eksak namun juga algoritma heuristik. Berikut diantaranya hasil penelitian yang cukup dikenal tentang *Knapsack problem.*

Martello *et al* (2000) melaporkan beberapa algoritma eksak untuk menangani *0-1 Knapsack problem*. Algoritma tersebut yaitu *Branch-and-bound*, *Dynamic programming*, *Core algorithms* dan *Tighter bounds*. Dalam penelitian lainnya, Martello *et al* mengkombinasikan *dynamic programming* dengan *tight bounds* serta menghasilkan suatu algoritma yang diberi nama *combo*. Penelitian dilakukan dengan membangkitkan beberapa data, kemudian data dikelompokkan kedalam tujuh tipe data. Hasil penelitian Martello *et al* melaporkan bahwa dari keempat algoritma yang dibandingkan, *combo* merupakan algoritma yang mampu memecahkan permasalahan dalam berbagai tipe dan pada *range* data kecil maupun besar.

Zhao *et al* (2009) menghadirkan cara baru untuk menyelesaikan 0-1 *Knapsack problem*. GA dan strategi *greedy* dikombinasikan dengan tujuan agar mengurangi waktu penyelesaian serta untuk memperbaiki akurasi solusi yang dihasilkan dengan mengurangi iterasi.

Singh (2011) menjelaskan tentang implementasi dari *Knapsack problem* menggunakan GA. Tujuan utama dari paper ini adalah untuk mengimplementasikan *Knapsack problem* menggunakan suatu algoritma yaitu G*enetic Algorithm*. *Knapsack problem* diselesaikan dengan menggunakan tiga metode seleksi yaitu *Roulette-Wheel, Tournament Selection* dan *Stochastic Selection*. Singh berhasil membuktikan bahwa GA mampu menangani NP *Complete Knapsack problem* dengan menggunakan tiga metode seleksi tersebut.

2.3. *Genetic Algorithm* (GA)

GA dikenal sebagai salah satu metode pencarian heuristik yang dikembangkan berdasarkan prinsip genetika dan proses seleksi alamiah Teori Darwin. Proses pencarian penyelesaian atau proses terpilihnya suatu penyelesaian dalam algoritma ini berlangsung sama seperti terpilihnya suatu individu untuk bertahan hidup dalam proses evolusi.

Pencarian dalam GA bekerja dengan sekumpulan kandidat solusi *n* (kromosom) yang dikenal dengan istilah populasi (*population*). Masing-masing kromosom terdiri dari sejumlah gen yang merepresentasikan suatu beberapa solusi (*feasible / infeasible*) untuk persoalan yang akan diselesaikan.

Struktur umum GA yang diperkenalkan oleh Gen & Cheng (2000) ditunjukkan pada prosedur berikut.

```
Prosedur: Genetic Algorithm
begin
initialize P(t);
evaluate P(t);
while (not termination condition) do
begin
   recombine P(t) to yield C(t);
   evaluate C(t);
   select P(t + 1) from P(t) and C(t);
   t ← t + 1;
   end
end
```

Menurut Goldberg (1989), GA melakukan pencarian terhadap solusi optimal dengan empat cara sebagai berikut:

1. GA bekerja dengan proses *coding* dari parameter.
2. Proses pencarian dalam GA menggunakan sekumpulan kandidat solusi (kromosom).
3. Fungsi tujuan merupakan informasi penting dalam GA dan bukan fungsi turunan dan sebagainya.
4. Aturan probabilitas digunakan didalam GA.

GA dikenal sebagai suatu metode yang berhasil memecahkan permasalahan pencarian optimasi, dengan begitu GA tentunya memiliki beberapa struktur dasar pembentuk. Menurut Haupt dan Haupt (2004), struktur dasar GA terdiri atas beberapa langkah seperti berikut.

1. Inisialisasi populasi
2. Evaluasi populasi
3. Seleksi populasi yang akan dikenai operator genetika
4. Proses penyilangan pasangan kromosom tertentu
5. Proses mutasi pada kromosom tertentu
6. Evaluasi populasi yang baru
7. Ulangi dari langkah 3 selama syarat berhenti belum terpenuhi.

Beberapa istilah-istilah penting dalam GA ditunjukkan pada Tabel 2.1. Istilah-istilah ini merupakan hasil pemetaan proses alamiah kedalam proses komputasi.

Tabel 2.2. Pemetaan Proses Alamiah ke dalam GA (Syarif, 2014).

Individu / Kromosom	Kandidat solusi dari masalah yang akan diselesaikan
Populasi	Himpunan dari kromosom
Gen	Bagian dari kromosom (satu kromosom terdari berapa gen)
Parent	Kromosom yang terpilih untuk proses reproduksi (baik itu *crossover* maupun *mutation*)
Offspring	Keturunan dari hasil reproduksi
Fitness	Suatu nilai yang melambangkan kualitas suatu kromosom
Crossover	Proses reproduksi yang dilakukan dengan proses perkawinan silang antara dua kromosom
Mutation	Proses reproduksi yang dilakukan dengan memodifikasi gen yang ada didalam kromosom.

2.3.1. Inisialisasi Populasi

Proses awal GA adalah inisialisasi populasi. Proses inisialisasi merupakan proses yang paling penting dalam GA, karena pada proses ini semua titik-titik yang mungkin menjadi solusi optimal akan dijadikan sebagai kandidat solusi atau disebut kromosom (Zukhri, 2013). Selain itu, proses ini sangat mempengaruhi efektivitas dan efesiensi GA.

Langkah awalnya ialah dengan memilih representasi kromosom berdasarkan pada masalah yang dihadapi. Hal ini lah yang menjadikan proses ini sulit, karena saat

menemukan masalah yang berbeda dibutuhkan representasi kromosom yang berbeda pula. Representasi kromosom pada satu masalah belum tentu cocok untuk masalah yang lainnya. Jika pembangkitan kromosom secara acak maka langkah selanjutnya bangkitkan bilangan acak (*random*) sebanyak ukuran populasi yang diinginkan dan sebanyak gen yang dibutuhkan untuk memenuhi satu kromosom. Beberapa model representasi (*encoding*) kromosom yaitu *binary encoding, permutation encoding, value encoding, tree encoding* dan *matrix encoding*.

Tabel 2.3. Kromosom pada *Knapsack Problem*.

Kromosom	001101001								
Gen	0	0	1	1	0	1	0	0	1

Pada *knapsack problem*, kromosom menggambarkan sekelompok *item-item* yang mungkin dapat dimasukkan kedalam *knapsack*. Sedangkan gen yang berisi bit 1 pada urutan ke-*i* menggambarkan *item* ke-*i* dimasukkan kedalam *knapsack*.

2.3.2. Evaluasi

Setiap kromosom yang dibangkitkan secara acak oleh sistem komputer memiliki sifat layak dan tidak layak. Jika kromosom dibangkitkan diarahkan keruang solusi yang layak maka otomatis semua kromosom awal adalah kromosom yang layak. Namun akibat dari proses reproduksi (*crossover* dan *mutation*) kromosom yang dihasilkan kemungkinan menjadi tidak layak. Maka dari itu, dilakukan evaluasi terhadap setiap kromosom.

Knapsack problem merupakan salah satu permasalahan yang dapat memiliki satu atau lebih kendala (*constraint*) seperti pada persamaan 2.2. Maka dari itu

diperlukan cara khusus untuk mengatasi hal ini. Ada beberapa cara yang dapat digunakan untuk mengatasi kendala pada permasalahan *knapsack*, yaitu dengan strategi penalti ataupun dengan strategi perbaikan. Pada penelitian ini digunakan metode strategi penalti.

Sebelum suatu kromosom diberikan nilai penalti, kromosom tersebut akan dihitung fungsi tujuannya (pers. 2.1.), agar dapat diketahui apakah kromosom ini layak atau tidak. Jika diketahui bahwa kromosom tersebut tidak layak maka nilai fungsi tujuannya akan diberikan nilai penalti. Hasil dari perhitungan fungsi tujuan juga disebut *fitness value*. *Fitness value* menunjukkan kualitas kromosom dalam populasi. Semakin besar *fitness value* maka semakin besar pula kemungkinannya untuk dipertahankan kedalam populasi selanjutnya (Zukhri, 2014).

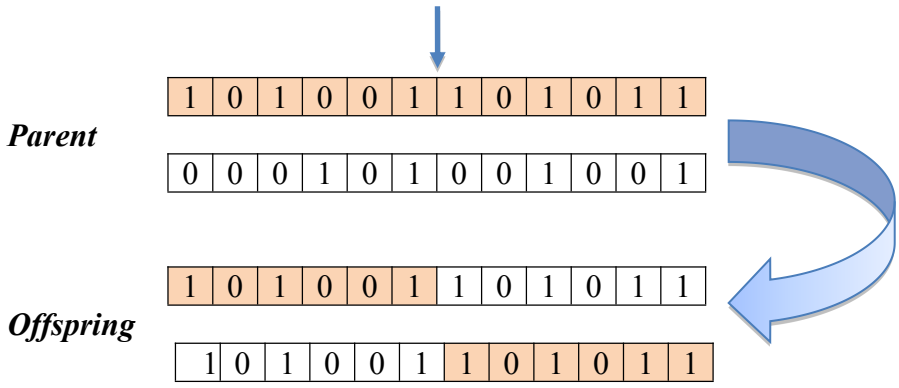
2.3.3. Seleksi

Proses selanjutnya dalam GA adalah proses seleksi. Proses dimana dilakukan seleksi pada masing-masing kandidat solusi (kromosom). Penyeleksian ini dilakukan agar hanya kromosom terbaiklah yang mampu bertahan hingga ke generasi selanjutnya. Beberapa metode seleksi yang terkenal ialah *Roulette Wheel Selection*, *Rank Base Selection, Elitist* dan *(μ + λ) selection* (Syarif, 2014). Metode-metode ini digunakan sesuai permasalahan yang akan diselesaikan agar algoritma lebih efektif.

Pada proses ini individu yang diseleksi berdasarkan pada *fitness value.* Individu-individu yang lolos membentuk generasi baru sebanyak populasi awal

(Singh, 2011). *Fitness value* merepresentasikan nilai pada masing-masing kromosom. Nilai inilah yang akan menjadi tolak ukur suatu kromosom dapat bertahan hingga ke generasi berikutnya.

2.3.4. Pindah Silang (*Crossover*)



Gambar 2.1. Metode *One-point Crossover*.

Crossover merupakan proses reproduksi dalam genetika. Proses ini dibentuk atas dua kromosom yang kemudian disilangkan sehingga melahirkan dua individu baru (*offspring*) dari kombinasi kedua kromosom tersebut (Zhao, 2009). Langkah sederhana untuk mencapai *crossover* yaitu dengan memilih titik potong secara acak dan membangkitkan *offspring* dengan mengkombinasikan bagian dari salah satu *parent* ke sebelah kiri titik potong dengan bagian *parent* yang lain ke sebelah kanan titik potong. Metode ini dinamakan O*ne-point Crossover*. Metode ini bekerja dengan baik pada representasi biner.

Probabilitas *crossover* dinotasikan dengan pC. Probabilitas *crossover* merupakan nilai yang mendeskripsikan seberapa besar kemungkinan kromosom tersebut akan mengalami *crossover*. Selain itu, pC menentukan seberapa banyak

kromosom yang di silangkan dalam satu populasi. Jika nilai pC bernilai 0 maka dapat disimpulkan tidak akan terjadi *crossover*. Biasanya pC dibuat bernilai 0<pC<1. Beberapa metode crossover yang terkenal ialah O*ne-point Crossover*, T*wo-point Crossover*, P*artial Mapped Crossover* (PMX), O*rder Crossover* (OX), P*osition-based Crossover* (PX), O*rder-based Crossover*, C*ycle Crossover* (CX) dan W*eight Mapping Crossover* (WMX).

2.3.5. Mutasi

Sama halnya mutasi gen didalam genetika. Mutasi didalam GA juga ditandai dengan mengubah satu atau lebih gen yang ada pada kromosom sehingga terbentuk individu baru (Zhao, 2009). Gen yang dipilih untuk dimutasi berdasarkan pada probabilitas mutasi yang diharapkan. Probalitas mutasi dinotasikan dengan pM. Jika pM bernilai 0,10 maka berarti hanya 10% dari kromosom yang akan mengalami mutasi. Beberapa metode yang sering digunakan ialah *Flip Mutation* (Metode Penggantian)*, Swap Mutation* (Metode Penukaran)*, Inversion Mutation* (Metode Pembalikan)*, Insertion Mutation* (Metode Penyisipan) dan *Displacement Mutation* (Metode Pemindahan).

Metode sederhana yang paling sering digunakan pada permasalahan yang menggunakan representasi biner adalah *flip mutation*. Metode ini dilakukan dengan mengubah gen terpilih pada kromosom. Jika gen yang terpilih berisi bit 0 maka gen diubah menjadi bit 1, dan sebaliknya. Gambar 2.2. merupakan ilustrasi metode *flip mutation*.

Parent **:**

1	0	1	0	0	1	1	0	1	0	1	1

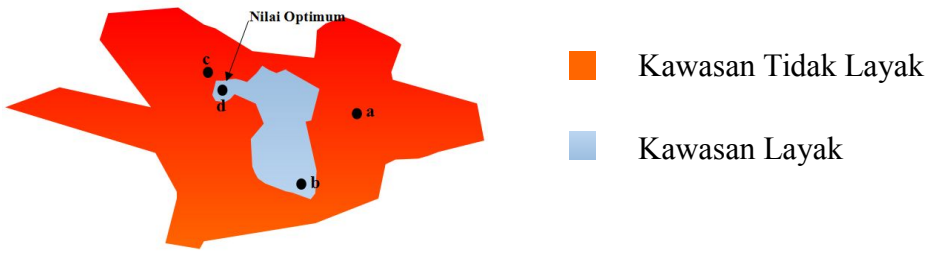
Offspring :

1	0	1	0	0	0	1	0	1	0	1	1

Gambar 2.2. Metode *Flip Mutation*.

2.4. *Penalty Strategy*

Strategi penalti merupakan salah satu teknik penanganan kendala dengan menghadirkan fungsi penalti.



Gambar 2.3. Kelayakan Solusi Persoalan Optimasi.

Pada Gambar 2.3. terlihat bahwa titik c yang berada pada kawasan yang tidak layak justru lebih dekat dengan titik d (titik d adalah nilai optimum). Hal ini menjelaskan bahwa keturunan yang tidak layak juga memiliki informasi yang berharga dalam proses pencarian untuk menuju ke titik optimum.

Fungsi penalti dapat ditambahkan dalam fungsi evaluasi menggunakan dua acara, yaitu:

1) Metode Penambahan

Penjumlahan antara fungsi tujuan dan fungsi penalti akan menghasilkan fungsi evaluasi, seperti berikut ini.

$$eval(x) = f(x) + p(x) \tag{2.12}$$

merupakan fungsi tujuan awal, jika kromosom tidak memenuhi fungsi pembatas maka nilai fungsi evaluasi ditambah dengan fungsi penalti $p(x)$. Terlihat jelas bahwa nilai fungsi evaluasi akan bernilai sama dengan nilai fungsi tujuan jika nilai dari $p(x)$ bernilai 0. Artinya kromosom tersebut memenuhi fungsi pembatas.

2) Metode Perkalian

Metode lain yang bisa digunakan adalah dengan mengalikan fungsi tujuan dengan fungsi penalti sebagai berikut.

$$eval(x) = f(x) * p(x) \tag{2.13}$$

Berbeda dengan metode sebelumnya, pada metode perkalian jika kromosom memenuhi pembatas (*feasible*) maka $p(x) = 1$. Nilai fungsi penalti untuk persoalan maksimasi haruslah $0 \le p(x) < 1$. Sebaliknya, untuk persoalan minimasi, nilai fungsi penalti haruslah $\sum_{i=1}^{n} w_i$.

Menurut laporan Anagun dan Sarac (2006), penambahan fungsi penalti pada masing-masing kromosom bernilai sama. Berikut merupakan prosedur penalti dari hasil penelitian tersebut.

Prosedur penalti :

$a = \sum_{i=1}^{n} w_i x_i$

if $a > c$ **then**

```
begin
```

$$p = 1 - \left(\frac{a}{5 * c} \right);$$

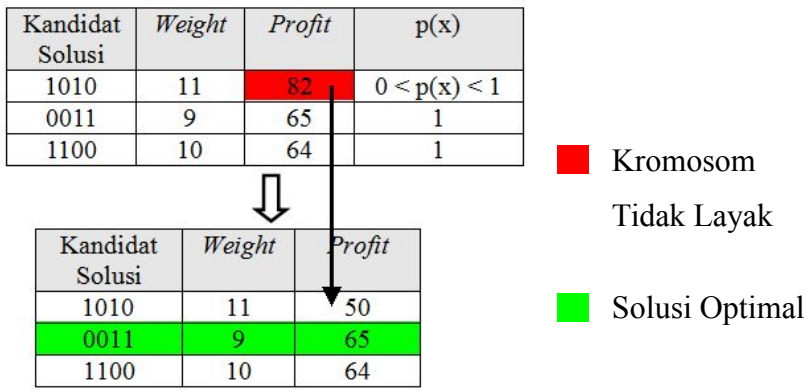
```
if p <= 0 then p:= 0.00001;
fitness value of string i= fitness value of string i*p;
end
```

Beberapa literatur seperti yang disebutkan oleh Olsen (1994) menjelaskan fungsi penalti memiliki kadar yang berbeda untuk setiap kromosom.

$$\delta = \min\left\{ c, \left| \sum_{i=1}^{n} w_i - c \right| \right\} \tag{2.14}$$

$$p(x) = 1 - \frac{\left| \sum_{i=1}^{n} w_i x_i - c \right|}{\delta} \tag{2.15}$$

Teknik penentuan fungsi penalti (Gen dan Yu, 2010) diawali dengan mendefinisikan skala penalti seperti pada persamaan 2.14. Langkah pembangkitan penalti dijelaskan pada Gambar 2.4. (Data lihat ditabel 2.1.)



Gambar 2.4. Ilustrasi Pembangkitan Nilai Penalti.

Pada Gambar 2.4. kandidat solusi (kromosom) 0011 dan 1100 adalah kromosom yang layak, karena kromosom tersebut memenuhi fungsi kendalanya (pers. 2.2),

namun kedua kromosom tersebut memiliki *profit* yang lebih kecil dibandingkan kromosom 1010. Sehingga ada kemungkinan kromosom 1010 yang akan terpilih sebagai solusi optimal meskipun kromosom ini tidak memenuhi kendala. Maka dari itu, untuk menghindari terjadinya permasalahan ini diberikan nilai penalti terhadap kromosom 1010 sehingga *profit*-nya turun menjadi 50. Karena $50 < 64 < 65$, maka dapat diketahui bahwa solusi optimal diperoleh pada kromosom 0011 dengan total *weight* = 9 dengan *profit* = 65.