PERMASALAHAN OPTIMASI 0-1 KNAPSACK DAN PERBANDINGAN BEBERAPA ALGORITMA PEMECAHANNYA

Fitriana Passa (13508036)

Program Studi Teknik Informatika
Institut Teknologi Bandungg
Jl. Ganesha 10 Bandung
Email: if18036@students.if.itb.ac.id

ABSTRAK

Knapsack adalah permasalahan mengenai optimasi dalam pemilihan barang dengan pembatasan kuota maksimum yang dapat ditampung. Pemilihan barang didasarkan pada kombinasi barang yang akan menghasilkan nilai tertinggi dan masih memenuhi batasan kuota. Ada beberapa cara pemecahan masalah Knapsack, diantaranya menggunakan strategi algoritma brute force, cara matematika, algoritma genetik, algoritma greedy, dan pemrograman dinamis. Cara yang berbeda akan menyebabkan perbedaan proses pencapaian optimasi. Selain harus memperhatikan kemungkinan kebocoran kasus menggunakan salah satu strategi algoritma, kompleksitas algoritma yang digunakan juga menjadi pertimbangan lain bagi pemilihan algoritma untuk penyelesaian persoalan Knapsack. Cara yang biasa digunakan adalah dengan penggunaan strategi algoritma Greedy dan Dynamic Programming (DP). Kedua cara tersebut memiliki kelebihan dan kekurangan masing-masing dalam memecahkan persoalan yang berhubungan dengan Optimalisasi, dalam makalah ini khususnya adalah permasalahan Knapsack.

Kata Kunci: Knapsack, brute force, algoritma genetik, algoritma greedy, pemrograman dinamis, kompleksitas.

1. PENDAHULUAN

Optimalisasi merupakan salah satu masalah klasik yang dihadapi dalam berbagai bidang ilmu, termasuk di bidang matematika maupun informatika. Banyak kasus yang membutuhkan optimalisasi yang baik dari berbagai kombinasi input yang mungkin diberikan untuk mendapatkan efisiensi dalam prosesnya serta menghasilkan keluaran yang benar-benar optimal. Di antara permasalahan yang ada, masalah

Beberapa contoh kasus yang termasuk dalam kasus optimasi diantaranya adalah masalah TSP *(Travelling Salesman Problem)*, MST *(Minimum Spanning Tree)*, dan Knapsack Problem. Beberapa kasus diatas merupakan kasus yang membutuhkan teknik optimasi dalam algoritmanya. Beberapa metode algoritma yang dipakai dalam menyelesaikan beberapa kasus optimasi diantaranya metode *Brute force*, *Greddy*, *Dynamic Programming* dan *Monte Carlo*. Setiap metode memiliki sifat dan ciri yang berbeda-beda.

Knapsack merupakan salah satu permasalahan yang dapat diselesaikan oleh beberapa cara yang berbeda. Knapsack merupakan salah satu permasalahan optimalisasi yang sering dihadapi. Logika knapsack banyak digunakan, tetapi sering kali banyak orang yang belum menyadarinya.

Masalah Knapsack merupakan suatu persoalan yang menarik. Dalam dunia nyata, permasalahan Knapsack ini sering sekali digunakan terutama pada bidang (jasa) pengangkutan barang (seperti pengangkutan peti kemas dalam sebuah kapal). Dalam usaha tersebut, diinginkan suatu keuntungan yang maksimal untuk mengangkut barang yang ada dengan tidak melebihi batas kapasitas yang ada. Berdasarkan persoalan tersebut, diharapkan ada suatu solusi yang secara otomatis dalam mengatasi persoalan itu. Contoh kasus lain yang menggunakan algoritma ini antara lain pengisian barang di bagasi, pengisian barang di suatu perusahaan, pengoptimalisasi karyawan dalam suatu badan usaha.

Pemilihan algoritma dalam penyelesaian berbagai kasus ada sangat penting. Penggunaan algoritma yang tepat akan membantu penyelesaikan kasus Knapsack dengan baik. Sebaliknya, ketidaktepatan memilih salah satu algoritma optimalisasi akan menyebabkan terhambatnya proses pengambilan keputusan, yang dalam makalah ini dikhususkan pada permasalahan Knapsack.

2. PEMBAHASAN

2.1. Knapsack

Knapsack adalah permasalahan mengenai optimalisasi kombinatorial. Knapsack merupakan suatu permasalahan bagaimana memilih objek dari sekian banyak dan berapa besar objek tersebut akan disimpan sehingga diperoleh suatu penyimpanan yang optimal dengan memperhatikan objek yang terdiri dari n objek (1,2,3,…,n) dimana setiap objek memiliki bobot (w_i) dan Nilai profit (p_i) dengan memperhatikan juga kapasitas dari media penyimpanan sebesar W dan nilai probabilitas dari setiap objek (X_i). Kita diberikan suatu set barang dengan masing-masing barang mempunyai nilai dan harga yang berbeda dan kita harus menebak jumlah barang yang harus dimasukkan kedalam knapsack sehingga total nilainya tidak melebihi batas yang diberikan, tetapi memiliki harga total tertinggi yang paling

memungkinkan.
Knapsack merupakan salah satu problematika maksimalisasi, yang merupakan bagian dari kombinatorial dan matematika terapan.
Knapsack problem memiliki tiga jenis persoalan, yaitu:

1. Knapsack 0-1
2. Knapsack *Bounded*
3. Knapsack *Unbounded*

Knapsack yang dibahas pada makalah ini merupakan jenis knapsack 0-1. variabel keputusan yang diperoleh yaitu X_i bernilai 1 jika objek itu dipilih dan X_i bernilai 0 jika objek tidak dipilih.

2.2. Beberapa Metode Pemecahan Masalah Knapsack

Algoritma Optimasi terbagi menjadi dua jenis, yaitu algoritma optimasi dengan pendekatan berbasis *deterministic* dan algoritma optimasi dengan pendekatan berbasis *probabilistic*. Yang termasuk kedalam algoritma berbasis *deterministic* diantaranya *State Space Search*, *Dynamic Programming*, dan *Branch and Bound*. Sedangkan algoritma optimasi yang termasuk kedalam algoritma yang berbasis pendekatan probabilistic adalah algoritma Monte Carlo dengan berbagai macam turunannya.
Pada makalah ini, tidak akan dibahas semua algoritma yang disebut di atas, dan hanya dibahas mengenai beberapa algoritma dan strategi pemecahan masalah yang berhubungan dengan optimasi Knapsack. beberapa di antaranya adalah:

1. *Brute Force*

Strategi algoritma yang dapat digunakan salah satunya adalah *brute force*. Algoritma ini dapat diibaratkan sebagai algoritma trial and error yang tidak memiliki sistematika yang baik dalam penyelesaikan berbagai kasus. strategi algoritma tersebut tidak efisien dan membutuhkan waktu yang lebih lama sehingga terkadang menyebabkan kasus *time limit exceeded* pada beberapa kesempatan yang membatasi waktu kompilasi dan runtime program.
Metode *brute force* tidak hanya digunakan dalam penyelesaian permasalahan Knapsack. Banyak kasus yang diselesaikan dengan menggunakan strategi ini. Akan tetapi, pendekatan yang digunakan dalam metode ini merupakan pendekatan *straightforward* dimana pada kasus Knapsack, kita menciptakan semua kombinasi barang yang mungkin dengan penanda 1/0 (true/false, ambil/tidak) untuk menentukan barang tersebut akan dimasukkan ke dalam list barang yang diambil atau tidak.

Dapat dilakukan pengujian, misalkan:
Jika kita mempunyai 3 benda dengan w_1=5 p_1=3, w_2=7 p_2=4 dan w_3=6 p_3=2, dengan kapasitas (W) = 11, maka terdapat 2^3 = 8 kombinasi keluaran yang mungkin:

Tabel 1. Kombinasi 3 input pada kasus knapsack dengan *brute force*

Benda 1	Benda 2	Benda 3	memenuhi?	Total profit
-	-	-	√	0
-	-	√	√	2
-	√	-	√	4
-	√	√	-	6
√	-	-	√	3
√	-	√	√	5
√	√	-	-	~
√	√	√	-	~

Dari tabel 1, didapatkan hasil knapsack yang diambil adalah pada baris ke-6, yaitu ambil benda 1 dan 3.
Permasalahan yang muncul adalah, terdapat banyak sekali kemungkinan kasus yang ada, yang besar kompleksitasnya dapat ditentukan dengan 2^n, n menyatakan jumlah barang. Barang yang diambil dinyatakan dengan 1 dan barang yang tidak diambil dinyatakan dengan 0. Strategi ini tidak mungkin masih efektif dijalankan pada kondisi n yang sangat kecil. Pada kasus n sangat besar, kompleksitas brute force sangat memperlambat proses pencarian solusi permasalahan Knapsack.

2. Cara Matematika

Pada cara ini, kita harus memperhatikan nilai probabilitas dari setiap barang, karena nilai inilah sebagai penentunya dengan memperhatikan nilai probabilitas (X_i) yaitu $0 \le X_i \le 1$. Nilai Xi bisa sangat beragam, dari 0, 0.1, 0.01, 0.001, ... , 1.
Dengan cara ini sulit untuk menentukan yang paling optimal sebab kita harus mencari nilai probabilitas yang tersebar antara 0 dan 1, $0 \le Xi \le 1$ untuk setiap objek. Cara ini disarankan tidak digunakan.

3. Algoritma Genetika

Algoritma genetik adalah teknik pencarian di dalam ilmu komputer untuk menemukan penyelesaian perkiraan untuk optimisasi dan masalah pencarian.[1]
Algoritma genetik merupakan kelas khusus dari algoritma evolusioner yang menggunakan teknik yang terinspirasi oleh biologi evolusioner seperti warisan, mutasi, seleksi alam dan rekombinasi (atau *crossover*)
Algoritma ini adalah algoritma komputer yang mencari suatu solusi baik dalam permasalah yang memiliki sejumlah besar kemungkinan pemecahan yang ada. Semua algoritma genetik dimulai dengan kumpulan solusi (yang diwakili oleh kromosom) yang biasa disebut populasi.
Suatu populasi baru diciptakan dari solusi-solusi yang ada dalam suatu populasi tua diharapan dapat menjadi suatu populasi lebih baik. Solusi-solusi yang telah dipilih dalam membentuk solusi baru (anak/*offspings*) akan diseleksi menurut *fitness* mereka. Semakin solusi-solusinya tersebut cocok maka akan lebih banyak kesempatan mereka dalam produksi kembali. Proses ini diulangi sampai kondisi yang diinginkan didapat.

[1] http://id.wikipedia.org/wiki/Algoritma_genetik

Representasikan barang dapat dilakukan dalam dua array, array pertama berisi berat barang, dan array kedua berisi profit barang. Contoh *constraint* yang digunakan pada kasus Knapsack adalah weight. Representasi kromosom dapat dilakukan dengan array 1 dimensi yang berisi 1 atau 0.
Misalkan**:**
Kromosom : 1 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 0 0
Arti : Barang 1, 4, 8, 9, 10, 12, 14, 16, 18 diambil
Barang 2, 3, 5, 6, 7, 11, 13, 15, 17, 19, 20 tidak diambil
Aplikasi ini akan selalu menemukan solusi, karena pengecekan apakah kromosom dalam suatu populasi dilakukan dua kali, yakni ketika inisialisasi populasi awal dan ketika kromosom-kromosom telah dimutasi.
Penerapan Algoritma Genetika dalam penyelesaian Knapsack Problem ini memiliki kelemahan yaitu ketidakpastian untuk menghasilkan solusi optimum global. Hal ini berlaku untuk semua kasus karena sebagian besar dari Algoritma Genetika ini berhubungan dengan bilangan random yang bersifat probabilistik.

4. Algoritma Greedy

Algoritma Greedy merupakan alternatif lain untuk memecahkan permasalahan Knapsack. Algoritma ini sering digunakan untuk memecahkan masalah optimasi dalam berbagai kasus seperti pada pohon bentangan terpendek (*minimum spanning tree*). Secara umum teknik ini menggunakan *heuristic* untuk mencari solusi suboptimum sehingga diharapkan solusi optimum.
Strategi greedy yang diterapkan pada 0-1 Knapsack:

- Pilih item yang memiliki nilai maksimum dari item-item yang tersedia, hal ini akan menambah nilai dari Knapsack dengan cepat.
- Pilih item yang memiliki bobot minimum dari item-item yang ada sehingga kapasitas terisi secara perlahan dan dapat memuat lebih banyak item.
- Pilih item yang memiliki nilai tinggi untuk bobot/berat

Setelah tiga strategi tersebut diterapkan dan diuji, maka didapat hasil terbaik dari aturan ketiga, yaitu memilih item bernilai tinggi dari rasio bobot terhadap berat.

5. Pemrograman Dinamik (*Dinamic Programming*)

Dynamic programming adalah metode pemecahan masalah dengan menguraikan solusi menjadi sekumpulan langkah (*step*) atau tahapan (*stage*) sedemikian sehingga solusi dari persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.
Pada penyelesaian persoalan dengan metode ini:

a. Terdapat sejumlah berhingga pilihan yang mungkin.
b. Solusi pada setiap tahap dibangun dari hasil solusi tahap sebelumnya.
c. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

Pada pemrograman dinamis, rangkaian keputusan yang optimal dibuat menggunakan prinsip optimalitas.
Prinsip Optimalitas mengandung ide bahwa *jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal*

- Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap $k + 1$, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal.
- ongkos pada tahap k +1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap $k + 1$)

Oleh karena itu, dynamic programming merupakan salah satu cara penanganan kasus Knapsack yang membutuhkan optimalisasi hasil.

2.3. Algoritma Greedy versus Dynamic Programming dalam pemecahan masalah 0-1 Knapsack

Algoritma greedy dan pemrograman dinamik merupakan dua cara untuk mengatasi masalah optimasi. Terkadang sebuah permasalahan dapat diselesaikan dengan kedua cara tersebut. Misalnya pada kasus *Single Source Shortest Path* yang dapat diselesaikan baik menggunakan Pemrograman Dinamik maupun strategi penyelesaian greedy. Namun algoritma pemrograman dinamis terkadang berlebihan dalam yang menghasilkan jalur terpendek dari semua sumber. Tidak ada cara untuk memodifikasi algoritma untuk menghasilkan lebih efisien hanya jalan terpendek dari satu sumber karena seluruh array diperlukan. Di sisi yang lain, terkadang lebih sulit untuk menentukan apakah algoritma greedy selalu menghasilkan sebuah solusi yang optimal. Sebuah bukti diperlukan untuk menunjukkan bahwa algoritma greedy pada kasus yang bersangkutan selalu menghasilkan sebuah solusi yang optimal.
Dibawah ini akan dibahas mengenai pencapaian kedua algoritma, Greedy dan Dynamic Programming dalam menangani permasalahan 0-1 Knapsack.

2.3.1. Strategi Greedy pada permasalahan 0-1 Knapsack

Sebuah contoh dari problem ini adalah mengenai seorang yang mencuri toko perhiasan dan membawa sebuah "knapsack" yang merupakan tas milik pencuri tersebut. Tas itu akan rusak jika total berat dari benda curian melebihi maksimum beban yang dapat dibawa oleh tas, sebut saja W. Setiap benda memiliki berat dan nilai. Permasalahan yang timbul adalah ketika kita harus memaksimalkan total nilai dari barang curian tersebut sepanjang tas tersebut masih mencukupi. Kasus ini disebut sebagai permasalahan 0-1 Knapsack.

Secara formal, dapat dituliskan sebagai:
$S = \{barang_1, barang_2, barang_3, ..., barang_n\}$
w_i= berat dari $barang_i$
p_i= profit dari $barang_i$
W= maksimum berat yang dapat ditahan oleh knapsack
Wi, Pi, W merupakan bilangan bulat positif. Akan didefinisikan suatu subset A dari S, sedemikian sehingga:

$\sum_{barang_i \in A} Pi$ (1)

dimaksimalkan terhadap:

$\sum_{barang_i \in A} w_i \leq W$ (2)

Strategi Greedy adalah mencuri barang dengan keuntungan terbesar pertama. yaitu, mencurinya dalam urutan keuntungan tak menaik. Bagaimanapun juga, strategi ini tidak akan bekerja sangat baik jika konten yang paling menguntungkan memiliki berat besar dibandingkan dengan keuntungan.

Sebagai contoh:
misalkan kita mempunyai 3 barang, w_1=25 p_1=10, w_2=w_3=10 p_2=p_3=9. Jika kapasitas W Knapsack adalah 30, strategi Greedy ini hanya akan memperoleh keuntungan 10, padahal solusi optimal keuntungan adalah 18. Cara lain dalam algoritma greedy adalah dengan pertama kali mengambil benda yang mempunyai bobot paling kecil. Strategi ini gagal diterapkan ketika benda-benda ringan mempunyai keuntungan kecil dibandingkan dengan beratnya.
Untuk mencegah kegagalan pada kedua algoritma greedy tersebut, sebuah algoritma greedy lain menggunakan strategi untuk mencari benda dengan rasio terbesar dari profit per berat benda. Algoritma ini menyusun benda dalam ukuran tidak menaik berdasarkan rasio tersebut, dan memilih mereka secara berurutan. Sebuah benda dimasukkan ke dalam Knapsack apabila berat benda tersebut tidak menyebabkan kelebihan pada kapasitas W.

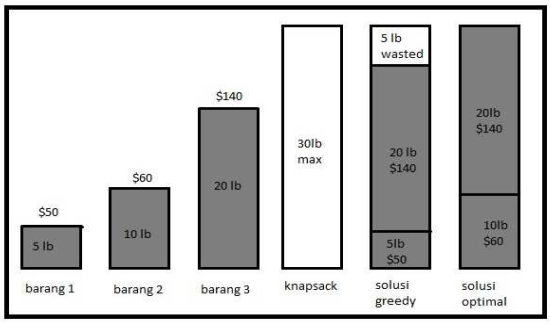
Misalkan suatu kasus uji diberikan:

$\text{Benda}_1 : \frac{\$50}{5} = \$10$

$\text{Benda}_2 : \frac{\$60}{10} = \$6$

$\text{Benda}_3 : \frac{\$140}{20} = \$7$

Akan ditempatkan ke dalam sebuah knapsack dengan beban maksimum (W) = 30lb.
Benda-benda yang ada kemudian disusun dengan urutan: *benda$_1$, benda$_3$, benda$_2$*.
Seperti terlihat pada Gambar 1, strategi greedy akan mengambil benda$_1$ dan benda$_3$ sebagai pilihan berdasarkan urutan perbandingan profit terhadap berat yang menghasilkan total profit sebesar \$190. Padahal, solusi optimal yang seharusnya adalah pemilihan terhadap benda$_2$ dan benda$_3$ yang akan menghasilkan profit sebesar \$200.
Masalah yang ada adalah bahwa terdapat kapasitas sisa sebesar 5 pon yang akhirnya dibuang begitu saja karena benda yang tersisa memiliki berat sebesar 10 pon. Meskipun algoritma ini tampak lebih baik dari algoritma greedy yang telah dibahas sebelumnya, tetapi algoritma ini masih gagal dalam menyelesaikan kasus 0-1 Knapsack.



Gambar 1. Solusi greedy dan solusi optimal untuk permasalahan 0-1 Knapsack

2.3.2. Strategi Pemrograman Dinamis (Dynamic Programming) pada permasalahan 0-1 Knapsack

Pemecahan masalah 0-1 Knapsack dengan menggunakan strategi pemrograman dinamis, atau lebih dikenal dengan singkatan DP (Dynamic Programming) meliputi beberapa tahap:

1. Tahap (misal: *k*) adalah proses memasukkan barang ke dalam karung (*knapsack*)
2. Status *(y)* menyatakan kapasitas muat karung yang tersisa setelah memasukkan barang pada tahap sebelumnya.

Dari tahap ke-1, kita masukkan objek ke-1 ke dalam karung untuk setiap satuan kapasitas karung sampai batas kapasitas maksimumnya. Karena kapasitas karung adalah bilangan bulat, maka pendekatan ini praktis.

- Misalkan ketika memasukkan objek pada tahap *k*, kapasitas muat karung sekarang adalah $y - w_k$.
- Untuk mengisi kapasitas sisanya, kita menerapkan prinsip optimalitas dengan mengacu pada nilai optimum dari tahap sebelumnya untuk kapasitas sisa $y - w_k$ (yaitu $f_{k-1}(y - w_k)$).
- Selanjutnya, kita bandingkan nilai keuntungan dari objek pada tahap *k* (yaitu p_k) plus nilai $f_{k-1}(y - w_k)$ dengan keuntungan pengisian hanya $k - 1$ macam objek, $f_{k-1}(y)$.
- Jika $p_k + f_{k-1}(y - w_k)$ lebih kecil dari $f_{k-1}(y)$, maka objek yang ke-*k* tidak dimasukkan ke dalam karung, tetapi jika lebih besar, maka objek yang ke-*k* dimasukkan.

- **Relasi rekurens untuk persoalan ini adalah**

$$f_0(y) = 0, \quad y = 0, 1, 2, \ldots, M \qquad \text{(basis)}$$
$$f_k(y) = -\infty, \quad y < 0 \qquad \text{(basis)}$$

$$f_k(y) = \max\{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\}, \quad k = 1, 2, \ldots, n \qquad \text{(rekurens)}$$

Gambar 2. Proses rekursi untuk DP

- $f_k(y)$ adalah keuntungan optimum dari persoalan 0/1 *Knapsack* pada tahap *k* untuk kapasitas karung sebesar *y*.
- $f_0(y)$ = 0 adalah nilai dari persoalan *knapsack* kosong (tidak ada persoalan *knapscak*) dengan

kapasitas y,

- $f_k(y) = -\infty$ adalah nilai dari persoalan knapsack untuk kapasitas negatif. Solusi optimum dari persoalan 0/1 *Knapsack* adalah $f_n(W)$.
- Dengan prinsip optimalitas ini dijamin bahwa pengambilan keputusan pada suatu tahap adalah keputusan yang benar untuk tahap-tahap selanjutnya.

Misalkan diberikan sebuah kasus uji:
Diketahui n=3, W=5

Tabel 2. Spesifikasi barang untuk kasus uji-1 DP

Barang ke-	w_i	p_i
1	2	65
2	3	80
3	1	30

Penyelesaian kasus per tahap:
Tahap 1:
$f_1(y) = \max \{f_0(y), p_1 + f_0(y-w_1)\}$
$= \max \{f_0(y), 65 + f_0(y-2)\}$

Tabel 3. Tahap 1 untuk kasus uji-1 DP

y	$f_0(y)$	$65 + f_0(y-2)$	Solusi Optimum	
			$f_1(y)$	(x_1,x_2,x_3)
0	0	$-\infty$	0	(0,0,0)
1	0	$-\infty$	0	(0,0,0)
2	0	65	65	(1,0,0)
3	0	65	65	(1,0,0)
4	0	65	65	(1,0,0)
5	0	65	65	(1,0,0)

Tahap 2:
$f_2(y) = \max \{f_1(y), p_2 + f_1(y-w_2)\}$
$= \max \{f_1(y), 80 + f_1(y-3)\}$

Tabel 4. Tahap 2 untuk kasus uji-1 DP

y	$f_1(y)$	$80 + f_1(y-3)$	Solusi Optimum	
			$f_2(y)$	(x_1,x_2,x_3)
0	0	$80+(-\infty)=-\infty$	0	(0,0,0)
1	0	$80+(-\infty)=-\infty$	0	(0,0,0)
2	65	$80+(-\infty)=-\infty$	65	(1,0,0)
3	65	80+0=80	80	(0,1,0)
4	65	80+0=80	80	(0,1,0)
5	65	80+65=145	145	(1,1,0)

Tahap 3:
$f_3(y) = \max \{f_2(y), p_3 + f_2(y-w_3)\}$
$= \max \{f_2(y), 30 + f_2(y-1)\}$

Tabel 4. Tahap 3 untuk kasus uji-1 DP

y	$f_1(y)$	$30 + f_2(y-1)$	Solusi Optimum	
			$f_3(y)$	(x_1,x_2,x_3)
0	0	$30+(-\infty)=-\infty$	0	(0,0,0)
1	0	$30+(-\infty)=-\infty$	0	(0,0,0)
2	65	30+0=30	65	(1,0,0)
3	80	30+65=95	95	(1,0,1)
4	80	30+80=110	110	(0,1,1)
5	145	80+65=145	145	(1,1,0)

Solusi optimalnya adalah benda ke 1 dan ke 2. Jumlah total profit yang dipilih: $\sum p = f = 145$.
Proses yang terdapat dalam algoritma ini memang tergolong panjang, tetapi pencapaian yang diperoleh benar dan sesuai dengan hasil optimasi seharusnya. Algoritma ini pun masih lebih singkat dibandingkan dengan algoritma brute force yang notabene mempunyai kompleksitas eksponensial sebesar 2^n.
Dari kasus uji pada algoritma Greedy dan pemrograman dinamis, didapatkan pencapaian yang berbeda pada kasus uji yang berbeda pula. Perbedaan kasus uji yang ada tidak dimaksudkan untuk membedakan hasil yang diperoleh dengan kesulitan yang berbeda yang diberikan pada kedua macam algoritma. Akan tetapi, lebih kepada pengambilan kasus uji secara acak, sehingga didapatkan hasil yang lebih umum.
Dari analisis di atas, tampak bahwa strategi pemrograman dinamis (*dynamic programming*) lebih cocok diterapkan pada masalah knapsack. Hal itu tidak berarti bahwa strategi DP (*Dynamic Programming*) lebih mangkus digunakan untuk segala persoalan optimasi.

3. KESIMPULAN

Berdasarkan analisis dan pembahasan, dapat diambil kesimpulan:

- Knapsack merupakan salah satu permasalahan optimasi yang sering dihadapi dalam berbagai bidang.
- Terdapat berbagai algoritma pemecahan masalah knapsack yang memiliki alur (proses) yang berbeda.
- Pemilihan algoritma yang tepat sangat membantu dalam mencapai optimasi yang benar dan pencapaian waktu proses yang lebih singkat.
- Pada metode *greedy* hanya satu rangkaian keputusan yang pernah dihasilkan, sedangkan pada metode program dinamis lebih dari satu rangkaian keputusan. Hanya rangkaian keputusan yang memenuhi prinsip optimalitas yang akan dihasilkan.
- metode *Brute Force* memiliki sifat optimal akan tetapi lama dalam waktu running, *Greedy* memiliki sifat cepat tapi tidak optimal, *Dynamic Programming* memiliki sifat optimal dan agak cepat dalam waktu running.

DAFTAR REFERENSI

[1] Naimipour, Kumarss., Neapolitan, Richard E., *Foundations of Algorithms*, D.C. Health and Company, 1996.
[2] Averbach, Bonnie., Chein, Orin., Problem Solving Through Recreational Mathematics, Dover Publications, Inc. 2000
[3] Hendra Prihandono. "Knapsack Problem dengan Algoritma dan Metode Greedy,"

http://hendryprihandono.wordpress.com/2009/01/03/knapsack-problem-dengan-algoritma-dan-metode-greedy (Tanggal akses: 19 Desember 2009, pukul 09.00 WIB)
[4] ----------. "Knapsack Problem," http://en.wikipedia.org/wiki/Knapsack_problem.htm (Tanggal akses: 19 Desember 2009, pukul 09.00 WIB)
[5] ----------. "Knapsack Algorithm," http://oc.its.ac.id/ambilfile.php?idp=667 (Tanggal akses: 19 Desember 2009, pukul 09.00 WIB)
[6] ----------. "Teknik Optimasi," http://ilmukuilmumu.wordpress.com/2009/11/12/teknik-optimasi/#more-76 (Tanggal akses: 19 Desember 2009, pukul 21.30 WIB)