PENERAPAN ALGORITMA *DYNAMIC PROGRAMMING* PADA PERMASALAHAN KNAPSACK 0-1

Irmeilyana[1)], Putra Bahtera Jaya Bangun[2)], Dian Pratamawati[3)], Winda Herfia Septiani[4)]
[1]Fakultas MIPA, Universitas Sriwijaya
email: imel_unsri@yahoo.co.id
[2]Fakultas MIPA, Universitas Sriwijaya
email: teger4959@ymail.com

Abstract

Knapsack problem is a optimization problem in the process of selection of goods loaded in container with maximum capacity restrictions that can be accommodated in a knapsack. The selected goods may not exceed the capacity of the knapsack, but they can maximize the total profit. In this research, the knapsack problem is solved by Dynamic Programming algorithm with forward recursive and backward recursive calculations. The algorithm is applied to the transport of agricultural goods by using a truck at UD. Subur Tani haul trucks with a maximum capacity of 6,000 kg. Optimal gains obtained by Dynamic Programming algorithm with forward recursive calculation is Rp 118,096,500 with a total by weight of goods transported is 5,981 kg. It is 99.683% of truck capacity. While the optimal profit gained by using Dynamic Programming algorithm with backward recursive equation is Rp 86,246,500 with a total weight of goods transported is 5,881 kg. It is 98.017% of truck capacity.

Keywords: *Knapsack, Maximum Capacity, Maximum Profit, Dynamic Programming, Recursive Equation.*

1. PENDAHULUAN

Masalah *knapsack* 0-1 merupakan salah satu contoh program *integer* [1]. Permasalahan *knapsack* terbagi menjadi tiga, yaitu *integer knapsack, bounded knapsack*, dan *unbounded knapsack*.

Setiap perusahaan pasti menginginkan keuntungan yang sebanyak-banyaknya dengan mengefisiensikan sumber daya yang dimiliki terhadap batasan-batasan yang ada. Salah satu contoh dalam kehidupan sehari-hari adalah permasalahan pada saat seseorang memilih objek dari sekumpulan objek yang masing-masing memiliki bobot/berat, volume, dan nilai untuk dimuat ke dalam sebuah media penyimpanan tanpa melebihi kapasitasnya sehingga diperoleh keuntungan yang maksimal dari pemilihan objek-objek. Permasalahan seperti ini merupakan permasalahan *knapsack*. Keterbatasan manusia dalam menyelesaikan masalah *knapsack* tanpa menggunakan alat bantu merupakan salah satu kendala dalam pencarian solusi yang optimal. Efisiensi waktu juga sering menjadi pertimbangan, maka dibutuhkan suatu metode sekaligus program aplikasi penerapan metode tersebut yang dapat membantu menyelesaikan masalah *knapsack*.

Terdapat beberapa strategi algoritma yang mempunyai karakteristik berbeda yang menghasilkan solusi optimal dalam menyelesaikan permasalahan *knapsack* diantaranya algoritma *Greedy*, algoritma *Dynamic Programming*, metode *Branch and Bound*, algoritma *Brute Force*, dan algoritma Genetika. Penelitian mengenai penyelesaian kasus *knapsack* 0-1 menggunakan algoritma *Dynamic Programming* rekursif maju telah dilakukan sebelumnya oleh [2], diterapkan pada komposisi makanan pasien rawat inap. Komposisi makanan mempunyai batasan minimal kalori yang diperbolehkan bagi pasien dan total nilai kolestrol paling minimal. Hasil optimal yang diperoleh memudahkan untuk mengetahui alternatif kombinasi makanan tersebut. Penyelesaian masalah *knapsack* 0-1 pada [3] dengan algoritma *Dynamic Programming* menghasilkan solusi yang lebih optimal dibandingkan dengan menggunakan algoritma *Brute Force*, algoritma genetika, dan algoritma *Greedy*.

Dynamic Programming digunakan untuk mengoptimalkan proses pengambilan keputusan secara bertahap, perhitungan di

tahap yang berbeda dihubungkan melalui perhitungan rekursif. Proses perhitungan dilakukan secara satu per satu dan menyeluruh di setiap tahap dengan memperhatikan kondisi kapasitas maksimal *knapsack*. Keputusan yang didapat pada setiap tahap digunakan untuk perhitungan selanjutnya guna mendapatkan solusi optimal yang mungkin bagi seluruh masalah.

[4] membahas tentang penyelesaian *knapsack* 0-1 dengan menggunakan algoritma *Greedy* dan metode *Branch and Bound*. Keuntungan maksimal yang didapat dengan menggunakan perhitungan algoritma *Greedy by Density* dibanding algoritma *Greedy by Weight* dan *Greedy by Density* tetapi keuntungan yang lebih optimal diperoleh dengan menggunakan metode *Branch and Bound* dengan total barang yang diangkut lebih besar.

Tujuan penelitian ini adalah mendapatkan solusi optimal pada masalah *knapsack* 0-1 dengan menggunakan algoritma *Dynamic Programming.* Selanjutnya, solusi optimal yang diperoleh dibandingkan dengan solusi optimal dari algoritma *Greedy* dan metode *Branch and Bound* hasil penelitian [4].

2. KAJIAN LITERATUR

Persoalan *Knapsack* 0-1 pada wadah terbuka ditulis sebagai berikut [1]:
Maksimum :

$$Z = \sum_{i=1}^{n} p_i x_i \qquad (1)$$

dengan kendala:

$$\sum_{i=1}^{n} w_i x_i \leq M \qquad (2)$$

Keterangan:
$i = 1, 2, 3, \dots, n$,
z = keuntungan total,
n = banyaknya jenis objek,
p_i = keuntungan per satuan objek ke-*i*,
w_i = berat objek ke-*i*, dan
M = kapasitas maksimal *knapsack*.

$$x_i = \begin{cases} 0, \text{jika objek} - i \text{ tidak dimasukkan} \\ 1, \text{jika objek} - i \text{ dimasukkan} \end{cases}$$

Pendekatan *Dynamic Programming* Secara Rekursif [5]

(i). Perhitungan Rekursif Maju

Model untuk rekursif maju:

$$f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \cdots \rightarrow f_n$$

Keterangan:
f_n = keputusan dari tahap perhitungan, dan
n = jumlah objek yang dinotasikan dengan i.

Persamaan (1) dan Persamaan (2) digunakan untuk menyelesaikan perhitungan *Dynamic Programming*, dengan menetapkan $f_i(y)$ sebagai keuntungan maksimal yang dapat diperoleh dari keadaan *i*, *i*+1, ... , *n*. Jika *y* satuan dari sumber yang akan dialokasikan ke keadaan *i*, *i*+1, ... , *n* maka dapat dituliskan secara umum formulasi *Dynamic Programming* rekursif maju yaitu:

$$f_i(y) = \text{maks}\ \{f_{i-1}(y),\ f_{i-1}(y - w_i) + p_i\} \qquad (3)$$

Keterangan:
$i = 1, 2, 3, \dots, n$,
f_i = nilai optimal dari permasalahan yang diselesaikan pada tahap *i*,
y = kapasitas *knapsack* pada tahap *i*,
$f_i(y)$ = nilai optimal dari permasalahan *knapsack* 0-1 yang diselesaikan pada tahap i dengan kapasitas *knapsack* sebesar *y*.

Jika $f_{i-1}(y - w_i) + p_i > f_{i-1}(y)$ (4)
maka objek pada tahap-*i* dimasukkan ke dalam *knapsack*, dan sebaliknya untuk
$f_{i-1}(y - w_i) + p_i < f_{i-1}(y)$ (5)
maka objek pada tahap-*i* tidak dimasukkan ke dalam *knapsack*.

(ii) Perhitungan Rekursif Mundur

Model untuk rekursif mundur:

$$f_n,\ f_{n-1},\ f_{n-2}, \cdots, f_1$$

$$f_i(y) = \text{maks}\ \{f_{i+1}(y),\ f_{i+1}\,(y - w_i) + p_i\} \qquad (6)$$

dengan $i = n, n-1, n-2, n-3, \cdots, 1$
Jika $f_{i+1}(y - w_i) + p_i > f_{i+1}(y)$ (7)
maka objek pada tahap-*i* dimasukkan ke dalam *knapsack*, dan sebaliknya untuk:
$f_{i+1}(y - w_i) + p_i < f_{i+1}(y)$ (8)
maka objek pada tahap-*i* tidak dimasukkan ke dalam *knapsack*.
Untuk $y < 0$, maka $f_i(y)$ dinotasikan:
$f_i(y) = -\infty$ (9)
$f_0(y) = 0$, jika nilai dari persoalan *knapsack* kosong (tidak ada persoalan *knapsack*) dengan kapasitas *y*.
$f_i(y) = -\infty$, jika nilai dari persoalan *knapsack* untuk kapasitas negatif.

3. METODE PENELITIAN

Penelitian ini menggunakan data sekunder pada penelitian [4] yang diperoleh dari UD. Subur Tani Makmur pada bongkar muat pupuk dan kebutuhan pertanian. Adapun langkah-langkah yang dilakukan sebagai berikut :

1. Merumuskan model masalah *knapsack* yang diperoleh [4] ke dalam persamaan *Dynamic Programming* untuk setiap penyelesaian tahap dengan perhitungan rekursif maju yang ditulis pada Persamaan (3) dan perhitungan rekursif mundur pada Persamaan (6).
2. Menghitung solusi optimal masalah *knapsack* 0-1 di UD. Subur Tani Makmur berdasarkan algoritma *Dynamic Programming* dengan menggunakan pendekatan rekursif maju pada Persamaan (3).

2.1 Untuk perhitungan pertama belum ada produk atau barang yang akan dimasukkan ke dalam *knapsack,* sehingga $f_0(y) = 0$.

2.2 Melakukan perhitungan untuk tahap selanjutnya hingga ke-*n* data dengan menggunakan hasil dari solusi optimal sebelumnya.

2.3 Mencari nilai keuntungan tiap barang pada tahap ke-*i* dengan kapasitas berat barang ke-*y* dengan menyelesaikan semua perhitungan untuk kapasitas maksimal barang (*M*). Jika nilai keuntungan diperoleh pada kapasitas *y* bernilai negatif maka solusi yang diperoleh sesuai dengan Persamaan (9).

2.4 Bandingkan nilai keuntungan yang didapat sekarang dengan hasil yang diperoleh sebelumnya pada tahap ke-*i* dengan menggunakan Persamaan (4) dan Persamaan (5).

2.5 Analisis keuntungan yang didapat untuk setiap tahap perhitungan sehingga didapat keuntungan maksimal untuk tahap ke-1, 2, 3, …, *n*.

2.6 Definisikan tiap barang (x_i) yang akan dimasukkan ke dalam *knapsack* yaitu 1 dan yang tidak dimasukkan ke dalam *knapsack* adalah 0 dengan menganalisis tiap keuntungan yang didapat dengan kapasitas berat barang sebesar *y* selama proses perhitungan.

3. Menghitung solusi optimal dengan menggunakan pendekatan rekursif mundur Persamaan (6).

3.1 Perhitungan tahap ke-1 dimulai dari data ke-19 hingga data ke-1, sehingga $f_{20}(y) = 0$.

3.2 Mencari nilai keuntungan tiap barang pada tahap ke-*i* dengan kapasitas berat barang ke-*y* dengan menyelesaikan semua perhitungan untuk kapasitas maksimal barang (*M*). Jika nilai keuntungan diperoleh pada kapasitas *y* bernilai negatif maka solusi yang diperoleh sesuai dengan Persamaan (9).

3.3 Bandingkan nilai keuntungan yang didapat sekarang dengan hasil yang diperoleh sebelumnya pada tahap ke-*i* dengan menggunakan Persamaan (7) dan Persamaan (8).

3.4 Lakukan perhitungan untuk tahap selanjutnya hingga data ke-1 dengan menggunakan hasil keuntungan optimal yang diperoleh sebelumnya dan digunakan untuk perhitungan guna memperoleh keuntungan selanjutnya.

3.5 Analisis keuntungan yang didapat untuk setiap tahap perhitungan sehingga didapat keuntungan maksimal untuk tahap ke-*n*-1, *n*-2, …, 1.

3.6 Definisikan tiap barang (x_i) yang akan dimasukkan ke dalam *knapsack* yaitu 1 dan yang tidak dimasukkan ke dalam *knapsack* adalah 0 dengan menganalisis tiap keuntungan yang didapat dengan kapasitas berat barang sebesar *y* selama proses perhitungan.

4. Analisis barang yang dapat dimasukkan ke dalam *knapsack* pada Langkah 3.6 dan Langkah 4.6.
5. Menghitung keuntungan maksimal dengan menggunakan bantuan *software Java Netbeans* untuk mengetahui keuntungan maksimal yang didapat.
6. Membandingkan solusi optimal yang diperoleh dari algoritma *Dynamic Programming* dengan perhitungan rekursif maju dan perhitungan rekursif mundur dengan hasil penelitian [4].

4. HASIL DAN PEMBAHASAN

4.1. Deskripsi Data

Perusahaan UD. Subur Tani mempunyai permasalahan untuk mengangkut barang berupa pupuk dan kebutuhan pertanian

sebanyak 19 item barang yang diangkut dengan sebuah truk dengan kapasitas maksimum sebesar 6.000 kg. Dalam hal ini item barang diurutkan dari item yang mempunyai keuntungan yang tertinggi, seperti pada Tabel 1.

Tabel 1. Data Jenis Barang, Berat, dan Keuntungan Total

i	Variabel	Berat (w_i)	Keuntungan (p_i)
1	x_1	1000	35.000.000
2	x_2	750	22.500.000
3	x_3	750	18.750.000
4	x_4	375	9.375.000
5	x_5	750	11.250.000
6	x_6	100	1.500.000
7	x_7	750	9.375.000
8	x_8	150	1.500.000
9	x_9	200	2.000.000
10	x_{10}	100	1.000.000
11	x_{11}	500	3.500.000
12	x_{12}	100	500.000
13	x_{13}	150	637.500
14	x_{14}	300	1.200.000
15	x_{15}	400	1.600.000
16	x_{16}	200	800.000
17	x_{17}	150	450.000
18	x_{18}	150	300.000
19	x_{19}	6	9.000

***Keterangan**:* Nama barang : x_1 = pupuk NPK basf, x_2 = pupuk borax, x_3 = KCI merauke, x_4 = plastik mulsa 25 kg, x_5 = pupuk yaramila mutiara, x_6 = plastik mulsa 10 kg, x_7 = pupup merauke TSP 46, x_8 = antracol 70 WP, x_9 = amcozep, x_{10} = dithome M-45, x_{11} = ferterra 0,4 gr, x_{12} = kresnakum, x_{13} = furadan 3 gr kecil, x_{14} = polybag 30x35 cm, x_{15} = primodon 2 kg, x_{16} = furadan 3 gr besar, x_{17} = mineral feed, x_{18} = sidafur, x_{19} = top mix (pakan ayam).

4.2. Penerapan Algoritma *Dynamic Programming* pada Permasalahan *Knapsack* 0-1

Diketahui terdapat 19 jenis barang (n = 19) yang dinotasikan dengan **x = (**x_1 x_2, …,x_{19}). Berat barang i dinotasikan dengan w_i yaitu **w** = (w_1, w_2, … , w_{19}) = (1.000, 750, 750, …, 6). Kapasitas angkut dinotasikan dengan M = 6.000. Tujuan dari penggunaan *Dynamic Programming* yaitu mencari total keuntungan yang maksimal dari pengangkutan barang.

4.2.1 Algoritma *Dynamic Programming* Perhitungan Rekursif Maju

Tahap ke-1:

Keuntungan maksimal yang diperoleh untuk i = 0 adalah $f_0(y) = 0$.

Berdasarkan Tabel 1, berat barang 1 (w_1) sebesar 1.000 kg dan keuntungannya (p_1) sebesar Rp 35.000.000, dengan menggunakan Persamaan (3) maka

$$f_1(y) = \text{maks}\ \{f_0(y),\ f_0(y - w_1) + p_1\}$$
$$f_1(y) = \text{maks}\{f_0(y), f_0(y - 1.000) + 35.000.000\ \}$$

Hasil dari perhitungan barang ke-1 dapat dilihat pada Tabel 2 berikut:

Tabel 2. Hasil Perhitungan *Dynamic Programming* pada Tahap ke-1

y	$f_0(y)$	$f_0(y - 1000) + 35.000.000$	$f_1(y)$
0	0	-∞	0
1	0	-∞	0
2	0	-∞	0
⋮	⋮	⋮	⋮
999	0	-∞	0
1.000	0	35.000.000	35.000.000
1.001	0	35.000.000	35.000.000
⋮	⋮	⋮	⋮
6.000	0	35.000.000	35.000.000

Keterangan: $f_i(y) = -\infty$; untuk y < 0 (nilai dari persoalan *knapsack* untuk kapasitas negatif).

Diperoleh hasil untuk mengetahui keuntungan maksimal sebesar :

$$f_1(y) = \text{maks}\quad \{f_0(y),\ f_0(y - 1.000) + 35.000.000\ \}$$
$$f_1(y) = \text{maks}\,\{0,\ 35.000.000\ \}=35.000.000$$

Keuntungan maksimal untuk perhitungan pada tahap ke-1 sebesar Rp 35.000.000. Dari hasil keuntungan maksimal yang didapat, barang yang akan diangkut ke dalam truk yaitu x_1 (bernilai 1). Sedangkan untuk $x_2, x_3, \ldots, x_{19}$ bernilai 0 artinya barang tidak diangkut ke dalam truk. Solusi optimal berdasarkan perhitungan algoritma *Dynamic Programming* pada tahap ke-1 adalah:

$\boldsymbol{x}$ = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Tahap ke-2:

Diketahui pada data jenis barang ke-2 pada Tabel 1 memiliki berat barang (w_i) sebesar 750 kg dan keuntungan barang (p_i) sebesar Rp 22.500.000, dengan menggunakan Persamaan (3),

$$f_2(y) = \text{maks}\,\{f_1(y), f_1(y - 750) + 22.500.000\ \}$$

Perhitungan untuk penyelesaian barang ke-2 dapat dilihat pada Tabel 3.

Tabel 3. Hasil Perhitungan *Dynamic Programming* pada Tahap ke-2

y	$f_1(y)$	$f_1(y-750)+22.500.000$	$f_2(y)$
0	0	-∞	0
1	0	-∞	0
2	0	-∞	0
⋮	⋮	⋮	⋮
749	0	-∞	0
750	0	22.500.000	22.500.000
⋮	⋮	⋮	⋮
1.000	35.000.000	22.500.000	22.500.000
⋮	⋮	⋮	⋮
1.749	35.000.000	22.500.000	35.000.000
1.750	35.000.000	57.500.000	57.500.000
1.751	35.000.000	57.500.000	57.500.000
⋮	⋮	⋮	⋮
6.000	35.000.000	57.500.000	57.500.000

Diperoleh hasil untuk mengetahui keuntungan maksimal sebesar :

$f_2(y)$ = maks {35.000.000, 57.500.000}
= 57.500.000

Solusi optimal pada tahap ke-2 adalah:

$\boldsymbol{x}$ = (1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Dengan cara yang sama dilakukan juga sampai tahap ke-19, sehingga diperoleh Tabel 4.

Tabel 4. Rekapitulasi Solusi Optimal dari Perhitungan Rekursif Maju

Tahap	Barang yang Diangkut	Keuntungan Maksimal
6	$x_1, x_2, \ldots, x_6$	98.375.000
7	$x_1, x_2, \ldots, x_7$	107.750.000
8	$x_1, x_2, \ldots, x_8$	109.250.000
9	$x_1, x_2, \ldots, x_9$	111.250.000
10	$x_1, x_2, \ldots, x_{10}$	112.250.000
11	$x_1, x_2, \ldots, x_{11}$	115.750.000
12	$x_1, x_2, \ldots, x_{12}$	116.250.000
13	$x_1, x_2, \ldots, x_{13}$	116.887.000
14	$x_1, x_2, \ldots, x_{14}$	118.087.500
15	$x_1, x_2, \ldots, x_{14}$	118.087.500
16	$x_1, x_2, \ldots, x_{14}$	118.087.500
17	$x_1, x_2, \ldots, x_{14}$	118.087.500
18	$x_1, x_2, \ldots, x_{14}$	118.087.500

Keterangan: Barang-barang (item) yang diangkut bernilai 1.
Keuntungan (dalam Rp)

Berdasarkan Tabel 4, keuntungan maksimal dari hasil perhitungan penyelesaian tahap ke-6 sampai pada tahap ke-14 terjadi peningkatan dengan jumlah berat barang maksimal yang dapat diangkut pada tahap ke-14 yaitu 5.975 kg. Pada tahap ke-15 sampai tahap ke-18 keuntungan maksimal yang didapat sama dengan keuntungan maksimal yang didapat pada tahap ke-14 karena berat barang yang diangkut melebihi kapasitas maksimal angkut truk.

Penyelesaian untuk tahap ke-19 sebagai berikut:

Tahap ke-19

Diketahui pada data jenis barang ke-19 pada Tabel 1 memiliki berat barang (w_i) sebesar 6 kg dan keuntungan barang (p_i) sebesar Rp 9.000, dengan menggunakan Persamaan (3) maka

$$f_{19}(y) = \text{maks}\,\{f_{18}(y), f_{18}(y-6) + 9.000\,\}$$

Perhitungan untuk penyelesaian barang ke-19 dapat dilihat pada Tabel5.

Tabel 5. Hasil Perhitungan *Dynamic Programming* pada Tahap ke-19

y	$f_{18}(y)$	$f_{18}(y-6)+9.000$	$f_{19}(y)$
0	0	-∞	0
1	0	-∞	0
2	0	-∞	0
⋮	⋮	⋮	⋮
250	3.000.000	9000	3.000.000
⋮	⋮	⋮	⋮
1..176	36.500.000	45.500.000	45.500.000
⋮	⋮	⋮	⋮
1.800	57.500.000	66.500.000	66.500.000
⋮	⋮	⋮	⋮
2.050	60.500.000	60.509.000	60.509.000
⋮	⋮	⋮	⋮
2.125	66.875.000	70.500.000	70.500.000
⋮	⋮	⋮	⋮
2.500	76.250.000	70.884.000	76.250.000
⋮	⋮	⋮	⋮
2.874	80.250.000	80.259.000	80.259.000
2.875	85.625.000	80.259.000	85.625.000
⋮	⋮	⋮	⋮
3.625	96.875.000	92.271.500	96.875.000
⋮	⋮	⋮	⋮

4.375	106.250.000	103.521.500	103.521.500
⋮	⋮	⋮	⋮
5.974	117.850.000	117.859.000	117.859.000
5.975	118.087.500	117.859.000	118.087.500
⋮	⋮	⋮	⋮
6.000	118.087.500	118.096.500	118.096.500

4.2.2 Perhitungan Rekursif Mundur

Perhitungan dengan rekursif mundur dimulai dari data terakhir yaitu data ke-19 sampai pada data ke-1 dengan *n*, *n*-1, *n*-2, *n*-3, …, 1. Dengan cara yang analog dengan perhitungan rekursif maju, maka didapat:

Tahap ke-1:

Untuk *i* = 20, keuntungan maksimal yang diperoleh adalah $f_{20}(y) = 0$

Berdasarkan Tabel 1 , diketahui jenis barang ke-19 memiliki berat barang sebesar 6 kg dan keuntungan sebesar Rp 9.000, dengan menggunakan Persamaan (6)

Untuk *i* = 19, maka

$$f_{19}(y) = \text{maks}\ \{f_{20}(y),\ f_{20}(y-6) + 9.000\}$$

Hasil dari perhitungan barang ke-19 dapat dilihat pada Tabel 6.

Tabel 6. Hasil Perhitungan *Dynamic Programming* pada Tahap ke-1

y	$f_{20}(y)$	$f_{20}(y-6) + 9.000$	$f_{19}(y)$
0	0	-∞	0
1	0	-∞	0
2	0	-∞	0
⋮	⋮	⋮	⋮
6	0	9.000	9.000
7	0	9.000	9.000
8	0	9.000	9.000
⋮	⋮	⋮	⋮
6.000	0	9.000	9.000

Keuntungan maksimal yang didapat pada perhitungan tahap ke-1 sebesar Rp 9.000, dengan solusi optimal $\boldsymbol{x}$ = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1).

Selaanjutnya, dengan cara yang sama diperoleh rekapitulasi seperti pada Tabel 7.

Tabel 7. Rekapitulasi Solusi Optimal dari Perhitungan Rekursif Mundur

Tahap	Barang yang Diangkut	Keuntungan Maksimal
6	$x_{14}, x_{15}, \ldots, x_{19}$	4.359.000
7	$x_{13}, x_{14}, \ldots, x_{19}$	4.996.500
8	$x_{12}, x_{13}, \ldots, x_{19}$	5.496.500
9	$x_{11}, x_{12}, \ldots, x_{19}$	8.996.500
10	$x_{10}, x_{11}, \ldots, x_{19}$	9.996.500
11	$x_{19}, x_{10}, \ldots, x_{19}$	11.996.500
12	$x_{8}, x_{9}, \ldots, x_{19}$	13.496.000
13	$x_{7}, x_{8}, \ldots, x_{19}$	22.871.500
14	$x_{6}, x_{7}, \ldots, x_{19}$	24.371.500
15	$x_{5}, x_{6}, \ldots, x_{19}$	35.621.500
16	$x_{4}, x_{5}, \ldots, x_{19}$	44.996.500
17	$x_{3}, x_{4}, \ldots, x_{19}$	63.476.500
18	$x_{2}, x_{3}, \ldots, x_{19}$	82.246.500

Keterangan: Barang-barang (item) yang diangkut bernilai 1.

Berdasarkan Tabel 7, terjadi peningkatan keuntungan maksimal dari perhitungan tahap ke-6 sampai pada tahap ke-18 dengan jumlah berat barang yang diangkut yaitu 5.881 kg. Pada perhitungan tahap ke-19 tidak terjadi peningkatan keuntungan maksimal karena berat barang melebihi kapasitas angkut truk sehingga barang ke-19 tidak dimasukkan. Keuntungan maksimal yang didapat dalam penyelesaian kasus muat barang menggunakan algoritma *Dynamic programming* perhitungan rekursif mundur sebesar Rp 86.246.500.

4.3 Perbandingan Hasil Algoritma *Dynamic Programming* dengan Algoritma *Greedy* dan Metode *Branch and Bound*

Perbandingan hasil [4] dengan hasil perhitungan menggunakan *Dynamic Programming* dapat dilihat pada Tabel 8.

Tabel 8. Perbandingan Hasil Perhitungan *Knapsack* 0-1

	Septiani (2014)				Algoritma *Dynamic Programming*	
	Algoritma Greedy			Metode *B & B*	Rekursif Maju	Rekursif Mundur
	by Weight	*by Profit*	*by Density*			
A	86,2465	116,35	118,0875	118,0965	118,0965	86,2465
B	X_1	X_{10}, X_{12}, X_{13}, X_{14}, X_{16}, X_{17}, X_{18}, X_{19}	X_{15}, X_{16}, X_{17}, X_{18}, X_{19}	X_{15}, X_{16}, X_{17}, X_{18}	X_{15}, X_{16}, X_{17}, X_{18}	X_1
C	5.881	5.725	5.975	5.981	5.981	5.881
D	98,017	95,417	99,583	99,683	99,683	98,017

Keterangan: A = Keuntungan Maksimal,
B = Barang yang Tidak Diangkut,
C = Jumlah Bobot Barang yang Diangkut (dalam kg),
D = Persentase Barang yang Diangkut

B & B = *Branch and Bound*
Persentase barang yang diangkut adalah persentase muatan yang didasarkan kapasitas truk (6.000 kg). Keuntungan dalam juta rupiah.

Berdasarkan Tabel 8, diketahui keuntungan maksimal untuk masalah pengangkutan barang di UD. Subur Tani adalah sama antara hasil yang diperoleh dari perhitungan menggunakan metode *Branch and Bound* dan algoritma *Dynamic Programming* rekursif maju yaitu Rp 118.096.500. *Dynamic Programming* perhitungan rekursif mundur dan algoritma *Greedy by Weight* menghasilkan solusi keuntungan, jenis barang yang tidak terangkut, dan persentase barang yang terangkut yang sama. Solusi keuntungan kedua metode ini paling kecil dibanding solusi dari metode atau algoritma yang lain. Jumlah dan Jenis barang yang tidak diangkut dan jumlah bobot barang yang diangkut dari hasil algoritma *Dynamic Programming* perhitungan rekursif maju dan metode *Branch and Bound* adalah sama.

Dengan demikian, metode *Branch and Bound* dan algoritma *Dynamic Programming* perhitungan rekursif maju menghasilkan solusi keuntungan yang paling maksimal dibanding algoritma *Greedy* (baik *by Weight*, *by Profit* dan *by Density*) dan algoritma *Dynamic Programming* perhitungan rekursif mundur.

5. KESIMPULAN

Kesimpulan dari penelitian ini adalah:

1. Keuntungan di UD. Subur Tani Makmur berdasarkan algoritma *Dynamic Programming* perhitungan rekursif maju yaitu sebesar Rp 118.096.500 dengan total berat barang yang diangkut adalah 5.981 kg sehingga memenuhi 99,683 % dari kapasitas truk.
2. Keuntungan berdasarkan algoritma *Dynamic Programming* perhitungan rekursif mundur yaitu sebesar Rp 86.246.500 dengan total berat barang yang diangkut adalah 5.881 kg sehingga memenuhi 98,017 % dari kapasitas truk.
3. Penggunaan algoritma *Dynamic Programming* perhitungan rekursif maju menghasilkan solusi keuntungan yang sama dengan metode *Branch and Bound* dan merupakan solusi paling maksimal dibanding algoritma *Greedy*.
4. Algoritma *Dynamic Programming* perhitungan rekursif mundur dan algoritma *Greedy by Weight* menghasilkan solusi keuntungan yang sama dan merupakan solusi yang paling minimal.

6. REFERENSI

[1] Martello, S & P. Toth. 1990. *Knapsack Problem*. John Wiley & Sons, Singapore.

[2] Rahajoe, A & Afrizal, A. 2013. Pendekatan Maju (Forward) *Dynamic Programming* untuk Permasalahan *MinMax Knapsack* 0-1. *Prosiding Seminar Teknik Elektro dan Pendidikan Teknik Elektro*. Surabaya: Universitas Negeri Surabaya.

[3] Passa, F. 2009. Permasalahan Optimasi 0-1 *Knapsack* dan Perbandingan Beberapa Algoritma Pemecahannya. *Makalah IF2091 Struktur Diskrit*. Bandung: Institut Teknologi Bandung.

[4] Septiani, W.H. 2014. Implementasi Algoritma *Greedy* dan Metode *Branch and Bound* dalam Persoalan *Knapsack* 0-1 di UD. Subur Tani Makmur. *Skripsi*. Indralaya: Universitas Sriwijaya.

[5] Horowitz. E, Sahni. S & Rajasekaran. S. 1988. *Computer Algorithms*. New York: Computer Science Press.