Aplikasi Strategi Algoritma *Greedy* dan Branch and Bound Dalam Perencanaan Pembelian Bahan Baku di Pasar

Disusun Oleh :

Kelompok 2

Anyelir Belia Azzahra - 1301200048

Anisa Adelya Ayuputri - 1301204225

Faiha Adzra Darmawan - 1301202434

Program Studi S1 Informatika

Fakultas Informatika

Telkom University

2022

Aplikasi Strategi Algoritma *Greedy* dan Branch and Bound

Dalam Perencanaan Pembelian Bahan Baku di Pasar

Abstrak - Pembelian bahan baku di pasar merupakan kegiatan yang dapat mempengaruhi proses produksi. Maka dari itu diperlukan adanya perencanaan pembelian bahan baku agar diperoleh solusi optimal untuk kuantitas bahan baku dengan jumlah uang yang terbatas. Hal ini dilakukan dengan tujuan untuk meningkatkan efektivitas kegiatan pembelian bahan baku di pasar. Dalam studi ini akan dibahas strategi untuk menyelesaikan permasalahan ini dengan pendekatan *greedy* dan *branch and bound.*

Kata kunci: *Greedy*, *Branch and Bound*, Bahan Baku, Pasar

PENDAHULUAN

Pembelian bahan baku merupakan kegiatan yang dapat mempengaruhi proses produksi. Bahan baku merupakan bahan yang akan digunakan untuk membuat suatu produk. Menurut (Stevenson & Chuong, 2014), pengertian bahan baku adalah sesuatu yang digunakan untuk membuat barang jadi, bahan pasti menempel menjadi satu dengan barang jadi. Maka dari itu perlu adanya perencanaan bahan baku yang matang agar persediaan tidak mengalami kekurangan dan kelebihan yang membuat total biaya persedian menjadi tinggi serta untuk meningkatkan efektivitas kegiatan pembelian bahan baku di pasar. Pada aplikasi ini kami menerapkan dua algoritma untuk perencanaan pembelian bahan baku di pasar, algoritma pertama menggunakan *greedy* dan algoritma kedua kami menggunakan *branch and bound.*

I. PENJELASAN STUDI KASUS

Sering kali dalam pembelian bahan baku di pasar, kita menggunakan metode kira kira, tanpa memperhatikan jumlah atau kuantitas barang yang kita dapatkan.

Berdasarkan masalah tersebut, dapat dicari solusi untuk mendapatkan kuantitas optimal dengan jumlah uang yang terbatas. Pada makalah ini kami melakukan pencarian solusi menggunakan algoritma *greedy* dan *branch and bound*.

Gambar 1. Timbangan untuk bahan baku

II. STRATEGI ALGORITMA

Algoritma *greedy* merupakan algoritma yang bersifat heuristik dan urutan logisnya disusun berdasarkan langkah-langkah penyelesaian masalah yang disusun secara sistematis. Pada algoritma *greedy*, solusi yang terbentuk adalah optimum lokal, dibentuk dari setiap langkah yang diambil dan pada setiap langkahnya harus dibuat keputusan terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah berikutnya.

Algoritma *greedy* disusun berdasarkan 5 komponen, yaitu:

1) Himpunan Kandidat

   Berisi elemen-elemen pembentuk solusi

2) Fungsi Solusi

   Fungsi yang menentukan akhir dan iterasi

3) Fungsi Seleksi

   Memilih kandidat yang paling memungkinkan mendapatkan solusi optimal dan kandidat yang sudah terpilih pada satu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

4) Fungsi Kelayakan

   Memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yaitu kandidat yang bersama-sama dengan himpunan solusi yang sudah terbentuk tidak

melanggar *constraints* (dalam hal ini *constraint* yang digunakan adalah jumlah uang yang terbatas).

5) Fungsi Objektif

Fungsi yang akan memaksimumkan atau meminimumkan nilai solusi (misal: harga barang, keuntungan barang, dsb).

Branch and Bound biasanya digunakan untuk menyelesaikan persoalan optimasi, yaitu permasalahan meminimalkan atau memaksimalkan suatu fungsi objektif, yang tidak melanggar batasan dari suatu permasalahan. Umumnya pada algoritma ini menggunakan pendekatan seperti *Best-First Search* dan pencarian solusi menggunakan *state-space-tree.*

Pada permasalahan perencanaan pencarian bahan baku ini, level pohon menyatakan urutan objek, dan simpul anak menyatakan kemungkinan terpilih atau tidaknya objek tersebut.

Permasalahan ini setara dengan permasalahan knapsack yang merupakan masalah maximasi, dimana dapat ditinjau menggunakan batas atas (*upper bound*). Total biaya tidak akan melebihi batas atas (*upper bound*), oleh karena itu kita perlu mengurutkan objek berdasarkan *density* secara *descending* atau menurun.

2.1 Pemodelan Menggunakan Algoritma *Greedy*

Pada makalah ini akan dibahas bagaimana pengimplementasian algoritma *greedy* dalam perencanaan pembelian bahan baku sehingga didapatkan solusi optimal, yaitu semakin banyaknya bahan baku yang dapat dibeli dengan jumlah uang yang terbatas.

Dalam merencanakan pembelian bahan baku tersebut, hal yang perlu diperhitungkan adalah harga dan berat bahan.

Berdasarkan persoalan tersebut, maka setiap bahan baku akan memiliki properti seperti tabel berikut.

Nama Bahan	Berat Bahan	Harga Bahan

Tabel 1. Tabel bahan baku

Pemodelan matematis *fractorial* pada masalah ini mirip dengan permasalahan knapsack sebagai berikut.

Maximasi

$$F = \sum_{i=1}^{n} p_i x_i$$

dengan batasan

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

dimana untuk setiap $i = 1,\ldots, n,$

$$0 \leq x_i \leq 1$$

Dimana P_i adalah subset dari berat bahan baku, w_i adalah harga barang, x_i menandakan terpilihnya bahan baku tersebut, serta K adalah total uang yang dimiliki untuk berbelanja.

Pada makalah ini contoh bahan baku yang kami gunakan adalah sebagai berikut.

Nama Bahan	Berat Bahan (/g)	Harga Bahan (Rp)	*Density* (Berat/Harga (g))
Beras	3000	36000	0,083
Telur	1000	20000	0,05
Tepung Terigu	750	10000	0,075
Daging Ayam	900	30000	0,03
Gula	200	13000	0,015

Tabel 2. Daftar bahan baku dan *density*

P : Berat bahan
H : Harga bahan = 50000
P/H : *Density*

Nama Bahan	Properti			*Greedy By*		
	P	H	P/H	P	H	P/H
Beras	3000	36000	0,083	1	0	1
Telur	1000	20000	0,05	0,3	1	0,2
Tepung Terigu	750	10000	0,075	0	1	1
Daging Ayam	900	30000	0,03	0	0,766	0
Gula	200	13000	0,015	0	1	0
Total harga bahan				50000	50000	50000
Total berat bahan				3300	2639,4	3950

Tabel 3. Daftar bahan baku dan pemodelan *greedy*

Perhitungan pada pemodelan di atas dilakukan pemodelan *greedy* berdasarkan berat bahan yang dipilih dengan mengutamakan berat objek paling besar dengan harga yang minimum. Sehingga hasil *greedy* berdasarkan berat bahan didapatkan :

a. Total harga bahan : 50000

b. Total berat bahan : 3300

Sedangkan saat dilakukan pemodelan *greedy* berdasarkan harga, mengutamakan objek yang memiliki harga paling rendah. Sehingga, hasil *greedy* berdasarkan harga bahan didapatkan :

a. Total harga bahan : 50000

b. Total berat bahan : 2639,4

Selanjutnya saat dilakukan pemodelan *greedy* berdasarkan *density*, mengutamakan objek yang memiliki *density* atau keuntungan paling besar. Sehingga, hasil *greedy* berdasarkan *density* didapatkan :

c. Total harga bahan : 50000

d. Total berat bahan : 4550

Berdasarkan ketiga pemodelan *greedy* tersebut, didapatkan solusi optimal dengan bahan maksimum dan harga minimum dari pemodelan *greedy* berdasarkan *density*, yaitu diperoleh total berat bahan 4550 g dan total uang yang dikeluarkan Rp50000,-.

2.2 Pemodelan Menggunakan Algoritma Branch and Bound

Pada makalah ini algoritma kedua yang kami gunakan adalah *Branch and Bound* untuk menyelesaikan persoalan optimasi dalam memaksimalkan fungsi objektif yang sesuai dengan *constraint*. Dalam merencanakan pembelian bahan baku tersebut, terdapat beberapa hal yang perlu diperhitungkan, yaitu harga, berat bahan, dan *density*.

Dalam algoritma *branch and bound* masalah ini merupakan masalah maximasi, sehingga ditinjau batas atas (*upper bound*) untuk persoalan ini.

Total biaya yang didapatkan oleh solusi dari permasalahan ini, tidak akan lebih besar daripada total biaya yang telah didapatkan dari solusi parsial yang terbentuk, ditambah dengan sisa kapasitas dikalikan dengan densitas terbesar objek yang belum terpilih.

Oleh karena itu, kita perlu mengurutkan bahan makanan yang ada berdasarkan *density*, secara menurun (*descending*), karena bahan makanan dengan *density* terbesar akan dipertimbangkan terlebih dahulu.

Berdasarkan persoalan tersebut, maka setiap bahan baku akan memiliki properti seperti tabel berikut.

Nama Bahan	Berat Bahan	Harga Bahan

Tabel 4. Tabel bahan baku

Rumus untuk mencari nilai batas atas (*upper bound*) yang kami gunakan adalah sebagai berikut.

$$ub = p_i + ((k - w_i) \times (\frac{w_{i+1}}{P_{i+1}}))$$

dimana,

p_i : berat bahan
K : jumlah uang
w_i : harga bahan

Contoh bahan makanan yang kami gunakan untuk laporan ini tercantum pada tabel berikut.

Nama Bahan	Berat Bahan (/g)	Harga Bahan (Rp)
Beras	3000	36000
Telur	1000	20000
Tepung Terigu	750	10000
Daging Ayam	900	30000
Gula	200	13000

Tabel 5. Daftar bahan baku

Berdasarkan tabel tersebut, akan dilakukan pencarian solusi menggunakan algoritma *branch and bound* dengan *Best-First Search*.

Langkah pertama yang kami lakukan adalah mengurutkan bahan makanan berdasarkan nilai *density* secara menurun *(descending)*.

1) **Beras**

$p_1 = 3000$

$w_1 = 36000$

$p_1/w_1 = 0{,}083$

2) **Tepung Terigu**

$p_2 = 750$

$w_2 = 10000$

$p_2/w_2 = 0{,}075$

3) **Telur**

$p_3 = 1000$

$w_3 = 20000$

$p_3/w_3 = 0{,}05$

4) **Daging Ayam**

$p_4 = 900$

$w_4 = 30000$

$p_4/w_4 = 0{,}03$

5) **Gula**

$p_5 = 200$

$w_5 = 13000$

$p_5/w_5 = 0{,}015$

Langkah kedua, menghitung nilai upper bound menggunakan rumus yang tertera di atas.

untuk daun ke-0 :

$$ub = 0 + ((50000 - 0)\ x\ (0.083))$$
$$= 4150$$

untuk daun ke-1:

$$ub = 3000 + ((50000 - 36000)\ x\ (750/10000))$$
$$= 3000 + (14000\ x\ 0,075)$$
$$= 4050$$
$$ub = 0 + ((50000 - 0)\ x\ (750/10000))$$
$$= 0 + (50000\ x\ 0,075)$$
$$= 3750$$

untuk daun ke-2 :

$$ub = 3750 + ((50000 - 46000)\ x\ (1000/20000))$$
$$= 3750 + (4000\ x\ 0,05)$$
$$= 3950$$
$$ub = 750 + ((50000 - 36000)\ x\ (1000/20000))$$
$$= 750 + (14000\ x\ 0,05)$$
$$= 1450$$

untuk daun ke-3 :

Karena nilai W > K maka tidak diambil.

$$ub = 3750 + ((50000 - 46000)\ x\ (0.03))$$

$= 3750 + (4000\ x\ 0,03)$

$= 3870$

untuk daun ke-4:

Karena nilai W > K maka tidak diambil.

$ub = 3750 + ((50000 - 46000)\ x\ (0.015))$

$= 3750 + (4000\ x\ 0,03)$

$= 3810$

untuk daun ke-5:

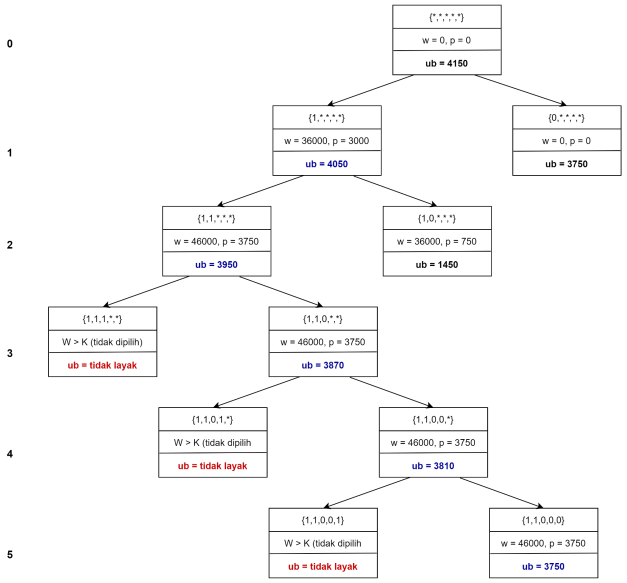
Karena nilai W > K maka tidak diambil.

$ub = 3750 + ((50000 - 46000)\ x\ (0))$

$= 3750 + (4000\ x\ 0)$

$= 3750$

Berikut ini *state-space-tree* menggunakan best-first-search.



Gambar 2. State-space-tree Algoritma Branch and Bound

Daun yang terpilih akan mempresentasikan solusi utuh, yang berarti daun lainnya tidak tidak menghasilkan solusi solusi atau dengan kata lain tidak ada nilai ub yang lebih besar dari 3750. Jadi, berdasarkan penjabaran solusi di atas, didapatkan solusi [1, 1, 0, 0, 0] dengan bahan yang terpilih yaitu beras dan tepung terigu dengan total berat bahan 3750 gram.

III. FUNGSIONALITAS PROGRAM

3.1 Fungsionalitas Program Menggunakan Algoritma Greedy

Fungsi *fractional knapsack* ini merupakan fungsi dari algoritma menggunakan *greedy* yang digunakan untuk mencari nilai optimal dari permasalahan perancangan bahan baku. Dimana tidak melebihi batas uang yang ditentukan dengan mendapatkan bahan baku maksimum.

```
def fractional_knapshack(value, weight, capacity):
  index = list(range(len(value)))
  ratio = [v/w for v, w in zip(value, weight)]
  index.sort(key=lambda i: ratio[i], reverse=True)

  max_value = 0
  fractions = [0]*len(value)
  for i in index:
    if weight[i] <= capacity:
      fractions[i] = 1
      max_value += value[i]
      capacity -= weight[i]
    else :
      fractions[i] = capacity/weight[i]
      max_value += value[i]*capacity/weight[i]
      break
  return max_value, fractions
```

Gambar 3. Source Code Algoritma Greedy

Dibawah ini merupakan program untuk meminta inputan dari user berupa jumlah bahan baku, berat, dan harga dari masing-masing bahan baku, serta total uang maksimum.

```
n = int(input('Enter number of items : '))
value = input('Enter the values of the {} items(s) in order : '.format(n)).split()
value = [int(v) for v in value]
weight = input('Enter the positive weights of the {} item(s) in order : '.format(n)).split()
weight = [int(w) for w in weight]
capacity = int(input('Enter maximum weight : '))
```

Gambar 4. Source Code Algoritma Greedy

```
max_value, fractions = fractional_knapshack(value, weight, capacity)
print('The maximum value of items that can be carried : ', max_value)
print('The Fractions in which the items should be taken : ', fractions)
```

Gambar 5. Source Code Algoritma Greedy

Berdasarkan daftar bahan pada tabel 2, maka didapatkan nilai optimal dari program tersebut sebagai berikut.

```
Masukkan banyak nya bahan baku : 5
Masukkan berat masing masing bahan baku : 3000 1000 750 900 200
Masukkan harga dari masing masing bahan baku : 36000 20000 10000 30000 13000
Masukkan total uang maksimum yang dimiliki : 50000
Nilai maksimal berat bahan yang dapat dibeli :  3950.0 gram
Nilai pembagian bahan yang diambil :  [1, 0.2, 1, 0, 0]
```

Gambar 6. Output dari Source Code Algoritma Greedy

3.2 Fungsionalitas Program Menggunakan Algoritma Branch and Bound

```
n = 5
W = 50000
p = [3000, 750, 1000, 900, 200]
w = [36000, 10000, 20000, 30000, 13000]
p_per_w = [0.083, 0.075, 0.05, 0.03, 0.015]


class Priority_Queue:
    def __init__(self):
        self.pqueue = []
        self.length = 0

    def insert(self, node):
        for i in self.pqueue:
            get_bound(i)
        i = 0
        while i < len(self.pqueue):
            if self.pqueue[i].bound > node.bound:
                break
            i+=1
        self.pqueue.insert(i,node)
        self.length += 1

    def print_pqueue(self):
        for i in list(range(len(self.pqueue))):
            print ("pqueue",i, "=", self.pqueue[i].bound)
```

Gambar 7. Source Code Algoritma Branch and Bound

```
    def remove(self):
        try:
            result = self.pqueue.pop()
            self.length -= 1
        except:
            print("Priority queue is empty, cannot pop from empty list.")
        else:
            return result

class Node:
    def __init__(self, level, profit, weight):
        self.level = level
        self.profit = profit
        self.weight = weight
        self.items = []
```

Gambar 8. Source Code Algoritma Branch and Bound

```
def get_bound(node):
    if node.weight >= W:
        return 0
    else:
        result = node.profit
        j = node.level + 1
        totweight = node.weight
        while j <= n-1 and totweight + w[j] <= W:
            totweight = totweight + w[j]
            result = result + p[j]
            j+=1
        k = j
        if k<=n-1:
            result = result + (W - totweight) * p_per_w[k]
        return result
```

Gambar 9. Source Code Algoritma Branch and Bound

```
nodes_generated = 0
pq = Priority_Queue()

v = Node(-1, 0, 0)
nodes_generated+=1
maxprofit = 0
v.bound = get_bound(v)

pq.insert(v)
```

Gambar 10. Source Code Algoritma Branch and Bound

```
while pq.length != 0:

    v = pq.remove()

    if v.bound > maxprofit:
        u = Node(0, 0, 0)
        nodes_generated+=1
        u.level = v.level + 1
        u.profit = v.profit + p[u.level]
        u.weight = v.weight + w[u.level]

        u.items = v.items.copy()
        u.items.append(u.level)
        if u.weight <= W and u.profit > maxprofit:
            maxprofit = u.profit
            bestitems = u.items
        u.bound = get_bound(u)
        if u.bound > maxprofit:
            pq.insert(u)
        u2 = Node(u.level, v.profit, v.weight)
        nodes_generated+=1
        u2.bound = get_bound(u2)
        u2.items = v.items.copy()
        if u2.bound > maxprofit:
            pq.insert(u2)
```

Gambar 11. Source Code Algoritma Branch and Bound

```
print("\nBerat Maksimal yang didapatkan = ", maxprofit)
print("Jumlah Node = ", nodes_generated)
print("Bahan yang terpilih = ", bestitems)
```

Gambar 12. Source Code Algoritma Branch and Bound

```
Berat Maksimal yang didapatkan =  3750
Jumlah Node =  11
Bahan yang terpilih =  [0, 1]
```

Gambar 13. Output dari Source Code Algoritma Branch and Bound

IV. HITUNG KOMPLEKSITAS WAKTU

4.1 Kompleksitas waktu algoritma Greedy

Pada setiap langkah, kita memilih (bagian) barang dengan *density* terbesar yang masih bisa dipilih, berdasarkan kapasitas *knapsack*, di antara semua barang yang belum dipilih.

Untuk mengoptimalkan proses pemilihan bahan berdasarkan *density*, maka bahan perlu diurutkan berdasarkan besar densitas secara menurun. Jika waktu pengurutan tidak dihitung, maka kompleksitas algoritma greedy by *density* untuk masalah *fractional knapsack* adalah $T(n) = O(n)$.

4.2 Kompleksitas waktu algoritma Branch and Bound

Pada setiap langkah, kita akan melakukan looping sebanyak n bahan dan dari setiap bahan memiliki 2 kemungkinan yaitu antara dipilih atau tidak dipilih untuk menjadi solusi optimal. Oleh karena itu nilai kompleksitas waktu algoritma yang didapatkan jika menggunakan Branch and Bound adalah $T(n) = O(n.2^n)$.

V. PENUTUP

5.1 Kesimpulan

Jadi, berdasarkan perbandingan kompleksitas waktu dari kedua algoritma tersebut, diperoleh nilai kompleksitas waktunya lebih rendah jika dibandingkan dengan algoritma *Branch and Bound*. Maka, algoritma yang lebih baik untuk digunakan dalam aplikasi perencanaan pembelian bahan baku adalah algoritma *Greedy*. Berdasarkan uji coba dua algoritma tersebut, juga didapatkan bahwa perolehan berat bahan yang optimal didapatkan jika menggunakan algoritma *Greedy Fractional Knapsack*.

VI. REFERENSI

[1] Dzakiy, M Irfan, “Aplikasi Strategi Algoritma Greedy Dalam Perencanaan Pembelian Bahan Baku di Pasar”, 2020.

[2] Slide Perkuliahan. “Slide CII2K3 Strategi Algoritma Greedy”, Prodi S1 Informatika: Telkom University.

[3] Slide Perkuliahan. “Slide CII2K3 Strategi Algoritma Branch and Bound”, Prodi S1 Informatika: Telkom University.

VII. LAMPIRAN

Video presentasi :
https://drive.google.com/file/d/1gp1QI_AFIGjM5lKHmojw9EzwZPHC5vg3/view?usp=sharing

Source code algoritma *greedy fractional knapsack* :
https://colab.research.google.com/drive/1IwEcZVSHQmhaQjjeHtM5g-NhYZbQW-AE?usp=sharing

Source code algoritma Branch and Bound :
https://colab.research.google.com/drive/1XcXKZMVQpCfL2WdGZPf4o6GUFusJRCov?usp=sharing

Slide presentasi :
https://www.canva.com/design/DAFEe1n3u_M/6koQRRo6l9as9jOpomw9iA/edit?utm_content=DAFEe1n3u_M&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton