Laporan Tugas Besar
Mata Kuliah Strategi Algoritma

Perbandingan Algoritma *Backtracking* dan *Branch and Bound* untuk Menyelesaikan Permasalahan *Smart Grid Environment*

Kelompok 12

1301218598	**Mohammad Andiez Satria Permana**
1301218601	**Wandi Yusuf Kurniawan**
1301218603	**HNW Syahuda Nahatmasuni**

S1 Informatika - Fakultas Informatika
Universitas Telkom
Bandung

2022

I. PENDAHULUAN

Knapsack Problem (KP) adalah masalah penempatan item (barang) ke dalam suatu tempat (biasa disebut Knapsack) yang mempunyai kapasitas tertentu, dimana setiap item memiliki berat dan nilai, sehingga total berat dari item-item yang ditempatkan tidak melebihi kapasitas Knapsack dan nilai yang didapatkan maksimum. {0,1}-*Knapsack Problem* ({0,1}-KP) adalah kasus khusus dari KP dimana setiap item hanya tersedia 1 unit, sehingga keputusannya adalah untuk memasukkan item tersebut ke dalam Knapsack (x=1) atau tidak (x=0). Beberapa solusi untuk menyelesaikan permasalahan ini adalah dengan menggunakan algoritma *backtracking* dan algoritma *branch and bound*.

Pada laporan ini, akan dibahas mengenai permasalahan *smart grid environment* dengan menerapkan *0/1 Knapsack* dan solusi algoritma untuk menyelesaikan permasalahan ini, serta menganalisis waktu yang dibutuhkan oleh masing-masing algoritma untuk menemukan solusinya. Algoritma yang digunakan adalah algoritma *backtracking* dan algoritma *branch and bound*.

II. DASAR TEORI

2.1 *0/1 Knapsack Problem*

0/1 Knapsack Problem merupakan jenis *knapsack problem* dimana hanya terdapat 1 buah dari setiap barang/item yang ada dan tidak bisa diambil fraksinya alias barang tetap utuh. Sehingga pilihannya hanya dua, yaitu dibawa (1) atau tidak dibawa (0). Memasukkan sejumlah barang yang memiliki bobot (w_i) dan keuntungan (p_i) tertentu ke dalam suatu karung dengan kapasitas maksimal *W*, sedemikian sehingga keuntungan yang didapat adalah maksimal.

Permasalahan knapsack 0-1 memiliki solusi penyelesaian yang dinyatakan sebagai himpunan:

$$X = x_1, x_2, x_3, \dots, x_n$$

dimana *xi* = 1 jika benda ke-*i* dimasukkan kedalam media penyimpanan, dan *xi* = 0 jika benda ke-*i* tidak dimasukkan kedalam media penyimpanan. Misalkan solusi dari suatu permasalahan knapsack adalah *X* = {0,1,0,1}, itu berarti benda ke-1 dan ke-3 tidak dimasukkan ke dalam media penyimpanan sedangkan benda ke-2 dan ke-4 dimasukkan ke dalam media penyimpanan.

2.2 Strategi Algoritma

Untuk menyelesaikan permasalahan *0/1 Knapsack Problem* ini ada dua strategi algoritma yang digunakan yaitu algoritma *backtracking* dan algoritma *branch and bound*.

A. Algortima *Backtracking*

Backtracking adalah algoritma yang berbasis pada *Depth First Search* (DFS) untuk mencari solusi persoalan lebih cepat. Runut balik yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan langkah ini tidak perlu memeriksa semua kemungkinan solusi yang ada. Akibatnya waktu pencarian dapat dihemat. *Backtracking* lebih alami disebut sebagai algoritma *rekursif*. Kadang disebutkan pula bahwa *Backtracking* merupakan bentuk tipikal dari algoritma *rekursif*.

Cara kerja algoritma *backtracking* pada permasalahan *0/1 Knapsack* adalah memasukkan sejumlah barang yang memiliki bobot (w_i) dan keuntungan (p_i) tertentu ke dalam suatu karung dengan kapasitas maksimal *W*, sedemikian sehingga keuntungan yang didapat adalah maksimal. Dengan ketentuan pohon ruang status sebagai berikut :

1. Tiap barang menempati level berbeda pada pohon.
2. Dari akar, kunjungi anak kiri untuk memasukkan barang ke karung; kunjungi anak kanan jika tidak ingin memasukkan barang ke karung. Lakukan untuk tiap level.
3. Kandidat solusi: jalur dari simpul akar ke simpul daun.

B. Algoritma Branch and Bound

Algoritma Branch and Bound merupakan algoritma yang membagi permasalahan menjadi sub masalah lebih kecil yang mengarah ke solusi dengan pencabangan (branching) dan melakukan pembatasan (bounding) untuk mencapai solusi optimal. Pencabangan (branching) yaitu proses membentuk permasalahan

ke dalam bentuk struktur pohon pencarian (search tree). Proses Pencabangan dilakukan untuk membangun semua cabang pohon yang menuju solusi, sedangkan proses pembatasan dilakukan dengan menghitung estimasi nilai (cost) simpul dengan memperhatikan batas.

Cara kerja algoritma *branch and bound* pada permasalahan *0/1 Knapsack* adalah dengan mengisi solusi per komponen untuk mendapatkan solusi parsial per tahapnya, artinya pada setiap langkah ke-*i,* kita meninjau objek ke-*i* sehingga level pohon menyatakan urutan objek, dan (simpul) anak menyatakan kemungkinan terpilih atau tidaknya objek tersebut.

2.3 *Smart Grid Environment*

Smart Grid Environment adalah bentuk jaringan dari alat - alat elektronik yang secara cerdas dapat mengintegrasikan masukan dari pengguna yang tersambung ke jaringan untuk kebutuhan yang tahan lama, murah, dan aman. Permintaan energi selama jam kerja meningkat rata - rata sebanyak 25% tiap tahunnya mulai dari tahun 1982. Dengan menggunakan *smart grid*, emisi gas CO_2 dan konsumsi energi rumah berkurang masing - masing sebanyak 9% dan 10%.

Knapsack digunakan pada smart grid untuk memberikan pilihan alat - alat elektronik apa saja yang dapat digunakan pada waktu itu. Knapsack memungkinkan masyarakat untuk menggunakan energi secara efisien dalam rangka mencapai tujuan mereka. Tidak hanya meminimalkan tagihan pelanggan tetapi memaksa mereka untuk menggunakan peralatan berat mereka selain jam sibuk. Diperkirakan kebutuhan energi secara keseluruhan di seluruh dunia meningkat 25 persen per tahun, jadi tantangan besar bagi kami untuk memenuhi permintaan.

III. IMPLEMENTASI

3.1 Penyelesaian Kasus *0/1 Knapsack Problem* dengan Strategi Algoritma *Backtracking*

Dalam *Smart Grid Environment* akan menggunakan strategi *backtrack* untuk menyelesaikan masalah 0/1 knapsack. Backtracking adalah algoritma

penyelesaian masalah secara rekursif dengan mencoba membangun solusi secara menaik satu bagian, dan kembali lagi ke bagian sebelumnya jika bagian yang diperiksa tidak memenuhi konstrain dari masalah yang akan diselesaikan. Strategi ini dapat direpresentasikan dengan *tree* dan didasarkan dengan algoritma pencarian *Depth First Search* (DFS). Dalam *0/1 Knapsack* dengan strategi ini, tree merepresentasikan barang yang dipilih atau tidak. Tiap level dari tree merepresentasikan barang dengan bobot dan keuntungan yang sudah diurutkan dari terkecil ke terbesar berdasarkan keuntungan dibagi dengan keuntungan ($\frac{p_i}{w_i}$).

DFS sendiri adalah algoritma untuk mencari data dalam graph atau tree dengan menelusuri node anak paling kiri terlebih dahulu hingga paling dalam dan kembali untuk memeriksa node anak kanannya. Dalam knapsack problem, DFS ini digunakan untuk menelusuri barang yang dipilih dengan tanda node anak kiri dan barang yang tidak dipilih dengan tanda node anak kanan pada setiap level. Dengan menggunakan bound untuk menentukan barang tersebut layak atau tidak untuk dimasukkan sebagai solusi. Jika tidak layak maka program akan kembali (backtracking) ke node sebelumnya dan menelusuri node anak lainnya. Rumus bound sendiri adalah sebagai berikut:

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

$$bound = (profit + \sum_{j=i+1}^{k-1} p_j) + (W - totweight) \times \frac{p_k}{w_k}$$

Dimana k adalah barang ke-k yang menyebabkan knapsack *overweight*.

Dalam code yang akan dijalankan memerlukan bantuan queue untuk merepresentasikan tree yang digunakan. *Code* untuk algoritma backtracking adalah sebagai berikut:

```
# Memilih cabang traversal DepthFirst untuk melanjutkan eksplorasi node
elif traversal == 'DepthFirst':  # strategi DepthFirst cenderung berhasil eksplorasi yang cukup dalam hingga mencapai dead end (jalan buntu).
    # Algoritma tidak akan pergi ke cabang kanan sebelum menyelesaikan semua simpul di cabang kiri
    if self.length >= 2:
        if self.q[-1].side == 'right' and self.q[-2].side == 'left':
            result = self.q.pop(self.length - 2)
        else:
            result = self.q.pop()
        self.length -= 1
        return result
    elif self.length == 1:
        result = self.q.pop()
        self.length -= 1
        return result
```

Code di atas akan eksplorasi ke node anak kiri terlebih dahulu hingga yang paling terdalam jika tidak memenuhi batasan yang ditentukan. Saat dijalankan terhadap array dari 15 alat elektronik seperti gambar berikut:

```
kapasitas = 300
berat = [82, 70, 25, 55, 32, 32, 38, 28, 30, 30, 65, 10, 25, 10, 1]
keuntungan = [208, 180, 158, 130, 118, 101, 100, 90, 77, 73, 66, 58, 50, 10, 1]
jumlah berat = 533
kepadatan = [2.537, 2.571, 6.32, 2.364, 3.688, 3.156, 2.632, 3.214, 2.567, 2.433, 1.015, 5.8, 2.0, 1.0, 1.0]
```

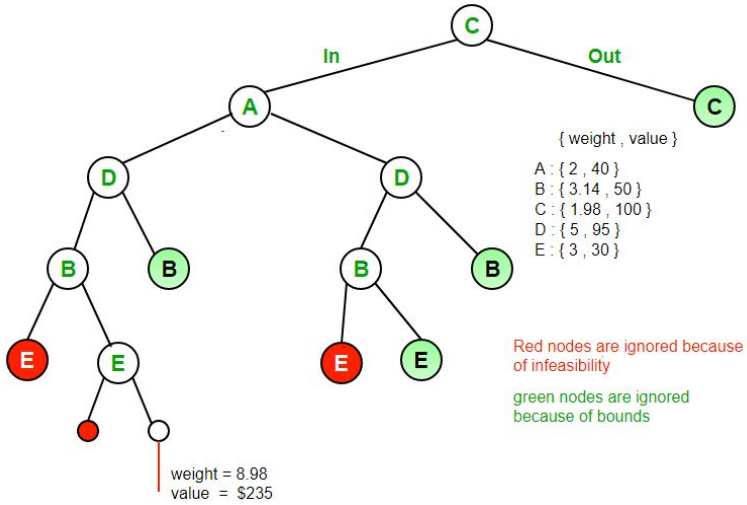
Program akan menghasilkan alat yang diambil untuk kebutuhan efisiensi penggunaan energi rumah seperti berikut:

```
Item yang bisa diambil adalah item ke-[1, 2, 4, 5, 6, 7, 8, 9, 11, 14]
Ada 10 dari 15 item yang bisa dimasukkan ke knapsack
Keuntungan maksimum yang bisa diraih adalah 956 dan jumlah simpul di pohon adalah 299
Waktu eksekusi adalah 0.01297 detik
```

3.2 Penyelesaian Kasus *0/1 Knapsack Problem* dengan Strategi Algoritma *Branch and Bound*

Pada masalah *Smart Grid Environment* dapat juga diselesaikan dengan menggunakan algoritma *branch and bound.* Algoritma *branch and bound* merupakan algoritma yang membagi permasalahan menjadi sub masalah yang lebih kecil yang mengarah ke solusi dengan pencabangan (*branching*) dan melakukan pembatasan (*bounding*) untuk mencapai solusi optimal. Dalam *0/1 Knapsack Problem* state space tree meninjau kapasitas K serta n objek, dimana setiap objek ke-i, memiliki berat w_i dan profit p_i . Solusi dari *0/1 Knapsack Problem* dapat ditulis sebagai $\{x1, x2, \dots, xn\}$, dimana $x_i = 0$ jika objek ke-i tidak terpilih dalam solusi, dan $x_i = 1$ jika objek ke-i masuk kedalam solusi. Dimana kita dapat mencari solusi dengan mengisi solusi per komponen untuk mendapatkan solusi parsial per tahapnya. Sehingga, level pohon menyatakan

urutan objek, dan simpul anak menyatakan kemungkinan terpilih atau tidaknya objek tersebut. Contoh state space tree akan terlihat seperti gambar berikut.



Sumber: https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/

Nilai *bound 0/1 Knapsack Problem* perlu kita tinjau *upper bound* karena kasus ini merupakan permasalahan maximasi. Total keuntungan yang didapat tidak akan lebih besar dari total biaya yang telah didapatkan dari solusi parsial yang terbentuk, ditambah dengan sisa kapasitas dikalikan densitas objek yang belum terpilih. Sehingga kita perlu mengurutkan objek yang ada berdasarkan densitas, secara menurun (karena objek dengan densitas terbesar sebaiknya kita pertimbangkan terlebih dahulu). Penyelesaian *Smart Grid Environment* dengan algoritma *branch and bound* dalam bentuk kodingan seperti berikut :

```
# Memilih cabang traversal BestFirst untuk melanjutkan eksplorasi node
if traversal == 'BestFirst':
    if self.length != 0:  # strategi BestFirst cenderung memilih node paling optimis untuk memulai eksplorasi
        result = self.q.pop()
        self.length -= 1
        return result

# Memilih cabang traversal BreadthFirst untuk melanjutkan eksplorasi node
elif traversal == 'BreadthFirst':  # strategi BreadthFirst cenderung memilih node yang paling lama berada dalam antrian
    if self.length != 0:
        result = self.q.pop(0)
        self.length -= 1
        return result
```

IV. ANALISIS

Disediakan 3 set array berat dan nilai/*profit* (dalam hal ini adalah energi) yang masing-masing berisi 10, 15, dan 25 barang elektronik serta kapasitas *knapsack* dengan rincian sebagai berikut:

Set 1:
weight = [70, 50, 48, 60, 32, 22, 50, 25, 10, 4]
value = [192, 162, 122, 110, 96, 82, 70, 63, 25, 5]
capacity = 200
Set 2:
weight = [82, 70, 25, 55, 32, 32, 38, 28, 30, 30, 65, 10, 25, 10, 1]
value = [208, 180, 158, 130, 118, 101, 100, 90, 77, 73, 66, 58, 50, 10, 1]
capacity = 300
Set 3:
weight = [80, 85, 72, 66, 55, 50, 40, 50, 22, 30, 40, 35, 25, 30, 50, 45, 60, 20, 20, 30, 20, 15, 10, 4, 2]
value = [220, 198, 185, 165, 160, 155, 125, 120, 115, 105, 100, 98, 95, 88, 80, 75, 72, 69, 65, 60, 56, 30, 15, 8, 3]
capacity = 500

Kemudian array tersebut dimasukkan dan diproses dalam sebuah program dengan menggunakan tiga algoritma, yaitu Depth First Search, Breadth First Search, dan Best First Search. Setelah selesai dieksekusi dari masing-masing set dan algoritma, semuanya berhasil menemukan solusi terbaik dengan rincian sebagai berikut:

	Backtracking (Depth First)	Branch and Bound (Breadth First)	Branch and Bound (Best First)
Set 1 (10 barang, kapasitas = 200)	Solusi = [1, 1, 0, 0, 1, 1, 0, 1, 0, 0] Keuntungan = 595		
Set 2 (15 barang, kapasitas = 300)	Solusi = [0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1] Keuntungan = 956		
Set 3 (25 barang, kapasitas = 500)	Solusi = [1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0] Keuntungan = 1536		

Selain itu, terdapat perbandingan penyelesaian masalah dari setiap strategi yang digunakan dapat berbeda-beda, untuk mengetahui kompleksitas waktu mana yang merupakan kompleksitas waktu terbaik dalam menyelesaikan *0/1 Knapsack Problem*. Dapat dilihat perbandingan waktu pada tabel berikut.

	Backtracking (Depth First)	Branch and Bound (Breadth First)	Branch and Bound (Best First)
Set 1 (10 barang, kapasitas = 200)	33 simpul 0.00500 detik	49 simpul 0.00500 detik	33 simpul 0.00300 detik
Set 2 (15 barang, kapasitas = 300)	299 simpul 0.00599 detik	393 simpul 0.02099 detik	87 simpul 0.00598 detik
Set 3 (25 barang, kapasitas = 500)	735 simpul 0.02896 detik	5473 simpul 0.30483 detik	89 simpul 0.00999 detik

Berdasarkan tabel di atas, algoritma Best First Search menjadi algoritma terbaik dalam penyelesaian masalah s*mart grid environment* karena memiliki jumlah simpul dan eksekusi waktu yang paling sedikit dibandingkan dua algoritma lainnya. Kompleksitas waktu terbaik untuk semua algoritma adalah O(n) karena cukup satu cabang yang dieksplor setiap levelnya. sedangkan yang terburuk bisa sampai O(2^n) karena semua cabang dieksplor sampai keturunan setiap levelnya lengkap.

V. KESIMPULAN

Masalah *smart grid environment* dengan *0/1 Knapsack* dapat diselesaikan dengan beberapa cara, contohnya dengan algoritma *backtracking* dan *branch and bound.* Keduanya akan menghasilkan hasil yang optimal. Namun, kompleksitas waktu dari kedua algoritma tersebut bisa jadi berbeda bergantung dengan jumlah barang, berat dan *profit* setiap barangnya, serta kapasitas knapsacknya.

DAFTAR PUSTAKA

Iryanto, M. P., & Mardiyati, S. (2017, February). Penyelesaian {0, 1}-Knapsack Problem dengan Algoritma Soccer League Competition. In *PRISMA, Prosiding Seminar Nasional Matematika* (pp. 688-700).

https://www.researchgate.net/publication/275195981_Home_Energy_Management_and_Knapsack_Technique_in_Smart_Grid_Environment

https://www.geeksforgeeks.org/backtracking-algorithms/

https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/

LAMPIRAN

Link Source Code : https://github.com/onedeetelyu/TubesSA