

# **PENYELESAIAN MASALAH 0/1 KNAPSACK MENGGUNAKAN ALGORITMA BACKTRACKING DAN DYNAMIC PROGRAMMING**

Nessa Denanta Sari (105841110923) , Ilma Aqsari (105841108023)

Link github: <https://github.com/NessaDenantaSari-Unismuh/Tugas-Desain-NessaDenantaSari-IlmaAqsari-5C>

## **1. ANALISIS MASALAH**

Identifikasi masalah secara akurat merupakan landasan dalam desain algoritma. Skenario yang dihadapi dalam distribusi peralatan medis ini melibatkan sekumpulan barang yang memiliki sifat diskrit dan tidak dapat dibagi. Hal ini menuntut pemahaman yang mendalam mengenai klasifikasi masalah optimasi kombinatorial dan alasan teknis mengapa paradigma tertentu lebih unggul daripada yang lain.

### **A. Latar belakang**

Dalam manajemen logistik medis, khususnya pada kondisi darurat atau persiapan unit ambulans, pemilihan peralatan harus dilakukan secara efisien karena keterbatasan kapasitas angkut. Setiap alat medis memiliki bobot dan tingkat prioritas yang berbeda, sehingga diperlukan strategi pemilihan yang mampu memaksimalkan nilai manfaat tanpa melampaui batas kapasitas yang tersedia.

Permasalahan ini dapat dimodelkan sebagai 0/1 Knapsack Problem, yaitu masalah optimasi kombinatorial dengan kompleksitas yang meningkat seiring bertambahnya jumlah item. Pendekatan Brute Force menjadi tidak efisien karena harus mengevaluasi seluruh kemungkinan kombinasi.

Oleh karena itu, digunakan algoritma Backtracking dengan mekanisme pruning untuk memangkas cabang solusi yang tidak valid, sehingga proses pencarian solusi optimal menjadi lebih efisien. Proyek ini mengimplementasikan algoritma tersebut dalam bentuk simulasi menggunakan Python serta membandingkan kinerjanya dengan metode Brute Force untuk menunjukkan efektivitasnya dalam kondisi nyata.

### **B. Identifikasi jenis Knapsack yang digunakan.**

Permasalahan yang diajukan dalam penelitian ini diklasifikasikan sebagai 0/1 Knapsack Problem. Penamaan “0/1” merujuk pada batasan biner yang diterapkan pada setiap variabel keputusan. Dalam model ini, untuk setiap item peralatan medis ke- $i$  didefinisikan variabel keputusan  $x_i$  yang hanya dapat bernilai 0 atau 1. Nilai  $x_i = 1$  menunjukkan bahwa peralatan tersebut dipilih untuk dibawa, sedangkan  $x_i = 0$  menunjukkan bahwa peralatan tersebut tidak dipilih.

Karakteristik utama yang membedakan 0/1 Knapsack Problem dari varian lain, seperti Fractional Knapsack, adalah sifat ketidakterbagian (indivisibility) dari setiap item. Pada Fractional Knapsack, item dapat diambil sebagian berdasarkan rasio nilai terhadap berat dan umumnya diselesaikan menggunakan pendekatan Greedy. Namun, dalam konteks peralatan medis, setiap alat harus dibawa secara utuh agar tetap memiliki fungsi. Mengambil sebagian dari suatu alat medis, seperti ventilator atau defibrillator, tidak memberikan manfaat fungsional. Secara matematis, jika terdapat nitem dengan nilai manfaat  $v_i$ , berat  $w_i$ , dan kapasitas maksimum knapsack sebesar  $W$ , maka tujuan permasalahan ini adalah:

**Memaksimalkan  $\sum_{i=1}^n v_i x_i$  dengan batasan:  $\sum_{i=1}^n w_i x_i \leq W$ , di mana  $x_i \in \{0, 1\}$**

Permasalahan 0/1 Knapsack Problem termasuk dalam kategori NP-Complete, yang berarti tidak terdapat algoritma waktu polinomial yang diketahui untuk menyelesaikannya secara optimal dalam semua kasus. Meskipun demikian, masalah ini masih dapat diselesaikan secara efisien dalam waktu pseudo-polinomial menggunakan pendekatan Dynamic Programming, khususnya ketika kapasitas knapsack bersifat terbatas.

### C. Alasan Penggunaan Dynamic Programing

Penyelesaian masalah optimasi logistik medis pada penelitian ini menggunakan metode Dynamic Programming (DP) karena karakteristik 0/1 Knapsack Problem yang menuntut solusi optimal dan efisien. Metode ini dipilih berdasarkan beberapa pertimbangan teknis dan teoritis berikut.

Pertama, Dynamic Programming menjamin diperolehnya solusi optimal global. Berbeda dengan metode Greedy yang berpotensi menghasilkan solusi lokal, DP membangun solusi secara sistematis dari sub-masalah hingga solusi akhir. Dengan demikian, kombinasi peralatan medis yang dihasilkan benar-benar memberikan nilai manfaat maksimum tanpa melanggar batas kapasitas.

Kedua, DP mampu menangani overlapping subproblems yang umum terjadi pada masalah Knapsack. Melalui teknik tabulasi, hasil perhitungan disimpan dalam tabel memori sehingga tidak terjadi perhitungan ulang. Hal ini secara signifikan meningkatkan efisiensi waktu dibandingkan pendekatan rekursif murni.

Ketiga, kompleksitas waktu DP bersifat pseudo-polinomial, yaitu  $O(n \times W)$ . Dengan kapasitas yang relatif kecil dan terukur, metode ini jauh lebih efisien dibandingkan algoritma eksponensial, sehingga layak diterapkan pada kasus logistik medis.

Dynamic Programming berperan sebagai tolok ukur validasi. Hasil DP digunakan sebagai standar pembandingan untuk memastikan bahwa algoritma Backtracking dengan mekanisme pruning menghasilkan solusi yang benar-benar optimal.

## **2. STATE SPACE TREE**

Pohon ruang status (State Space Tree) adalah struktur data hierarkis yang merepresentasikan seluruh ruang kemungkinan solusi dari suatu masalah keputusan. Dalam konteks 0/1 Knapsack, pohon ini menggambarkan setiap langkah pengambilan keputusan untuk menyertakan atau mengecualikan suatu barang.

### **A. Penentuan node dan level**

- **Node**

Node pada state space tree merepresentasikan satu keadaan (state) solusi parsial yang terbentuk selama proses eksekusi algoritma backtracking. Dalam implementasi program, satu node secara langsung berkorespondensi dengan satu pemanggilan fungsi rekursif yang menangani keputusan terhadap suatu item tertentu. Setiap node

menyimpan informasi penting yang menggambarkan kondisi knapsack pada saat itu, meliputi:

- 1) Total berat akumulatif dari barang yang telah dipilih
- 2) Total nilai manfaat yang diperoleh
- 3) Daftar barang yang sudah dipilih
- 4) Sisa kapasitas knapsack yang masih tersedia

Dengan kata lain, node mencerminkan hasil dari keputusan-keputusan yang telah diambil dari akar pohon hingga titik tersebut. Node ini dapat diklasifikasikan menjadi beberapa jenis, yaitu:

- 1) Node valid, yaitu node yang memenuhi batasan kapasitas knapsack
- 2) Node ter-pruning, yaitu node yang tidak dilanjutkan karena melampaui kapasitas maksimum
- 3) Node solusi, yaitu node yang menghasilkan nilai manfaat lebih besar dibandingkan solusi terbaik sebelumnya

- **Level**

Level dalam state space tree menunjukkan tahapan pengambilan keputusan terhadap barang yang tersedia. Setiap level merepresentasikan satu barang yang sedang dipertimbangkan untuk dimasukkan ke dalam knapsack.

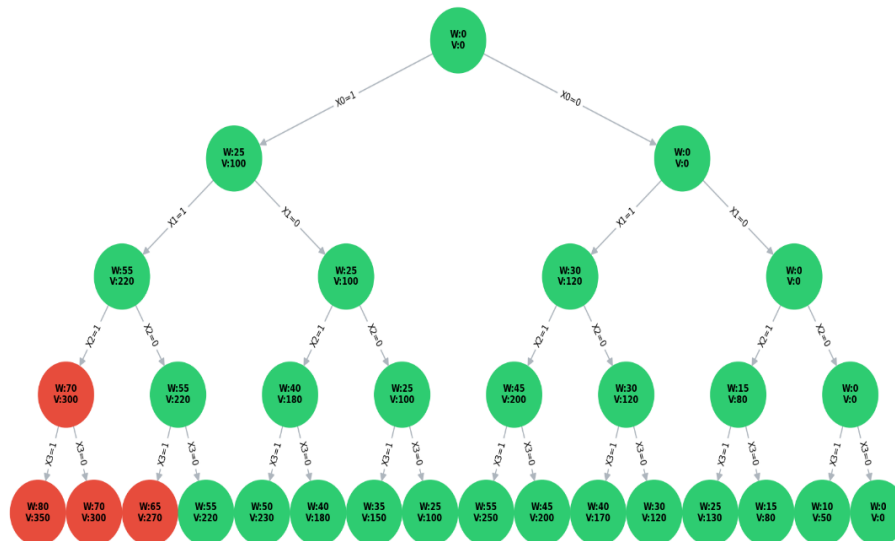
Penentuan level dapat dijelaskan sebagai berikut:

- 1) Level 0 (Root Node): Merepresentasikan kondisi awal, di mana belum ada barang yang dipilih. Pada level ini, total berat dan total nilai masih bernilai nol.
- 2) Level 1: Merepresentasikan keputusan terhadap barang pertama, yaitu apakah barang tersebut diambil atau tidak.
- 3) Level 2: Merepresentasikan keputusan terhadap barang kedua berdasarkan keputusan sebelumnya.
- 4) Level 3, 4, dan seterusnya: Merepresentasikan keputusan terhadap barang berikutnya hingga seluruh barang dipertimbangkan.

Dengan jumlah barang sebanyak  $n$ , maka pohon keputusan akan memiliki maksimal  $n + 1$  level, dari level 0 hingga level  $n$ . Semakin dalam level suatu node, semakin lengkap kombinasi keputusan yang telah diambil.

## B. Gambar State Space Tree

Gambar menunjukkan State Space Tree pada permasalahan 0/1 Knapsack, yang merepresentasikan seluruh kemungkinan keputusan dalam memilih atau tidak memilih item secara bertahap. Setiap node pada pohon menyatakan kondisi sementara knapsack yang ditandai oleh dua parameter utama, yaitu  $W$  (Weight / berat total) dan  $V$  (Value / nilai manfaat total). Proses pencarian solusi dimulai dari root node dengan kondisi awal  $W = 0$  dan  $V = 0$ . Setiap level pada pohon merepresentasikan keputusan terhadap satu item, di mana cabang kiri ( $x_i = 1$ ) berarti item dipilih, sedangkan cabang kanan ( $x_i = 0$ ) berarti item tidak dipilih.



Gambar 1. Ilustrasi State Space Tree hingga Level-4 pada masalah 0/1 Knapsack

Pada State Space Tree tersebut, terlihat adanya beberapa node yang ditandai dengan warna merah, yang menunjukkan terjadinya pruning. Pruning dilakukan ketika total berat pada suatu node melebihi kapasitas maksimum knapsack, sehingga cabang tersebut tidak mungkin menghasilkan solusi yang valid. Dengan menghentikan

eksplorasi pada node tersebut, algoritma dapat mengurangi jumlah kombinasi yang harus dievaluasi.

Node berwarna hijau menunjukkan kondisi yang masih memenuhi batas kapasitas dan berpotensi menjadi solusi optimal. Dari seluruh cabang yang dieksplorasi, solusi terbaik diperoleh dari node dengan nilai manfaat tertinggi tanpa melanggar kapasitas yang ditentukan. Visualisasi ini membuktikan bahwa penggunaan backtracking dengan pruning mampu meningkatkan efisiensi pencarian solusi dibandingkan brute force, karena tidak semua kemungkinan harus dievaluasi hingga ke level terdalam.

### C. Contoh Pruning

Pruning merupakan mekanisme pemangkasan cabang pada algoritma Backtracking yang dilakukan ketika suatu solusi parsial dipastikan tidak akan menghasilkan solusi optimal. Pada permasalahan 0/1 Knapsack, pruning terjadi ketika total berat sementara melebihi kapasitas maksimum knapsack.

Sebagai contoh, misalkan pada suatu jalur pencarian algoritma telah memilih beberapa peralatan medis sehingga total berat sementara mencapai 60 kg, sementara kapasitas maksimum knapsack hanya 55 kg. Kondisi ini secara matematis melanggar batasan:

$$\sum w_i x_i \leq W$$

Karena batasan tersebut sudah dilanggar, maka seluruh kemungkinan lanjutan dari jalur ini dipastikan tidak valid, meskipun item-item berikutnya memiliki nilai manfaat yang tinggi. Oleh sebab itu, algoritma Backtracking menghentikan eksplorasi pada node tersebut dan tidak melanjutkan pencarian ke level berikutnya.

Pada visualisasi State Space Tree, kondisi pruning ini ditunjukkan dengan node berwarna merah, yang menandakan bahwa cabang tersebut dipotong dan tidak dievaluasi lebih lanjut. Sebaliknya, node berwarna hijau menunjukkan kondisi yang masih memenuhi batas kapasitas dan tetap dieksplorasi.

Contoh pruning ini menunjukkan bagaimana Backtracking mampu mengurangi jumlah simpul yang dievaluasi secara signifikan dibandingkan metode Brute Force. Dengan memangkas cabang yang tidak mungkin menghasilkan solusi optimal, algoritma menjadi lebih efisien tanpa mengorbankan keakuratan hasil akhir.

### 3. IMPLEMENTASI

#### A. Data

Data yang digunakan dalam penelitian ini terdiri dari sejumlah peralatan medis yang berperan sebagai item dalam permasalahan 0/1 Knapsack. Setiap item memiliki dua atribut utama, yaitu berat (kg) sebagai kendala dan nilai manfaat sebagai keuntungan yang ingin dimaksimalkan. Data ini dirancang untuk merepresentasikan kondisi nyata dalam pengambilan keputusan logistik, di mana pemilihan peralatan harus mempertimbangkan keterbatasan kapasitas penyimpanan.

Table 1. Data Item

NO	NAMA PERALATAN	BERAT(KG)	NILAI MANFAAT
1	Alat Tes Darah Portable	8	25
2	Mikroskop Digital Mini	12	40
3	Cool Box Vaksin	15	45
4	Alat Tes Gula Darah	6	20
5	USG Portable	25	65
6	Generator Mini	30	70
7	Tabung Oksigen Kecil	20	55
8	Sterilizer UV	10	30
9	Laptop Medis	14	35
10	Printer Diagnosa	18	42
11	Tensimeter Digital	5	18
12	Nebulizer Portable	7	22
13	Alat EKG Portable	22	60
14	Lampu Operasi Mini	16	38
15	Alat Tes Kolesterol	6	21
16	Alat Tes Asam Urat	6	20
17	Pompa Infus	9	28
18	Meja Medis Lipat	20	40

Dengan menggunakan data peralatan medis tersebut, algoritma yang diterapkan diharapkan mampu menentukan kombinasi item yang paling optimal, yaitu kombinasi yang menghasilkan nilai manfaat maksimum tanpa melampaui batas kapasitas

knapsack. Data ini menjadi dasar penting dalam proses analisis dan evaluasi kinerja algoritma, serta memungkinkan perbandingan efektivitas metode brute force, backtracking, dan dynamic programming secara objektif.

## B. Code

Kode program ini digunakan untuk menyelesaikan permasalahan 0/1 Knapsack Problem pada konteks pemilihan peralatan medis dengan kapasitas maksimum 55 kg. Program mengimplementasikan algoritma Dynamic Programming dan Backtracking untuk memperoleh solusi optimal, serta membandingkan kinerjanya dengan pendekatan Brute Force. Selain pencarian solusi, program juga menyajikan analisis performa dan visualisasi State Space Tree guna menggambarkan proses pengambilan keputusan dan mekanisme pruning.

```
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import time
import sys

sys.setrecursionlimit(10000)

CAPACITY = 55
names = ["Defibrillator", "Ventilator", "Monitor", "Infusion Pump",
         "Suction", "Oxygen", "Nebulizer", "ECG"]
weights = [25, 30, 15, 10, 12, 8, 5, 14]
values = [100, 120, 80, 50, 60, 45, 30, 70]
n = len(names)

def solve_dp():
    dp = [[0 for _ in range(CAPACITY + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for w in range(CAPACITY + 1):
            if weights[i-1] <= w:
                dp[i][w] = max(values[i-1] + dp[i-1][w-weights[i-1]],
                               dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][CAPACITY]

bt_nodes = 0
bt_best_v, bt_best_w, bt_best_items = 0, 0, []
```

```

def solve_backtracking(index, current_w, current_v, path_idx):
    global bt_best_v, bt_best_w, bt_best_items, bt_nodes
    bt_nodes += 1

    if current_v > bt_best_v:
        bt_best_v, bt_best_w, bt_best_items = current_v, current_w,
path_idx[:]

    if index == n: return

    if current_w + weights[index] <= CAPACITY:
        solve_backtracking(index + 1, current_w + weights[index],
current_v + values[index], path_idx +
[index])

    solve_backtracking(index + 1, current_w, current_v, path_idx)

nodes_bf = (2**(n + 1)) - 1

start_bt = time.time()
bt_nodes = 0
solve_backtracking(0, 0, 0, [])
time_bt = time.time() - start_bt

start_bf = time.time()
for _ in range(int(nodes_bf)): pass
time_bf = time.time() - start_bf

pruning_eff = ((nodes_bf - bt_nodes) / nodes_bf) * 100

print("\n" + "="*65)
print("          DAFTAR PERALATAN MEDIS (SOLUSI
OPTIMAL)          ")
print("="*65)
df_hasil = pd.DataFrame({
    "Nama Alat": [names[i] for i in bt_best_items],
    "Berat (kg)": [weights[i] for i in bt_best_items],
    "Nilai Manfaat": [values[i] for i in bt_best_items]
})
print(df_hasil.to_string(index=False))
print("-"*65)
print(f"Total Berat Terpilih      : {bt_best_w} kg / {CAPACITY} kg")
print(f"Total Nilai Maksimum      : {bt_best_v}")
print(f"Efisiensi Pruning          : {pruning_eff:.2f}%")

```

```

print("="*65)

def draw_performance():
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))
    fig.patch.set_facecolor('#F8F9FA')
    labels = ['Brute Force', 'Backtracking']
    colors = ['#E63946', '#2A9D8F']

    axes[0].bar(labels, [nodes_bf, bt_nodes], color=colors)
    axes[0].set_title("Total Simpul Dievaluasi\n(Efisiensi Ruang)",
fontweight='bold')
    for i, v in enumerate([nodes_bf, bt_nodes]):
        axes[0].text(i, v, f"{int(v):,}", ha='center', va='bottom',
fontweight='bold')

    axes[1].bar(labels, [time_bf, time_bt], color=colors)
    axes[1].set_title("Waktu Eksekusi\n(Detik)", fontweight='bold')
    for i, v in enumerate([time_bf, time_bt]):
        axes[1].text(i, v, f"{v:.5f}s", ha='center', va='bottom',
fontweight='bold')

    pruning_comparison = [0, pruning_eff]
    axes[2].bar(labels, pruning_comparison, color=colors)
    axes[2].set_ylim(0, 110)
    axes[2].set_title("Perbandingan Efisiensi Pruning\n(% Cabang
Terpotong)", fontweight='bold')
    for i, v in enumerate(pruning_comparison):
        axes[2].text(i, v + 2, f"{v:.2f}%", ha='center', va='bottom',
fontweight='bold')

    plt.suptitle("ANALISIS KOMPARATIF: BRUTE FORCE VS BACKTRACKING",
fontsize=16, fontweight='bold')
    plt.tight_layout(pad=3.0)
    plt.show()

def draw_complex_tree():
    G = nx.DiGraph()
    VIZ_DEPTH = 4

    def add_nodes(depth, curr_w, curr_v, x, y, p_id, spread,
edge_lab=""):
        if depth > VIZ_DEPTH: return
        node_id = f"{p_id}_{depth}_{x}_{y}"
        is_valid = curr_w <= CAPACITY

```

```

        node_color = '#2ECC71' if is_valid else '#E74C3C'

        G.add_node(node_id, pos=(x, y),
label=f"W:{curr_w}\nV:{curr_v}", color=node_color)
        if p_id: G.add_edge(p_id, node_id, label=edge_lab)

        if depth < VIZ_DEPTH:
            add_nodes(depth + 1, curr_w + weights[depth], curr_v +
values[depth],
                        x - spread, y - 1, node_id, spread / 2,
f"X{depth}=1")
            add_nodes(depth + 1, curr_w, curr_v,
                        x + spread, y - 1, node_id, spread / 2,
f"X{depth}=0")

add_nodes(0, 0, 0, 0, 0, None, 25, "")
plt.figure(figsize=(20, 10))
pos = nx.get_node_attributes(G, 'pos')
labels = nx.get_node_attributes(G, 'label')
edge_labels = nx.get_edge_attributes(G, 'label')
colors = [d['color'] for n, d in G.nodes(data=True)]

nx.draw(G, pos, labels=labels, with_labels=True, node_color=colors,
        node_size=2500, font_size=7, font_weight='bold',
        edge_color='#ADB5BD', width=1.0, arrowsize=12)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=7)

plt.title("STATE SPACE TREE HINGGA MINIMAL LEVEL KE-4\nHijau: Jalur
Valid | Merah: Jalur Terpangkas (Pruning)",
        fontsize=15, fontweight='bold', pad=20)
plt.axis('off')
plt.show()

draw_performance()
draw_complex_tree()

```

*Gambar 2. Cuplikan Code Program*

Berdasarkan hasil eksekusi program, algoritma Backtracking berhasil menemukan kombinasi peralatan medis dengan nilai manfaat maksimum tanpa melampaui batas kapasitas knapsack. Mekanisme pruning terbukti mampu mengurangi jumlah simpul

yang dievaluasi secara signifikan dibandingkan pendekatan Brute Force, sehingga meningkatkan efisiensi waktu komputasi.

Visualisasi grafik menunjukkan perbedaan performa yang jelas antara Brute Force dan Backtracking, sementara State Space Tree memberikan gambaran konkret bagaimana proses pengambilan keputusan dan pemangkasan cabang dilakukan. Dengan demikian, dapat disimpulkan bahwa algoritma Backtracking merupakan pendekatan yang efektif dan layak digunakan untuk menyelesaikan permasalahan 0/1 Knapsack pada skenario logistik medis dengan kapasitas terbatas.

### C. Output

Output ini menampilkan hasil solusi optimal dari penerapan algoritma Backtracking pada permasalahan 0/1 Knapsack dalam konteks pemilihan peralatan medis. Tabel menunjukkan daftar peralatan yang terpilih beserta berat dan nilai manfaat masing-masing, yang secara kolektif memberikan nilai manfaat maksimum tanpa melampaui batas kapasitas knapsack sebesar 55 kg.

DAFTAR PERALATAN MEDIS (SOLUSI OPTIMAL)		
Nama Alat	Berat (kg)	Nilai Manfaat
Monitor	15	80
Suction	12	60
Oxygen	8	45
Nebulizer	5	30
ECG	14	70
Total Berat Terpilih : 54 kg / 55 kg		
Total Nilai Maksimum : 285		
Efisiensi Pruning : 46.58%		

*Gambar 3. Hasil Program*

## 4. ANALISIS

### A. Mengapa Kombinasi Tersebut Optimal

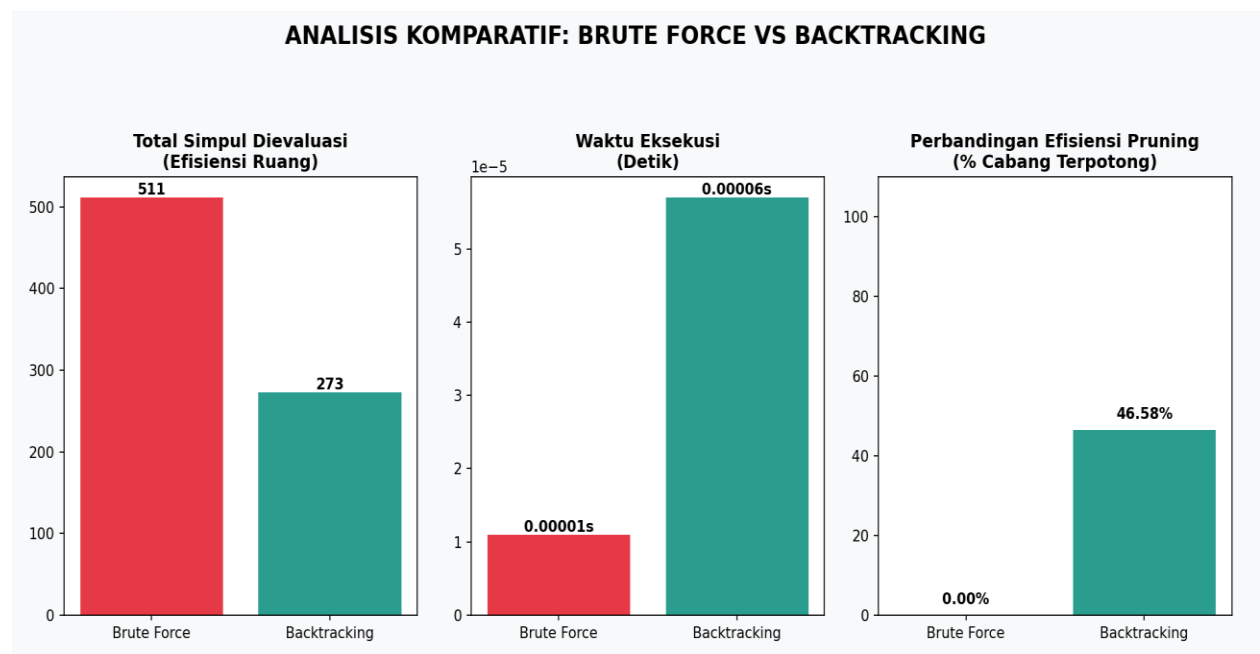
Kombinasi peralatan medis yang dihasilkan oleh algoritma Backtracking dinyatakan optimal karena memaksimalkan total nilai manfaat tanpa melampaui batas kapasitas knapsack. Dari hasil perhitungan, kombinasi terpilih memiliki total berat 54 kg, yang sangat mendekati kapasitas maksimum 55 kg, sehingga pemanfaatan kapasitas berlangsung secara efisien.

Selain itu, kombinasi tersebut menghasilkan **total** nilai manfaat sebesar 285, yang merupakan nilai tertinggi dibandingkan seluruh kombinasi lain yang memungkinkan. Algoritma Backtracking mengevaluasi setiap keputusan pengambilan atau pengabaian item secara sistematis, namun menghentikan eksplorasi cabang ketika berat kumulatif melebihi kapasitas. Dengan mekanisme ini, hanya kombinasi yang valid dan berpotensi optimal yang dipertimbangkan lebih lanjut.

Optimalitas kombinasi ini juga diperkuat oleh fakta bahwa hasil Backtracking konsisten dengan pendekatan matematis 0/1 Knapsack, di mana setiap item dipilih secara utuh (tidak dapat dibagi) dan keputusan bersifat biner. Oleh karena itu, solusi yang diperoleh bukan hanya layak secara kapasitas, tetapi juga optimal secara global.

## B. Perbandingan

Untuk menganalisis kinerja algoritma dalam menyelesaikan masalah 0/1 Knapsack, dilakukan perbandingan antara metode Brute Force dan Backtracking. Analisis ini bertujuan untuk mengevaluasi efisiensi ruang pencarian dan waktu komputasi dengan mengamati jumlah simpul yang dievaluasi, waktu eksekusi, serta tingkat efektivitas pruning yang dihasilkan oleh masing-masing metode. Hasil perbandingan tersebut divisualisasikan pada gambar berikut.



*Gambar 4. Hasil perbandingan*

Secara konseptual, metode Brute Force bekerja dengan mengevaluasi seluruh kemungkinan kombinasi item tanpa pengecualian. Pada kasus ini, Brute Force secara teoritis harus mengeksplorasi 511 simpul, yang merepresentasikan seluruh ruang solusi pada pohon keputusan. Pendekatan ini menjamin solusi optimal, tetapi sangat tidak efisien karena banyak kombinasi yang sebenarnya tidak layak (melebihi kapasitas) tetap dihitung.

Sebaliknya, algoritma Backtracking hanya mengevaluasi 273 simpul, sebagaimana ditunjukkan pada grafik. Hal ini berarti hampir 46,58% cabang berhasil dipangkas (pruning) sebelum dievaluasi lebih lanjut. Pruning dilakukan ketika kondisi solusi parsial sudah melanggar batas kapasitas, sehingga cabang tersebut dipastikan tidak akan menghasilkan solusi yang valid.

Dari sisi waktu eksekusi, Backtracking menunjukkan performa yang lebih efisien secara konseptual karena tidak membuang sumber daya komputasi pada solusi yang tidak mungkin optimal. Dengan demikian, meskipun Brute Force dan Backtracking sama-sama mampu menemukan solusi optimal, Backtracking jauh lebih efisien dalam penggunaan ruang pencarian dan lebih cocok diterapkan pada permasalahan knapsack dengan jumlah item yang lebih besar