**Jurnal Kajian dan Terapan Matematika**
**Volume 11, Edisi 1, Bulan Maret 2025, 64 – 72**
**http://journal.student.uny.ac.id/ojs/index.php/jktm**
**http://dx.doi.org/10.21831/jktm.v11i1.22844**

# OPTIMASI MASALAH KNAPSACK 0-1 DENGAN MENGGUNAKAN PEMROGRAMAN DINAMIS

## *OPTIMIZATION OF THE 0-1 KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING*

Zahra Zharifah Anwar *, Program Studi Matematika Universitas Negeri Yogyakarta, Indonesia
Caturiyati, Program Studi Matematika Universitas Negeri Yogyakarta, Indonesia
*e-mail korespondensi: zahrazharifah.2020@student.uny.ac.id

**Abstrak**
Optimasi merupakan aspek yang mendasar dalam pengambilan keputusan, di mana tujuannya adalah untuk mengidentifikasi solusi optimal dari sekumpulan alternatif yang kompleks. Contoh masalah sehari-hari yang memerlukan optimasi adalah masalah pemuatan barang (masalah knapsack), yang berdampak langsung pada efisiensi operasi logistik dan pada akhirnya berpengaruh pada biaya operasional. Salah satu variasi utama dalam masalah knapsack adalah masalah knapsack 0-1. Setiap item pada masalah knapsack 0-1 memiliki pilihan untuk dimuat ke dalam knapsack atau tidak. Masalah knapsack 0-1 merupakan masalah optimasi kombinatorial yang paling banyak dipelajari. Sebagai masalah yang tergolong NP-hard, masalah knapsack 0-1 memiliki tingkat kerumitan yang tinggi. Salah satu algoritma penyelesaian masalah knapsack 0-1 adalah pemrograman dinamis.
**Kata kunci:** masalah knapsack, masalah knapsack 0-1, np-hard, pemrograman dinamis.

***Abstract***
*Optimization is a fundamental aspect of decision-making, where the goal is to identify the optimal solution from a complex set of alternatives. An example of an everyday problem that requires optimization is the loading of goods problem (knapsack problem), which has a direct impact on the efficiency of logistics operations and ultimately on operational costs. One of the main variations of the knapsack problem is the 0-1 knapsack problem. Each item in the 0-1 knapsack problem has the option to be loaded into the knapsack or not. The 0-1 knapsack problem is the most studied combinatorial optimization problem. As an NP-hard problem, the 0-1 knapsack problem has a high level of complexity. One of the algorithms for solving the 0-1 knapsack problem is dynamic programming.*
***Keywords**: knapsack problem, knapsack problem 0-1, np-hard, dynamic programming.*

## PENDAHULUAN

Masalah pengambilan keputusan sering kali melibatkan proses optimasi dengan mencari solusi terbaik dari semua kemungkinan yang ada. Optimasi merupakan salah satu cabang dari riset operasi yang menyediakan metode penyelesaian untuk memecahkan suatu masalah (Della Croce et al., 2017). Salah satu contoh dalam kehidupan sehari-hari yang membutuhkan proses optimasi yaitu pada masalah pemuatan barang. Masalah pemuatan barang membutuhkan proses optimasi karena secara langsung memiliki dampak pada efisiensi pemuatan logistik yang berpengaruh pada biaya operasional (Cattaruzza et al., 2017). Dalam pemuatan barang ke truk, kontainer, atau pesawat secara efisien akan memaksimalkan pemanfaatan ruang dan memastikan barang yang akan dimuat dapat terpilih sesuai kebutuhan, serta meningkatkan perlindungan terhadap barang yang bersifat rapuh. Masalah ini dipelajari dalam ilmu optimasi sebagai masalah *knapsack* (Eilon & Christofides, 1971).

Pada tahun 1990 muncul sebuah persepsi umum yang menyatakan bahwa definisi dasar dari masalah *knapsack* sangat sederhana dan mudah dipahami oleh siapa pun (Kellerer et al., 2004). Bagi pengamat awam, masalah ini mungkin tidak terlihat sebagai tantangan penelitian

yang signifikan atau membutuhkan studi yang mendalam. Namun, persepsi ini telah ditantang oleh berbagai penelitian. Para peneliti telah mengeksplorasi kompleksitas masalah *knapsack* dengan memperluasnya ke dimensi yang lebih tinggi, menambahkan beberapa kendala, memperkenalkan beberapa *knapsack*, dan memodifikasi struktur set item atau fungsi objektif (Kellerer et al., 2004). Perluasan dan variasi ini memiliki implikasi praktis dan telah memicu penelitian yang intensif, yang mengungkapkan bahwa masalah ini jauh lebih rumit dan bervariasi dari yang terlihat pertama kali. Terlepas dari definisinya yang sederhana, masalah *knapsack* terus menjadi area yang kaya akan penelitian, menghadirkan banyak variasi dan tantangan yang relevan dengan konteks teoritis dan terapan.

Masalah *knapsack* diterapkan secara luas pada skenario dunia nyata karena prinsip utamanya yaitu mengoptimalkan kapasitas ruang yang terbatas untuk mendapatkan profit maksimal sangat relevan di berbagai bidang (Angelelli et al., 2010). Masalah *knapsack* dapat diterapkan dalam manajemen operasi, termasuk di dalamnya adalah berbagai bidang seperti investasi modal, pemotongan bahan baku, pemuatan kargo, dan berbagai aplikasi lainnya (Kellerer et al., 2004). Hal ini menunjukkan bagaimana prinsip dari masalah *knapsack* sangat penting untuk membuat keputusan yang tepat untuk mencapai hasil yang optimal dalam menyelesaikan persoalan pengambilan keputusan dengan batasan kapasitas di kehidupan nyata.

Penerapan prinsip-prinsip masalah *knapsack* dalam masalah optimasi sangat dibutuhkan karena membantu dalam membuat keputusan yang optimal ketika kapasitas ruang terbatas dan pilihannya kompleks (Cook et al., 2009). Meskipun dalam pembuatan keputusan dapat menggunakan akal sehat, penerapan prinsip-prinsip masalah *knapsack* memastikan pendekatan yang lebih sistematis dan optimal, terutama pada pengambilan keputusan yang melibatkan banyak variabel dan kendala (Taha, 2007). Memahami masalah *knapsack* dapat membantu menghindari inefisiensi dengan menyediakan strategi sistematis (Vanderbei, 2020). Selain itu, dapat meningkatkan kemampuan teoritis dengan memperdalam pengetahuan tentang teori optimasi, algoritma, dan kompleksitas komputasi, yang sangat penting untuk majukan penelitian. Sedangkan secara praktis, pemahaman ini menghasilkan keterampilan pengambilan keputusan yang lebih baik.

Masalah *knapsack* sama halnya masalah *knapsack* 0-1, memiliki pengertian yang sama yaitu memilih sejumlah item untuk dimuat ke dalam *knapsack* dengan kapasitas terbatas supaya total profit maksimum tercapai, dengan setiap item hanya dapat dipilih satu atau tidak sama sekali. Namun, dalam masalah nyata, tidak hanya memilih sejumlah item ke dalam suatu *knapsack* saja tetapi ada kendala tambahan lainnya. Faktor-faktor tambahan ini menyebabkan beberapa variasi dan perluasan dari masalah *knapsack* (Kellerer et al., 2004). Sehingga dari perspektif matematis, berbagai variasi masalah *knapsack* dapat dilihat sebagai generalisasi dari masalah *knapsack* 0-1, karena memperluas batasan atau tujuannya.

Masalah *knapsack* 0-1 adalah dasar dari banyak variasi masalah *knapsack*. Masalah ini mempelajari ide dasar dalam memilih item untuk memaksimalkan profit dengan tetap berada dalam batas kapasitas. Prinsip-prinsip ini berlaku untuk sebagian variasi, bahkan ketika ada kompleksitas baru yang ditambahkan, seperti beberapa *knapsack* atau batasan tambahan. Memahami masalah *knapsack* 0-1 adalah kunci untuk menguasai berbagai variasi lainnya.

## METODE

Penelitian ini bertujuan untuk mengoptimalkan penyelesaian masalah knapsack 0-1 menggunakan pendekatan pemrograman dinamis. Secara umum, metode penelitian ini meliputi tahapan sebagai berikut:

1. Studi Literatur

Tahap awal penelitian dilakukan melalui studi literatur terhadap konsep dasar masalah knapsack 0-1, karakteristik masalah optimasi kombinatorial, konsep NP-hard, serta prinsip-prinsip pemrograman dinamis. Studi literatur juga meliputi pengumpulan referensi dari buku teks, artikel jurnal internasional, dan sumber terpercaya lainnya yang relevan dengan topik penelitian.

2. Perumusan Model Matematika Masalah Knapsack 0-1

Permasalahan knapsack 0-1 diformulasikan secara matematis dengan mempertimbangkan:

- Variabel keputusan biner $x_i$, di mana $x_i = 1$ jika item iii dimasukkan ke dalam knapsack dan $x_i = 0$ jika tidak.
- Fungsi objektif untuk memaksimalkan total profit dari item-item yang dipilih.
- Kendala berupa kapasitas maksimum knapsack yang tidak boleh dilampaui.

Model matematika umum yang digunakan adalah:

$$\text{Maximize } \sum_{i=1}^{n} p_i\, x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_i x_i \le C, \quad x_i \in (0,1)$$

di mana $p_i$ adalah profit item i, $w_i$ adalah berat item i, dan C adalah kapasitas knapsack.

3. Penerapan Algoritma Pemrograman Dinamis

Penyelesaian masalah dilakukan dengan menerapkan algoritma pemrograman dinamis, yang melibatkan langkah-langkah berikut:

- Menyusun tabel dua dimensi yang mencatat nilai optimal untuk setiap submasalah, yaitu untuk setiap jumlah item dan kapasitas knapsack tertentu.
- Menggunakan rumus rekursif Bellman untuk mengisi tabel:

$$V(i,j) = \begin{cases} V(i-1,j), & ,jika\ wi > j \\ max\ (V(i-1,j), V(i-1,j-wi) + pi) & ,jika\ wi \le j \end{cases}$$

Inisialisasi nilai awal untuk kapasitas 0 dan tanpa item, kemudian mengisi tabel secara bertahap.

4. Simulasi Studi Kasus

Untuk memperjelas penerapan metode, diberikan studi kasus berupa pemilihan kombinasi barang tambahan yang akan dibawa dalam ransel dengan kapasitas terbatas, sebagaimana dicontohkan pada kegiatan hiking di SMP Indonesia Terpadu. Simulasi ini dilakukan baik secara manual menggunakan tabel perhitungan, maupun secara komputasional menggunakan bahasa pemrograman Python, berdasarkan algoritma pseudocode yang telah dirumuskan.

5. Validasi dan Interpretasi Hasil

Hasil solusi yang diperoleh dari perhitungan manual dan implementasi program Python dibandingkan untuk memastikan konsistensi. Interpretasi hasil meliputi:

- Identifikasi kombinasi item yang memberikan profit maksimum.
- Verifikasi bahwa solusi memenuhi batasan kapasitas knapsack.
- Evaluasi efisiensi metode pemrograman dinamis dalam menyelesaikan masalah optimasi berukuran kecil hingga menengah.

Metode ini dipilih karena pemrograman dinamis menawarkan pendekatan sistematis untuk memperoleh solusi optimal dengan membangun solusi dari submasalah-submasalah yang lebih kecil, sesuai dengan karakteristik masalah knapsack 0-1.

## HASIL DAN PEMBAHASAN

Masalah *knapsack* pertama kali dipublikasikan pada tahun 1957. George Dantzig, seorang pelopor dalam pemrograman linear dan riset operasi, mengemukakan bahwa versi kontinu dari masalah *knapsack* dapat diselesaikan dengan memilih item berdasarkan rasio nilai terhadap beratnya (Dantzig, 1957). Richard Bellman, seorang tokoh penting dalam riset operasi, memperkenalkan pemrograman dinamis untuk memecahkan masalah *knapsack*, yang mirip dengan menemukan jalur terpanjang dalam sebuah jaringan. Pada tahun 1980-an, para peneliti mengembangkan cara-cara yang lebih cepat untuk menyelesaikan masalah *knapsack*. Pada tahun 1975, Sahni menciptakan salah satu skema pendekatan polinomial pertama untuk masalah *knapsack* (Zhang & Zhang, 1999). Skema aproksimasi polinomial adalah algoritma untuk masalah optimasi yang sulit.

Perkembangan penelitian seputar masalah *knapsack* menjadi salah satu perhatian yang cukup komprehensif bagi para peneliti sejak buku yang ditulis oleh Martello dan Toth pada tahun 1990 diterbitkan (Kellerer et al., 2004). Para peneliti kemudian menangani masalah *knapsack* yang lebih besar. Penelitian lanjutan dilakukan dengan menggunakan masalah *knapsack* sebagai dasar untuk pengembangan algoritma solusi yang baru dan metode komputasi. Sejak saat itu, masalah *knapsack* digunakan dalam manajemen operasi, termasuk investasi modal, pemotongan bahan baku, pemuatan kargo, dan aplikasi lainnya (Kellerer et al., 2004).

Asal mula atau bentuk paling dasar dari banyak masalah optimasi dikenal sebagai masalah *knapsack* (Martello & Toth, 1990). Seringkali masalah *knapsack* disebut juga sebagai masalah *knapsack* 0-1 atau masalah *knapsack* dasar. Masalah ini dianggap sebagai "nenek moyang" karena menjadi dasar dari banyaknya variasi masalah *knapsack* yang lebih kompleks. Masalah yang sederhana namun mendasar ini telah memunculkan banyak perluasan variasi masalah yang dibangun di atas prinsip-prinsip dasarnya.

Masalah *knapsack* 0-1 dapat diadaptasi dengan mengubah parameter tertentu (Soukaina et al., 2018). Misalnya, dengan melibatkan penjumlahan jumlah item yang harus dipertimbangkan, jumlah tujuan yang harus dipenuhi, atau jumlah *knapsack* yang tersedia untuk digunakan. Dengan menyesuaikan parameter-parameter ini, berbagai variasi masalah dapat dibuat. Masing-masing dengan tantangan dan kompleksitas yang beragam. Fleksibilitas ini memungkinkan masalah *knapsack* 0-1 untuk diadaptasi ke berbagai skenario. Sehingga dapat diterapkan ke berbagai tantangan dunia nyata.

Setiap item pada masalah *knapsack* 0-1 memiliki pilihan untuk dimuat ke dalam *knapsack* atau tidak. Oleh karena itu, disebut "0-1" karena memiliki keputusan yang bersifat biner. Artinya, setiap item berada dalam salah satu dari dua kondisi. Kondisi tersebut mengasumsikan keterpilihan item untuk dipilih sepenuhnya atau tidak (Zhou et al., 2023). Variabel keputusan pada suatu item akan bernilai satu apabila item tersebut terpilih untuk dimuat ke dalam *knapsack* dan nol apabila tidak.

Pemilihan setiap item dibatasi dengan menyertakan atau mengecualikan setiap item secara keseluruhan (Pan & Zhang, 2018; Soukaina et al., 2018). Item yang dimuat ke dalam *knapsack* tetap dalam keadaan utuh dan bukan merupakan sebagian dari item tersebut. Tidak

ada pilihan untuk mengambil sebagian kecil dari suatu item untuk dimuat ke dalam *knapsack*. Apabila sebuah item tidak dapat sepenuhnya masuk ke dalam ruang yang tersisa di dalam *knapsack*, item tersebut tidak diperbolehkan untuk diambil sebagian saja dan harus memutuskan untuk tidak akan mengambil seluruh item. Sehingga, dalam masalah *knapsack* 0-1, tidak ada item yang dimuat dalam bentuk pecahan (Pisinger, 1999).

Model masalah masalah *knapsack* 0-1 diformulasikan sebagai masalah optimasi di mana terdapat sekumpulan $n$ item, masing-masing dengan profit $p_j$ dan berat $w_j$ $(j = 1,2, \dots, n)$ tertentu, harus dipilih untuk memaksimalkan total profit tanpa melebihi kapasitas *knapsack* $c$ (Kellerer et al., 2004). Dalam model ini, variabel keputusan biner $x_j$ digunakan, dimana $x_j = 1$ menunjukkan bahwa item $j$ dimuat ke dalam *knapsack* dan $x_j = 0$ jika tidak. Fungsi objektif didefinisikan untuk memaksimalkan total dari profit item yang termuat di dalam *knapsack*. Sedangkan kendalanya pada berat total item yang dipilih, untuk memastikan bahwa beratnya tidak melebihi kapasitas *knapsack*. Selain itu, pada item yang akan dipilih, tidak dapat dibagi atau diambil dalam jumlah pecahan. Hal ini mengarah pada tantangan untuk memilih kombinasi item terbaik untuk profit maksimum. Model matematika pada masalah *knapsack* 0-1 dapat diformulasikan sebagai berikut.

memaksimumkan

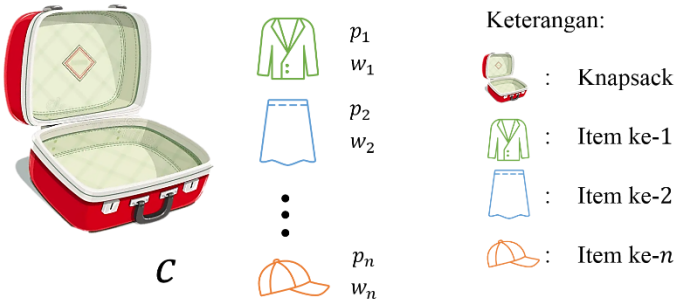
$$\sum_{j=1}^{n} p_j x_j$$

tergantung pada

$$\sum_{j=1}^{n} w_j x_j \leq c$$

$$x_j = \{0,1\} \qquad j = 1, 2, \dots, n$$

Bentuk masalah yang dapat dikatakan sebagai masalah *knapsack* 0-1 ditunjukkan pada gambar berikut.




**Gambar 1.** Ilustrasi Masalah Knapsack 0-1

Gambar tersebut menunjukkan sekumpulan item yang akan dipilih untuk dimuat ke dalam kanpsack. Tujuannya untuk menghasilkan profit maksimum tanpa melebihi kapasitas *knapsack* $c$.

Menurut (Kellerer et al., 2004) terdapat beberapa asumsi yang harus dipertimbangkan dalam menyelesaikan masalah *knapsack* 0-1, diantaranya sebagai berikut.

1. Jumlah item

   Masalah *knapsack* hanya akan bermakna jika setidaknya ada dua item $n \geq 2$. Jika masalahnya hanya melibatkan satu item, akan menghasilkan solusi trivial di mana pilihan biner sederhana (memasukkan atau mengeluarkan item) sudah cukup.

2. Batasan berat item

   Berat setiap item $w_j$ tidak boleh melebihi kapasitas *knapsack*, yaitu $w_j \leq c$ untuk semua item $j$. Jika ada item yang beratnya melebihi kapasitas, secara otomatis iem tersebut dikeluarkan dari pertimbangan dengan mengatur variabel keputusan $x_j$ ke 0. Selain itu, jika berat gabungan dari semua item kurang dari atau sama dengan kapasitas *knapsack*, yaitu $\sum_{j=1}^{n} w_j \leq c$, solusinya trivial karena semua item dapat masuk ke dalam *knapsack* tanpa melanggar batasan kapasitas.
3. Profit dan berat non-negatif

   Semua profit item $p_j$ dan berat $w_j$ diasumsikan positif, yaitu $p_j > 0$ dan $w_j > 0$ untuk semua item $j$. Hal ini untuk menghindari kasus dimana ada item yang memiliki profit nol atau negatif, yang tidak akan memberikan profit atau mengurangi profit total. Namun, Jika ada item yang memiliki profit atau berat yang tidak positif, maka masalahnya bisa diubah untuk mengecualikan item-item tersebut atau menanganinya dengan cara yang berbeda.
   a. Jika $p_j = 0$ dan $w_j \leq 0$, maka item tersebut dikeluarkan karena tidak memberikan kontribusi pada profit maupun berat yang berarti.
   b. Jika $p_j \leq 0$ dan $w_j > 0$, item tersebut dikeluarkan karena Memasukkan item ini tidak akan memberikan keuntungan atau justru mengurangi total keuntungan, namun tetap menambah bobot pada ransel. Sehingga, lebih baik untuk tidak memasukkan item ini.
   c. Jika $p_j < 0$ dan $w_j \leq 0$, item tersebut dapat dimasukkan hanya jika membantu meningkatkan profit secara keseluruhan dengan menghilangkan dampak negatifnya.
4. Perumusan ulang masalah

   Untuk menangani kasus dengan koefisien negatif atau nol, sebuah formulasi ulang disarankan dengan menggunakan satu set variabel biner $y_j$ yang baru. Dalam formulasi ini, $y_j = 0$ mengindikasikan bahwa sebuah item dimasukkan ke dalam *knapsack*, sedangkan $y_j = 1$ berarti item tersebut dikeluarkan. Transformasi ini membantu mengubah semua koefisien menjadi nilai positif, sehingga menyederhanakan proses optimasi.
5. Menangani koefisien rasional

   Jika profit atau berat diberikan sebagai bilangan rasional (pecahan), maka bisa dikalikan dengan penyebut yang sama untuk mengubahnya menjadi bilangan bulat.

Masalah *knapsack* 0-1 adalah jenis masalah optimasi sulit yang dikenal sebagai masalah NP-*hard* (*Non-deterministic Polynomial-time hard*) (Kellerer et al., 2004). Akibatnya, tidak ada algoritma efisien yang dapat menghitung solusi optimal dalam waktu yang singkat (Scheithauer, 2018). Ketika ukuran masalah meningkat, tidak ada algoritma yang dijamin dapat menyelesaikan masalah ini secara efisien dalam jangka waktu yang sesuai. Oleh karena itu, algoritma penyelesaian dibutuhkan sebagai pemecah masalah karena solusi praktis harus ditemukan untuk aplikasi kehidupan nyata. Meskipun solusi terbaik tidak selalu bisa didapatkan, perkiraan yang baik dapat diberikan oleh algoritma ini, sehingga menjadikannya penting dalam praktiknya.

Berbagai algoritma solusi telah dikembangkan untuk mengatasi kompleksitas masalah *knapsack* 0-1. Mulai dari algoritma eksak hingga pendekatan heuristik dan metaheuristik (Jalali Varnamkhasti, 2012). Algoritma eksak, seperti pemrograman dinamis, efektif untuk kasus-kasus berukuran kecil hingga menengah, tetapi kesulitan dengan masalah yang lebih besar karena peningkatan eksponensial dalam waktu komputasi. Artinya, seiring bertambahnya jumlah item atau kendala, waktu yang dibutuhkan untuk menyelesaikan masalah meningkat secara signifikan.

Pemrograman dinamis adalah pendekatan umum yang dapat digunakan di berbagai bidang riset operasi. Hal ini sangat berguna dalam masalah-masalah dimana solusi optimal dapat dibangun dengan menggabungkan solusi optimal dari submasalah (Bellman, 1960). Ketika diterapkan pada masalah *knapsack* 0-1, pendekatan ini melibatkan pembagian masalah

yang lebih besar menjadi submasalah yang lebih kecil dan dapat dikelola dan menyelesaikan setiap submasalah secara iteratif untuk membangun solusi akhir (Kellerer et al., 2004).

Diasumsikan bahwa solusi opimal telah dihitung untuk sebuah subset item dan semua kapasitas sampai dengan $c$. Kemudian, jika terdapat item baru yang ditambahkan, maka akan dilakukan pengecekan apakah solusi optimal sebelumnya perlu diperbarui untuk subset yang diperbesar. Sebuah submasalah dari masalah *knapsack* 0-1, dinotasikan sebagai $KP_j(g)$, dengan demikian dapat didefinisikan untuk himpunan item $\{1,..,j\}$ dan kapasitas *knapsack* $g \leq c$. Tujuannya adalah untuk memaksimalkan nilai total dari item-item yang dikemas sambil memastikan berat total tidak melebihi kapasitas $g$. Untuk setiap kapasitas dan himpunan item, solusinya akan dihitung secara rekursif menggunakan rumus rekursi Bellman, sebagai berikut.

$$z_j(g) = \begin{cases} z_{j-1}(g) & g < w_j \\ \max\{z_{j-1}(g), z_{j-1}(g - w_j) + p_j\} & g \geq w_j \end{cases}$$

Perulangan ini memastikan bahwa akan ada kemungkinan item $j$ dikeluarkan dari *knapsack* jika tidak sesuai atau mempertimbangkan apakah memasukkannya akan meningkatkan total profit. Proses iteratif ini membangun solusi optimal untuk masalah *knapsack* penuh dari sub-sub masalah.

Dengan menginisialisasi solusi untuk nol item dan secara bertahap memasukkan lebih banyak item, pada akhirnya akan didapatkan solusi optimal untuk keseluruhan masalah. Untuk menyelesaikan masalah *knapsack* 0-1 menggunakan pemrograman dinamis, prosesnya dimulai dengan menetapkan nilai solusi ke nol untuk nol item dan untuk setiap kapasitas *knapsack* yang mungkin dari 0 hingga kapasitas maksimum $c$ (Dreyfus, 2002). Selanjutnya, rumus rekursif diterapkan untuk menghitung nilai solusi selangkah demi selangkah, dengan mempertimbangkan setiap item secara progresif, mulai dari item pertama hingga item ke-$n$. Pendekatan sistematis ini memastikan bahwa nilai solusi seara keseluruhan untuk masalah *knapsack* 0-1, yang dinotasikan sebagai $z_n(c)$, yang mewakili nilai maksimum yang dapat dicapai ketika semua item dan kapasitas maksimum diperhitungkan.

Algoritma untuk pendekatan ini, yang disebut sebagai Algoritma Pemrograman Dinamis, dijelaskan pada Gambar 2.

**Pemrograman Dinamis**

for $g \coloneqq 0$ to $c$ do
    $z_0(g) \coloneqq 0$  // inisialisasi
for $j \coloneqq 1$ to $n$ do
    for $g \coloneqq 0$ to $w_j - 1$ do  //item $j$ terlalu besar unuk dimuat
        $z_j(g) \coloneqq z_{j-1}(g)$
    for $d \coloneqq w_j$ to $c$ do  //item $j$ dapat dimuat
        if $z_{j-1}(g - w_j) + p_j > z_{j-1}(g)$ then
            $z_j(g) \coloneqq z_{j-1}(g - w_j) + p_j$
        else $z_j(g) \coloneqq z_{j-1}(g)$
$z^* \coloneqq z_n(c)$

**Gambar 2.** Pseudocode Pemrograman Dinamis untuk Masalah Knapsack 0-1 (Sumber: Kellerer et al., 2004)

Berikut ini akan diberikan contoh penerapan pemrograman dinamis pada masalah *knapsack* 0-1 yang akan diselesaikan secara manual dan dengan menggunakan Python, sesuai dengan Pseudocode pada Gambar 2.

**Contoh 1.** Dalam memperingati acara HUT Kemerdekaan RI, SMP Indonesia Terpadu mengadakan kegiatan hiking. Setiap siswa masing-masing dibagi dalam beberapa kelompok dan setiap kelompok mendapatkan dua buah ransel. Barang-barang yang dibutuhkan selama kegiatan hiking telah disediakan oleh panitia dimuat dalam ransel pertama. Kemudian setiap

kelompok diperbolehkan untuk membawa barang tambahan pada ransel kedua dengan kapasitas berat maksimum 10 kg. Daftar barang tambahan yang boleh dibawa telah disediakan, masing-masing dengan berat dan profit tertentu. Berikut ini daftar barang tambahan yang perlu dipertimbangkan oleh setiap kelompok.

1. Bahan makan: profit = 34, berat = 4 kg
2. Peralatan masak: profit = 23, berat 4 kg
3. Bahan bakar: profit = 27, berat = 5 kg
4. Minuman: profit = 20, berat 3 kg

Kombinasi barang apa saja pada ransel kedua agar suatu kelompok mendapatkan profit maksimum dan berat tidak melebihi kapasitas ransel?

Telah diketahui bahwa terdapat *knapsack* dengan kapasitas $c = 10$ dan $n = 4$ dengan profit dan berat sebagai berikut:

**Tabel 1.** Data Awal Masalah Pada Pemrograman Dinamis

| $j$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $p_j$ | 34 | 23 | 27 | 20 |
| $w_j$ | 4 | 4 | 5 | 3 |

Dengan menggunakan penyelesaian secara menual, akan dihitung solusi optimal $z_j(g)$ dengan algoritma pemrograman dinamis. Komputasi dilakukan secara per baris dari kiri ke kanan dan di dalam baris dari atas ke bawah. Kolom ke-0 dan baris ke-0 diinisialisasi dengan entri yang sama dengan 0. Nilai-nilai di dalam kotak sesuai dengan profit dari himpunan solusi optimal.

**Tabel 2.** Nilai Solusi Optimal Pada Pemrograman Dinamis

| $j\backslash g$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 34 | 34 | 34 | 34 | 34 | 34 | 34 |
| 2 | 0 | 0 | 0 | 0 | 34 | 34 | 34 | 34 | 57 | 57 | 57 |
| 3 | 0 | 0 | 0 | 0 | 34 | 34 | 34 | 34 | 57 | 61 | 61 |
| 4 | 0 | 0 | 0 | 20 | 34 | 34 | 34 | 54 | 57 | 61 | 61 |

Penyelesaian algoritma pemrograman dinamis memilih item 1 dan 3 untuk dimuat ke dalam *knapsack* dengan total berat 9 kg dan menghasilkan profit sebanyak 61.

Hasil dari penyelesaian masalah Contoh 1 menggunakan Python diperoleh item yang terpilih yaitu "Bahan bakar" dan "Bahan makan". Berat gabungan dari barang-barang ini adalah 9 kg, yang berada di bawah batas maksimum kapasitas *knapsack*, sehingga memungkinkan untuk dibawa tanpa melebihi kapasitas. Total profit dari barang-barang yang dipilih ini adalah 61. Baik penyelesaian manual maupun menggunakan Python menghasilkan hasil yang sama.

## SIMPULAN

Masalah knapsack adalah salah satu masalah yang paling sering dipelajari dalam riset operasi. Masalah knapsack 0-1 atau masalah knapsack dasar, yang juga dikenal dengan sebutan biner. Munculnya variasi dari masalah knapsack merupakan bentuk generalisasi dari masalah knapsack 0-1. Dalam menyelesaikan masalah knapsack 0-1, pemrograman dinamis menjamin optimal dengan menyelesaikan submasalah secara iteratif, meskipun membutuhkan lebih banyak sumber daya komputasi.

**DAFTAR PUSTAKA**

Angelelli, E., Mansini, R., & Grazia Speranza, M. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. Computers and Operations Research, 37(11), 2017–2026. https://doi.org/10.1016/j.cor.2010.02.002

Bellman, R. (1960). Sequential Machines, Ambiguity, and Dynamic Programming. Journal of the ACM, 7(1), 24–28. https://doi.org/10.1145/321008.321011

Cattaruzza, D., Absi, N., Feillet, D., & González-Feliu, J. (2017). Vehicle routing problems for city logistics. EURO Journal on Transportation and Logistics, 6(1), 51–79. https://doi.org/10.1007/s13676-014-0074-0

Cook, W. J., Lovász, L., & Vygen, J. (2009). Research trends in combinatorial optimization: Bonn 2008. In Research Trends in Combinatorial Optimization: Bonn 2008. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-76796-1

Dantzig, G. B. (1957). Discrete-Variable Extremum Problems. Operations Research, 5(2), 266–288. https://doi.org/10.1287/opre.5.2.266

Della Croce, F., Salassa, F., & Scatamacchia, R. (2017). An exact approach for the 0–1 knapsack problem with setups. Computers & Operations Research, 80, 61–67. https://doi.org/10.1016/j.cor.2016.11.015

Dreyfus, S. (2002). Richard Bellman on the Birth of Dynamic Programming. Operations Research, 50(1), 48–51. https://doi.org/10.1287/opre.50.1.48.17791

Eilon, S., & Christofides, N. (1971). The Loading Problem. Management Science, 17(5), 259–268. https://doi.org/10.1287/mnsc.17.5.259

Jalali Varnamkhasti, M. (2012). Overview of the Algorithms for Solving the Multidimensional Knapsack Problems. In Advanced Studies in Biology (Vol. 4, Issue 1).

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Knapsack Problems. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24777-7

Martello, S., & Toth, P. (1990). Knapsack Problem Algorithms and Computer Implementations. John Wiley & Sons.

Pan, X., & Zhang, T. (2018). Comparison and Analysis of Algorithms for the 0/1 Knapsack Problem. Journal of Physics: Conference Series, 1069(1). https://doi.org/10.1088/1742-6596/1069/1/012024

Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. European Journal of Operational Research, 114(3), 528–541. https://doi.org/10.1016/S0377-2217(98)00120-9

Scheithauer, G. (2018). Knapsack problems. In International Series in Operations Research and Management Science (Vol. 263, pp. 19–45). Springer New York LLC. https://doi.org/10.1007/978-3-319-64403-5_2

Soukaina, L., Mohamed, N., Hassan, E. A., & Boujemâa, A. (2018, May 2). A hybrid genetic algorithm for solving 0/1 knapsack problem. ACM International Conference Proceeding Series. https://doi.org/10.1145/3230905.3230907

Taha, H. A. (2007). Operation Research: An Introduction. New Jersey : Pearson Education Inc., 2007.

Vanderbei, R. J. (2020). Linear Programming (Vol. 285). Springer International Publishing. https://doi.org/10.1007/978-3-030-39415-8

Zhang, L., & Zhang, Y. (1999). Approximation for knapsack problems with multiple constraints. Journal of Computer Science and Technology, 14(4), 289–297. https://doi.org/10.1007/BF02948730

Zhou, Y., Shi, Y., Wei, Y., Luo, Q., & Tang, Z. (2023). Nature-inspired algorithms for 0-1 knapsack problem: A survey. Neurocomputing, 554. https://doi.org/10.1016/j.neucom.2023.126630