BAB II
TINJAUAN PUSTAKA

2.1 Logistik dan Layanan Ekspedisi

Logistik merupakan suatu upaya yang mencakup proses perencanaan, implementasi, hingga pengawasan pada dalam kegiatan distribusi produk barang atau jasa dari pemasok sampai kepada konsumen. Upaya ini dilakukan dengan untuk memastikan ketersediaan produk barang serta menjamin pengiriman yang tepat waktu kepada penerima. Agar sistem logistik dapat berjalan dengan efektif dan efisien, diperlukan beberapa komponen penting yang saling mendukung, meliputi lokasi, transportasi, manajemen pengadaan persediaan, komunikasi, dan penyimpanan.

Salah satu bagian penting dalam sistem loistik adalah layanan ekspedisi. Ekspedisi merupakan suatu layanan atau jasa yang dapat membantu pengiriman barang dari satu lokasi ke lokasi lainnya (Santoso, Indrasari, Komari, & Tripariyanto, 2023). Layanan ini berperan sebagai jembatan antara penjual dan pembeli agar barang yang dipesan dapat tiba tepat waktu dalam kondisi aman dan biaya yang yang efisien.

Dalam kegiatan pengiriman barang tersebut, pengelolaan distribusi sering kali berkaitan dengan keterbatasan kapasitas tranportasi dan kebutuhan untuk memaksimalkan efisiensi pengiriman. Permasalahan tersebut dapat dimodelkan sebagai permasalahan *knapsack*. Dimana permasalahan *knapsack* ini dapat digunakan untuk menentukan kombinasi barang yang terbaik dengan mempertimbangkan batas kapasitas yang tersedia (Prayoga & Sormin, 2025).

2.2 Normalisasi Data (*Linear Sum Normalization*)

Dalam permasalahan optimasi yang melibatkan lebih dari satu kriteria, setiap kriteria sering kali memiliki satuan dan rentang nilai yang berbeda. Perbedaan kriteria ini dapat memicu bias, di mana kriteria dengan angka nominal besar akan mendominasi hasil akhir secara tidak seimbang. Oleh karena itu, diperlukan proses normalisasi untuk menyetarakan seluruh

skala data ke dalam rentang yang sebanding tanpa menghilangkan informasi pentimg yang ada di dalamnya.

Salah satu pendekatan normalisasi yang efektif adalah *Linear Sum Normalization* (*LSN*). Menurut Rozkowska (2026), *LSN* merupakan metode yang mereduksi skala nilai dengan cara membandingkannya terhadap total kumulatif nilai pada kriteria yang bersangkutan. Nilai yang besar akan tetap memberikan pengaruh yang dominan pada hasil normalisasi. Nilai yang lebih kecil juga tidak akan terabaikan, melainkan tetap berkontribusi secara adil sesuai dengan porsinya.

Pada kriteria keuntungan, dimana nilai hasil normalisasi diperoleh dengan membagi nilai alternatif individu dengan total keseluruhan nilai pada kriteria tersebut. Secara matematis, metode ini dirumuskan sebagai:

$$n_{ij} = \frac{x_{ij}}{\sum_{i=1}^{m} x_{ij}} \tag{2.2.1}$$

dengan

n_{ij} = nilai hasil normalisai alternatif ke- *i* pada kriteria ke- *j*

x_{ij} = nilai awal alternatif ke- i pada kriteria ke- *j*

m = banyaknya alternatif

Secara teknis, salah satu ciri khas metode *LSN* adalah menghasilkan nilai ternormalisasi yang jika dijumlahkan pada setiap kriteria akan selalu bernilai sama dengan satu. Dengan demikian, setiap nilai hasil normalisasi secara langsung mencerminkan seberapa besar proporsi atau andil suatu alternatif terhadap total keseluruhan nilai pada kriteria tersebut. Oleh karena itu, metode ini banyak digunkaan pada model agregasi berbobot (*weighted aggregation models*). Hal ini dikarenakan mampu menghasilkan data dengan skala yang sebanding, sehingga sesuai untuk proses pembobotan dan penggabungan berbagai kriteria.

Pada penelitian ini, metode *LSN* digunakan untuk menormalisasi variabel nilai barang dan berat barang. Kedua variabel tersebut memiliki

satuan yang berbeda, yaitu rupiah (Rp) dan kilogram (kg), sehingga tidak dapat dioperasikan secara langsung dalam fungsi objektif. Melalui proses normalisasi, kedua variabel diubah menjadi nilai tak berdimensi (*dimensionless*). Nilai tersebut merepresentasikan kontribusi relatif masing – masing terhadap total nilai pada kriterianya. Dengan demikian, hasil normalisasi dapat diberikan bobot sesuai preferensi pengambilan keputusan. Selanjutnya digunakan sebagai masukan dalam algoritma pemrograman dinamis. Secara matematis, data normalisasi didapatkan dengan rumus:

$$vt = \frac{v_i}{\sum v} \tag{2.2.2}$$

$$wt = \frac{w_i}{\sum w} \tag{2.2.3}$$

dengan keterangan:

vt = nilai barang yang telah dinormalisasi
v_i = nilai barang ke- i
$\sum v$ = total nilai barang
wt = berat barang ynag telah dinormalisasi
w_i = berat barang ke- i
$\sum w$ = total berat barang

Persamaan tersebut menunjukkan bahwa setiap nilai dan berat barang dinormalisasi dengan membagi nilai awal terhadap jumlah total pada masing – masing kriteria. Hasil normalisasi berupa nilai relatif pada rentang 0 hingga 1. Dengan demikian, kedua variabel menjadi sebanding dan dapat digunakan dalam proses pembobotan tanpa dipengaruhi oleh perbedaan satuan pengukuran.

2.3 Metode Pembobotan

Metode pembobotan merupakan metode dalam pengambilan keputusan dengan memberikan bobot pada masing – masing fungsi *objektif* yang melibatkan berbagai faktor. Pada metode ini, bobot yang diberikan akan mempertimbangkan faktor secara bersamaan. Bobot

tersebut kemudian digunakan untuk menggunakan seluruh fungsi tujuan menjadi satu fungsi *objektif* tunggal. Secara matematis, fungsi *objektif* gabungan diperoleh dari penjumlahan setiap fungsi tujuan yang telah dikalikan dengan bobot masing – masing dapat ditulis dalam bentuk berikut.

$$Maks/Min\, WZ = \sum_{j=1}^{n} \lambda Z \tag{2.3.1}$$

dengan kendala:

$$w_j \geq 0 \tag{2. 3.2}$$

$$\sum_{j=1}^{n} w_j = 1 \tag{2. 3.3}$$

keterangan

WZ = fungsi *objektif* tunggal hasil pembobotan

Z = fungsi tujuan (*objektif*) ke- *j*

λ = bobot yang diberikan pada fungsi tujuan ke- *j*

j = indeks fungsi tujuan, dengan *j* =1,2, ... , *n*

$\lambda \geq 0$ = bobot setiap fungsi tujuan harus bernilai non-negatif

Pada persamaan (2.3.1), seluruh fungsi tujuan (Z_j) digabungkan menjadi satu fungsi *objektif* baru WZ melalui penjumlahan. Penjumlahan tersebut merupakan hasil perkalian antara bobot (λ) dan fungsi tujuan (Z_j). Bobot harus memenuhi syarat $\lambda \geq 0$ dan jumlah seluruh bobot sama dengan satu ($\sum_{j=1}^{n} \lambda = 1$). Ketentuan ini bertujuan agar proporsi kepentingan setiap tujuan dapat diukur secara konsisten (Anggi, Prihandono, & Kiftiah, 2022).

2.4 Optimasi Kombinatorik

Optimasi adalah cabang ilmu matematika yang berkaitan dengan pencarian nilai maksimum atau minimum dari suatu fungsi tujuan dengan atau tanpa kendala. Optimasi sering kali digunakan untuk meningkatkan kinerja, efisiensi, dan efektivitas dalam berbagai bidang.

Optimasi kombinatorik merupakan cabang dari optimasi matematika yang melibatkan pencarian objek terbaik dari sekumpulan objek terbatas, dimana himpunan solusi yang memenuhi syarat bersifat diskrit. Salah satu contoh permasalahan optimasi kombinatorik adalah permasalahan *knapsack* (*knapsack problem*). Dimana permasalahan *knapsack* bertujuan untuk menentukan kombinasi barang terbaik dengan mempertimbangkan batas kapasitas yang tersedia (Azwar, Gultom, & Sawaluddin, 2022).

Permasalahan optimasi kombinatorik dapat diselesaikan menggunakan algoritma pemrograman dinamis. Hal ini dikarenakan solusi optimasi dari suatu permasalahan dapat diperoleh dengan kombinasi solusi optimal submasalahnya. Oleh karena itu, pemrograman dinamis sesuai untuk diterapkan sebagai penyelesaian masalah optimasi kombinatorik yang memiliki banyak kemungkinan solusi.

2.5 Permasalahan *Knapsack*

Knapsack dapat diartikan sebagai tas, karung, atau kantung. Menurut pendapat Ilham, et. al (2024), permasalahan *knapsack* merupakan masalah optimasi kombinasi dengan tujuan memaksimalkan total nilai dari barang – barang yang dimasukkan ke dalam *knapsack* atau suatu wadah tanpa melewati kapasitasnya.

Gambar 2.1. Gambaran permasalahan *knapsack*

Ada beberapa jenis permasalahan *knapsack*, diantaranya sebagai berikut.

a. *Unbounded knapsack problem*. Apabila tidak ada batasan jumlah barang untuk setiap barang.
b. *Integer knapsack*. Apabila jumlah barang untuk setiap barang hanya boleh 0 atau 1 (Rois, Maslihah, & Cahyono, 2019).
c. *Fractional knapsack problem*. Apabila jumlah barang untuk setiap barang boleh dipecah.

Dalam permasalahan ini, setiap barang atau item memiliki dua fakotr utama, yaitu berat (*weight*) barang dan nilai (*value*) barang. Adapun fungsi tujuan dari permasalahan *knapsack* adalah:

$$\sum_{i=1}^{n} v_i \times x_i \qquad (2.5.1)$$

dengan kendala :

$$\sum_{i=1}^{n} x_i \times w_i \leq M \qquad (2.5.2)$$

keterangan :

n = jumlah total barang

i = indeks barang (barang ke- 1, ke- 2, ..., ke- n)

v_i = nilai (*benefit/profit/value*) dari barang ke- i

x_i = variabel keputusan (biasanya 0 atau 1)

w_i = ukuran/berat/volume barang ke- i

M = kapasitas maksimum *knapsack*

Formulasi yang umum dan sering kali digunakan dari masalah *knapsack* adalah *knapsack problem* 0-1 (Devita & Wibawa, 2020). Hal tersebut membatasi kemungkinan pilihan yang diambil antara 1 atau 0.

2.6 Algoritma Pemrograman Dinamis

Pemrograman dinamis merupakan metode pemecahan masalah menggunakan prinsip optimalitas dengan menguraikan solusi menjadi

beberapa tahapan *(stage)* sedemikian sehingga solusinya dapat dipandang dari serangkaian keputusan yang saling berkaitan (Atiqah, 2021). Pola umum dalam pemilihan barang yaitu untuk setiap barang ke- i, program akan selalu memastikan barang ke- i memiliki nilai total terbaik yang dapat dicapai dari barang ke- 1 sampai ke- $(i-1)$ dengan sisa kapasitas yang ada. Kemudian barang selanjutnya akan mempertimbangkan hasil optimal dari barang sebelumnya (Gao, 2023).

Algoritma pemrograman dinamis efektif untuk digunkana karena setiap kombinasi barang dan kapasitas hanya dihitung satu kali saja dan proses yang dijalankan menjadi lebih efisien dan terstruktur.

2.7 Algoritma Pemrograman Dinamis pada Permasalahan *Knapsack*

Pemrograman dinamis (*dynamic programming*) adalah suatu teknik matematis yang digunakan untuk membuat suatu keputusan dari serangkaian keputusan yang saling berkaitan. Strategi pemrograman dinamis adalah membangun masalah optimasi bertingkat. Masalah optimasi bertingkat yaitu masalah yang dapat digambarkan dalam bentuk serangkaian tahapan (*stage*) yang saling mempengaruhi. Pemrograman dinamis memiliki karakteristik *overlapping subproblem* dan *optimal substructure*. Metode pemrograman dinamis dibagi menjadi dua jenis penyelesaian yaitu pemrograman dinamis rekursif maju dan pemrograman dinamis rekursif mundur. Penyelesaian masalah yang dilakukan secara rekursif berarti setiap kali mengambil keputusan harus memperhatikan keadaan yang dihasilkan oleh keputusan terbaik dari tahap sebelumnya dan menjadi landasan bagi keputusan terbaik pada tahap selanjutnya. Langkah penyelesaian pemrograman dinamis menggunakan rekursif maju dimulai dari iterasi ke-1 sampai iterasi ke-n. Sedangkan penyelesaian pemrograman dinamis menggunakan rekursif mundur dimulai dari iterasi ke- n sampai iterasi ke- 1.

Model algoritma pemrograman dinamis pada permasalahan *knapsack* disadarkan pada fungsi nilai optimal $F(i, y)$. Fungsi ini

menyatakan nilai maksimum agar dapat diperoleh dari i barang pertama dengan kapasitas *knapsack* sebesar y. Nilai tersebut dihitung secara rekursif dengan mempertimbangkan dua kemungkinan keputusan, yaitu memilih atau tidak memilih barang ke-i (Gao, 2023).

Diketahui jumlah barang dinyatakan dengan n. Setiap barang ke- i mempunyai berat sebesar w_i dan keuntungan sebesar v_i yang akan dipilih untuk dimasukkan ke dalam wadah (*knapsack*) dengan batas kapasitas sebesar M. Model matematis algoritma pemrograman dinamis pada permasalahan *knapsack* dinyatakan sebagai berikut.

$$f(i,y) = \begin{cases} f_{i-1}(y) & w_i \geq y \\ \max\{f_{i-1}(y), v_i + f_{i-1}(y-w_i)\} & w_i < y \end{cases} \quad (2.7.1)$$

keterangan :

$f(i,y)$ = solusi optimal

f_i = solusi pada tahap ke- i

f_{i-1} = solusi pada tahap sebelumnya

i = barang ke- i

w_i = berat barang ke- i

v_i = keuntungan barang ke- i

y = kapasitas *knapsack* yang tersedia

Pada kondisi awal, jika tidak ada barang yang dipertimbangkan atau kapasitas *knapsack* sama dengan nol, maka nilai maksimum yang dapat diperoleh adalah nol. Barang tidak dapat dimasukkan jika berat barang ke-i lebih besar dari kapasitas yang tersedia. Jika berat barang ke-i masih dapat dimasukkan ke dalam *knapsack*, maka terdapat dua kemungkinan, yaitu memilihnya atau tidak (Ilham, Natha, Jannah, & Christian, 2024). Nilai optimal diperoleh dari nilai maksimum antara kedua kemungkinan tersebut. Oleh karena itu, nilai optimal sama dengan nilai optimal tanpa mempertimbangkan barang tersebut.

Ketika diketahui sekumpulan barang dengan jumlah barang sebesar *n*, berat barang sebesar *w*, dan nilai dari barang tersebut sebesar *v* akan dipilih untuk dimasukkan ke dalam wadah (*knapsack*). Pemilihan barang perlu dilakukan dengan tepat agar total kapasitas barang yang dipilih tidak kapasitas wadah (*knapsack*) yang tersedia. Langkah penyelesaian masalah menggunakan algoritma pemrograman dinamis sebagai berikut.

1. Menentukan parameter masalah. Parameter tersebut meliputi jumlah barang (n), berat barang (w_i), nilai barang (v_i), kapasitas *knapsack* (M). Parameter ini menjadi dasar dalam proses perhitungan karena menentukan tujuan dan batasan optimasi yang akan dicapai
2. Membuat tabel data. Tabel ini umumnya memiliki ukuran $(n+1) \times (M+1)$, di mana baris menunjukkan jumlah barang yang dipertimbangkan dan kolom menunjukan kapasitas *knapsack* yang tersedia. Setiap elemen pada tabel menyatakan nilai maksimum yang dapat diperoleh dari sejumlah barang dengan kapasitas tertentu.
3. Mengisi tabel data secara bertahap. tahapan ini dimulai dari submasalah yang paling sederhana hingga permasalahan utama. Setiap tahap dilakukan dengan membandingkan dua kemungkinan, yaitu memilih barang ke-i atau tidak. Nilai maksimum dari kedua kemungkinan tersedut kemudian disimpan pada tabel.
4. Menentukan nilai optimal. Setelah seluruh tabel terisi, nilai optimal dari permasalahan *knapsack* dapat diperoleh dari elemen terakhir pada tabel yaitu pada posisi $[n][M]$. Nilai ini menunjukkan total nilai maksimum yang dapat diperoleh dari seluruh barang yang tersedia tanpa melebihi kapasitas *knapsack.*
5. Menentukan barang yang dipilih. Hal ini dilakukan melalui proses *backtracking*, yaitu menelusuri kembali nilai – nilai pada tabel dari posisi terakhir menuju posisi awal. Dengan cara ini, maka dapat diketahui barang yang dipilih dan tidak dalam kombinasi optimal.

6. Menampilkan hasil akhir barang yang dipilih. Hasil yang ditampilkan berupa nama, berat, nilai, total nilai, total berat dari barang yang terpilih. Selain itu juga dapat menampilkan sisa kapasitas *knapsack* yang tersedia.

2.8 Simulasi *Google Colaboratory*

Simulasi merupakan proses pembuatan model sistem nyata yang berjalan secara virtual untuk mengeksplorasi perilaku, menguji simulasi, atau mengoptimalkan keputusan di bawah ketidakpastian. Simulasi meniru dinamika sistem kompleks seperti logistik untuk memprediksi hasil tanpa resiko nyata. Proses simulasi dalam pemrograman dinamis pada *knapsack problem* digunakan untuk menghitung dan mengekstrak solusi optimal dari tabel data melalui iterasi rekursif yang mensimulasi pemilihan barang secara bertahap. Simulasi ini muncul ketika menelusuri kembali (*backtracking*) tabel data untuk rekonstruksi solusi yang diperoleh (item mana yang dipilih) dalam pohon keputusan tanpa mengulang langkah yang sama (Gao, 2023).

Data yang sesuai untuk simulasi dalam algoritma pemrograman dinamis pada *knapsack problem* adalah himpunan item berupa diskrit dengan variabel berat (*weights*) dan nilai (*values*) yang non negatif beserta kapasitas maksimal *knapsack*. Data ini cocok untuk permasalahan *knapsack* 0-1 di mana setiap item hanya dipilih satu kali dan tidak dapat dibagi. Dengan data ini tabel data dapat diisi secara iteratif untuk mensimulasikan solusi optimal dari submasalah dan menghasilkan solusi maksimal tanpa melebihi kapasitas *knapsack* (Gao, 2023). Proses simulasi akan dilakukan menggunakan bantuan bahasa pemrograman *Python* pada *google colab. Google colab* atau *google colaboratory* merupakan *platform* berbasis *cloud* gratis dari *google* untuk menulis, menjalankan, dan berbagi kode *Python* melalui *browser web*.