Aplikasi *Travelling Salesman Problem* untuk Optimalisasi Rute Penjelajahan Wisata Alam Pegunungan Bandung

Andrew Tedjapratama - 13523148[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]andrewtedj@gmail.com, 13523148@std.stei.itb.ac.id

Abstract*—Pariwisata alam di kawasan pegunungan Bandung merupakan salah satu daya tarik utama bagi wisatawan lokal maupun mancanegara. Dengan banyaknya destinasi wisata yang tersebar di wilayah ini, perencanaan rute perjalanan yang optimal menjadi tantangan penting untuk menghemat waktu dan biaya perjalanan. Makalah ini membahas optimisasi rute penjelajahan wisata alam di pegunungan Bandung menggunakan pendekatan Travelling Salesman Problem (TSP) yang diselesaikan dengan metode Dynamic Programming (DP). Hasil penelitian diharapkan dapat memberikan rekomendasi rute perjalanan terpendek yang efektif dan efisien bagi wisatawan.

***Keywords*—Graf, Rute Optimal, *Travelling Salesman Problem*, Wisata Alam Pegunungan**

I. PENDAHULUAN

Kawasan pegunungan di Bandung merupakan salah satu destinasi wisata yang paling banyak dijumpai di Indonesia yang terkenal dengan pemandangan alam, kesejukan udara segar, dan aktivitas wisata lainnya. Terdapat lebih dari 700 pegunungan tersebar di berbagai daerah di Bandung Raya. Beberapa lokasi wisata alam pegunungan populer di Bandung seperti Kawah Putih, Tangkuban Perahu, dan Tebing Keraton menjadi daya tarik utama bagi wisatawan. Namun, dengan bertumbuhnya angka wisatawan, timbul juga tantangan yang dihadapi para wisatawan dalam menentukan rute perjalanan optimal untuk mengunjungi tempat-tempat yang diinginkan dalam waktu yang terbatas.

Rute perjalanan yang tidak terencana dengan baik akan menyebabkan masalah seperti pemborosan waktu perjalanan dan pengalaman wisata yang kurang maksimal. Jarak antar lokasi wisata pegunungan di Bandung juga sangat bervariasi mulai dari 5 km hingga lebih dari 150km. Oleh karena itu, dibutuhkan suatu pendekatan yang sistematis dalam menangani masalah ini karena optimalisasi rute perjalanan dapat memberikan dampak yang signifikan dalam mengurangi jarak tempuh rute perjalanan.

Setiap lokasi-lokasi wisata pegunungan yang akan dikunjungi dimodelkan sebagai simpul dalam sebuah graf berbobot. Setiap simpul dihubungkan dengan simpul lain dengan sebuah sisi yang menyatakan jarak antar simpul tersebut. Dengan pemodelan ini, permasalahan optimalisasi rute dapat diselesaikan menggunakan pendekatan *Travelling Salesman Problem* menggunakan algoritma *Dynamic Programming*.

Makalah ini ditulis dengan tujuan mengoptimalisasi rute perjalanan antar beberapa wisata alam pegunungan di kawasan Bandung. Optimalisasi ini dilakukan dengan metode *Travelling Salesman Problem* (TSP). Dengan demikian, diharapkan pendekatan ini dapat memberikan rekomendasi rute perjalanan terbaik untuk wisatawan di Bandung.

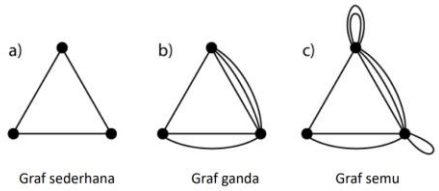
II. LANDASAN TEORI

A. Graf

Graf didefinisikan sebagai suatu struktur atau objek diskrit berupa kumpulan simpul (*vertices*) yang terhubung melalui himpunan sisi (*edges*). Secara formal, graf G disajikan dalam bentuk $G = (V, E)$, dimana untuk himpunan n benda, himpunan V akan berisi $\{v_1, v_2, \ldots, v_n\}$, dan untuk himpunan relasi m, himpunan E akan berisi $\{e_1, e_2, \ldots, e_m\}$ yang menghubung sepasang simpul dalam graf.

Graf juga dapat terdiri dari dua jenis, yaitu graf sederhana (*simple graph*) dan graf tak-sederhana (*unsimple graph*). Perbedaan kedua jenis tersebut adalah graf sederhana tidak memiliki sisi ganda dan sisi gelang (*loop*), sedangkan graf tak-sederhana mengandung sisi ganda atau sisi gelang. Sisi ganda didefinisikan oleh dua simpul yang memiliki dua atau lebih sisi satu sama lain, sedangkan sisi gelang didefinisikan oleh simpul yang memiliki satu atau lebih sisi terhadap dirinya sendiri.

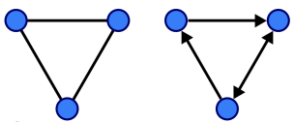
Graf tak-sederhana yang mengandung sisi ganda disebut graf ganda (*multi-graph*), sedangkan graf tak-sederhana yang mengandung sisi gelang disebut graf semu (*pseudo-graph*).



Gambar 2.1 (a) Graf sederhana (b) Graf tak-sederhana (c) Graf-semu (Sumber : [1])

Berdasarkan orientasi arah sisi, graf dapat dikelompokkan menjadi:

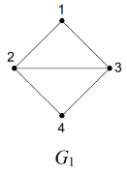
1. Graf tak berarah (*undirected graph*), yaitu graf yang sisinya tidak mempunyai orientasi arah.
2. Grah berarah (*directed graph*), yaitu graf yang setiap sisinya diberikan orientasi arah.



Gambar 2.2 (a) Contoh graf berarah dan graf tidak berarah (Sumber : [2])

Berikut adalah beberapa terminologi dalam teori graf yang relevan dengan penelitian ini:

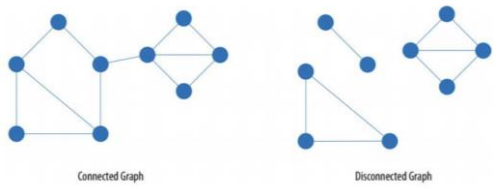
1. Ketetanggan (*adjacency*)
   Dua buah simpul dalam graf bertetangga bila keduanya terhubung langsung.



G_1

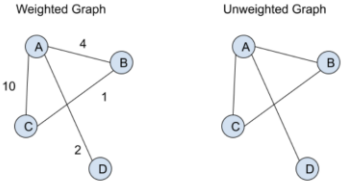
Gambar 2.3. (a) Contoh graf G1
(Sumber : [1])

2. Bersisian (*incidency*)
   Bersisian berarti terhubung oleh garis. Sebuah sisi e dikatakan bersisian dengan simpul v_i dan v_j jika sisi tersebut menghubungkan simpul v_i dengan v_j.
3. Derajat (*degree*)
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
4. Lintasan (*path*)
   Lintasan dengan barisan sisi yang menghubungkan barisan simpul, baik terbatas maupun tidak terbatas. Lintasan dengan panjang n ialah barisan berselang seling simpul dan sisi-sisi yang dilaluinya v_0 , e_1, v_1, e_2, v_2, ... , e_n, v_n yang menghubungkan simpul awal v_0 ke simpul tujuan v_n. Panjang lintasan n adalah jumlah sisi dalam lintasan tersebut.
5. Siklus (*cycle*) atau sirkuit (*circuit*)
   Siklus atau sirkuit merupakan lintasan yang berawal dan berakhir pada simpul yang sama. Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut.
6. Keterhubungan (*connected*)
   Dua simpul v_0 dan v_1 dikatakan terhubung jika terdapat lintasan dari v_1 ke v_2. G disebut graf terhubung (connected graph) jika untuk setiap pasang simpul v_i dan v_j dalam himpunan V terdapat lintasan dari v_i ke v_j.



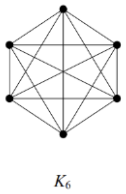
Gambar 2.4 (a) Contoh graf terhubung (connected graph) dan graf tak-terhubung (disconnected graph)
(Sumber : [1])

7. Graf berbobot (*weighted graph*)
   Graf berbobot adalah graf yang setiap sisinya diberi nilai numerik yang disebut bobot.



Gambar 2.5 (a) Contoh graf berbobot (weighted graph) dan graph tak-berbobot (unweighted graph)
(Sumber : [1])

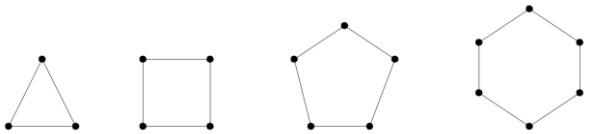
8. Graf lengkap (*complete graph*)
   Graf lengkap adalah graf sederhana yang setiap simpulnya mempunyai sisi ke semua simpul lainnya. Graf lengkap dengan n buah simpul dilambangkan dengan K_n. Jumlah sisi pada graf lengkap yang terdiri dari n buah simpul adalah $\frac{n}{2}(n-1)$.



K_6

Gambar 2.6 (a) Contoh graf lengkap dengan jumlah simpul 6
(Sumber : [1])

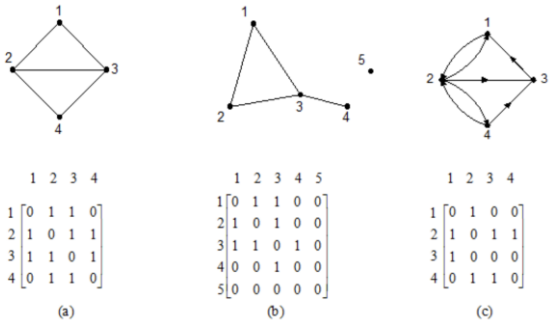
9. Graf lingkaran (*circle graph)*
   Graf lingkaran adalah graf sederhana yang setiap simpulnya berderajat dua. Graf lingkaran dengan n simpul dilambangkan dengan C_n.



Gambar 2.7 (a) Contoh graf lingkaran dengan simpul 3,4,5,6 secara urut
(Sumber : [1])

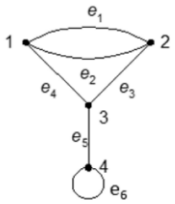
Representasi graf yang efektif sangat penting dalam menyelesaikan masalah *Travelling Salesman Problem* (TSP). Berikut adalah beberapa cara untuk merepresentasikan graf dengan matriks:

1. Matriks ketetanggaan (*adjacency matrix)*
   Matriks ketetanggaan adalah representasi graf dalam bentuk matriks dua dimensi berukuran n×n, di mana n adalah jumlah simpul dalam graf. Elemen matriks $M[i][j]$ menyatakan keberadaan sisi antara simpul i dan simpul j.
   Untuk graf berarah, elemen $M[i][j] = 1$ jika terdapat sisi dari simpul i ke simpul j, dan $M[i][j] = 0$ jika tidak ada sisi. Untuk graf berbobot, elemen $M[i][j]$ berisi bobot sisi antara simpul i dan j. Jika tidak ada sisi, elemen diisi dengan nilai tak hingga (∞) untuk menggambarkan bahwa simpul tidak terhubung.



Gambar 2.8 Representasi matriks ketetanggaan (adjacency matrix) (a)graf tidak berarah terhubung, (b) graf tidak berarah tak-terhubung, (c) graf berarah
(Sumber : [3])

2. Matriks bersisian (*incidency matrix*)
   Matriks bersisian merepresentasikan sebuah graf tidak berarah yang memiliki nnn simpul dan mmm sisi dalam bentuk matriks berukuran $n \times m$. Setiap elemen matriks pada baris ke-i dan kolom ke-j bernilai 1 jika simpul ke-i terhubung dengan sisi ke-j, dan bernilai 0 jika tidak terdapat hubungan.



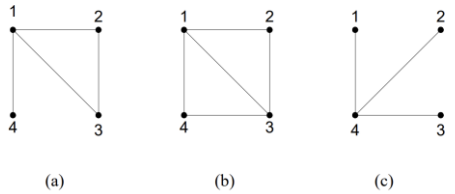
	e1	e2	e3	e4	e5	e6
1	1	1	0	1	0	0
2	1	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1

Gambar 2.9 Representasi matriks bersisian dari graf (Sumber : [3])

B. Lintasan dan Sirkuit Hamilton

Lintasan Hamilton adalah istilah yang digunakan didalam teori graf untuk sebuah lintasan yang melalui tiap simpul dalam graf tepat satu kali saja. Lintasan ini dinamai sesuai dengan nama seorang matematikawan berkebangsaan Irlandia bernama Sir William Rowan Hamilton. Lintasan Hamilton seringkali digunakan untuk menemukan jalur terpendek antara dua titik dalam sebuah sistem.

Sirkuit Hamilton adalah sirkuit yang melalui tiap simpul dalam graf tepat satu kali, kecuali simpul asal (sekaligus simpul akhir) yang dilalui dua kali. Dengan kata lain, sirkuit Hamilton merupakan lintasan Hamilton yang melalui setiap titik dalam sebuah sistem dan kembali ke titik awal, sehingga membentuk sebuah sirkuit. Graf yang mempunyai sirkuit Hamilton disebut graf Hamilton, sedangkan graf yang mempunyai lintasan Hamilton saja disebut graf semi-Hamilton.



Gambar 2.10 (a) Graf semi-Hamilton (b) Graf Hamilton (c) Bukan Graf semi-Hamilton ataupun graf Hamilton (Sumber:[4])

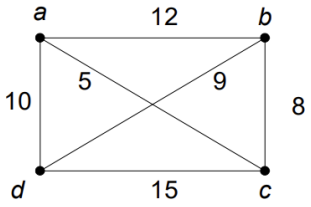
Sebuah graf sederhana G dengan n (≥ 3) buah simpul adalah graf Hamilton bila: $r \geq n/2$ untuk setiap simpul di G, dengan r adalah derajat setiap simpul.

Setiap graf lengkap adalah graf Hamilton. Di dalam graf lengkap G dengan n buah simpul ($n \geq 3$), terdapat $(n-1)!/2$ buah sirkuit Hamilton. Jika n genap dan $n \geq 4$, maka di dalam G terdapat $(n-2)/2$ buah sirkuit Hamilton yang saling lepas.

C. Travelling Salesman Problem (TSP)

Travelling Salesman Problem (TSP) atau Persoalan Pedagang Keliling merupakan sebuah contoh masalah optimasi untuk mencari rute terpendek yang perlu dilalui seorang pedagang untuk mengunjungi beberapa kota dan kembali ke kota awal. Dengan kata lain, TSP merupakan contoh masalah yang berhubungan ketat dan dapat diselesaikan dengan teori lintasan Hamilton.

Misalnya, diberikan sejumlah kota dan diketahui jarak antarkota. Dalam kasus ini, seorang pedagang harus mengunjungi semua kota dan kembali ke kota asalnya. Kita perlu mencari jalur terpendek yang dapat dilalui pedagang tersebut untuk menghemat waktu dan biaya perjalanan. Untuk menyelesaikan masalah ini, dibuatlah sebuah graf dengan setiap kota sebagai simpul dan jalan yang menghubungkan tiap kota sebagai sisi. Bobot setiap sisi merupakan jarak antarkota yang dihubungkan oleh sisi tersebut.



Gambar 2.11 Contoh graf berbobot untuk permasalahan TSP (Sumber : [4])

Diketahui untuk sebuah graf lengkap, jumlah sirkuit Hamiltonnya adalah $\frac{1}{2}(n-1)!$, sehingga permasalahan TSP dapat diselesaikan dengan menentukan sirkuit Hamilton yang memiliki bobot minimum. Contohnya graf pada gambar 2.11 memiliki $\frac{1}{2}(4-1)! = 3$ buah sirkuit Hamilton. Dengan mengecek bobot dari semua sirkuit, didapatkan rute terpendek yaitu $a-c-b-d-a$ dengan bobot $10+5+9+8=32$.

Sirkuit Hamilton dengan bobot terendah memberikan solusi optimal untuk *Travelling Salesman Problem* pada graf ini. Pendekatan ini bekerja dengan memeriksa semua kemungkinan lintasan, yang bisa dilakukan untuk menyelesaikan masalah dengan jumlah simpul yang kecil. Jika dicermati, algoritma sebelumnya merupakan algorima *Brute Force* dengan kompleksitas $O(n!)$, di mana n adalah jumlah simpul dalam graf.

Untuk jumlah simpul besar, pendekatan ini tidak efisien disebabkan perlunya menghitung bobot dari banyak sirkuit Hamilton. Oleh karena itu, dalam makalah ini, pendekatan seperti *Dynamic Programming* atau dapat digunakan untuk meningkatkan efisiensi perhitungan khususnya dalam permasalahan TSP.

C. Algoritma Dynamic Programming untuk TSP

Dynamic Programming (DP) merupakan sebuah teknik pemrograman dinamis yang digunakan untuk menyelesaikan masalah dengan banyak submasalah yang saling terkait. Algoritma ini juga dikenal sebagai Algoritma Held-Karp, dengan kompleksitas waktu $O(n2 \cdot 2n^2)$, di mana n adalah jumlah simpul dalam graf.

Penyelesaian masalah menggunakan pemrograman dinamis dilakukan dengan membagi suatu masalah besar menjadi sub-masalah yang lebih kecil. Solusi dari setiap sub-masalah yang lebih kecil tersebut lalu disimpan. Dengan cara ini, pemrograman dinamis menghindari pengulangan perhitungan sub-masalah yang sudah dihitung dan mencapai solusi yang lebih efisien. Optimisasi dengan pemrograman dinamis mengurangi kompleksitas waktu dari eksponensial menjadi polinomial.

Travelling Salesman Problem (TSP) dapat diselesaikan dengan pemrograman dinamis untuk menghindari perhitungan ulang pada sub-rute yang tumpang tindih (*overlapping subproblems*). Dalam pendekatan ini, lintasan minimum dihitung dengan fungsi rekursif $dp(pos, mask)$:

- pos : menunjukkan kota terakhir yang dikunjungi.
- $mask$: mewakili kumpulan atau himpunan kota yang telah dikunjungi menggunakan *bitmasking*.

Bitmask merupakan bilangan biner yang menunjukkan status kunjungan atau apakah suatu kota sudah pernah dikunjungi. Jika bit ke-i pada b bernilai 1, maka kota i telah dikunjungi. Sebaliknya jika bit ke-i bernilai 0, maka kota i belum dikunjungi Sebagai contoh, misalnya diberikan *bitmask* 10010_2, maka itu menunjukkan bahwa kota pertama dan keempat telah dikunjungi. Fungsi dp menghitung total jarak minimum berdasarkan sub-rute yang telah dikunjungi.

Fungsi ini dirumuskan secara matematis dengan mencari Basis, yaitu ketika semua kota telah dikunjungi, jarak antar kota terakhir ke kota awal dihitung.

$$dp(pos, 2n-1) = dist[pos][0]$$

Lalu relasi rekursif menambahkan jarak kota terakhir ke kota berikutnya, dengan ditambah jarak minimum antar kota yang tersisa. Hubungan rekursif untuk masalah TSP didefinisikan sebagai:

$$dp(pos, mask) = min(dist[pos][nxt] + dp(nxt, mask \mid (1 << nxt)))$$

Dimana:

- pos adalah kota saat ini yang dikunjungi
- mask mewakili kota-kota yang telah dikunjungi
- $(dist[pos][nxt]$ adalah jarak perjalanan dari kota saat ini (pos) ke kota berikutnya (nxt)
- $dp(nxt, mask \mid (1 << nxt)))$ mewakili rekursif perhitungan jarak minimum dari kota berikutnya (nxt) setelah memperbarui status kunjungan (menandai kota tersebut sudah dikunjungi).

III. METODOLOGI

A. Pendahuluan

Makalah ini bertujuan menghadapi masalah utama yaitu bagaimana cara menentukan rute perjalanan yang optimal untuk mengunjungi sejumlah lokasi wisata dengan jarak tempuh total yang seminimal mungkin. Permasalahan ini dimodelkan menggunakan *Travelling Salesman Problem* (TSP) dengan pendekatan *Dynamic Programming* (DP), penelitian ini bertujuan untuk memberikan solusi rute perjalanan terbaik bagi wisatawan di kawasan pegunungan Bandung. Metodologi ini mencakup pemodelan masalah, pengumpulan data, dan implementasi algoritma, dengan evaluasi hasil berada pada bab berikutnya.

B. Pemodelan Masalah

Dalam penelitian ini, lokasi wisata dimodelkan sebagai simpul (*node*) dalam suatu graf lengkap berbobot dan bobot sisi (*edge*) merepresentasikan jarak antar lokasi wisata. Data jarak antar lokasi yang dipakai untuk eksperimen diwakili dengan matriks berbobot (*graph[i][j]*). Dataset yang digunakan merupakan jarak aktual yang didapatkan dari observasi melalui sumber Google Maps. Setelah itu, tetap ada asumsi dan batasan yang perlu diketahui terlebih dahulu.

Untuk menentukan fokus dan menjaga ruang lingkup penelitian dalam makalah ini, terdapat beberapa batasan masalah untuk menyederhanakan masalah penentuan rute perjalanan. Batasan-batasan tersebut antara lain:

1. Lokasi wisata yang dipertimbangkan
   Penelitian ini hanya mencakup 10 lokasi wisata alam pegunungan di kawasan Bandung yang dipilih berdasarkan popularitas dan aksesibilitas.
2. Asumsi jarak statis
   Data jarak antar lokasi dianggap tetap dan tidak memperhitungkan faktor dinamis seperti kondisi lalu lintas, cuaca, dan lain-lain.
3. Komparasi dan validasi
   Komparasi dan validasi dilakukan dengan membandingkan hasil algoritma *Dynamic Drogramming* (DP) dengan metode *Brute Force*.
4. Tidak memperhatikan waktu dan biaya kunjungan
   Penelitian ini tidak memperhitungkan waktu dan biaya kunjungan di setiap lokasi, hanya fokus pada jarak antar lokasi wisata dalam kilometer.

C. Dataset yang digunakan

Data yang digunakan merupakan data aktual dikumpulkan dari observasi lapangan sendiri melalui Google Maps, dengan mencari 10 lokasi wisata pegunungan terpopuler di Bandung lalu mencari jarak antar lokasinya dalam kilometer.

TABEL 1. INDEKS SIMPUL DAN NAMA LOKASI WISATA

No.	Nomor Simpul (Indeks)	Nama Lokasi
1.	0	Tebing Keraton
2.	1	Kawah Putih Ciwidey
3.	2	Tangkuban Perahu
4.	3	Gunung Putri Lembang
5.	4	Taman Hutan Raya
6.	5	Gunung Manglayang
7.	6	Bukit Moko
8.	7	Curug Maribaya
9.	8	Kebun Teh Sukawana
10.	9	Gunung Patuha

Lokasi-lokasi tersebut dicari jarak antar lokasinya dan direpresentasi bentuk matriks ketetanggaan dimana setiap elemen mewakili jarak dalam kilometer.

TABEL 2. REPRESENTASI MATRIKS JARAK ANTAR LOKASI WISATA

Lokasi	Tebing Keraton	Kawah Putih Ciwidey	Tangkuban Perahu	Gunung Putri Lembang	Taman Hutan Raya	Gunung Manglayang	Bukit Moko	Curug Maribaya	Kebun Teh Sukawana	Gunung Patuha
Tebing Keraton	0	65.0	28.1	17.6	5.0	54.0	7.7	14.1	24.7	64.3
Kawah Putih Ciwidey	65.0	0	78.7	61.8	60.0	82.9	66.7	60.4	70.9	1.0
Tangkuban Perahu	28.1	78.7	0	14.7	23.0	82.2	27.8	14.8	12.0	76.8
Gunung Putri Lembang	17.6	61.8	14.7	0	12.2	71.4	19.5	8.3	13.6	66.9
Taman Hutan Raya	5.0	60.0	23.0	12.2	0	48.9	9.3	9.0	19.6	59.0
Gunung Manglayang	54.0	82.9	82.2	71.4	48.9	0	39.1	54.6	73.5	82.1
Bukit Moko	7.7	66.7	27.8	19.5	9.3	39.1	0	16.3	29.4	65.9
Curug Maribaya	14.1	60.4	14.8	8.3	9.0	54.6	16.3	0	15.9	65.0
Kebun Teh Sukawana	24.7	70.9	12.0	13.6	19.6	73.5	29.4	15.9	0	62.6
Gunung Patuha	64.3	1.0	76.8	66.9	59.0	82.1	65.9	65.0	62.6	0

C. Implementasi Algoritma TSP dengan Bahasa Python

Dalam penelitian ini, algoritma untuk optimalisasi rute perjalanan menggunakan diimplementasikan menggunakan bahasa pemrograman Python. Tautan repository GitHub untuk implementasi program dapat diakses di Repository TSP Travel Optimizer [6]

Data jarak antar lokasi tempat wisata akan dimodelkan sebagai matriks yang akan diproses melalui dua pendekatan yaitu pendekatan *Dynamic Programming (Held-Karp Algorithm)* dan pendekatan *Brute Force*. Pendekatan ini dilakukan dengan tujuan mencari keunggulan metode *Dynamic Programming* dalam TSP dengan membandingkannya dengan metode umum seperti metode *Brute Force*.

1. Representasi Data

Langkah implementasi diawali dengan dataset jarak antar lokasi wisata yang dimodelkan sebagai matriks berbobot (*weighted adjacency matrix)* berukuran 10x10 yang merepresentasikan jarak antar 10 lokasi wisata. Contoh representasi data jarak adalah sebagai berikut.

```
graph = [
    [0, 65.0, 28.1, 17.6, 5.0, 54.0, 7.7, 14.1, 24.7, 64.3],
    [65.0, 0, 78.7, 61.8, 60.0, 82.9, 66.7, 60.4, 70.9, 1.0],
    [28.1, 78.7, 0, 14.7, 23.0, 82.2, 27.8, 14.8, 12.0, 76.8],
    [17.6, 61.8, 14.7, 0, 12.2, 71.4, 19.5, 8.3, 13.6, 66.9],
    [5.0, 60.0, 23.0, 12.2, 0, 48.9, 9.3, 9.0, 19.6, 59.0],
    [54.0, 82.9, 82.2, 71.4, 48.9, 0, 39.1, 54.6, 73.5, 82.1],
    [7.7, 66.7, 27.8, 19.5, 9.3, 39.1, 0, 16.3, 29.4, 65.9],
    [14.1, 60.4, 14.8, 8.3, 9.0, 54.6, 16.3, 0, 15.9, 65.0],
    [24.7, 70.9, 12.0, 13.6, 19.6, 73.5, 29.4, 15.9, 0, 62.6],
    [64.3, 1.0, 76.8, 66.9, 59.0, 82.1, 65.9, 65.0, 62.6, 0]
]
```

Gambar 3.1 Graf ketetanggaan dalam bentuk matriks jarak antar lokasi wisata (dalam kilometer)

Bobot atau elemen dalam matriks tersebut menunjukkan jarak antar lokasi wisata dalam satuan kilometer. Lokasi-lokasi wisata tersebut juga diindeks untuk kemudahan identifikasi.

```
index_to_location = {
    0: "Tebing Keraton",
    1: "Kawah Putih Ciwidey",
    2: "Tangkuban Perahu",
    3: "Gunung Putri Lembang",
    4: "Taman Hutan Raya",
    5: "Gunung Manglayang",
    6: "Bukit Moko",
    7: "Curug Maribaya",
    8: "Kebun Teh Sukawana",
    9: "Gunung Patuha"
}
```

Gambar 3.2 Dictionary index_to_location untuk identifikasi indeks dari lokasi-lokasi wisata

2. Algoritma Brute Force

Pendekatan *Brute Force* digunakan pada penelitian ini sebagai pendekatan banding dengan pendekatan *Dynamic Programming* dengan mencari efisiensi pencarian rute terdekat. Pendekatan ini digunakan dengan mengevaluasi seluruh kemungkinan lintasan yang mengunjungi semua simpul tepat satu kali dan kembali ke simpul awal. Dalam implementasinya, digunakan library *itertools.permutations* untuk membantu menghasilkan semua permutasi lintasan. Kompleksitas algoritma dari pendekatan *Brute Force* adalah $O(n!)$ yang membuatnya tidak efisien pada kalkulasi jarak dan rute terdekat pada graf. Kode implementasi algoritma *Brute Force* adalah sebagai berikut.

```
from itertools import permutations

def brute_force_tsp(graph, start):
    n = len(graph)
    nodes = list(range(n))
    nodes.remove(start)

    min_distance = float('inf')
    best_path = []

    for perm in permutations(nodes):
        # Kalkulasi permutasi total jarak
        current_distance = graph[start][perm[0]]  # start dari simpul awal
        for i in range(len(perm) - 1):
            current_distance += graph[perm[i]][perm[i + 1]]
        current_distance += graph[perm[-1]][start]  # balik ke simpul awal

        if current_distance < min_distance:
            min_distance = current_distance
            best_path = [start] + list(perm) + [start]

    return min_distance, best_path
```

Gambar 3.3 Implementasi algoritma Brute Force untuk TSP dalam Python

3. Algoritma Dynamic Programming (Held-Karp)

Dynamic Programming merupakan pendekatan yang lebih efisien dengan menutupi kelemahan pendekatan *Brute Force* dengan mengurangi perhitungan berulang menggunakan memoization. Implementasi menggunakan teknik bitmasking untuk merepresentasikan kota yang telah dikunjungi dan dibantu dengan library *NumPy* untuk perhitungan. Kode implementasi algoritma *Dynamic Programming* adalah sebagai berikut:

```
import numpy as np

def dp_tsp(graph, start):
    n = len(graph)
    graph = np.array(graph)
    memo = np.full((n, 1 << n), -1, dtype=float)
    parent = np.full((n, 1 << n), -1, dtype=int)

    def visit(mask, pos):
        if memo[pos, mask] != -1:
            return memo[pos, mask]

        if mask == (1 << n) - 1:
            return graph[pos, start]

        min_distance = float('inf')
        for city in range(n):
            if mask & (1 << city) == 0:
                distance = graph[pos, city] + visit(mask | (1 << city), city)
                if distance < min_distance:
                    min_distance = distance
                    parent[pos, mask] = city

        memo[pos, mask] = min_distance
        return min_distance

    total_distance = visit(1 << start, start)

    # Reconstruct path
    best_path = [start]
    mask = 1 << start
    pos = start
    while parent[pos, mask] != -1:
        next_pos = parent[pos, mask]
        best_path.append(next_pos)
        mask |= (1 << next_pos)
        pos = next_pos

    best_path.append(start)
    return total_distance, best_path
```

Gambar 3.4 Implementasi algoritma Dynamic Programming untuk TSP dalam Python

Dalam algoritma ini, *memo* merupakan matriks berukuran 2D untuk menyimpan hasil submasalah. Elemen $memo[pos][mask]$ menyimpan jarak minimum dari simpul *pos* dengan kumpulan simpul yang dikunjungi direpresentasikan oleh mask.

Selain itu, *parent* merupakan matriks 2D yang menyimpan kota sebelumnya untuk membantu rekonstruksi jalur optimal setelah solusinya ditemukan. Elemen $parent[pos][mask]$ menyimpan simpul berikutnya yang harus dikunjungi dari simpul *pos* dengan *mask* tertentu. Pada program ini, *mask* merepresentasikan bitmask untuk kota-kota yang telah dikunjungi. Misalnya, jika ada 4 kota dan $mask = 0110$ (dalam biner), artinya kota ke-1 dan ke-2 telah dikunjungi.

Setelah itu, ada fungsi rekursif *visit* dengan *memoization* yang menghitung jarak minimum dari kota *pos* dengan kumpulan kota yang telah dikunjungi sesuai *mask*. Oleh karena itu, fungsi *visit* menjadi inti dari implementasi *Dynamic Programming*.

4. Integrasi dan Perbandingan

Kedua algoritma diintegrasikan dalam program *main.py* utama untuk melakukan perbandingan hasil jarak total dan jalur optimal yang dihasilkan oleh masing-masing metode dengan tambahan runtime untuk membandingkan efisiensi algoritma. Berikut adalah code untuk integrasi dan perbandingan:

```python
start = 0  # titik awal perjalanan

# Dynamic Programming
start_time = time.time()
total_distance_dp, path_dp = dp_tsp(graph, start)
end_time = time.time()
runtime_dp = end_time - start_time
print(color("\nTSP - Dynamic Programming", 94))
print(f"Jarak total perjalanan: {(total_distance_dp)} km")
print(f"\nJalur optimal (simpul): {' -> '.join(map(str, path_dp))}")

path_with_names = [index_to_location[i] for i in path_dp]
print(color(" -> ".join(path_with_names), 93))

# Runtime DP
print(color(f"Runtime: {end_time - start_time} detik", 92))
print()

# Brute Force
start_time = time.time()
total_distance_bf, path_bf = brute_force_tsp(graph, start)
end_time = time.time()
runtime_bf = end_time - start_time
print(color("TSP - Brute Force", 94))
print(f"Jarak total perjalanan: {total_distance_bf} km")
print(f"\nJalur optimal (simpul): {' -> '.join(map(str, path_bf))}")
path_with_names = [index_to_location[i] for i in path_bf]
print(color(" -> ".join(path_with_names), 93))

# Runtime BF
print(color(f"Runtime: {end_time - start_time} detik", 91))
print()

print(f"Perbandingan runtime DP vs BF (approximately): {runtime_bf/runtime_dp:.2f} kali lebih cepat")
```

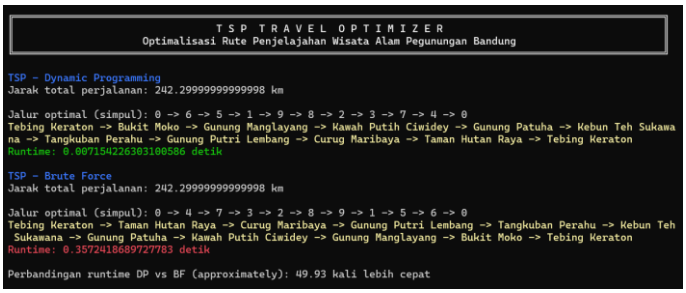
Gambar 3.5 Implementasi main.py untuk perbandingan jarak, rute tercepat, dan runtime antara algoritma Dynamic Programming dan Brute Force.

Dengan integrasi ini, program dapat memberikan perbandingan langsung antara efisiensi metode *Dynamic Programming* dengan metode *Brute Force*.

IV. Diskusi dan Pembahasan

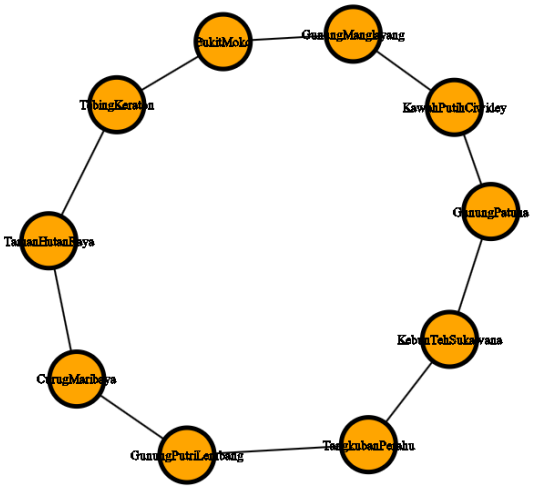
Bab ini bertujuan untuk mengevaluasi hasil dan algoritma yang telah diimplementasikan di bab sebelumnya, yaitu *Dynamic Programming (DP)* dan perbandingan efisiensinya dengan *Brute Force*, dalam penyelesaian masalah menggunakan *Travelling Salesman Problem* (TSP). Pembahasan dan evaluasi mencakup analisis performa runtime dan efektivitas jalur optimal.

Dari hasil percobaan, dengan menjalankan program main didapatkan berbagai perbandingan hasil seperti jarak total perjalanan, jalur optimal dalam indeks atau simpul, jalur optimal dalam lokasi, dan runtime untuk masing-masing metode yaitu *Dynamic Programming* dan *Brute Force*. Setelah itu, diberikan perbandingan efisiensi runtime di akhir. Contoh tampilan hasil program dapat dilihat pada gambar berikut:



Gambar 4.1 Hasil implementasi dan perbandingan runtime antara Dynamic Programming dan Brute Force pada TSP

Dari hasil diatas, dapat divisualisasikan graf untuk hubungan antar lokasi wisata di kawasan pegunungan Bandung dengan jalur paling optimal menggunakan *Dynamic Programming*. Setiap simpul (*node*) merepresentasikan lokasi wisata dan setiap sisi (*edge*) menunjukkan koneksi langsung antar lokasi. Visualisasi struktur graf tersebut dapat dilihat dalam gambar berikut:



Gambar 4.2 Visualisasi rute optimal melalui representasi graf berarah

TABEL 3. PERBANDINGAN RUNTIME ANTARA BRUTE FORCE DAN DYNAMIC PROGRAMMING

No.	Runtime Brute Force (detik)	Runtime Dynamic Programming (detik)	Perbandingan
1.	0.2277	0.0055	41.15×
2.	0.2333	0.0055	42.18×
3.	0.2359	0.0053	43.75×
4.	0.2247	0.0051	44.17×
5.	0.2218	0.0049	44.38×
6.	0.2774	0.0060	46.23×
7.	0.2795	0.0060	46.54×

8.	0.2755	0.0058	47.09×
9.	0.2398	0.0050	47.95×
10.	0.3572	0.0071	49.93×
Rata-rata			45.34×

Tabel di atas merupakan hasil perbandingan kedua algoritma setelah 10 kali percobaan dan dapat dilihat bahwa algoritma *Dynamic Programming* secara signifikan lebih cepat dibandingkan *Brute Force* dalam penyelesaian masalah TSP dengan rata-rata 45.54 kali lebih cepat. Hal ini membuktikan bahwa *Dynamic Programming* optimal dalam memberikan hasil dengan runtime jauh lebih efisien.

V. KESIMPULAN

Penelitian ini berhasil mengimplementasikan algoritma *Dynamic Programming* dan *Brute Force* untuk menyelesaikan permasalahan *Travelling Salesman Problem* (TSP) dalam konteks rute perjalanan wisata di kawasan Pegunungan Bandung. Representasi graf berbobot lengkap dengan 10 simpul lokasi wisata digunakan sebagai basis perhitungan. Hasil implementasi menunjukkan bahwa *Dynamic Programming* mampu menghasilkan jalur optimal dengan runtime yang jauh lebih efisien dibandingkan *Brute Force*, terutama pada dataset dengan jumlah simpul yang lebih besar.

Dari evaluasi runtime, algoritma *Dynamic Programming* terbukti lebih unggul, dengan efisiensi hingga puluhan kali lipat dibandingkan *Brute Force*. Selain itu, jalur optimal yang dihasilkan oleh kedua metode menunjukkan konsistensi pada dataset kecil, membuktikan akurasi dari implementasi *Dynamic Programming*.

Penelitian ini menunjukkan bahwa pendekatan *Dynamic Programming* adalah pilihan yang lebih baik untuk menyelesaikan TSP pada graf dengan ukuran sedang, sementara *Brute Force* lebih cocok digunakan untuk validasi pada graf kecil. Untuk penelitian di masa depan, pengembangan dapat dilakukan dengan mempertimbangkan faktor dinamis seperti lalu lintas atau waktu tempuh untuk meningkatkan relevansi solusi dalam dunia nyata.

VI. LAMPIRAN

1. Repository GitHub untuk implementasi program pada makalah ini dapat diakses pada tautan berikut: https://github.com/andrewtedja/travel-optimizer-tsp
2. Link video: https://youtu.be/y3HqOIIaXus

VII. UCAPAN TERIMA KASIH

Penulis menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa karena atas berkat dan pemeliharaan-Nya, penulis dapat menyelesaikan makalah ini dengan baik.
2. Orang tua penulis yang selalu memberikan semangat dukungannya selama proses menulis makalah ini.
3. Bapak Dr. Ir. Rinaldi Munir, Arrival Dwi Sentosa, S.Kom., M.T. dan Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen mata kuliah IF 1220 Matematika Diskrit yang telah menjadi pembimbing serta memberikan segala ilmu dan pengetahuannya akan materi terkait sehingga penulis dapat lebih memahami dan memiliki kesempatan untuk makalah dan melakukan eksplorasi.

Akhir kata, penulis menyampaikan rasa syukur yang terbanyak serta berharap makalah ini dapat bermanfaat bagi pihak yang membacanya.

REFERENCES

[1] Munir, Rinaldi. (2024). "Graf: Bagian 1". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf (diakses tanggal 29 Desember 2024).

[2] Ipython Books, "Graphs, Geometry, and Geographic Information Systems," https://ipython-books.github.io/chapter-14-graphs-geometry-and-geographic-information-systems/ (diakses tanggal 29 Desember 2024),

[3] Munir, Rinaldi. (2024). "Graf: Bagian 2". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf (diakses tanggal 29 Desember 2024).

[4] Munir, Rinaldi. (2024). "Graf: Bagian 3". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf (diakses tanggal 29 Desember 2024).

[5] GeeksforGeeks, "Travelling Salesman Problem using Dynamic Programming," https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/?ref=ml_lbp (diakses tanggal 7 Januari 2025).

[6] Andrew Tedjapratama, "travel-optimizer-tsp," GitHub repository, https://github.com/andrewtedja/travel-optimizer-tsp (diakses tanggal 8 Januari 2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2024

Andrew Tedjapratama 13523148