Aproksimasi Traveling Salesman Problem dengan Minimum Spanning Tree dalam Mencari Tur Terpendek Benua Eropa

Rayhan Fadhlan Azka - 13522095
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
rayhanfazka07@gmail.com

***Abstrak*— Traveling Salesman Problem (TSP) adalah masalah optimasi yang mencari rute terpendek untuk mengunjungi setiap kota tepat satu kali dan kembali ke kota asal. TSP termasuk dalam kategori NP-hard, yang berarti tidak ada algoritma yang dapat menyelesaikan masalah ini secara polinomial dalam semua kasus. Makalah ini membahas penerapan pohon merentang minimum dan algoritma 2-Opt dalam mencari solusi mendekati optimal untuk rute mengelilingi seluruh negara di benua Eropa. Hasil pengujian menunjukkan bahwa kombinasi antara pohon minimum merentang dan algoritma 2-Opt menghasilkan Solusi yang lebih optimal dibandingkan hanya menggunakan pohon minimum merentang.**

Kata Kunci; travelling salesman problem, aproksimasi, eropa, two opt, prim, minimum spanning tree

I. PENDAHULUAN

A. Latar Belakang

Benua Eropa merupakan benua yang di dalamnya terdapat negara-negara yang indah dan sangat diminati turis. Eropa dikenal dengan kekayaan sejarah, budaya, arsitektur, dan keindahan alamnya yang memukau. Mulai dari kota-kota klasik seperti Paris, Roma, dan London hingga keajaiban alam seperti Pegunungan Alpen dan Fjord di Norwegia, Eropa menawarkan berbagai destinasi yang menarik bagi wisatawan dari seluruh dunia.

Gambar 1 : Peta Benua Eropa

Sumber : https://www.britannica.com/topic/European-countries-by-area

Secara geografis, Eropa merupakan salah satu benua yang relatif kecil namun padat penduduknya, dengan lebih dari 40 negara yang tersebar di seluruh wilayahnya. Setiap negara memiliki karakteristik unik dan daya tarik tersendiri, yang menjadikan perjalanan antar negara di Eropa sebagai pengalaman yang sangat menarik. Sistem transportasi yang efisien seperti jaringan kereta api cepat dan maskapai penerbangan berbiaya rendah semakin memudahkan wisatawan untuk berkeliling benua ini.

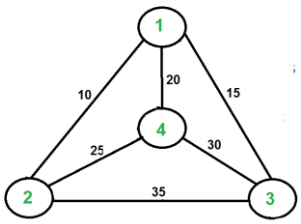
Perjalanan untuk keliling Eropa tentu saja membutuhkan biaya yang mahal, apalagi jika seseorang ingin mengunjungi seluruh negara di Eropa dalam sekali jalan, tentu saja uang yang dikeluarkan tidak sedikit. Oleh karena itu, perlu dicari rute dengan meminimalkan biaya yang harus dikeluarkan. Penulis mengasumsikan jarak tempuh yang lebih pendek akan membuat biaya yang dikeluarkan lebih sedikit (pada banyak kasus, maskapai penerbangan dan kereta api menetapkan harga berdasarkan jarak).

Dalam makalah ini, penulis akan membahas rute perjalanan tur terpendek benua Eropa yang mengunjungi seluruh negara di Eropa (44 negara)[1] dengan menggunakan salah satu teknik aproksimasi traveling salesman problem (TSP). Dalam mencari tur terpendek ini, tidak memungkinkan jika penulis menggunakan algoritma TSP yang bersifat *exact match*, dimana algoritma untuk menyelesaikan TSP dengan *exact match* terbaik saat ini, Held-Karp (algoritma pemrograman dinamis untuk menyelesaikan TSP), kompleksitas waktunya adalah eksponensial, yang di mana membutuhkan waktu bertahun-tahun jika ingin menyelesaikan TSP dengan jumlah kota lebih dari 40. Oleh karena itu, penulis akan menggunakan salah satu teknik aproksimasi dengan algoritma greedy dalam menyelesaikan masalah tersebut. Penyelesaian ini disebut teknik aproksimasi dikarenakan hasil yang didapat bukan merupakan hasil yang terbaik (bukan *exact match*), tetapi masih dalam batas wajar.

II. LANDASAN TEORI

A. *Traveling Salesman Problem (TSP)*

Traveling Salesman Problem (TSP) adalah suatu persoalan dengan bunyi sebagai berikut "Diberikan daftar kota dan daftar jarak antar masing-masing kota, carilah rute terpendek yang dapat mengunjungi setiap kota tepat satu kali dan kembali ke kota asal." Persoalan ini adalah salah satu masalah klasik dalam bidang optimasi kombinatorial dan teori graf, dan sangat relevan dalam berbagai aplikasi seperti perencanaan perjalanan, dan desain sirkuit.



Gambar 2 : Graf lengkap

Sumber : geeksforgeeks.com

Sebagai contoh, persoalan Traveling Salesman Problem pada graf diatas dapat diselesaikan dengan hasil rute terpendeknya adalah 80, dengan melewati simpul 1-2-4-3-1.

TSP merupakan salah satu persoalan NP-hard, yang dimana saat ini tidak ada algoritma polinomial untuk menyelesaikan permasalahan tersebut. Oleh karena itu, penyelesaian eksak untuk TSP sangat tidak praktis jika digunakan untuk kasus dimana jumlah kota yang besar karena membutuhkan waktu komputasi yang sangat tinggi. Sebagai contoh, untuk menyelesaikan TSP secara eksak dengan jumlah kota 50 menggunakan algoritma Held-Karp yang mempunyai kompleksitas waktu $O(n^2 2^n)$, maka dibutuhkan waktu 10^9 hari atau 8 juta tahun dengan asumsi sekali komputasi membutuhkan waktu 0,1 ms.

Algoritma untuk menyelesaiakan TSP dapat dibedakan menjadi dua, yaitu algoritma eksak dan algoritma aproksimasi. Pada algoritma eksak, hasil yang didapat pasti minimal dan akurat, beberapa contoh algoritma eksak adalah branch and bound, pemrograman dinamis (Held-Karp), dan brute force. Sedangkan untuk algoritma aproksimasi, hasil yang didapat tidak akurat tapi merupakan aproksimasi dengan batas yang wajar, keunggulan dari algoritma ini adalah waktu yang dibutuhkan untuk melakukan komputasi jauh lebih sedikit dibandingkan dengan algoritma eksak, beberapa contoh algoritma aproksimasi adalah Nearest Neighbor dan Minimum Spanning Tree. Pada makalah ini, penulis akan menggunakan Minimum Spanning Tree dalam menyelesaikan persoalan TSP.

B. *Algoritma Greedy*

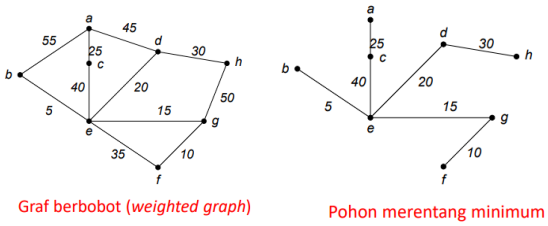
Algoritma Greedy merupakan algoritma yang secara umum digunakan untuk memecahkan permasalahan optimasi. Algoritma ini memecahkan persoalan secara langkah per langkah dan solusi dipilih pada saat itu juga tanpa mempertimbangkan masa depan. Setelah mengambil langkah yang dirasa terbaik pada saat itu, algoritma greedy berharap agar solusi yang dipilih (optimum lokal) juga merupakan optimum global.[3]

Algoritma greedy dapat dipetakan menjadi enam elemen, yaitu [3]

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah.
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih.
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif : memaksimumkan atau meminimumkan.

C. *Minimum Spanning Tree*

Minimum Spanning Tree (MST) atau pohon merentang minimum adalah subgraf dari sebuah graf berbobot yang menghubungkan semua simpul dalam graf tersebut dengan jumlah bobot sisi seminimal mungkin dan tanpa adanya siklus.



Gambar 3 : Pohon Merentang Minimum

Sumber : Munir, Rinaldi. 2023. Pohon (Bag. 1): Bahan Kuliah IF2120 Matematika Diskrit)

Pembentukan pohon merentang minimum dapat dilakukan dengan beberapa algoritma, seperti algoritma kruskal dan algoritma prim.

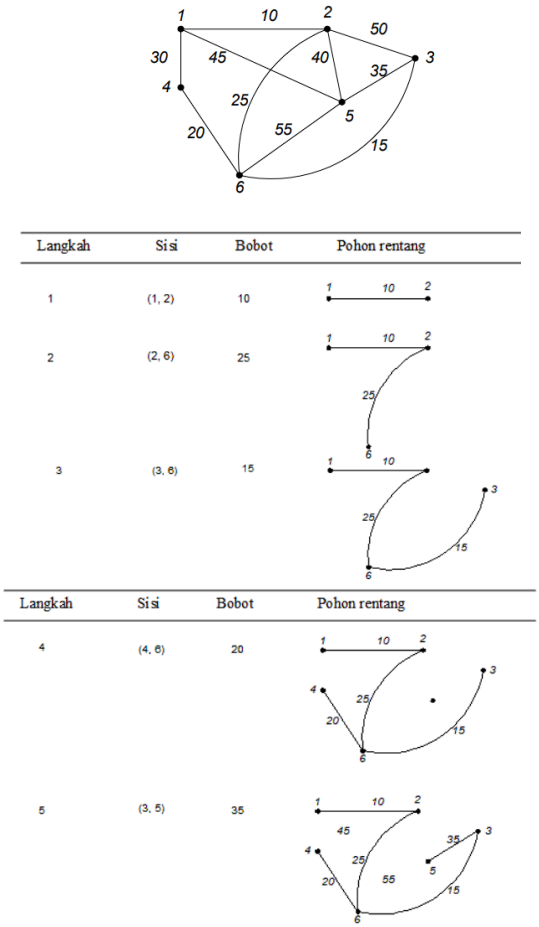
D. *Algoritma Prim*

Algoritma Prim adalah salah satu algoritma yang digunakan untuk menemukan Minimum Spanning Tree (MST) dalam sebuah graf berbobot yang tidak berarah. Algoritma ini termasuk kedalam kategori algoritma greedy karena dalam proses pembentukan MST, algoritma ini memilih sisi dengan bobot terkecil untuk di ekspan. Cara kerja algoritma prim adalah sebagai berikut :

1. Pilih sembarang simpul sebagai simpul awal dan masukkan ke dalam MST. Misalkan simpul awal ini adalah u.
2. Inisialisasi himpunan T yang akan menyimpan simpul-simpul yang termasuk dalam MST dengan T={u}
3. Temukan sisi dengan bobot terkecil yang menghubungkan simpul dalam T dengan simpul di luar T.

4. Tambahkan sisi ini ke dalam MST.
5. Tambahkan simpul yang dihubungkan oleh sisi tersebut (yang belum ada dalam T) ke dalam T.
6. Lakukan tahap 3 sampai 6 sampai seluruh simpul ada didalam T.

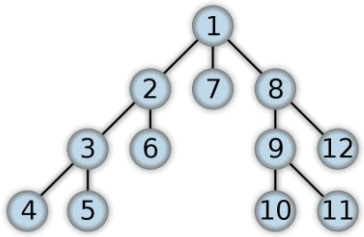
Ilustrasi dari pembentukan pohon merentang minimum dengan algoritma prim adalah sebagai berikut,



Gambar 4 : Algoritma Prim
Sumber : Munir, Rinaldi. 2023. Pohon (Bag. 1): Bahan Kuliah IF2120 Matematika Diskrit)

E. Algoritma Depth First Search

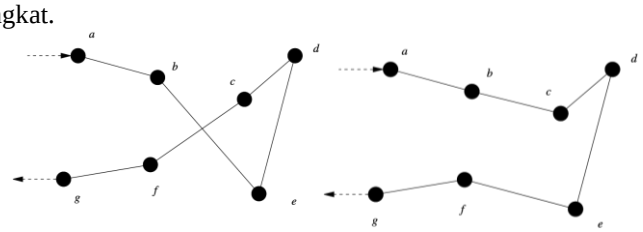
Algoritma Depth First Search (DFS) adalah salah satu algoritma yang digunakan untuk menjelajahi semua simpul dan sisi dalam sebuah graf. DFS bekerja dengan menjelajahi graf sedalam mungkin sebelum kembali dan menjelajahi cabang lain.



Gambar 5 : Penelusuran secara DFS
Sumber : https://en.wikipedia.org/wiki/Depth-first_search

F. Algoritma 2-Opt

Algoritma 2-Opt adalah salah satu algoritma yang digunakan untuk mencari solusi yang lebih baik pada masalah Traveling Salesman Problem (TSP). Algoritma ini bekerja dengan memperbaiki rute yang ada melalui serangkaian pertukaran sederhana yang disebut 2-Opt Swap. Ide dasar algoritma ini adalah menghilangkan persilangan yang terdapat pada rute TSP, sehingga membuat rute TSP tersebut lebih singkat.



Gambar 6 : Proses Algoritma 2-Opt
Sumber : https://en.wikipedia.org/wiki/2-opt#/media/File:2-opt_wiki.svg

Sebagai contoh, gambar diatas adalah sebuah rute sebelum dan sesudah diubah dengan algoritma 2-opt, pada awalnya rutenya adalah a-b-e-d-c-f-g dimana terjadi persilangan, sehingga dilakukan pembalikan pada subrute e-d-c menjadi c-d-e sehingga menghasilkan rute baru yang lebih optimal, a-b-c-d-e-f-g. Cara kerja algoritma 2-opt adalah sebagai berikut :

1. Misal terdapat rute R dan simpul v1 dan v2 dimana simpul diantara v1 dan v2 ingin dibalik.
2. Ambil rute dari start sampai v1 dan tambahkan ke rute baru dengan urutan sesuai.
3. Ambil rute dari v1+1 sampai v2 dan tambahkan ke rute baru dengan urutan dibalik.
4. Ambil rute dari v2+1 sampai start dan tambahkan ke rute baru dengan urutan sesuai.

III. Implementasi dan Pembahasan

Seperti yang sudah dijelaskan pada pendahuluan, penulis akan mencari aproksimasi rute terpendek tur lengkap di seluruh negara di Eropa, titik masing-masing negara yang penulis gunakan adalah ibu kota dari negara tersebut. Eropa sendiri terbagi menjadi beberapa wilayah, Eropa Timur, Eropa Utara, Eropa Selatan, dan Eropa Barat. Disini penulis telah membuat list masing-masing negara dan ibu kotanya di masing-masing wilayah di Eropa, berikut cuplikan dari list negara dan ibu kotanya

```
countries = {
    'Eastern Europe': [
        "Moscow Russia",
        "Warsaw Poland",
        "Kyiv Ukraine",
        "Bucharest Romania",
        "Prague Czech Republic (Czechia)",
        "Budapest Hungary",
        "Minsk Belarus",
        "Sofia Bulgaria",
        "Bratislava Slovakia",
        "Chişinău Moldova"
    ],
    'Northern Europe': [
        "London United Kingdom",
        "Stockholm Sweden",
        "Copenhagen Denmark",
        "Helsinki Finland",
        "Oslo Norway",
        "Dublin Ireland",
        "Vilnius Lithuania",
        "Riga Latvia",
        "Tallinn Estonia",
        "Reykjavik Iceland"
    ],
    'Southern Europe': [
        "Rome Italy",
          ...
          ...
```

Setelah itu, penulis mencari koordinat garis lintang (*latitude*) dan garis bujur (*longitude*) menggunakan kakas Geopy yang disediakan bahasa Python. Koordinat dari garis lintang dan garis bujur tersebut penulis simpan menjadi format JSON,

```
{
    "Eastern Europe": {
        "Moscow Russia": {
            "latitude": 55.625578,
            "longitude": 37.6063916
        },
        "Warsaw Poland": {
            "latitude": 52.2337172,
            "longitude": 21.071432235636493
        },
        "Kyiv Ukraine": {
            "latitude": 50.4500336,
            "longitude": 30.5241361
        },
        "Bucharest Romania": {
            "latitude": 44.4361414,
            "longitude": 26.1027202
        },
        "Prague Czech Republic (Czechia)": {
            "latitude": 50.0992702,
            "longitude": 14.3766834
        },
          ....
          ....
```

Setelah mendapatkan koordinat dari masing-masing ibukota negara, penulis menghitung jarak geografis dari satu negara ke seluruh negara lainnya, dengan menggunakan rumus Haversine (rumus untuk menghitung jarak antar dua titik koordinat pada sebuah bola (atau dalam kasus ini bola tersebut adalah bumi)). Jarak antar masing-masing negara disimpan pada sebuah *adjacecny matrix* berbobot. *Adjacency matrix* berbobot ini yang akan menjadi representasi graf lengkap untuk tiap 44 negara tersebut.

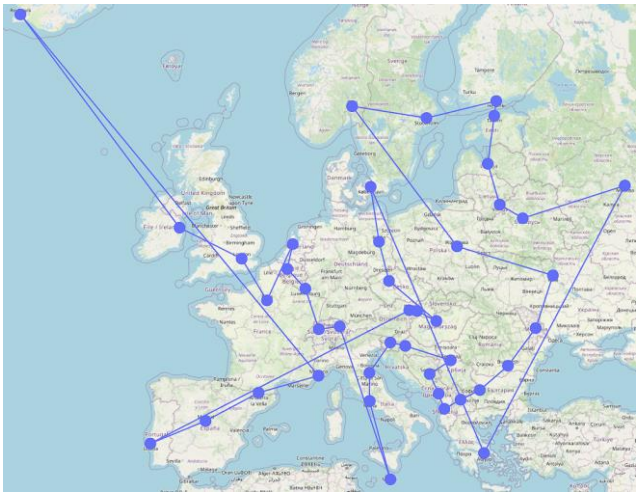
Dari graf lengkap tersebut, dicari pohon merentang minimumnya dengan algoritma Prim. Implementasi algoritma prim dalam pseudocode adalah sebagai berikut :

```
input
adj_matrix : a graph represented with adjacency matrix

output
mst : a graph of minimum spanning tree represented on a list of
tuple (from_node,to_node,weight)

function
function prim_mst(adj_matrix):
    initialize mst as an empty list
    initialize visited as an empty set
    initialize edges as an empty priority queue

    # Add start edges (edges from the first node)
    for each to, weight in enumerate(adj_matrix[0]):
        if weight > 0:
            push (weight, 0, to) onto edges

    while edges is not empty:
        (weight, frm, to) = pop from edges
        if to is in visited:
            continue
        add to to visited
        append (frm, to, weight) to mst
        for each to_next, wt_next in
enumerate(adj_matrix[to]):
            if wt_next > 0 and to_next is not in visited:
                push (wt_next, to, to_next) onto edges



    return mst
```

Algoritma Prim memanfaatkan *priority queue* dalam menyimpan sisi-sisi dari simpul yang sedang di ekspan, penggunaan *priority queue* itu bertujuan agar tiap simpul yang akan di ekspan adalah simpul dengan bobot sisi minimal. Representasi output yang digunakan menggunakan list of tuple agar output yang dikeluarkan tersortir dari bobot paling kecil untuk tiap simpul yang sama, hal ini penting agar saat melakukan pembangkitan rute dengan DFS nanti, rute yang dibangkitkan minimal. Pembentukan pohon merentang minimum ini sendiri bisa dipetakan menjadi elemen-elemen greedy, diantaranya :

1. Himpunan kandidat (C), yaitu sisi-sisi (negara) dari graf berbobot, yang nantinya beberapa sisi tersebut akan dipilih sehingga membentuk pohon merentang minimum.
2. Himpunan solusi (S), yaitu sisi-sisi yang membentuk pohon merentang minimum.
3. Fungsi solusi, yaitu fungsi yang menentukan apakah himpunan sisi yang dipilih sudah merupakan pohon merentang minimum,
4. Fungsi seleksi, yaitu fungsi prim sebagai fungsi yang menentukan himpunan solusi dari himpunan kandidat.
5. Fungsi kelayakan (feasible), memeriksa apakah sisi yang dimasukkan membentuk sirkuit di pohon merentang minimum
6. Fungsi obyektif, meminimumkan jarak antar negara pada pohon merentang minimum.

Setelah mendapatkan pohon merentang minimum, dilakukan pembentukan rute TSP dengan cara traversal pada pohon merentang minimum tersebut secara preorder atau DFS, dan tambahkan simpul awal agar rute tersebut menjadi rute TSP. Berikut adalah hasil percobaan dari pembentukan rute TSP dengan algoritma tersebut.



Gambar 6 : Rute TSP dengan Pohon Merentang Minimum

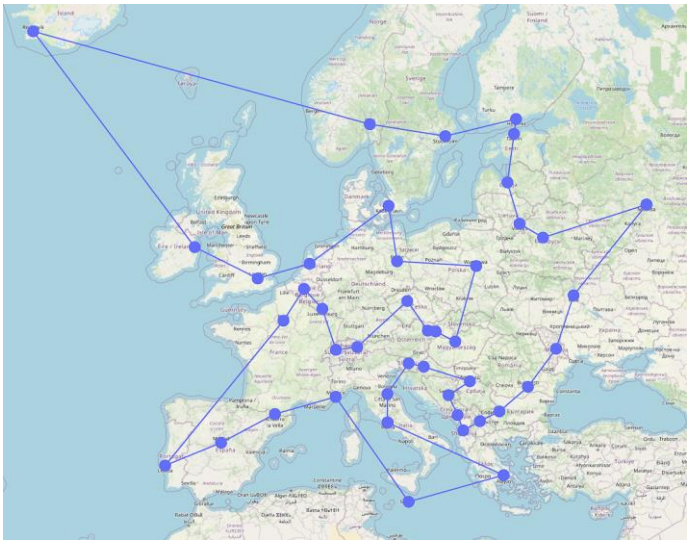
Sumber : Arsip Pribadi

Rute aproksimasi TSP ini menghasilkan *cost* yang dibutuhkan 24863 kilometer untuk mengunjungi seluruh 44 negara di Eropa dan kembali ke negara awal. Namun dapat dilihat bahwa dalam rute tersebut masih terdapat jalur-jalur yang tidak optimal dan terdapat persilangan seperti pada rute dari Islandia ke Manaco dengan Irlandia ke Inggris, dan masih banyak lagi. Rute ini masih bisa dioptimalkan lagi untuk menghilangkan seluruh persilangan tersebut dengan menggunakan algoritma 2-Opt. Pseudocode untuk pembangkitan rute baru dengan algoritma 2-Opt adalah sebagai berikut :

```
Input
Tour : array of the TSP tour
Tour length : TSP tour cost

Output
Tour : optimal tour where there are no crossing route

Function
1. function alter_tour(tour, tour_length):
 2.     n = length of tour
 3.     for i from 2 to n-1:
 4.         for j from i+1 to n:
 5.             altered_tour = swap_2opt(tour, i, j)
 6.             altered_tour_length =
calculate_tsp_cost(altered_tour, adj_matrix)
 7.
 8.             if altered_tour_length < tour_length:
 9.                 return alter_tour(altered_tour,
altered_tour_length)
10.     return tour
11.

Input
Tour : array of the TSP tour
i,j : index which subtour will be reversed

Output
Tour : Tour which the subtour is reversed

Function
12. function swap_2opt(tour, i, j):
13.     new_tour = empty list
14.     append elements tour[0] to tour[i-1] to new_tour
15.     append elements tour[i] to tour[j-1] in reverse order to
new_tour
16.     append elements tour[j] to tour[end] to new_tour
17.     return new_tour
18.
```

Setelah mengubah tur dengan minimum spanning tree tadi dengan menambahkan algoritma 2-Opt, didapatkan hasil tur sebagai berikut,

```
TSP Using Minimum Spanning Tree and 2-Opt

Moscow Russia -> Minsk Belarus -> Vilnius Lithuania ->
Riga Latvia -> Tallinn Estonia -> Helsinki Finland ->
Stockholm Sweden -> Oslo Norway -> Reykjavik Iceland ->
Dublin Ireland -> London United Kingdom -> Amsterdam
Netherlands -> Copenhagen Denmark -> Berlin Germany ->
Warsaw Poland -> Budapest Hungary -> Bratislava Slovakia
-> Vienna Austria -> Prague Czech Republic (Czechia) ->
Vaduz Liechtenstein -> Bern Switzerland -> Luxembourg
City Luxembourg -> Brussels Belgium -> Paris France ->
Lisbon Portugal -> Madrid Espana -> Andorra la Vella
Andorra -> Monaco Monaco -> Valletta Malta -> Athens
Greece -> Rome Italy -> Vatican City -> San Marino San
Marino -> Ljubljana Slovenia -> Zagreb Croatia ->
Belgrade Serbia -> Sarajevo Bosnia and Herzegovina ->
Podgorica Montenegro -> Tirana Albania -> Skopje North
Macedonia -> Sofia Bulgaria -> Bucharest Romania ->
Chişinău Moldova -> Kyiv Ukraine -> Moscow Russia

Tour cost: 19819
Execution time: 13.000011444091797 ms
```



Gambar 7 : Rute TSP dengan Pohon Merentang Minimum dan 2-Opt

Sumber : Arsip Pribadi

Dari hasil tersebut, dapat dilihat bahwa *cost* yang dihasilkan lumayan berbeda dimana dengan menggunakan pohon merentang minimum ditambah 2-Opt, *cost* yang dibutuhkan jauh lebih sedikit (19819 dibandingkan 24863), dan dari rute yang dihasilkan pun tidak lagi terdapat persilangan. Waktu eksekusi yang diperlukan juga sangat singkat, yaitu hanya 13 ms.

IV. Analisis

Penggunaan algoritma aproksimasi seperti pohon merentang minimum dan 2-Opt tidak selalu menghasilkan

solusi yang optimal, tetapi dapat memberikan solusi dalam batas yang wajar. Algoritma pohon merentang minimum tanpa menggunakan 2-Opt, dalam kasus terburuknya dapat menghasilkan *cost* dua kali solusi optimal [2]. Lalu, jika ditambahkan algoritma 2-Opt, *cost* yang dihasilkan oleh pohon merentang minimum tersebut dapat di optimasi dengan menghilangkan rute-rute silang.

Penulis telah melakukan analisis dengan membandingkan solusi yang dihasilkan oleh ketiga algoritma, TSP dengan Held-Karp (*exact solution*), TSP dengan pohon merentang minimum, dan TSP dengan pohon merentang minimum serta 2-Opt. Perbandingan tersebut dilakukan untuk jumlah negara 5 sampai 22, untuk negara dengan jumlah lebih dari 22, algoritma Held-Karp membutuhkan waktu yang lama karena kompleksitasnya yang eksponensial

Num Countries	DP Cost	MST Cost	MST Alter Cost	MST Ratio	MST Alter Ratio
5	4255	4546	4255	1,06839	1
6	4262	4552	4262	1,068043	1
7	4262	4906	4262	1,151103	1
8	4545	5189	4545	1,141694	1
9	4553	5197	4553	1,141445	1
10	4564	5783	4564	1,26709	1
11	6525	8167	6525	1,251648	1
12	7316	8326	7631	1,138054	1,043056315
13	7360	9686	7675	1,316033	1,042798913
14	7768	10470	8010	1,347837	1,03115345
15	8147	10849	8853	1,331656	1,086657665
16	8838	11604	9425	1,312967	1,066417742
17	8981	11738	9048	1,306981	1,007460194
18	9062	10725	9062	1,183514	1
19	9062	10725	9062	1,183514	1
20	11038	13164	11038	1,192607	1
21	12113	14751	12113	1,217783	1
22	13703	17162	14028	1,252426	1,023717434

Gambar 8 : Tabel perbandingan keoptimalan Aproksimasi TSP

Sumber : Arsip Pribadi

Berdasarkan pengujian tersebut dapat dilihat bahwa algoritma TSP dengan pohon merentang minimum serta 2-Opt menghasilkan solusi yang sangat baik dengan rata-rata rasio perbandingan solusi dengan solusi eksak adalah 1.016 (kolom MST Alter Ratio). Sedangkan algoritma TSP dengan pohon merentang minimum tanpa 2-Opt memiliki rasio rata-rata 1.215 (kolom MST Ratio).

Pembuatan pohon merentang minimum dengan algoritma Prim membutuhkan kompleksitas waktu $O(N^2 \log N)$, kompleksitas ini didapat dari jumlah sisi yang harus di telusuri (jumlah sisi pada graf komplit sebanyak N^2) dan memasukkan sisi ke *priority queue* membutuhkan waktu $O(\log N)$ dengan menggunakan heap. Penelusuran pohon merentang minimum dengan DFS membutuh kompleksitas waktu $O(V+E)$. Perubahan rute menggunakan 2-Opt rata-rata membutuhkan kompleksitas waktu $O(N^2)$. Dari seluruh operasi yang dijalani, pembentukan rute aproksimasi penyelesaian TSP dengan pohon merentang minimum dan 2-Opt membutuhkan kompleksitas waktu $O(N^2 \log N)$, jauh lebih cepat dibandingkan dengan penyelesaian TSP secara exact match dengan *tradeoff* solusi bukan merupakan solusi optimal.

V. KESIMPULAN

Benua Eropa memiliki banyak negara, dan menentukan rute optimal untuk mengunjungi semua negara dengan biaya minimal merupakan tantangan yang dapat diselesaikan menggunakan masalah Traveling Salesman Problem (TSP). Persoalan Traveling Salesman Problem dapat diselesaikan dengan banyak cara, salah satu diantaranya adalah dengan menggunakan algoritma aproksimasi. Penggunaan algoritma aproksimasi seperti pohon merentang minimum dan 2-Opt dapat membuat pemrosesan solusi menjadi jauh lebih cepat dan memungkinkan penyelesaian TSP untuk jumlah kota hingga ribuan, tetapi dengan konsekuensi solusi yang dihasilkan bukan merupakan solusi yang optimal.

LINK REPOSITORY GITHUB

https://github.com/RayhanFadhlan/TSP-Approximation

VIDEO LINK YOUTUBE

https://youtu.be/PIGEoJvRc70

UCAPAN TERIMA KASIH

Penulis mengucapkan ucapan terima kasih yang sebesar-besarnya kepada seluruh dosen mata kuliah Strategi Algoritma, terutama kepada Bapak Rinaldi Munir yang telah membimbing saya sehingga saya bisa menyelesaikan makalah ini.

REFERENSI

[1] "How many countries in Europe?" https://www.worldometers.info/geography/how-many-countries-in-europe/ , Diakses 10 Juni 2024

[2] "Some important heuristics for the TSP" https://ocw.mit.edu/courses/1-203j-logistical-and-transportation-planning-methods-fall-2006/03634d989704c2607e6f48a182d455a0_lec16.pdf , Diakses 10 Juni 2024

[3] Munir, Rinaldi. 2023. Greedy (Bag. 1): Bahan Kuliah Strategi Algoritma. Diakses pada tanggal 10 Juni 2024

[4] Munir, Rinaldi. 2023. Program Dinamis (Bag. 2): Bahan Kuliah Strategi Algoritma. Diakses pada tanggal 10 Juni 2024

[5] Munir, Rinaldi. 2023. Teori P,NP, dan NP-Complete (Bag. 2): Bahan Kuliah Strategi Algoritma. Diakses pada tanggal 10 Juni 2024

[6] Munir, Rinaldi. 2023. Pohon (Bag. 1): Bahan Kuliah Matematika Diskrit. Diakses pada tanggal 10 Juni 2024

[7] Shen, Max. The Traveling Salesman Problem https://aswani.ieor.berkeley.edu/teaching/FA13/151/lecture_notes/ieor151_lec17.pdf

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, 12 Juni 2024

Rayhan Fadhlan Azka 13522095