Optimalisasi Biaya Pembuatan Jaringan Transportasi Metro pada Game Cities Skylines dengan Pendekatan Travelling Salesman Problem

Muhammad Aufa Farabi - 13523023
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
iniabi838@gmail.com, 13523023@std.stei.itb.ac.id

***Abstract*—Cities Skylines merupakan suatu *game* simulasi pembangunan kota yang memungkinkan pemain untuk mengatur segala aspek Pembangunan dan manajemen kota. Untuk menciptakan kota yang berhasil, jaringan transportasi umum yang efisien dan menjangkau secara luas merupakan salah satu kunci utamanya. Salah satu transportasi umum di *game* Cities Skylines yang umum dibangun oleh pemain adalah transportasi Metro karena beberapa kelebihannya. Akan tetapi, pembangunan jaringan transportasi metro memerlukan biaya yang tidak sedikit sehingga pemain harus merencanakan rute jaringan transportasi Metro dengan biaya yang minimum agar pembangunan tersebut dapat tercapai dengan lebih cepat tanpa mengorbankan aspek-aspek lain pada manajemen kota. Permasalahan tersebut dapat dimodelkan dengan pendekatan *Travelling Salesman Problem* sebagai bagian dari konsep teori graf. Dengan menggunakan algoritma *Branch and Bound*, perencanaan jaringan Metro dapat diselesaikan dengan menemukan rute paling optimal yang mencakup seluruh stasiun dengan biaya pembangunan minimum.**

***Keywords*—Graf, Travelling Salesman Problem, Algoritma Branch and Bound, Cities Skylines, Jaringan Transportasi Metro, Biaya Pembangunan Minimum**

I. PENDAHULUAN

Gambar 1. *Game* Cities Skylines
Sumber : https://store.steampowered.com/app/255710/Cities_Skylines/

Cities Skylines adalah salah satu *game* simulasi pembangunan kota yang cukup populer saat ini. *Game* ini memungkinkan pemain untuk merancang dan mengelola kota sesuai keinginan dengan fleksibilitas yang diberikan. Hal ini memungkinkan pemain untuk mengatur tata letak jalan, zona wilayah, pengelolaan sumber daya kota, hingga transportasi umum pada kota. Salah satu elemen penting dalam pengembangkan kota di Cities Skylines adalah sistem transportasi publik yang dibangun secara efisien dan berhasil untuk mengakomodasi mobilitas penduduknya. Sistem transportasi publik yang efisien akan mengurangi tingkat kemacetan dalam kota pemain dan secara tidak langsung mampu meningkatkan pendapatan pada kota. Salah satu transportasi publik yang umum dibangun oleh pemain adalah transportasi Metro.

Gambar 2. Model stasiun Metro bawah tanah
Sumber : https://skylines.paradoxwikis.com/Transportation

Pada *game* Cities Skylines, Metro adalah suatu transportasi publik berupa kereta yang umumnya dibangun di bawah tanah atau melayang yang berfungsi untuk menghubungkan titik-titik lokasi pada pusat kota. Transportasi Metro ini serupa dengan transportasi MRT Jakarta pada dunia nyata yang memiliki jaringan stasiun bawah tanah. Metro merupakan salah satu pilihan transportasi andalan banyak pemain untuk dalam *game* karena proses pembangunannya tidak memerlukan lahan banyak dan memiliki kapasitas penumpang yang besar. Namun dengan fleksibilitas tersebut, pembangunan jaringan Metro memerlukan biaya yang tidak murah terutama pada pembuatan jalur rel bawah tanah yang memerlukan biaya lebih besar dengan penambahan biaya yang sebanding dengan panjang rel. Oleh karena itu, diperlukan perencanaan rute jaringan Metro yang efektif agar tidak memerlukan biaya Pembangunan yang besar sehingga pemain dapat mengalokasikan pendapatan dari kota nya pada hal-hal yang lain.

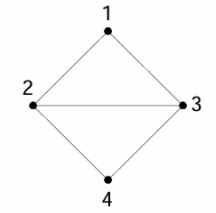
Permasalahan biaya sebagai salah satu masalah utama dalam pembuatan jaringan Metro di Cities Skylines dapat dimodelkan sebagai aplikasi *Travelling Salesman Problem* (*TSP)* menggunakan teori graf. Pada makalah ini, akan dibahas perancangan rute jaringan Metro dengan biaya yang teroptimal denganalgoritma *Branch and Bound* untuk menyelesaikan permasalahan *TSP* sebagai pemodelan dari masalah ini serta dilakukan analisis terhadap hasil implementasi algoritma tersebut untuk menentukan efektivitas dari solusi optimal yang

diinginkan.

II. LANDASAN TEORI

A. Definisi Graf

Graf didefinisikan sebagai suatu pasangan terurut dari dua buah himpunan, yakni himpunan tak kosong berisi simpul-simpul (*verticles*) dan himpunan berisi sisi (*edges)* yang menghubungkan sepasang simpul. Graf ditulis dengan notasi $G = (V,E)$, dengan V menyatakan simpul-simpul dan E sebagai sisi.

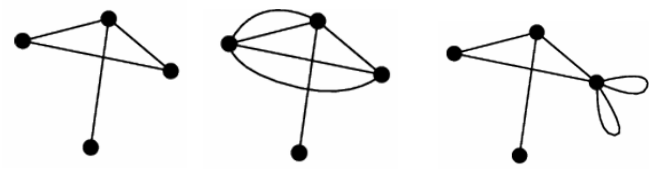


Gambar 3. Contoh graf
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

B. Jenis Graf

Graf dapat dikategorikan menjadi beberapa jenis berdasarkan sisi, arah, dan berat. Berdasarkan keberadaan sisi gelang atau sisi ganda, graf dapat dibedakan menjadi berikut:

1. Graf Sederhana (*simple graph)*, graf yang tidak mengandung baik sisi gelang maupun sisi ganda.
2. Graf tak-sederhana (*unsimple graph)*, graf yang mengandung sisi ganda atau sisi gelang. Graf yang mengandung sisi ganda disebut sebagai graf ganda (*multi-graph)*, sedangkan graf yang mengandung sisi ganda disebut sebagai graf semu (*pseudo-graph)*.

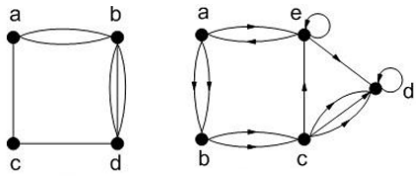


(a) (b) (c)

Gambar 4. (a) graf sederhana, (b) graf ganda, (b) graf semu
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Berdasarkan orientasi arah pada sisi, graf dikategorikan menjadi dua jenis, yakni

1. Graf tak-berarah (*undirected graph)*, graf yang sisinya tidak mempunyai orientasi arah.
2. Graf berarah (*directed graph* atau *digraph)*, graf yang setiap sisinya memiliki orientasi arah.

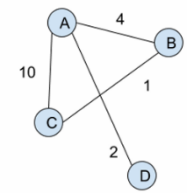


(a) (b)

Gambar 4. (a) graf tak-berarah, (b) graf berarah
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

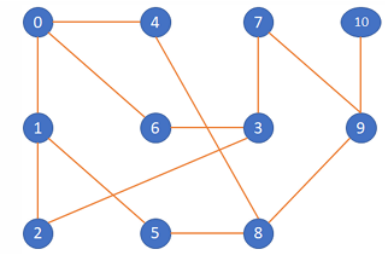
C. Terminologi Graf

Terdapat beberapa terminologi yang perlu diketahui dalam konsep graf. Pertama, Dua buah simpul pada graf dikatakan bertetangga (*adjacent*) apabila keduanya terhubung langsung melalui setidaknya satu sisi. Sebuah sisi e = (v_i, vj) dikatakan bersisian (*incidence*) dengan simpul v_i dan v_j jika sisi tersebut menghubungkan keduanya. Jumlah sisi yang berisisan pada suatu simpul dalam graf didefinisikan sebagai derajat (*degree*). Derajat pada suatu simpul v dinotasikan dalam bentuk d(v). Suatu graf dikatakan sebagai graf berbobot (*weighted graph*) apabila setiap sisi pada graf nya diberikan suatu nilai atau bobot (*weight*).



Gambar 5. Graf berbobot
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Graf memiliki lintasan dan bisa mengandung sirkuit. Lintasan (*path*) adalah suatu barisan berselang-seling atau rangkaian dari simpul dan sisi yang membentuk seperti v_o, e_1, v_1, e_2, v_2, …, v_n, e_n yang menghubungkan simpul awal v_0 ke simpul tujuan v_n. lintasan memiliki suatu Panjang yakni jumlah dari sisi yang terdapat pada suatu lintasan. Suatu lintasan yang dimulai dan berakhir di simpul yang sama disebut sirkuit (*circuit*) atau siklus (*cycle*). Contoh graf yang mengandung sirkuit dan siklus adalah seperti pada gambar berikut:



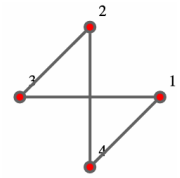
Gambar 6. Graf yang memiliki lintasan dan sirkuit
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Pada graf tersebut, lintasan 0, 4, 8, 5, 1, 0 membentuk suatu sirkuit dengan panjang sirkuitnya adalah 5.

D. Representasi Graf

Pada dasarnya, graf dapat direpresentasikan dengan tiga cara, yakni matriks ketetanggaan (*adjacency matrix)*, matriks bersisisan (*incidency matrix*), dan senarai ketetanggaaan (*adjacency list)*.

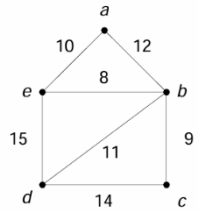
1. Matriks Ketetanggaan (*Adjacency Matrix*)
Matriks ketetanggaan merupakan matriks berukuran *n x n* dengan n sebagai jumlah simpul pada graf yang dinotasikan A = [a_{ij}]. Elemen a_{ij} merupakan tanda apakah simpul I bertetangga dengan simpul j dengan angka 1 menyatakan ada dan angka 0 menyatakan tidak ada. Untuk graf dengan sisi ganda, elemen a_{ij} diisi dengan jumlah sisi ganda, sedangkan pada graf berbobot elemen a_{ij} diisi dengan bobot sisi (i, j). Contoh representasi graf dengan matriks ketetanggaan adalah sebagai berikut



$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

(a) (b)

Gambar 7. (a) Graf tak-berbobot, (b) Matriks ketetanggaan dari graf (a)
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf

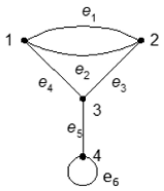


	a	b	c	d	e
a	∞	12	∞	∞	10
b	12	∞	9	11	8
c	∞	9	∞	14	∞
d	∞	11	14	∞	15
e	10	8	∞	15	∞

(a) (b)

Gambar 8. (a) graf berbobot, (b) matriks ketetanggaan dari graf (a)
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf

2. Matriks Bersisian (*incidency matrix*)
Matriks ketetanggaan merupakan matriks berukuran *n x m* dengan n adalah jumlah simpul dan m adalah jumlah sisi pada graf. Elemen b_{ij} berisi tanda apakah sisi j bersisian dengan simpul i dengan angka 1 menyatakan iya dan 0 menyatakan tidak. Contoh dari graf dengan matriks bersisiannya adalah seperti berikut:



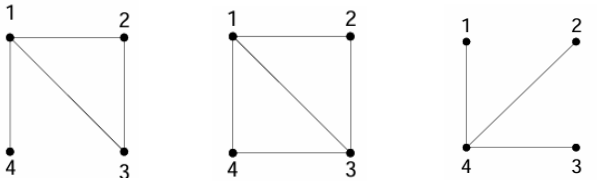
	e1	e2	e3	e4	e5	e6
1	1	1	0	1	0	0
2	1	1	1	0	0	0
3	0	0	1	1	1	0
4	0	0	0	0	1	1

(a) (b)

Gambar 9. (a) graf , (b) matriks bersisian dari graf (a)
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf

E. Lintasan dan Sirkuit Hamilton

Lintasan Hamilton adalah suatu istilah untuk lintasan pada graf yang melalui setiap simpul tepat sekali. Lintasan Hamilton yang melalui tiap simpul dalam graf tepat satu kali kecuali pada simpul asal atau akhir (kembali ke simpul awal) membentuk suatu sirkuit yang bernama sirkuit Hamilton. Graf yang memiliki sirkuit Hamilton dinamakan graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton saja disebut graf semi-hamilton. Graf yang memiliki sirkuit Hamilton pasti juga memiliki lintasan Hamilton tetapi tidak sebaliknya.



(a) (b) (c)

Gambar 10. (a) Graf Hamilton, (b) graf semi-hamilton, (c) bukan graf Hamilton atau graf semi-hamilton
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf

Untuk menentukan apakah suatu graf mengandung sirkuit Hamilton atau tidak, terdapat beberapa teori yang dapat membantu mengidentifikasi suatu graf Hamilton, yakni sebagai berikut.

- Apabila graf sederhana G memiliki n > 3 simpul dan setiap simpul memiliki derajat paling sedikit n/2 maka graf G merupakan graf Hamilton (Teorema Dirac)
- Setiap graf lengkap adalah graf Hamilton

Selain itu, terdapat beberapa teori yang bisa digunakan untuk menentukan jumlah sirkuit Hamilton pada graf Hamilton

dengan penjelasan seperti berikut

- Graf lengkap G mengandung (n – 1)!/2 buah sirkuit Hamilton apabila memiliki lebih dari sama dengan 3 simpul.
- Pada graf lengkap yang jumlah simpulnya ganjil dan lebih dari 3, terdapat (n - 1)/2 buah sirkuit Hamilton yabg saling lepas, sedangkan pada graf lengkap yang jumlah simpulnya genap dan lebih dari 4 maka terdapat (n – 2)/2 buah sirkuit Hamilton yang saling lepas.

F. Travelling Salesman Problem

Travelling Salesman Problem (TSP) adalah suatu permasalahan optimisasi yang berhubungan dengan konsep sirkuit Hamilton. Permasalahan ini dianalogikan sebagai cara seorang penjual untuk mengunjungi sejumlah kota tertentu tepat sekali saja lalu Kembali ke kota asal dengan total jarak yang terpendek. Hal ini juga dapat diartikan sebagai permasalahan dalam pencarian sirkuit Hamilton dari sejumlah simpul dan sisi pada suatu graf dengan bobot yang paling minimum.

Permasalahan *Travelling Salesman Problem* memiliki banyak terapan di dunia nyata, pada makalah ini masalah dari topik yakni pengembangan rute transportasi metro dengan biaya (sebanding dengan jarak atau bobot) optimal merupakan bentuk dari *Travelling Salesman Problem*. Untuk menyelesaikan permasalahan TSP, ada beberapa algoritma yang dapat digunakan, seperti *Branch and Bound*, *Nearest Neighbour*, metode *Dynamic Programming* (*DP)*. Pada makalah ini, *TSP* akan diselesaikan dengan pendekatan algoritma *Branch and Bound.*

G. Algoritma Branch and Bound

Algoritma *Branch and Bound* adalah suatu metode pencarian Solusi dengan cara membagi suatu permasalahan menjadi bagian-bagian masalah yang lebih kecil (*branch*) lalu mengevaluasi setiap bagian masalah tersebut berdasarkan nilai batas bawah solusi lokal dengan nilai batas atas solusi terbaik (*bound*) untuk menentukan apakah *branch* dapat dilanjutkan proses eksplorasi untuk mencapai solusi yang optimal. Metode *Branch and Bound* ini merupakan salah satu metode yang umum digunakan dalam permasalahan *Travelling Salesman Problem* (*TSP)* kareana metode ini menjamin solusi yang optimal dari permasalahan tersebut.

Untuk mencari solusi optimal pada permasalahan *TSP*, dilakukan langkah-langkah dalam proses algoritma *Branch and Bound* sebagai berikut;

1. Menentukan batas bawah dari simpul awal atau simpul "0" sebelum melakukan eksplorasi pada *branch* dengan menjumlahkan dua sisi dengan bobot minimum yang bertetangga ke setiap simpul lalu dibagi dengan dua seperti berikut

   Batas Bawah= (1/2) * Σ(Jumlah dari biaya dua sisi minimum yang bertetangga pada simpul u), untuk setiap simpul u

2. Selanjutnya adalah melakukan eksplorasi dari simpul "0" sebagai titik awal dari jalur secara rekursif. Pada simpul pertama atau "1" yang dikunjungi setelah simpul ke 0, Batas bawah simpul awal perlu diubah dengan menggunakan rumus berikut

   Batas Bawah Simpul 1= Batas Bawah Lama - ((biaya sisi minimum 0+ biaya sisi minimum 1) / 2) + (biaya sisi 0-1)

3. Dalam mengunjungi simpul-simpul berikutnya setelah simpul 1, kita juga perlu memperbarui simpul tersebut dengan cara yang serupa dengan langkah sebelumnya seperti berikut

   Batas Bawah Simpul (i > 1)= Batas Bawah Lama - ((biaya sisi minimum (i)+ biaya sisi minimum (i) / 2) + (biaya sisi (i-1)-1)

   dengan i adalah simpul yang sedang dikunjungi.

4. Proses penelusuran cabang seperti langkah sebelumnya akan terus berlanjut hingga terbentuk sirkuit hamilton. Jika suatu *branch* memiliki batas bawah yang melebihi solusi terbaik saat itu maka *branch* tersebut akan diabaikan. Hal ini akan terus berlanjut hingga menemuka solusi paling optimal dan tidak ada *branch* yang masih bisa dieksplorasi.

III. METODOLOGI

A. Batasan Masalah

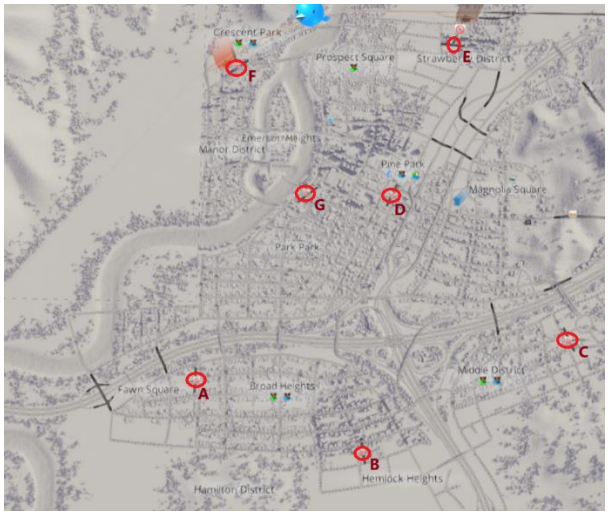
Terdapat batasan-batasan dalam menerapkan *Travelling Salesman Problem (TSP)* pada permasalahan optimalisasi biaya pembuatan jaringan transportasi Metro dalam *game* Cities Skylines. Batasan-batasan berikut di antaranya adalah

1. Stasiun Metro pada *game* yang direpresentasikan sebagai simpul diperlakukan sebagai titik untuk kemudahan perhitungan, meskipun secara visual dalam *game*, stasiun metro berbentuk persegi panjang.
2. Biaya pembangunan rel antarstasiun untuk rute yang representasi sebagai bobot tiap sisi diasumsikan sebanding dengan panjang rel atau jarak antar dua stasiun tanpa mempedulikan tingkat kedalaman rel.
3. Biaya pembangunan rute antarstasiun dihitung dengan menarik garis lurus rel antarstasiun untuk mengetahui biaya yang diperlukan.

B. Pengambilan Data Lokasi Titik Stasiun Metro dan Pemodelan Graf

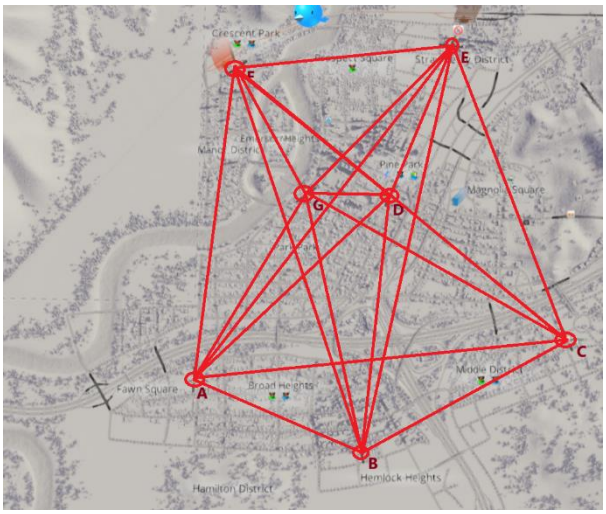
Stasiun-stasiun Metro (sebagai representasi simpul dalam pemodelan graf dibangun terlebih dahulu pada lokasi-lokasi yang strategis pada kota dalam *game*, seperti zona komersil atau *office*. Kemudian, data mengenai peta lokasi-lokasi stasiun

Metro yang sudah dibangun diperoleh dengan melakukan *screenshot* pada layar "kota" yang dibangun dalam *game* Cities Skylines.



Gambar 11. Peta letak stasiun-stasiun Metro yang dibangun
Sumber : Dokumen Pribadi

Stasiun Metro dibangun sebanyak 7 buah pada Lokasi-lokasi yang strategis, seperti pusat komersial, pada kota dalam *game*. Pada gambar peta di atas, setiap stasiun dilingkari dengan merah dan dinamai secara alfabetik secara *anti-clockwise* dimulai dari huruf "A" sampai huruf "G". Dari sini, bisa dibuat suatu pemodelan graf berbobot dengan stasiun-stasiun sebagai simpul dan lintasan antar stasiun yang dapat dibangun sebagai sisinya dengan biaya pembangunan jalur antar stasiun sebagai bobotnya.



Gambar 12. Pemodelan Graf Berbobot yang menghubungkan setiap titik Stasiun Metro
Sumber : Dokumen Pribadi

Hubungan ketetanggaan antar simpul beseerta bobot sisi-sisi pada graf yang sudah dimodelkan dapat direpresentasikan dengan matriks ketetanggaan (*adjacency matrix).* Matriks ketetanggaan ini akan berguna dalam implementasi algoritma *Branch and Bound* sebagai input pada program.

Awal/Akhir	A	B	C	D
A	0	51680	106400	74480
B	51680	0	66500	68400
C	106400	66500	0	64600
D	74480	68400	64600	0
E	120840	119320	90060	45600
F	90060	116660	121600	58520
G	57760	71440	82080	23940

Awal/Akhir	E	F	G
A	120840	90060	57760
B	119320	116660	71440
C	90060	121600	82080
D	45600	58520	23940
E	0	59280	58900
F	59280	0	44080
G	58900	44080	0

Tabel 1. Representasi graf berbobot jaringan Metro dengan matriks ketetanggaan

C. Implementasi Algoritma Branch and Bound pada Travelling Salesman Problem (TSP)

Untuk menentukan sirkuit Hamilton dengan biaya teroptimal dalam permasalahan *TSP* dari data graf perencanaan jaringan transportasi Metro yang sudah diperoleh sebelumnya, diimplementasikan algoritma *Branch and Bound* dalam bahasa Python yang akan mengembalikan rute jaringan Metro dengan biaya teroptimal beserta jumlah biaya tersebut.

Langkah pertama pada implementasi adalah inisialisasi matriks ketetanggaan (*adjacency matrix)* pada program dari data matriks ketetanggaan untuk graf transportasi Metro yang sudah dibuat sebelumnya, Di sini, dilakukan pemetaan dari nama simpul menjadi indeks angka dan sebaliknya untuk mempermudah proses algoritma tanpa mengubah hasil tampilan akhir.

```
# Inisialisasi Matriks ketetanggaan
simpulStasiun = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
adjacencyMatrix = [
    [0, 51680, 106400, 74480, 120840, 90060, 57760],
    [51680, 0, 66500, 68400, 119320, 116660, 71440],
    [106400, 66500, 0, 64600, 90060, 121600, 82080],
    [74480, 68400, 64600, 0, 45600, 58520, 23940],
    [120840, 119320, 90060, 45600, 0, 59280, 58900],
    [90060, 116660, 121600, 58520, 59280, 0, 44080],
    [57760, 71440, 82080, 23940, 58900, 44080, 0]
]
symbolToIndexMapper = {chr(65 + i): i for i in range(26)}  #
indexToSymbolMapper = {v: k for k, v in symbolToIndexMapper.items()}
```

Gambar 13. Bagian inisialisasi matriks ketetangaan dalam program
Sumber : Dokumen Pribadi

Dalam proses algoritma *Branch and Bound*, nilai batas bawah dari setiap simpul dihitung untuk memperkirakan biaya minimum rute perjalanan yang dicapai dari simpul tertentu hingga simpul dikunjungi dan kembali ke simpul awal. Perhitungan batas bawah tersebut diimplementasikan dengan fungsi yang mencari nilai bobot terminimum pertama dan kedua

pada bagian program yang ditampilkan di bawah.

```
maxsize = float('inf')

def findFirstMinimum(adjacencyMatrix, i):
    firstMinimum = maxsize
    for k in range(len(adjacencyMatrix)):
        if adjacencyMatrix[i][k] < firstMinimum and i != k:
            firstMinimum = adjacencyMatrix[i][k]
    return firstMinimum

def findSecondMinimum(adjacencyMatrix, i):
    firstminimum, secondMinimum = maxsize, maxsize
    for j in range(len(adjacencyMatrix)):
        if i != j:
            if adjacencyMatrix[i][j] <= firstminimum:
                secondMinimum = firstminimum
                firstminimum = adjacencyMatrix[i][j]
            elif adjacencyMatrix[i][j] <= secondMinimum and adjacencyMatrix[i][j] != firstminimum:
                secondMinimum = adjacencyMatrix[i][j]
    return secondMinimum
```

Gambar 14. Bagian fungsi pencarian nilai minimum pertama dan kedua dalam program
Sumber : Dokumen Pribadi

Maxsize di sini merupakan variabel yang menyimpan nilai numerik maksimum yang direpresentasikan oleh bahasa Python. Variabel-variabel yang akan diisi dengan nilai paling minimum akan diinisialisasi terlebih dahulu dengan nilai maxsize, baik pada bagian program ini maupun pada bagian program lain.

Dengan adanya fungsi pencarian nilai minimum pertama dan kedua untuk setiap simpul, sekarang fungsi rekursif dapat diimplementasikan yang bertujuan untuk menjelajahi semua kemungkinan pada jalur dari simpul-simpul yang dimulai dari simpul 0.

```
def branchandBoundRecursion(adjacencyMatrix, currentBound, currentBiaya, level, currentPath, locationVisited):
    global biayaMinimum

    # Basis
    if level == len(adjacencyMatrix):
        if adjacencyMatrix[currentPath[level - 1]][currentPath[0]] != 0:
            currentBiayaMinimum = currentBiaya + adjacencyMatrix[currentPath[level - 1]][currentPath[0]]
            if currentBiayaMinimum < biayaMinimum:
                finalPathResult(currentPath, finalPath, adjacencyMatrix)
                biayaMinimum = currentBiayaMinimum
        return

    for i in range(len(adjacencyMatrix)):
        if (adjacencyMatrix[currentPath[level-1]][i] != 0 and not locationVisited[i]):
            tempBound = currentBound
            currentBiaya += adjacencyMatrix[currentPath[level - 1]][i]

            # Update bound based on level
            if level == 1:
                currentBound -= (findFirstMinimum(adjacencyMatrix, currentPath[level - 1]) + findFirstMinimum(adjacencyMatrix, i)) / 2
            else:
                currentBound -= (findSecondMinimum(adjacencyMatrix, currentPath[level - 1]) + findSecondMinimum(adjacencyMatrix, i)) / 2

            # Rekurens
            if currentBound + currentBiaya < biayaMinimum:
                currentPath[level] = i
                locationVisited[i] = True
                branchandBoundRecursion(adjacencyMatrix, currentBound, currentBiaya, level + 1, currentPath, locationVisited)

            # Backtracking
            currentBiaya -= adjacencyMatrix[currentPath[level - 1]][i]
            currentBound = tempBound
            locationVisited = [False] * len(locationVisited)
            for j in range(level):
                if currentPath[j] != -1:
                    locationVisited[currentPath[j]] = True
```

Gambar 15. Bagian fungsi rekursif *Branch and Bound* dalam program
Sumber : Dokumen Pribadi

Ketika semua simpul telah dikunjungi sehingga terbentuk suatu rute, fungsi rekursif akan memanggil fungsi yang bernama "finalPathResult" untuk menyimpan rute tersebut ke dalam variabel finalpath lalu menambahkan simpul akhir ke variabel tersebut untuk membentuk sirkuit Hamilton. Fungsi ini akan terus dipanggil setiap kali terbentuk rute dengan bobot yang lebih minimum dibandingkan rute dengan bobot terminimum sementara hingga tidak ada rute yang bisa dieksplorasi.

```
def finalPathResult(currentPath, finalPath, adjacencyMatrix):
    for i in range (len(adjacencyMatrix)):
        finalPath[i] = currentPath[i]
    finalPath[len(adjacencyMatrix)] = currentPath[0]
```

Gambar 16. Bagian pembuatan rute final teroptimal pada program
Sumber : Dokumen Pribadi

Program implementasi algoritma *branch and bound* untuk penyelesaian *TSP* diawali dengan inisialisasi nilai-nilai variabel awal dengan nilai yang default pada *driver* program. Untuk kondisi awal yakni simpul awal atau "0", *driver* program menghitung batas bawah pertama dengan fungsi findFirstMinimum dan findSecondMinimum yang sudah dibuat sebelumnya. Setelah itu, driver program akan menjalankan fungsi rekursif untuk mengeksplorasi rute teroptimal untuk setiap simpul lalu mencetak hasil rute teroptimal untuk setiap simpul dengan biaya terminimum dari rute tersebut.

```
# Driver Program
biayaMinimum = maxsize
currentBound = 0
finalPath = [None] * (len(adjacencyMatrix) + 1)
currentPath = [-1] * (len(adjacencyMatrix) + 1)
locationVisited = [False] * len(adjacencyMatrix)

for i in range(len(adjacencyMatrix)):
    currentBound += (findFirstMinimum(adjacencyMatrix, i) + findSecondMinimum(adjacencyMatrix, i))
currentBound = math.ceil(currentBound / 2)

locationVisited[0] = True
currentPath[0] = 0

branchandBoundRecursion(adjacencyMatrix, currentBound, 0, 1, currentPath, locationVisited)

print("")
print("Rute Metro Teroptimal:", " - ".join(indexToSymbolMapper[node] for node in finalPath))
print("Total biaya minimum dari rute Metro teroptimal:", biayaMinimum)
print("")
```

Gambar 17. Bagian driver program
Sumber : Dokumen Pribadi

IV. HASIL DAN PEMBAHASAN

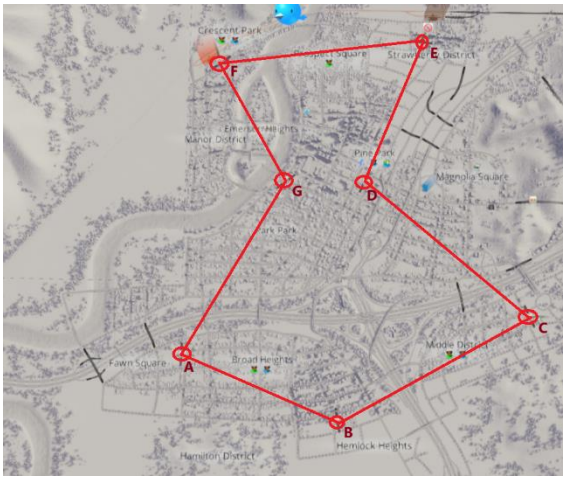
A. Hasil Eksekusi Program dan Pemodelan Rute Optimal

Dengan mengeksekusi program algoritma *Branch and Bound* pada data matriks ketetanggaan dari graf perencanaan rute jaringan transportasi Metro di game Cities Skylines, dihasilkan tampilan rute optimal dengan total biaya minimumnya sebagai berikut:

```
Rute Metro Teroptimal: A - B - C - D - E - F - G - A
Total biaya minimum dari rute Metro teroptimal: 389500
```

Gambar 18. Hasil eksekusi program
Sumber : Dokumen Pribadi

Pada gambar hasil program di atas, diketahui bahwa rute jaringan transportasi Metro dengan biaya optimal yang dimulai dari simpul (representasi dari stasiun) A adalah melalui jalur simpul B, C, D, E, F, G lalu kembali ke simpul A. Solusi rute Metro optimal ini akan tetap optimal jika dibalik, yakni A-G-F-E-D-C-B-A, karena matriks ketetanggaannya yang simetris. Total biaya minimum dari rute tersebut adalah 389.500.



Gambar 19. Pemodelan rute optimal jaringan transportasi Metro
Sumber : Dokumen Pribadi

Gambar di atas merupakan hasil pemodelan dari rute teroptimal untuk jaringan transportasi Metro berdasarkan hasil eksekusi program. Rute tersebut membentuk suatu sirkuit Hamilton pada graf berbobot dengan titik awal dan titik akhir merupakan stasiun A yang memiliki jumlah biaya minimum untuk pembangunannya rutenya. Dengan begitu, permasalahan *Travelling Salesman Problem* pada pembuatan rute jaringan Transportasi Metro dengan biaya optimal berhasil diselesaikan.

B. Analisis dan Pembahasan

Berdasarkan hasil eksekusi program algoritma *Branch and Bound* pada input matriks ketetanggaan (*adjacency matrix)* graf dari pemodelan jaringan rute transportasi Metro di game Cities Skylines, telah dihasilkan suatu solusi optimal dalam pencarian sirkuit hamilton dengan bobot terminimum pada konsep *Travelling Salesman Problem* sebagai representasi dari rute jaringan transportasi Metro dengan biaya minimum. Algoritma yang dipakai mampu menghasilkan suatu solusi dengan waktu eksekusi yang cepat (<2 detik) karena sehingga sangat efektif untuk kasus pada jumlah simpul sebanyak tujuh atau kurang. Namun, kompleksitas algoritma *Branch and Bound* yang bersifat eksponensial akan menyebabkan peningkatan waktu eksekusi yang signifikan untuk jumlah simpul yang besar.

Walaupun begitu, hasil akhir beserta pemodelan yang dibuat masih berupa aproksimasi saja karena beberapa keterbatasan yang ada pada *game Cities Skylines*. Pertama, ketidakakuratan dalam penghitungan biaya dari penarikan garis rel antarstasiun pada data yang yang disebabkan oleh bentuk representasi simpul menyebabkan perbedaan angka dari Solusi biaya optimum yang didapatkan dengan biaya pembangunan jaringan Metro sebenarnya pada *game*. Selain itu, mekanisme pada *game Cities Skylines* tidak memungkinkan fleksibilitas penuh pembuatan jaringan rel yang lurus antar stasiun untuk segala kondisi sehingga solusi teoritis yang dihasilkan oleh algoritma tidak akan sepenuhnya sesuai dengan implementasi praktikalnya dalam game.

V. KESIMPULAN

Pendekatan *Travelling Salesman Problem (TSP)* dengan memodelkan jaringan transportasi telah berhasil dalam menentukan rute jaringan transportasi Metro di game Cities Skylines dengan biaya pembangunan yang teroptimal. Penggunaan algoritma *Branch and Bound* merupakan pilihan metode yang tepat dalam menyelesaikan permasalahan *TSP* tersebut juga telah optimalisasi biaya jaringan transportasi Metro dengan solusi yang optimal serta waktu eksekusi yang cepat tanpa perlu mengeksplorasi semua kemungkinan rute.

Dari penerapan algoritma *Branch and Bound* yang dilakukan untuk mencari rute jaringan transportasi Metro dengan biaya optimal dengan pemodelan *Travelling Salesman Problem*, ditemukan solusi jaringan transportasi Metro dengan total biaya paling optimal adalah rute stasiun A -> stasiun B -> Stasiun C -> Stasiun D -> Stasiun E -> Stasiun F -> Stasiun G -> Stasiun A dengan total biaya pembangunan rutenya adalah sebesar 389.500. Walaupun dalam keadaan nyatanya pada *game* angka tersebut masih merupakan suatu aproksimasi, pendekatan *TSP* mampu memberikan gambaran yang kuat dalam perencanaan desain jaringan transportasi Metro yang efisien.

VI. LAMPIRAN

Implementasi program yang digunakan pada makalah ini dapat dilihat pada link berikut:
https://github.com/AgungLucker/Makalah-Matdis-TSP-Cities-Skylines-Metro-Transportation

VII. UCAPAN TERIMA KASIH

Pertama-tama, penulis ucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas berkat, dan rahmat-Nya sehingga makalah ini bisa terselesaikan dengan tepat waktu. Penulis juga hendak mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampu mata kuliah IF1220 kelas K-1 yang telah membimbing dan memberikan materi-materi ilmu yang berharga sebagai modal dalam penulisan makalah ini. Ucapan terima kasih penulis juga sampaikan kepada keluarga, teman-teman, dan pihak lain yang turut mendukung penulis selama proses pengembangan makalah ini. Semoga Tuhan Yang Maha Esa membalas semua kebaikan yang diberikan dengan kebaikan yang berlipat ganda.

REFERENSI

[1] Skylines Wiki,"Transportation," .[Online]. Available: https://skylines.paradoxwikis.com/Transportation [Diakses 28 Desember 2024].
[2] R. Munir, "Matematika Diskrit," 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf [Diakses 28 Desember 2024].
[3] R. Munir, "Matematika Diskrit," 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf [Diakses 3 Januari 2025].
[4] R. Munir, "Matematika Diskrit," 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf [Diakses 3 Januari 2025].
[5] Geeksforgeeks," Traveling Salesman Problem using Branch And Bound," .[Online]. Available: https://www.geeksforgeeks.org/traveling-

salesman-problem-using-branch-and-bound-2/ [Diakses 3 Desember 2024].

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, 8 Januari 2025

Muhammad Aufa Farabi 13523023