Optimasi Jalur Metro Loop dalam Cities: Skylines Menggunakan Algoritma Cheapest Link untuk Meminimalkan Biaya Terowongan

Nathanael Gunawan - 13524066
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nathanifolia@gmail.com , 13524066@std.stei.itb.ac.id

***Abstract*—Transportasi publik yang efisien merupakan elemen kunci dalam perencanaan kota, termasuk dalam permainan Cities: Skylines. Saat kota berkembang, kemacetan menjadi tantangan utama sehingga dibutuhkan solusi seperti metro bawah tanah. Namun, tingginya biaya konstruksi terowongan seringkali membuat pemain kehabisan dana sebelum seluruh jalur terhubung. Untuk itu, diperlukan metode optimasi untuk menentukan jalur metro yang menghubungkan seluruh stasiun dengan biaya minimal. Makalah ini menggunakan algoritma Cheapest Link, pendekatan terhadap permasalahan sirkuit Hamilton, untuk menemukan urutan pembangunan jalur metro yang paling efisien. Hasil simulasi menunjukkan bahwa algoritma ini mampu menghasilkan urutan rute dengan total panjang terowongan terpendek.**

Keywords—Cities: Skylines; Algoritma Cheapest Link; Metro bawah tanah; Sirkuit Hamilton; Perencanaan kota

I. PENDAHULUAN

Cities: Skylines adalah permainan video simulasi pembangunan kota yang dikembangkan oleh Collosal Order dan diterbitkan oleh Paradox Interactive. Dalam permainan video ini, pemain berperan sebagai wali kota yang bertugas merancang, membangun, dan mengelola sebuah kota dari awal, mulai dari infrastruktur, tata ruang, layanan publik, hingga ekonomi kota [1].

Dalam Cities: Skylines, saya bisa mengalami proses perkembangan kota secara menyeluruh, mulai dari sebuah desa kecil hingga menjadi kota metropolitan yang kompleks. Salah satu tantangan yang saya alami adalah seiring bertambahnya populasi dan aktivitas kota, kepadatan lalu lintas semakin memburuk yang dapat menghambat mobilitas penduduk serta mengganggu distribusi layanan publik. Untuk mengatasi hal ini, penerapan sistem transportasi umum, diperlukan di kotaku yang semakin berkembang. Untuk itu, aku memilih metro bawah tanah sebagai solusi karena tidak perlu mengubah tata letak kotaku untuk mengakomodasi jalur layang.

Saat aku melihat-lihat jalur metro di dunia nyata aku menemukan bentuk yang unik dan efisien dalam lingkungan perkotaan padat yaitu *loop line* atau jalur metro melingkar. Tidak seperti jalur metro konvensional yang membentuk sebuah garis lurus, *loop line* membentuk lingkaran yang mengelilingi pusat kota atau kawasan sub-pusat. Jalur ini telah diimplementasikan di berbagai kota besar dunia, seperti London, Moskow, Tokyo dan Beijing. Keunggulan utamanya dapat menghubungakan berbagai titik penting kota tanpa harus bergantung pada satu pusat transit utama [3].

Lalu, ketika aku mencoba membuatnya untuk pertama kali, aku menyadari bahwa konstruksi jalur bawah tanah jauh lebih mahal, sehingga tanpa kusadari uang telah habis, dan keadaan semakin parah. Oleh karena itu, selanjutnya perancangan perlu dilakukan dengan rencana yang seksama dan tidak gegabah, karena sama seperti di dunia nyata, dapat membawa masalah yang lebih besar.

Dari permasalahan optimasi jalur metro ini, saya menyadari bahwa hal ini berkaitan erat dengan teori graf dalam matematika diskrit. Secara khusus, konsep sirkuit Hamilton dan pendekatan dari Travelling Salesperson Problem (TSP) dapat diterapkan untuk menemukan urutan pembangunan jalur yang efisien. Dengan memodelkan stasiun sebagai simpul dan jalur sebagai sisi berbobot, solusi matematis dapat membantu menurunkan total biaya pembangunan, meminimalkan jarak tempuh, dan meningkatkan efisiensi operasional.

Walaupun Cities: Skylines merupakan sebuah permainan simulasi, pendekatan berbasis teori graf yang digunakan dalam makalah ini diharapkan dapat merepresentasikan bagaimana prinsip-prinsip ilmiah dapat diterapkan dalam perencanaan sistem transportasi nyata. Dengan demikian, makalah ini tidak hanya relevan dalam konteks hiburan digital, tetapi juga memiliki potensi kontribusi dalam pemahaman dan pendidikan perencanaan transportasi yang efektif dan efisien.

II. DASAR TEORI

A. Graf

Graf adalah kumpulan simpul (disebut juga *vertex* atau *node)* yang dihubungkan oleh sisi (*edge*) atau busur (*arc*). Dalam representasi visual, graf digambarkan sebagai sekumpulan titik yang dihubungkan oleh garis lurus untuk sisi dan panah arah untuk busur. Jika sebuah sisi menghubungkan sebuah simpul dengan dengan dirinya sendiri, maka struktur tersebut disebut sebagai *loop* atau gelang. Graf digunakan untuk menunjukkan

keterhubungan antar objek dengan menghubungkannya dengan objek lainnya dengan garis [3], [9].

Secara formal, graf didefinisikan sebagai pasangan G = (V,E), di mana V adalah himpunan simpul atau titik (*verices*) dan E adalah himpunan sisi, yakni pasangan yang menghubungkan sepasang simpul [9].

B. Jenis-Jenis Graf

B.1. Berdasarkan keberadaan gelang atau sisi ganda

1. Graf Sederhana

   Graf yang tidak mengandung gelang maupun sisi ganda

2. Graf Tak-sederhana

   Graf yang mengandung gelang atau sisi ganda

   a. Graf ganda (mengandung sisi ganda)

   b. Graf semu (mengandung sisi gelang)

B.2. Berdasarkan Orientasi Arah Pada Sisi

1. Graf Tak-berarah

   Graf yang sisinya tidak mempunyai orientasi arah

2. Graf berarah

   Graf yang sisinya diberikan orientasi arah

B.3. Berdasarkan Properti Khusus

1. Graf Lengkap

   Graf sederhana yang setiap simpulnya terhubung langsung dengan simpul lainnya dimana derajat setiap simpul adalah jumlah simpul dikurangi 1.

2. Graf Lingkaran

   Graf yang setiap simpulnya berderajat dua, sehingga bisa digambarkan membentuk suatu lingkaran.

3. Graf Teratur

   Graf yang setiap simpulnya memiliki derajat yang sama.

4. Graf Bipartite

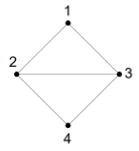
   Simpul dapat dibagi menjadi dua himpunan yang tidak saling terhubung dalam satu himpunan.

C. Terminologi Graf

Dalam implementasi algoritma pada graf, terutama untuk menyelesaikan masalah seperti Travelling Salesman Problem, diperlukan pemahaman terhadap berbagai terminologi dasar dalam teori graf. Beberapa terminologi yang relevan dalam konteks percobaan ini antara lain adalah ketetanggaan, bersisian, derajat, lintasan, sirkuit, keterhubungan, upagraf merentang, dan graf berbobot.

C.1. Ketetanggaan dan Bersisian

Ketetanggaan dan Bersisian adalah terminologi yang mendasari teori graf. Ketetanggan terjadi ketika dua simpul dalam graf terhubung langsung oleh sebuah sisi. Bersisian adalah ketika sisi dan simpul saling terhubung, misal terdapat sisi $e = (v_j,v_k)$ dikatakan e bersisian dengan simpul v_j dan v_k [3].



G_1

Gambar 2.1 Contoh graf sederhana untuk menggambarkan ketetanggan dan bersisian
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 17/06/2025)

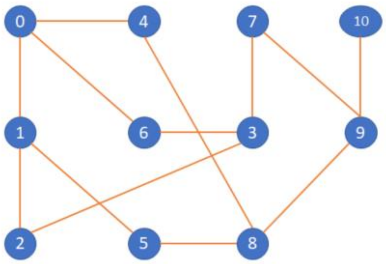
Gambar 2.1 adalah graf dimana simpul 1 bertetangga dengan simpul 2 dan 3. Untuk bersisian sisi (2,3) bersisian dengan simpul 2 dan 3.

C.2. Derajat

Derajat simpul adalah terminologi yang memainkan peran penting dalam sirkuit dikarenakan sirkuit Hamilton harus memiliki derajat 2, sehingga derajat mempermudah analisis tersebut. Derajat itu sendiri adalah jumlah sisi yang bersisian pada suatu simpul. Pada gambar 1 simpul 1 dan simpul 4 memiliki derajat 2 (d(1) = d(4) = 2) dan simpul 2 dan simpul 3 memiliki derajat 3 (d(2) = d(3) = 3).

C.3. Lintasan dan Sirkuit

Lintasan dan sirkuit adalah terminologi yang digunakan untuk Algoritma Cheapest Link karena berbasis dari sirkuit khusus yaitu sirkuit Hamilton, sedangkan lintasan digunakan untuk menggambarkan urutan simpul (stasiun) yang dikunjungi. Lintasan adalah urutan simpul-simpul yang saling terhubung oleh sisi-sisi, di mana tidak ada simpul yang dikunjungi lebih dari satu kali. Lintasan dari simpul v_1 ke simpul v_k adalah urutan $v_1, e_1, e_2, ..., e_{k-1}, v_k$ di mana setiap $e_i = \{v_i,v_{i+1}\}$ merupakan sisi dalam graf, dan setiap vi bersifat unik. Sirkuit adalah lintasan tertutup, yaitu lintasan yang dimulai dan diakhiri pada simpul yang sama.



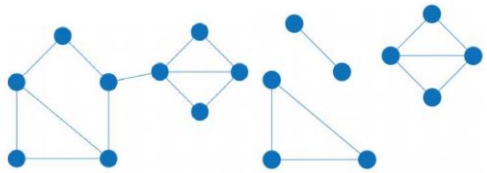
Gambar 2.2 Contoh graf menunjukkan lintasan dan sirkuit
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 17/06/2025)

Pada gambar 2.2, dapat ditinjau lintasan dan sirkuit yang terkandung didalamnya. Untuk lintasan, salah satu contohnya adalah lintasan 1, 5, 8, 9, 7. Untuk sirkuit, salah satu contohnya adalah lintasan 2, 3, 7, 9, 8, 5, 1, 2.

C.4. Keterhubungan

Tanpa keterhubungan solusi TSP tidak akan bisa diterapkan karena agar semua stasiun bisa dicapai dari simpul mana pun,

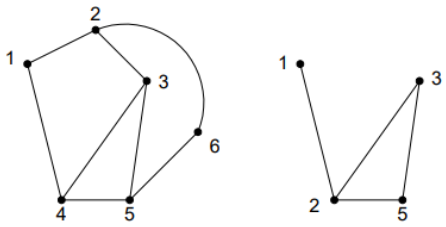
graf harus terhubung. Keterhubungan mengacu pada kemampuan simpul-simpul dalam graf untuk saling terhubung melalui lintasan. Graf dikatakan terhubung jika setiap pasangan simpul dapat dihubungkan oleh sedikitnya satu lintasan. Sebaliknya, graf dikatakan tidak terhubung (disconnected) apabila terdapat minimal satu pasangan simpul yang tidak memiliki lintasan penghubung di antaranya. Gambar 2.3 menunjukkan graf terhubung dan menunjukkan graf tak-terhubung.



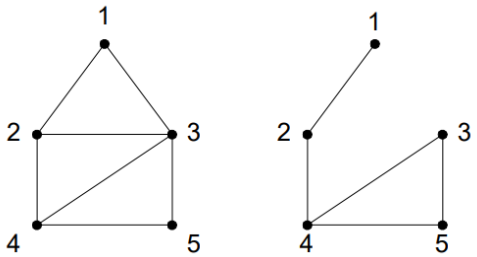
Gambar 2.3 Contoh graf terhubung (kiri) dan graf tidak terhubung (kanan)
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 17/06/2025)

C.5. Upagraf Merentang

Dalam penyelesaian TSP, tidak seluruh sisi dalam graf asal akan digunakan. Oleh karena itu, konsep upagraf (subgraph) menjadi relevan. Upagraf adalah subset dari sebuah graf, misalkan G = (V, E) adalah graf, sehingga G = (V_1, E_1) adalah upagraf jika a $V_1 \subseteq V$ dan $E_1 \subseteq E$. Upagraf merentang (spanning subgraph) adalah upagraf yang mencakup seluruh simpul dari graf asal, meskipun tidak semua sisi digunakan. Dalam konteks ini, jalur TSP dapat dipandang sebagai upagraf merentang yang optimal secara bobot.



Gambar 2.4 Graf (kiri) dan upagrafnya (kanan)
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 17/06/2025)



Gambar 2.5 Graf (kiri) dan upagraf merentangnya (kanan)
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses pada 17/06/2025)

C.6. Graf Berbobot

Graf berbobot adalah terminologi yang membuat perbedaan jarak pada kenyataan tidak perlu digambarkan. Graf berbobot adalah graf yang memiliki harga (bobot) di setiap sisinya. Bobot ini bisa merepresentasikan besar objek sesuai kebutuhan. Dengan ini, graf bisa digambarkan dengan efisien, tidak perlu menyesuaikan skala kenyataan. Pada percobaan ini, graf akan merepresentasikan jarak antar stasiun.

D. Sirkuit Hamilton

Lintasan Hamilton adalah lintasan dalam suatu graf yang melalui setiap simpul (vertex) tepat satu kali. Sementara itu, sirkuit Hamilton adalah lintasan tertutup yang melalui setiap simpul tepat satu kali dan kembali ke simpul awal, sehingga simpul awal dan akhir dilewati dua kali dalam konteks perjalanan. Graf yang memiliki sirkuit Hamilton disebut sebagai graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton tanpa kembali ke simpul asal disebut graf semi-Hamilton.

Secara formal, sebuah graf tidak berarah G = (V, E) dikatakan Hamilton jika terdapat sirkuit C ⊆ E yang melalui setiap simpul v ∈ V tepat satu kali dan kembali ke simpul awal, maka disebut sebagai lintasan Hamilton [5]. Namun dalam konteks pembangunan jalur metro berbentuk loop, fokus utama adalah pada sirkuit Hamilton.

Tidak seperti graf Euler yang memiliki karakteristik pasti (berdasarkan derajat simpul), tidak ada aturan deterministik sederhana untuk menentukan apakah suatu graf adalah graf Hamilton. Hal ini menjadikan persoalan pencarian sirkuit Hamilton termasuk dalam katergori NP-complete dalam ilmu komputer [10]. Beberapa ciri atau kondisi yang cukup (tetapi tidak perlu agar suatu graf memiliki sirkuit Hamilton, antara lain:

- Jika suatu graf sederhana dengan n simpul (n ≥ 3) memiliki setiap simpul dengan derajat ≥ n/2, maka graf tersebut Hamilton.
- Jika untuk setiap pasangan simpul tak bertetangga u dan v, berlaku deg(u) + deg(v) ≥ n, maka graf tersebut Hamilton.

Meskipun demikian, kondisi-kondisi tersebut hanya memberikan indikasi awal, dan pada kasus dunia nyata (seperti perancangan jalur metro dalam Cities: Skylines), sirkuit Hamilton biasanya dicari menggunakan algoritma heuristik atau pendekatan eksak seperti Cheapest Link atau Brute Force, tergantung pada kompleksitas dan jumlah simpul.

Dalam konteks percobaan ini, graf Hamilton digunakan sebagai model teoretis untuk menghubungkan seluruh stasiun metro dalam satu rangkaian loop dengan efisiensi tinggi, sehingga setiap lokasi hanya dikunjungi satu kali, mengurangi redundansi, serta menekan biaya konstruksi.

E. *Travelling Salesperson Problem (TSP)*

Travelling Salesperson Problem (TSP) merupakan salah satu optimasi paling klasik dalam teori graf dan riset operasi. Permasalahan ini pertama kali diperkenalkan pada awal abad ke-19 oleh William Rowan Hamilton dan Thomas Penyngton Kirkman. TSP menggambarkan situasi di mana seorang salesperson harus mengunjungi seluruh kota dalam suatu jaringan tepat satu kali dan kembali ke kota asal, dengan tujuan untuk meminimalkan total jarak atau biaya perjalanan.

TSP dapat dimodelkan sebagai permasalahan pencarian sirkuit Hamilton optimal dalam graf berbobot lengkap, di mana simpul merepresentasikan kota (atau lokasi seperti stasiun), dan sisi menggambarkan jarak atau biaya antar simpul.

TSP mengharuskan setiap kota hanya dikunjungi satu kali, dan lintasan harus kembali ke titik awal. TSP memiliki kemiripan dengan model rute terpendek (shortest path), namun berbeda dalam batasan bahwa seluruh simpul harus dilewati, bukan hanya dari satu simpul ke simpul tujuan.

Karakteristik umum dari TSP adalah:

- Perjalanan dimulai dan diakhiri di simpul yang sama.
- Seluruh simpul dikunjungi tepat satu kali.
- Tidak diperbolehkan mengunjungi simpul yang sama lebih dari satu kali sebelum seluruh simpul dikunjungi.

Dalam konteks penelitian ini, TSP dimodelkan dengan menganggap stasiun metro sebagai simpu, dan jarak antar stasiun sebagai bobot sisi. Tujuannya adalah menentukan jalur loop (sirkuit) yang paling efisien untuk menghubungkan seluruh stasiun tanpa redundansi dan dengan total jarak (biaya konstruksi) minimum.

Secara umum, formulasi matematis dari TSP adalah sebagai berikut [7]:

$$Minimalkan\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Dengan kendala:

- $\sum_{i=1,i\neq j}^{n} x_{ij} = 1,\ untuk\ semua\ j$
- $\sum_{j=1,j\neq 1}^{n} x_{ij} = 1, untuk\ semua\ i$
- $x_{ij} \in \{0,1\}, untuk\ semua\ i,j$

Keterangan :

- c_{ij} adalah bobot dari simpul i ke simpul j
- $x_{ij} = 1$ jika lintasan dari simpul i ke j dipilih sebagai bagian dari rute
- Fungsi tujuan meminimalkan total biaya lintasan
- Pembatas memastikan bahwa setiap simpul hanya dikunjungi satu kali

F. *Algoritma Cheapest Link*

Algoritma Cheapest Link adalah salah satu metode heuristik yang digunakan untuk menyelesaikan permasalahan Travelling Salesman Problem. Algoritma ini termasuk pendekatan yang bersifat greedy, di mana pada setiap langkahnya dipilih sisi (edge) dengan bobot paling kecil yang membentuk siklus parsial yang tidak valid, hingga terbentuk satu sirkuit Hamilton. Meskipun tidak menjamin solusi optimal secara global, algoritma ini cukup efisien dan praktis digunakan untuk graf berukuran kecil hingga menengah, khusudnya dalam konteks simulasi atau perancangan sistem [6].

Menurut Kusrini dan Istiyanto (2007), algoritma Cheapest Insertion yang merupakan salah satu bentuk konkret dari algoritma Cheapest Link bekerja sebagai berikut [6]:

1. Diberikan graf berbobot lengkap $G = (V, E)$ di mana setiap sisi memiliki bobot c_{ij}.
2. Dua simpul awal dengan jarak terpendek dipilih untuk membentuk sirkuit awal.
3. Pada setiap iterasi, simpul baru disisipkan pada posisi antara dua simpul dalam tur yang menyebabkan penambahan total jarak terkecil.
4. Proses diulang hingga seluruh simpul telah dimasukkan ke dalam tur.

III. METODE DAN IMPLEMENTASI

A. *Alur Penelitian*

Penelitian ini dilakukan untuk menerapkan algoritma Cheapest Link sebagai solusi dari permasalahan Travelling Salesperson Problem (TSP) yang dimodelkan dalam konteks pembangunan jalur mtero melingkar di dalam permainan Cities: Skylines.

Metode ini digunakan untuk menentukan urutan pembangunan antar stasiun metro dengan total biaya konstruksi terowongan seminimal mungkin, sekaligus memastikan bahwa seluruh stasiun terhubung dalam satu sirkuit tertutup.

Tahapan dari penelitian secara umum sebagai berikut:

1. Pengumpulan data.
2. Representasi masalah ke dalam graf berbobot.
3. Implementasi algoritma Cheapest Link.
4. Analisis hasil.

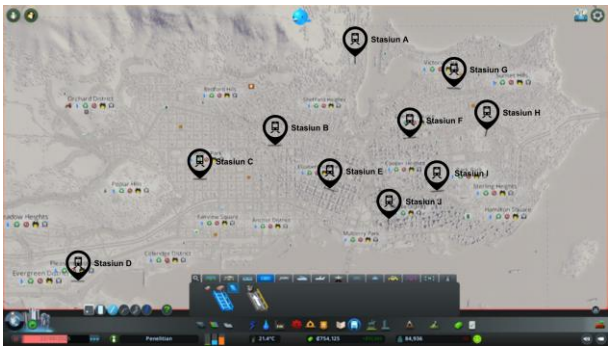
B. *Representasi Data*

Dalam penelitian ini, data dikumpulkan memalui simulasi pembangunan jalur metro bawah tanah dalam permainan Cities: Skylines. Permainan ini memungkinkan pemain membangun sistem transportasi, termasuk jalur metro, namun tidak menyediakan koordinat geografis eksplisit antar stasiun. Oleh karena itu, pendekatan yang digunakan adalah dengan mencatat biaya konstruksi yang muncul saat dua stasiun dihubungkan langsung. Biaya ini bersifat proporsional terhadap jarak antar stasiun dan digunakan sebagai bobot dalam graf.

B.1. Simpul dan Sisi

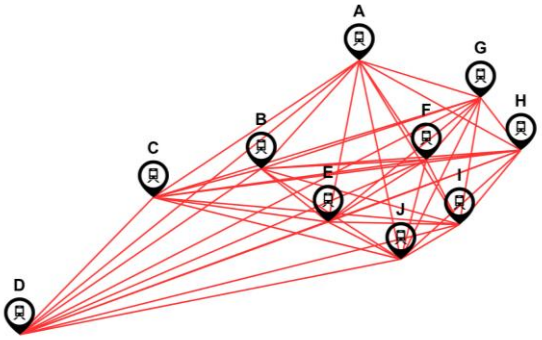
Setiap stasiun metro dianggap sebagai simpul (node) dalam graf tak berarah. Hubungan langsung antar dua stasiun

direpresentasikan dengan sisi (edge) berbobot, dengan nilai bobot setara dengan biaya pembangunan terowongan antar stasiun.



Gambar 3.1 Pemetaan lokasi stasiun untuk dihubungkan dengan satu sama lain

Gambar 3.1 menunjukkan 10 stasiun metro yang akan digunakan untuk penelitian ini. Penempatan dari stasiun ini tidak ada aturan hanya memiliki jarak antar stasiun yang logis (tidak terlalu dekat). Dari gambar dibuatkan graf yang merepresentasikan hubungan dari setiap stasiun ke setiap stasiun lainnya, dengan hasilnya pada gambar .



Gambar 3.2 Graf lengkap yang menggambarkan semua kemungkinan jalur

B.2. Matriks Biaya Antar Stasiun

Matriks berukuran 10 × 10 dengan entri simetris karena graf tak berarah. Tabel 1 adalah matriks berbentuk tabel yang merepresentasikan biaya konstruksi antar stasiun.

Tabel 3.1 Hasil pengumpulan data jarak antar stasiun

Node Tujuan → Node Asal ↓	A	B	C	D	E	F	G	H	I	J
A	0	61180	104120	193420	75240	49400	54340	79420	83600	82840
B	61180	0	38380	125400	33060	67260	96520	111720	85880	70300
C	104120	38380	0	82460	64980	109820	140980	159600	123120	104500
D	193420	125400	82460	0	141740	191900	225340	237500	196460	185440
E	75240	33060	64980	141740	0	47880	83220	88540	51680	30780
F	49400	67260	109820	191900	47880	0	31920	38000	30780	41800
G	54340	96520	140980	225340	83220	31920	0	24320	55860	76760
H	79420	111720	159600	237500	88540	38000	24320	0	38000	66880
I	83600	85880	123120	196460	51680	30780	55860	38000	0	26600
J	82840	70300	104500	185440	30780	41800	76760	66880	26600	0

C. Implementasi Algoritma Cheapest Link

Implementasi algoritma dilakukan dalam bahasa Python dan dibagi ke dalam beberapa bagian fungsional. Setiap bagian memiliki peran spesifik yang mendukung proses pemilihan sisi berbobot terendah tanpa membentuk siklus prematur, hingga terbentuk sirkuit Hamilton.

C.1. Inisialisasi Simpul dan Matriks Biaya

```python
import numpy as np

# Daftar simpul (stasiun)
nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
n = len(nodes)

# Matriks biaya antar simpul (dari data Cities: Skylines)
graph = np.array([
    [0,     61180, 104120, 193420, 75240, 49400, 54340, 79420, 83600, 82840],
    [61180, 0,     38380,  125400, 33060, 67260, 96520,111720, 85880, 70300],
    [104120,38380, 0,      82460,  64980,109820,140980,159600,123120,104500],
    [193420,125400,82460,  0,     141740,191900,225340,237500,196460,185440],
    [75240, 33060, 64980, 141740, 0,      47880, 83220, 88540, 51680, 30780],
    [49400, 67260,109820, 191900, 47880, 0,      31920, 38000, 30780, 41800],
    [54340, 96520,140980, 225340, 83220, 31920, 0,      24320, 55860, 76760],
    [79420,111720,159600, 237500, 88540, 38000, 24320, 0,      38000, 66880],
    [83600, 85880,123120, 196460, 51680, 30780, 55860, 38000, 0,     26600],
    [82840, 70300,104500, 185440, 30780, 41800, 76760, 66880, 26600, 0]
])
```

Gambar 3.3 Kode inisialisasi simpul dan matriks biaya (Sumber: Kode Penulis)

Pada bagian awal kode, didefinisikan daftar simpul, serta matriks bobot menggunakan numpy. Matriks ini diisi secara manual dari hasil pencatatan biaya konstruksi antar stasiun di Cities: Skylines.

C.2. Fungsi cheapest_link_algorithm

```python
def cheapest_link_algorithm():
    # Simpan semua sisi (cost, node1, node2)
    edges = []
    for i in range(n):
        for j in range(i + 1, n):
            edges.append((graph[i][j], i, j))

    # Urutkan berdasarkan cost terkecil
    edges.sort()

    # Array untuk menyimpan derajat tiap simpul
    degree = [0] * n
    # Array sisi hasil terpilih
    selected_edges = []

    # Union-Find untuk deteksi siklus prematur
    parent = list(range(n))

    def find(x):
        if parent[x] != x:
            parent[x] = find(parent[parent[x]])
        return parent[x]

    def union(x, y):
        px, py = find(x), find(y)
        if px != py:
            parent[py] = px
            return True
        return False

    def would_create_premature_cycle(u, v):
        """Cek apakah menambah edge (u,v) akan membuat siklus prematur"""
        return find(u) == find(v) and len(selected_edges) < n - 1

    # Cheapest Link Main Logic
    for cost, u, v in edges:
        # Aturan 1: Tidak boleh ada simpul berderajat > 2
        if degree[u] >= 2 or degree[v] >= 2:
            continue

        # Aturan 2: Tidak boleh membuat siklus prematur (kecuali edge terakhir)
        if would_create_premature_cycle(u, v):
            continue

        # Tambahkan edge
        selected_edges.append((u, v, cost))
        degree[u] += 1
        degree[v] += 1
        union(u, v)

        # Jika sudah n edges, selesai
        if len(selected_edges) == n:
            break

    return selected_edges
```

Gambar 3.4 Kode fungsi cheapest_link_algorithm
(Sumber: Kode Penulis)

Fungsi ini mengimplementasikan algoritma Cheapest Link untuk memilih sisi terbaik.

C.2.1 Pengumpulan dan Pengurutan Sisi

Sebelum melakukan pencarian, algoritma Cheapest Link membutuhkan data yang telah terurut dengan rapi dari nilai (bobot) terkecil hingga terbesar. Pada bagian ini, sisi dikumpulkan ke dalam list *edges* dan diurutkan berdasarkan nilai terendah.

C.2.2 Inisialisasi Derajat dan Union-Find

Degree menyimpan jumlah sisi yang telah terhubung ke setiap simpul. Parent digunakan untuk struktur disjoint set (union-find) guna mengecek apakah penambahan sisi akan membentuk siklus prematur.

C.2.3 Logika Pemilihan Sisi

Bagian penambahan sisi hanya jika Kedua simpulnya belum memiliki derajat > 2 dan tidak membentuk siklus sebelum simpul terakhir

C.3. Fungsi build_hamilton_circuit

```python
def build_hamiltonian_circuit(selected_edges):
    # Bangun adjacency list
    adj = [[] for _ in range(n)]
    for u, v, cost in selected_edges:
        adj[u].append(v)
        adj[v].append(u)

    # Cek apakah setiap simpul berderajat 2 (syarat sirkuit Hamiltonian)
    for i in range(n):
        if len(adj[i]) != 2:
            print(f"Warning: Simpul {nodes[i]} berderajat {len(adj[i])}, bukan 2")
            return None

    # Bangun sirkuit dengan mengikuti jalur
    circuit = [0]  # mulai dari A
    visited = {0}
    current = 0

    while len(circuit) < n:
        # Cari tetangga yang belum dikunjungi
        next_node = None
        for neighbor in adj[current]:
            if neighbor not in visited:
                next_node = neighbor
                break

        if next_node is None:
            print("Error: Tidak bisa melanjutkan sirkuit")
            return None

        circuit.append(next_node)
        visited.add(next_node)
        current = next_node

    # Tutup sirkuit kembali ke awal
    circuit.append(0)
    return circuit
```

Gambar 3.5 Kode fungsi build_hamitonian_circuit
(Sumber: Kode Penulis)

Fungsi ini menyusun rute dalam bentuk urutan simpul.

C.3.1 Membuat Adjacency List dan Validasi Derajat

Membentuk representasi graf tak berarah agar traversal jalur bisa dilakukan.Validasi memastikan bahwa semua simpul punya derajat tepat 2, syarat wajib sirkuit Hamilton.

C.3.2 DFS Sederhana untuk Menyusun Jalur

Ini adalah proses mengikuti graf dari simpul ke simpul (DFS ringan) untuk menyusun rute berurutan dari edge terpilih.

C.4. Bagian Eksekusi dan Output

```python
print("=== CHEAPEST LINK ALGORITHM ===")
selected_edges = cheapest_link_algorithm()

print(f"\nEdges terpilih ({len(selected_edges)} edges):")
total_construction = 0
for u, v, cost in selected_edges:
    print(f"{nodes[u]} - {nodes[v]}: {cost:,}")
    total_construction += cost

print(f"\nTotal biaya konstruksi: {total_construction:,}")

# Bangun sirkuit
circuit = build_hamiltonian_circuit(selected_edges)

if circuit:
    print(f"\nSirkuit Hamiltonian:")
    print(" → ".join(nodes[i] for i in circuit))

    # Hitung total biaya perjalanan
    total_travel = 0
    for i in range(len(circuit) - 1):
        u, v = circuit[i], circuit[i+1]
        total_travel += graph[u][v]

    print(f"Total biaya perjalanan: {total_travel:,}")

    # Verifikasi
    if total_travel == total_construction:
        print("✓ Verifikasi berhasil: biaya konstruksi = biaya perjalanan")
    else:
        print("✗ Error: biaya tidak cocok!")
else:
    print("Gagal membentuk sirkuit Hamiltonian!")
```

Gambar 3.6 Kode eksekusi dan output
(Sumber: Kode Penulis)

Bagian terakhir menjalankan fungsi algoritma, mencetak edge terpilih, menyusun rute akhir, dan menghitung total biaya. Bagian ini juga melakukan verifikasi bahwa total biaya dari edge terpilih sama dengan total biaya dari sirkuit, sebagai bentuk pengecekan keakuratan.

IV. PERCOBAAN

A. *Rancangan Percobaan*

Percobaan ini bertujuan untuk membandingkan efisiensi pemilihan jalur hasil optimasi menggunakan Cheapest Link dengan hasil tanpa optimasi.

Percobaan dilakukan dengan dua skenario, rute pertama yaitu rute optimasi di mana jalur metro dibentuk berdasarkan hasil algoritma Cheapest Link untuk membentuk sirkuit Hamiltoni berbiaya minimum. Skenario kedua yaitu rute non-optimasi dimana jalur metro dibangun secara intuitif/manual tanpa metode matematis khusus.

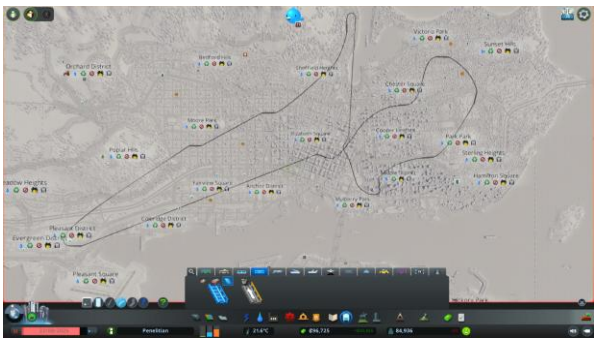
Aspek yang dibandingkan, yaitu total biaya konstruksi jalur metro, waktu tempuh kereta untuk satu putaran penuh, distribusi derajat simpul dan efisiensi relatif.

B. *Hasil Jalur Non-Optimasi*

Pada skenario ini, jalur metro dibuat secara bebas oleh pemain tanpa mempertimbangkan metode optimasi. Urutan stasiun yang dikunjungi adalah sebagai berikut:

Rute Manual (Non-Optimasi):

A → B → C → D → E → F → G → H → I → J → A



Gambar 4.1 Jalur metro bawah tanah non-optimasi

Total Biaya konstruksi : 754.125 – 96.725 = 657.400

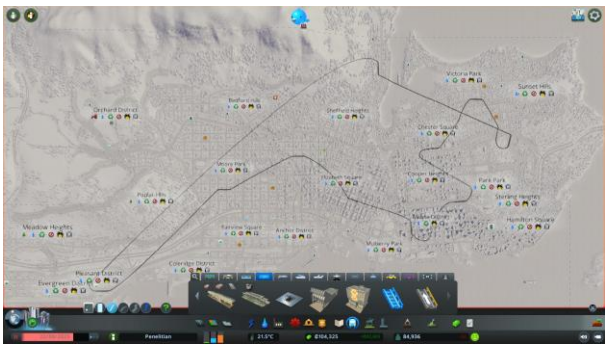
Waktu tempuh : 423 detik

Jarak 14,4 km

C. *Hasil Jalur Optimasi*

Rute Hasil Optimasi:

A → H → G → F → I → J → E → B → C → D → A



Gambar 4.2 Jalur metro bawah tanah hasil optimasi

Total Biaya konstruksi : 754.125 – 104.325 = 649.800

Waktu tempuh : 438 detik

Jarak : 13,3 km

D. *Perbandingan Hasil Percobaan*

Tabel 4.1 Tabel perbandingan hasil percobaan

Aspek	Manual Value	Optimized Value	Selisih	Persen Efisiensi (%)
Biaya Konstruksi	657.400	649.800	7.600	1,169
Waktu Tempuh (detik)	438	423	44	3,546
Panjang jalur (km)	14,4	13,3	1,1	8,271
Derajat (rata-rata)	2	2	0	Sama

E. *Analisis Hasil Percobaan*

Hasil percobaan menunjukkan bahwa panjang jalur dengan optimasi 8% lebih efisien, tetapi biaya konstruksi dan waktu tempuh tidak memiliki keefisienan yang sama dengan panjangnya. Hal ini bisa disebabkan oleh beberapa faktor, untuk biaya bisa dikarenakan persimpangan terjadi satu kali lebih banyak, ketika ada persimpangan maka jalur harus dibuat lebih dalam yang membuat biayanya meningkat. Untuk waktu tempuh, bisa disebabkan karena faktor stasiun yang terlalu dekat sehingga memerlukan banyak tikungan yang memperlambat jalan kereta, sehingga tidak seefisien panjang.

Namun, dengan optimasi seluruh aspek biaya, waktu tempuh, dan panjang jalur lebih efisien dibandingkan tanpa optimasi.

V. KESIMPULAN

Pembangunan jalur bawah tanah memiliki biaya konstruksi yang tinggi, sehingga diperlukan perencanaan yang matang dan efisien. Berdasarkan hasil percobaan, penerapan konsep Sirkuit Hamilton dan penyelesaian Travelling Salesperson Problem (TSP) melalui algoritma Cheapest Link terbukti menjadi strategi perencanaan yang efektif dalam mengoptimalkan pembangunan jalur metro melingkar (loop line) pada permainan Cities: Skylines. Dengan memanfaatkan data biaya konstruksi antar stasiun yang dikumpulkan secara manual dari simulasi permainan, algoritma ini berhasil menghitung rute berbiaya minimum yang menghubungkan semua stasiun dan kembali ke titik awal.

Optimasi ini tidak hanya menurunkan biaya pembangunan, tetapi juga berdampak pada penurunan jarak tempuh keseluruhan. Konsekuensinya, biaya operasional turut

menurun, serta efisiensi perjalanan meningkat. Rute yang lebih pendek memungkinkan waktu tempuh kereta menjadi lebih cepat, sehingga meningkatkan efektivitas layanan dan menjadikan transportasi metro sebagai pilihan yang lebih kompetitif dibandingkan moda transportasi pribadi.

Implementasi dari algoritma ini, tidak hanya untuk meningkatkan performa dalam konteks simulasi permainan., tetapi juga memiliki potensi penerapan dalam skenario perencanaan kota dan transportasi di dunia nyata. Dalam konteks tersebut, efisiensi menjadi faktor krusial. Pembangunan yang tidak efisien dapat menyebabkan pemborosan anggaran, waktu konstruksi yang lebih lama, serta rendahnya daya tarik pengguna terhadap transportasi publik. Oleh karena itu, pendekatan matematis seperti algoritma Cheapest Link dapat berkontribusi dalam mendukung perencanaan infrastruktur yang lebih optimal, terukur, dan berkelanjutan.

VI. Saran

Bagi peneliti atau praktisi yang ingin menerapkan studi serupa atau melakukan pengembangan lebih lanjut, disarankan untuk menggunakan perangkat dengan spesifikasi performa tinggi guna menghindari lag atau keterbatasan kinerja saat menjalankan simulasi Cities: Skylines. Selain itu, dianjurkan untuk menggunakan save file dengan jumlah bangunan dan populasi yang masih relatif sedikit agar proses simulasi berjalan lebih lancar dan tidak membebani sistem.

Dalam pengukuran waktu tempuh kereta, penggunaan kecepatan permainan pada tingkat 3× (tiga kali lipat) direkomendasikan untuk mengefisienkan waktu eksperimen, dengan catatan bahwa hasilnya tetap dicatat dan disesuaikan secara proporsional terhadap kecepatan normal. Pendekatan ini dapat mempercepat proses pengumpulan data tanpa mengurangi akurasi hasil pengamatan.

VII. Ucapan Terima Kasih

Penulis dapat menyelesaikan makalah ini dengan baik tidak lepas dari kontribusi dan bantuan berbagai pihak yang terus mendukung penulis. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa
2. Ayah dan Ibu Penulis
3. Dosen pengampu mata kuliah IF 1220 Matematika Diskrit
4. Teman-teman penulis.
5. AE Nino dan party anomali.

References

[1] Paradox Interactive. *Cities: Skylines – About the Game*. https://www.paradoxinteractive.com/games/cities-skylines/about. Diakses pada 17 Juni 2025.

[2] Kühne, Matthias. *Metro Rings and Loops*. https://mic-ro.com/metro/metrorings.html. Diakses pada 17 Juni 2025.

[3] Munir, Rinaldi. 2024. *Teori Graf (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf. Diakses pada 17 Juni 2025.

[4] Munir, Rinaldi. 2024. *Teori Graf (Bagian 2)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/21-Graf-Bagian2-2024.pdf. Diakses pada 17 Juni 2025.

[5] Munir, Rinaldi. 2024. *Teori Graf (Bagian 3)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/22-Graf-Bagian3-2024.pdf. Diakses pada 17 Juni 2025.

[6] Kusrini, dan Jazi Eko Istiyanto. 2007. *Penyelesaian Travelling Salesman Problem dengan Algoritma Cheapest Insertion Heuristics dan Basis Data*. Jurnal Informatika, Fakultas Teknologi Industri – Universitas Kristen Petra. http://puslit.petra.ac.id/journals/informatics. Diakses pada 17 Juni 2025.

[7] Fachrudin, Henri. 2019. *Optimasi Penentuan Rute Perjalanan Sales pada UD. Aster*. Skripsi Sarjana, Universitas Islam Majapahit Mojokerto. Diakses pada 17 Juni 2025.

[8] GeeksforGeeks. *Graph – Types and Applications*. https://www.geeksforgeeks.org/dsa/graph-types-and-applications/. Diakses pada 17 Juni 2025.

[9] Departemen Matematika FMIPA UGM. *Teori Graf – Matematika Diskrit*. https://matematikadiskrit.mipa.ugm.ac.id/teori-graf/. Diakses pada 17 Juni 2025.

[10] Ashe, Jeffrey. *Hamiltonian Paths and Circuits – Lecture Notes*. https://www.cs.toronto.edu/~ashe/ham-path-notes.pdf. Diakses pada 18 Juni 2025.

Lampiran

Video makalah dapat diakses melalui link berikut:
https://youtu.be/6y0ltbR_Mn4
Github yang berisi kode implementasi algoritma Cheapest Link dapat diakses melalui link berikut:
https://github.com/NathanaelGun/Optimasi-Loop-Metro#

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.

Sumedang, 20 Juni 2025

Nathanael Gunawan, 13524066