Optimasi Transportasi Terintegrasi di Jakarta dengan Pendekatan Spanning Tree: Minimasi Biaya dan Maksimasi Efisiensi Waktu

Muhammad Farrel Wibowo - 13523153[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]faawibowo@gmail.com, 13523153@std.stei.itb.ac.id

***Abstrak*—pertumbuhan kota Jakarta yang pesat menumbuhkan tantangan yang besar dalam menghadapi masalah mobilisasi Masyarakat. Transportasi umum, sebagai salah satu opsi Masyarakat dalam mendukung mobilisasi di sehari hari memerlukan sistem yang terintegrasi dan efisien. Masalah seperti kemacetan lalu lintas, tingginya biaya operasional, dan waktu tempuh yang tidak optimal menjadi isu utama yang membutuhkan solusi inovatif. Penelitian ini menawarkan solusi untuk mengoptimalkan jaringan transportasi terintegrasi Jakarta menggunakan *Spanning Tree-Based Genetic Algorithm* (STGA). Hasil penelitian menunjukkan kemampuan algoritma ini dalam menghasilkan konfigurasi optimal yang beragam, memberikan fleksibilitas bagi pembuat kebijakan untuk memprioritaskan efisiensi biaya atau efektivitas waktu. Studi ini menyoroti potensi STGA dalam meningkatkan perencanaan transportasi perkotaan dan menyediakan kerangka kerja yang dapat diterapkan di wilayah metropolitan lain dengan tantangan serupa.**

***Kata kunci*—Transportasi terintegrasi, pohon rentang, algoritma genetika, Pareto optimal, Jakarta.**

I. PENDAHULUAN

Transportasi merupakan salah satu hal krusial dalam mendukung keberjalanan aktivitas Masyarakat baik dalam ekonomi, social, dan budaya, terutama di kota metropolitan seperti Jakarta yang memiliki penduduk sebanyak 10.672.100 jiwa (BPS DKI Jakarta 2023) dan dengan luas kota sebesar 661,5 km^2. Tentunya hal ini menjadi sebuah tantangan besar dalam menyediakan sistem transportasi yang efisien dan terintegrasi. Masalah utama yang sering muncul adalah kemacetan lalu lintas, tingginya biaya operasional, dan waktu tempuh yang tidak optimal. Hal ini menimbulkan kebutuhan akan solusi yang mampu mengoptimalkan jaringan transportasi agar lebih efisien baik dari segi biaya maupun waktu.

Jakarta sebagai kota metropolitan memiliki transportasi umum yang terintegrasi dan terhubung di setiap daerahnya. Berbagai opsi transportasi dimiliki kota Jakarta untuk berpindah dari suatu tempat menuju tempat lainnya. Mulai dari Transjakarta, KRL, MRT, LRT, Mikrotrans dan banyak opsi lainnya. Namun, integrasi antar moda tersebut masih memiliki kelemahan, seperti ketidaksesuaian rute, waktu tempuh yang tidak efisien, dan biaya operasional yang tinggi. Oleh karena itu, diperlukan pendekatan berbasis data dan algoritma untuk merancang jaringan transportasi yang lebih optimal.

Pendekatan berbasis graf, khususnya pohon spanning minimum (Minimum Spanning Tree/MST), menawarkan solusi matematis untuk memodelkan dan mengoptimalkan jaringan transportasi. Dengan menggunakan MST, semua lokasi penting dalam jaringan dapat terhubung dengan biaya minimum tanpa siklus, sehingga efisiensi operasional dapat tercapai. Untuk masalah transportasi yang melibatkan lebih dari satu tujuan, seperti minimasi biaya dan maksimasi efisiensi waktu, algoritma genetika berbasis pohon spanning (Spanning Tree-Based Genetic Algorithm/STGA) menjadi metode yang potensial.

Makalah ini bertujuan untuk mengaplikasikan pendekatan STGA pada jaringan transportasi terintegrasi di Jakarta. Dengan memanfaatkan model graf berbobot yang merepresentasikan node sebagai lokasi transportasi dan edge sebagai rute, solusi optimal dalam bentuk Pareto dapat diperoleh. Hal ini memungkinkan pembuat kebijakan untuk memilih solusi terbaik berdasarkan prioritas, baik itu efisiensi biaya maupun waktu tempuh. Penelitian ini diharapkan dapat berkontribusi dalam perencanaan transportasi yang lebih efektif, efisien, dan terintegrasi di Jakarta.

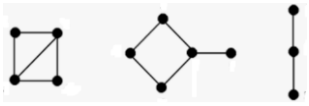
II. LANDASAN TEORI

A. Teori Graf

A.1. Definisi Graf

Graf umumnya digunakan untuk merepresentasikan diskrit dan hubungan antara objek objek tersebut. Graf didefinisikan sebagai G = (V,E), dengan V sebagai vertices atau simpul dan E sebagai edges atau sisi yang menghubungkan sepasang simpul. V didefinisikan sebagai himpunan tidak kosong dari simpul-simpul (vertices), $V = \{v_1,v_2,..,v_n\}$. Himpunan V tidak boleh kosong yang artinya sebuah graf tidak boleh tidak mengandung simpul. Sedangkan E didefinisikan sebagai himpunan sisi (edges) yang menghubungkan sepasan simpul $E = \{e_1,e_2,..,e_n\}$. Jika pada V himpunan tidak boleh kosong pada E himpunan boleh kosong, yang berarti graf boleh tidak mengandung sisi satu buah pun.

Berdasarkan ada tidaknya gelang atau sisi ganda di dalam graf, graf digolongkan menjadi dua macam yaitu graf sederhana dan graf tak sederhana. Graf sederhana *(simple graph)* merupakan graf yang tidak mengandung gelang maupun sisi ganda.

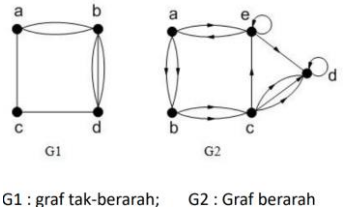


Gambar 1. Contoh Graf Sederhana
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

sedangkan graf tak-sederhana *(unsimple-graph)* merupakan graf yang mengandung sisi ganda atau gelang.

Gambar 2. Contoh Graf-tak-sederhana
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Selain itu graf juga dibedakan berdasarkan orientasinya yaitu graf tak-berarah *(undirected graph)* dan graf berarah *(directed graph atau digraph)*. Graf tak-berarah merupakan graf yang sisinya tidak mempunyai orientasi arah. Sedangkan graf berarah merupakan graf yang setiap sisinya diberikan orientasi arah.

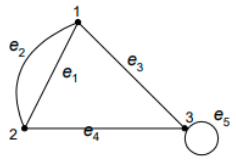


G1 : graf tak-berarah; G2 : Graf berarah

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

A.2. Terminologi di dalam graf

Terminologi didalam graf yang pertama adalah ketetanggaan (*Adjacent)*, Dua buah simpul dikatakan bertetangga bila keduanya terhubung langsung. Kemudian terdapat terminologi bersisian *(Incidency)* adalah untuk sembarang sisi $e = (v_j , v_k)$ dikatakan *e* bersisian dengan simpul v_j, atau *e* bersisian dengan simpul v_k. Terdapat juga terminologi derajat, Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut, dengan notasi *d(v)*.

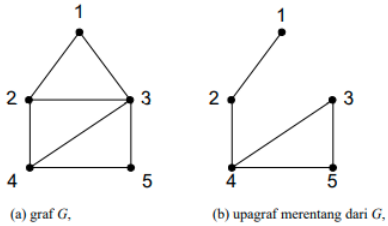


Gambar 4. Contoh Graf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Sebagai contoh pada gambar 4 dapat dilihat pada graf tersebut terdapat simpul-simpul yang bertetanggaan dan sisi yang bersisian dengan simpul. Seperti simpul 1 bertetanggaan dengan simpul 3 dan simpul 3 bertetanggaan dengan simpul 2. Kemudian ditinjau dari sisi, sisi e3 bersisian dengan simpul 1 dan 3, begitu juga dengan sisi e2 bersisian dengan simpul 1 dan 2. Pada graf gambar 4, *d(1)* = 3 dan *d(3)* = 4.

Dalam sebuah graf, istilah *lintasan* mengacu pada urutan langkah dari simpul awal V_0 ke simpul tujuan Vn dalam graf G. Lintasan ini terdiri dari rangkaian simpul dan sisi yang tersusun secara bergantian dalam bentuk $V_0,e_1,V_1,e_2,V_2,\ldots,V_{n-1}$, dengan ketentuan bahwa $e_1=(V_0,V_1)$, $e_2=(V_1,V_2)$, dan seterusnya hingga $e_n=(V_{n-1},V_n)$, di mana semua sisi tersebut merupakan bagian dari graf G dan panjang lintasan adalah jumlah sisi dalam lintasan tersebut.

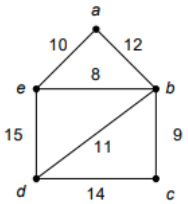
Kemudian terdapat terminologi upagraf, upagraf adalah bagian dari graf yang terdiri dari himpunan simpul dan himpunan sisi yang merupakan bagian dari graf asli. Misalkan terdapat graf G=(V,E), di mana V adalah himpunan simpul dan E adalah himpunan sisi. Sebuah upagraf G1=(V1,E1) dari G adalah graf yang memenuhi syarat V1⊆V dan E1⊆E. Komponen graf atau komponen terhubung adalah bagian dari graf yang terdiri dari simpul-simpul yang saling terhubung satu sama lain, sehingga dalam komponen tersebut setiap simpul dapat dijangkau dari simpul lainnya melalui jalur yang ada dalam graf tersebut. Mengetahui komponen terhubung dalam graf sangat berguna dalam berbagai aplikasi, seperti analisis jaringan sosial dan jaringan komputer. Upagraf merentang adalah upagraf G1=(V1,E1) dari graf G=(V,E) yang mengandung semua simpul V dari graf G. Salah satu contoh spesifik dari upagraf merentang adalah pohon rentang, yang merupakan upagraf merentang dari graf yang bersifat pohon dan tidak memiliki siklus, tetapi menghubungkan semua simpul dalam graf tersebut.



Gambar 5. Ilustrasi graf G (a) dan upagraf merentang dari G (b)
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

Terakhir terdapat terminologi graf berbobot (*weighted graf)* adalah jenis graf di mana setiap sisi (edge) memiliki nilai atau berat tertentu yang disebut bobot (weight). Bobot ini sering kali digunakan untuk mewakili biaya, jarak, durasi, atau ukuran lainnya yang relevan dengan konteks graf tersebut.

Misalkan terdapat graf G = (V, E, W), di mana V adalah himpunan simpul (vertices), E adalah himpunan sisi (edges), dan W adalah fungsi yang memetakan setiap sisi ke bobot tertentu. Contohnya, jika u dan v adalah simpul dalam V, dan terdapat sisi e = (u, v) dalam E, maka W(e) menyatakan bobot dari sisi tersebut.



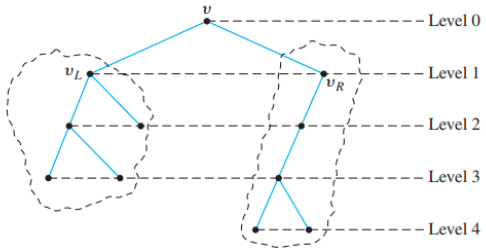
Gambar 6. Ilustrasi graf berbobot
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf

B. Teori Pohon

B.1. Definisi Pohon

Pada dasarnya pohon adalah graf tak-berarah yang terhubung dan tidak mengandung sirkuit. Terdapat 3 syarat pohon, yaitu merupakan graf tidak berarah, harus terhubung dan tidak memiliki sirkuit (siklus).

Sebuah pohon memiliki properti akar, daun, dan level. Akar merupakan satu simpul di dalam pohon yang ditunjuk sebagai akar yang berfungsi sebagai titik awal dari mana semua simpul lainnya dapat dicapai. Daun adalah simpul dengan derajat 1 (memiliki satu sisi yang menghubungkannya) disebut daun (leaf) atau simpul terminal (terminal vertex). Simpul dengan derajat lebih dari 1 disebut simpul internal (internal vertex) atau simpul cabang (branch vertex). Dalam pohon trivial (pohon dengan satu simpul), simpul unik ini juga disebut daun atau simpul terminal. Kemudian Level adalah jumlah sisi yang harus dilalui untuk mencapai simpul tersebut dari akar. Ini menunjukkan kedalaman atau jarak simpul dari akar.

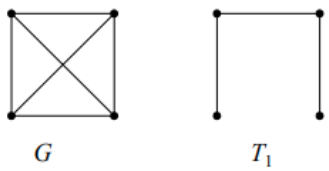


Gambar 7. Ilustrasi Pohon

Epp, S. S. (2010). *Discrete Mathematics with Applications* (pp. 737). Pacific Grove, CA: Brooks/Cole Publishing Co.

B.2. Pohon Merentang(spanning tree)

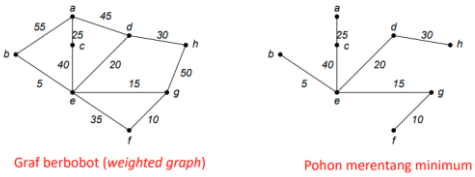
Pohon merentang adalah sejenis upagraf merentang yang berbentuk pohon dalam sebuah graf terhubung. Upagraf merentang (spanning subgraph) adalah upagraf dari graf G yang mencakup semua simpul dalam graf G. Untuk mendapatkan pohon merentang dari graf G, sirkuit di dalam graf tersebut harus dihapus. Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang sedangkan pada graf tak-terhubung dengan *k* komponen mempunyai *k* buah hutan merentang yang disebut hutan merentang *(spanning forest)*



Gambar 8. Ilustrasi graf G dan Pohon merentang dari graf G (T1)

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf

Pada sebuah graf terhubung-berbobot terdapat kemungkinan untuk mempunyai lebih dari 1 pohon merentang. Pohon merentang minimum merupakan pohon merentang yang memiliki bobot pohon paling minimum.



Gambar 9. Ilustrasi graf berbobot dan pohon merentang minimumnya

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf

C. Teori Spanning Tree-based Genetic Algorithm

Spanning Tree-Based Genetic Algorithm (STGA) adalah metode optimasi berbasis algoritma genetika yang dirancang khusus untuk menyelesaikan permasalahan optimasi jaringan, seperti transportation problem. Transportation problem adalah masalah distribusi logistik di mana sejumlah sumber (origins) harus mengirimkan barang ke sejumlah tujuan (destinations) dengan tujuan meminimalkan biaya atau mengoptimalkan kriteria lain, seperti waktu pengiriman atau keandalan jaringan. Salah satu ciri utama dari transportation problem adalah kebutuhan untuk menjaga keseimbangan antara total pasokan dan permintaan. Jika tidak seimbang, node tambahan (dummy node) dapat ditambahkan untuk memastikan formulasi masalah menjadi seimbang.

Solusi dari transportation problem dapat direpresentasikan sebagai spanning tree, yaitu subgraf berbasis pohon yang menghubungkan semua node tanpa membentuk siklus. Dalam STGA, solusi spanning tree ini direpresentasikan menggunakan teknik encoding yang disebut *Prufer number*. Teknik ini memungkinkan representasi setiap spanning tree secara unik dan efisien, hanya memerlukan m+n−2 unit memori untuk merepresentasikan graf dengan mmm sumber dan n tujuan. Berbeda dengan metode representasi matriks yang membutuhkan m×n unit memori, penggunaan *Prufer number* menjadikan STGA lebih efisien secara memori dan lebih cepat dalam proses komputasi. Selain itu, representasi ini memungkinkan solusi untuk secara langsung memuat informasi derajat koneksi antar node, sehingga setiap perubahan dalam representasi dapat dijamin menghasilkan solusi yang valid.

STGA terdiri dari beberapa tahap utama, yaitu representasi solusi, operasi genetik, dan seleksi. Operasi genetik melibatkan *crossover* dan *mutation* yang dirancang untuk menghasilkan individu baru yang valid dalam bentuk spanning tree. Prosedur ini memastikan bahwa solusi tetap memenuhi batasan pasokan dan permintaan. Pada tahap seleksi, metode kombinasi $\mu + \lambda$ dan *roulette wheel* digunakan untuk memilih solusi terbaik sekaligus menjaga keragaman populasi. Dengan struktur seperti ini, STGA mampu mencari solusi Pareto optimal untuk masalah multi-kriteria, di mana peningkatan satu tujuan hanya mungkin dilakukan dengan mengorbankan tujuan lain.

Berikut merupakan Metode algoritma genetika berbasis pohon rentang (*Spanning Tree-Based Genetic Algorithm*) yang diusulkan oleh Gen et al. (2000) menggunakan representasi Prufer number untuk encoding solusi spanning tree.

Procedure STGA
1. Initialize population P(t) with random feasible chromosomes (Prufer numbers).
2. Evaluate each chromosome in P(t) by decoding it into a spanning tree and calculating fitness.
3. Determine the Pareto solution set E(t) for the current population.
4. Repeat until termination condition is met:
   4.1. Recombine population P(t) using crossover and mutation to generate offspring C(t).
      - Ensure all offspring chromosomes are feasible.
   4.2. Evaluate the offspring in C(t) by decoding and calculating fitness.
   4.3. Update the Pareto solution set E(t) with solutions from both P(t) and C(t).
   4.4. Select the next generation population P(t+1) using a mixed strategy:
      - $\mu + \lambda$ selection to keep the best chromosomes.
      - Roulette wheel selection to maintain diversity.
5. Output the final Pareto solution set E(t).

III. METODE

A. Pengumpulan Data

Dalam penelitian ini akan diidentifikasi simpul atau node di dalam jaringan transportasi Jakarta. Simpul-simpul ini meliputi lokasi-lokasi penting seperti stasiun MRT, KRL, LRT, halte TransJakarta, dan terminal feeder. Setiap simpul akan mewakili titik-titik utama di mana moda transportasi tersedia atau bertemu.

Namun pada penelitian ini hanya akan diambil 3 daerah dari setiap kota administratif (Jakarta Pusat, Jakarta Utara, Jakarta Barat, Jakarta Timur, Jakarta Selatan) sebagai representatif dari daerah lainnya. Daerah-daerah tersebut antara lain adalah:

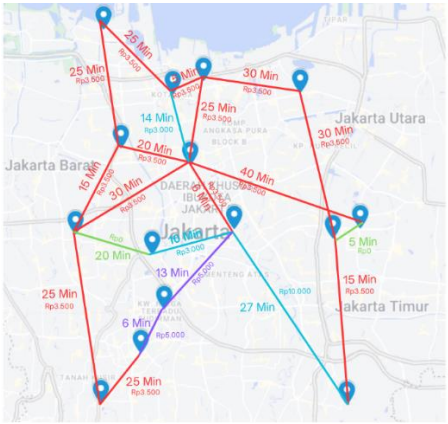
Kota Administratif	Daerah
Jakarta Pusat	Dukuh Atas, Gambir, Tanah Abang
Jakarta Utara	Tanjung Priok, Pluit, Ancol,
Jakarta Barat	Kota Tua, Grogol, Kebon Jeruk,
Jakarta Timur	Pulogadung, Rawamangun, Halim
Jakarta Selatan	Senayan, Pondok Indah, Blok M

Tabel 1. Kota administratif dan daerah yang digunakan

Dari daftar daerah tersebut dikumpulkan halte dan stasiun yang ada pada daerah tersebut dan dijadikan simpul. Kemudian dari setiap simpul tersebut akan dihubungkan berdasarkan peta pada gambar 10 dan dibuat graf berbobot. Setiap sisinya akan memiliki 2 atribut yaitu biaya dan lama waktu perjalanan sesuai dengan tujuan penelitian ini yaitu minimasi biaya dan efisiensi waktu.

B. Pembentukan Graf Berbobot

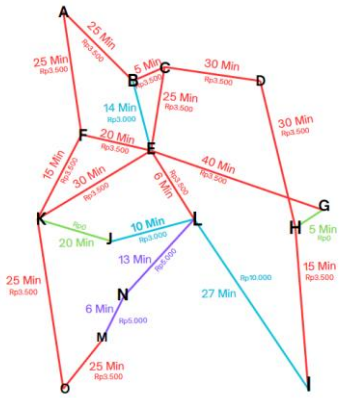
Berikut merupakan hasil visualisasi graf dari pengumpulan data:



Gambar 10. Ilustrasi graf berbobot jaringan transportasi

Berdasarkan gambar 11 ilustrasi graf yang dibuat garis merah menunjukkan garis transportasi umum yang digunakan dengan Transjakarta atau BRT, garis ungu merupakan garis lajur MRT, garis biru merupakan garis yang menggunakan KRL atau commuter line sedangkan garis hijau merupakan garis yang menggunakan mikrotrans. Setiap jenis transportasi umum memiliki biaya yang berbeda beda dan disesuaikan juga dengan jaraknya. Untuk memudahkan perhitungan Lokasi-lokasi yang sebagai node akan dimisalkan sebagai huruf.

A: Pluit, B: Kota tua, C: Ancol, D: Tanjung Priok, E: Gambir, F: Grogol, G: Pulo Gadung, H: Rawamangun, I: Halim, J: Tanah Abang, K: Kebon Jeruk, L: Dukuh Atas, N: Senayan, O: Pondok Indah.



Gambar 11. Ilustrasi graf berbobot jaringan transportasi dengan huruf

Selanjutnya untuk menjalankan algoritma Spanning Tree-based genetic Algorithm diperlukan konversi dari graf kedalam matriks bobot biaya dan waktu. berikut merupakan tabel matriks dari bobot waktu dan biaya.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A		3500				3500									
B	3500		3500		3000										
C		3500		3500	3500										
D			3500					3500							
E		3000	3500			3500	3500				3500	3500			
F	3500				3500						3500				
G					3500			0							
H				3500			0		3500						
I								3500				10000			
J											0	3000			
K					3500	3500				0					3500
L					3500				10000	3000				5000	
M														5000	3500
N												5000	5000		
O											3500		3500		

Tabel 2. Matrix Biaya Transportasi

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A		25				25									
B	35		5		14										
C		5		30	25										
D			30					30							
E		14	25			20	40				30	6			
F	25				30						15				
G					40			5							
H				30			5		15						
I								15				27			
J											20	10			
K					30	15				20					25
L					6				27	10				13	
M														6	25
N												13	6		
O											25		25		

Tabel 3. Matrix Waktu Transportasi

D. Penggunaaan Spanning Tree-Based genetic Algorithm

Setelah dilakukannya pengumpulan data dan data-data graf tersebut dikonversi menjadi sebuah matriks dan kemudian data-data tersebut akan dimasukkan ke dalam algoritma *spanning tree-based genetic algorithm* untuk menentukan sebuah rute terbaik dengan mempertimbangkan biaya dan waktu yang paling optimum.

Proses dimulai dengan inisialisasi, di mana program menerima input berupa matriks biaya dan matriks waktu yang mewakili hubungan antar node dalam jaringan. Dengan parameter tertentu, seperti jumlah populasi awal, jumlah generasi, dan probabilitas mutasi, program membangun populasi awal yang terdiri dari solusi acak berupa spanning tree yang valid. Solusi ini menghubungkan semua node tanpa membentuk siklus.

Pada setiap generasi, algoritma melakukan evolusi dengan tiga tahap utama. Pertama, crossover dilakukan dengan menggabungkan edge dari dua solusi (parent) untuk menghasilkan solusi baru (offspring), yang kemudian diperbaiki agar tetap menjadi spanning tree valid. Kedua, mutasi dilakukan pada beberapa solusi offspring dengan peluang tertentu untuk menambahkan variasi, di mana edge dihapus dan diganti dengan edge baru sambil menjaga validitas spanning tree. Ketiga, setiap solusi dievaluasi untuk menghitung total biaya dan waktu berdasarkan edge yang dipilih. Solusi yang tidak terdominasi oleh solusi lain, yaitu Pareto optimal, dipilih untuk membentuk populasi baru.

Setelah semua generasi selesai, algoritma menghasilkan solusi Pareto optimal yang mencerminkan trade-off terbaik antara biaya dan waktu. Solusi ini memberikan berbagai alternatif yang dapat dipilih sesuai dengan prioritas pengguna, seperti meminimalkan biaya atau waktu pengiriman. Berikut merupakan program algoritmanya.

```python
import numpy as np
from typing import List, Tuple
import random

class STGA:
    def __init__(self, cost_matrix: np.ndarray, time_matrix: np.ndarray):
        self.cost_matrix = cost_matrix
        self.time_matrix = time_matrix
        self.num_nodes = len(cost_matrix)

```

Gambar 12. *Program Algoritma Spanning tree based genetic Algorithm bag 1*

```python
def create_random_tree(self) -> List[Tuple[int, int]]:
    """Create a random valid spanning tree."""
    edges = []
    nodes = set(range(self.num_nodes))
    connected = {random.choice(list(nodes))}
    unconnected = nodes - connected

    while unconnected:
        node1 = random.choice(list(connected))
        node2 = random.choice(list(unconnected))

        # Check if valid connection exists
        if self.cost_matrix[node1][node2] > 0 or self.time_matrix[node1][node2] > 0:
            edges.append((node1, node2))
            connected.add(node2)
            unconnected.remove(node2)

    return edges

def evaluate_solution(self, edges: List[Tuple[int, int]]) -> Tuple[float, float]:
    """Calculate total cost and time for a solution."""
    total_cost = sum(self.cost_matrix[u][v] for u, v in edges)
    total_time = sum(self.time_matrix[u][v] for u, v in edges)
    return total_cost, total_time

def crossover(self, parent1: List[Tuple[int, int]], parent2: List[Tuple[int, int]]) -> List[Tuple[int, int]]:
    """Perform crossover between two parent solutions."""
    # Take random edges from both parents
    combined_edges = list(set(parent1 + parent2))
    child_edges = []
    nodes = set()

    # Start with a random edge
    start_edge = random.choice(combined_edges)
    child_edges.append(start_edge)
    nodes.update(start_edge)

    # Add edges until we have a spanning tree
    while len(nodes) < self.num_nodes:
        valid_edges = [
            edge for edge in combined_edges
            if (edge[0] in nodes) != (edge[1] in nodes)  # Only one end connected
            and edge not in child_edges
        ]

        if not valid_edges:
            # If stuck, add a random valid edge
            remaining_nodes = set(range(self.num_nodes)) - nodes
            connected_node = random.choice(list(nodes))
            new_node = random.choice(list(remaining_nodes))
            if self.cost_matrix[connected_node][new_node] > 0:
                edge = (connected_node, new_node)
                child_edges.append(edge)
                nodes.add(new_node)
            continue

        new_edge = random.choice(valid_edges)
        child_edges.append(new_edge)
        nodes.update(new_edge)

    return child_edges

def mutate(self, solution: List[Tuple[int, int]]) -> List[Tuple[int, int]]:
    """Perform mutation on a solution."""
    if random.random() > 0.3:  # 30% mutation rate
        return solution

    # Remove a random edge and add a new valid one
    edge_to_remove = random.choice(solution)
    new_solution = [edge for edge in solution if edge != edge_to_remove]

    # Find nodes that were connected by the removed edge
    all_nodes = set()
    for edge in new_solution:
        all_nodes.update(edge)

    # Add a new valid edge
    remaining_nodes = set(range(self.num_nodes)) - all_nodes
    if remaining_nodes:
        node1 = random.choice(list(all_nodes))
        node2 = random.choice(list(remaining_nodes))
        if self.cost_matrix[node1][node2] > 0:
            new_solution.append((node1, node2))
        else:
            return solution  # If can't find valid edge, return original

    return new_solution

def is_dominated(self, sol1: Tuple[float, float], sol2: Tuple[float, float]) -> bool:
    """Check if solution 1 is dominated by solution 2."""
    return (sol1[0] >= sol2[0] and sol1[1] > sol2[1]) or \
           (sol1[0] > sol2[0] and sol1[1] >= sol2[1])
```

Gambar 13. *Program Algoritma Spanning tree based genetic Algorithm bag 2*

```python
def get_pareto_front(self, population: List[List[Tuple[int, int]]]) -> List[List[Tuple[int, int]]]:
    """Get non-dominated solutions."""
    pareto_front = []
    for solution1 in population:
        is_dominated = False
        obj1 = self.evaluate_solution(solution1)

        for solution2 in population:
            if solution1 != solution2:
                obj2 = self.evaluate_solution(solution2)
                if self.is_dominated(obj1, obj2):
                    is_dominated = True
                    break

        if not is_dominated:
            pareto_front.append(solution1)

    return pareto_front

```

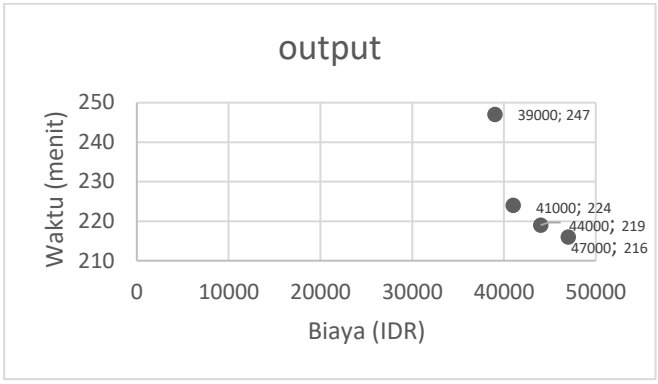
Gambar 14. *Program Algoritma Spanning tree based genetic Algorithm bag 3*

```python
    def run(self, population_size: int = 50, generations: int = 30) -> List[List[Tuple[int, int]]]:
        """Run the STGA algorithm."""
        # Initialize population
        population = [self.create_random_tree() for _ in range(population_size)]

        for gen in range(generations):
            # Create offspring through crossover
            offspring = []
            for _ in range(population_size // 2):
                parent1, parent2 = random.sample(population, 2)
                child = self.crossover(parent1, parent2)
                offspring.append(child)

            # Apply mutation
            mutated = [self.mutate(solution) for solution in offspring]

            # Combine populations and select the best
            population.extend(offspring)
            population.extend(mutated)

            # Get Pareto front
            population = self.get_pareto_front(population)

            # If population too small, add random solutions
            while len(population) < population_size:
                population.append(self.create_random_tree())

            print(f"Generation {gen+1}: Found {len(population)} Pareto solutions")

        return self.get_pareto_front(population)



# Example usage
if __name__ == "__main__":
    # Using the existing cost and time matrices
    cost_matrix = np.array([
        # A     B     C     D     E     F     G     H     I     J     K     L     M     N     O
    [   0, 3500,    0,    0,    0, 3500,    0,    0,    0,    0,    0,    0,    0,    0],  # A
    [3500,    0, 3500,    0, 3000,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # B
    [   0, 3500,    0, 3500, 3500,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # C
    [   0,    0, 3500,    0,    0,    0,    0, 3500,    0,    0,    0,    0,    0,    0],  # D
    [   0, 3000, 3500,    0,    0, 3500, 3500,    0,    0,    0, 3500, 3500,    0,    0],  # E
    [3500,    0,    0,    0, 3500,    0,    0,    0,    0,    0, 3500,    0,    0,    0],  # F
    [   0,    0,    0,    0, 3500,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # G
    [   0,    0,    0, 3500,    0,    0,    0,    0, 3500,    0,    0,    0,    0,    0],  # H
    [   0,    0,    0,    0,    0,    0,    0, 3500,    0,    0,    0,    0,    0,    0],  # I
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0, 3000,    0,    0],  # J
    [   0,    0,    0,    0, 3500, 3500,    0,    0,    0,    0,    0,    0, 3500,    0],  # K
    [   0,    0,    0,    0, 3500,    0,    0,    0,10000, 3000,    0,    0,    0, 5000],  # L
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0, 5000, 3500],  # M
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,5000, 5000,    0],  # N
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0, 3500,    0, 3500,    0],  # O
    ])

    time_matrix = np.array([
        # A     B     C     D     E     F     G     H     I     J     K     L     M     N     O
    [   0,   25,    0,    0,    0,   25,    0,    0,    0,    0,    0,    0,    0,    0],  # A
    [  35,    0,    5,    0,   14,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # B
    [   0,    5,    0,   30,   25,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # C
    [   0,    0,   30,    0,    0,    0,    0,   30,    0,    0,    0,    0,    0,    0],  # D
    [   0,   14,   25,    0,    0,   20,   40,    0,    0,    0,   30,    6,    0,    0],  # E
    [  25,    0,    0,    0,   30,    0,    0,    0,    0,    0,   15,    0,    0,    0],  # F
    [   0,    0,    0,    0,   40,    0,    0,    5,    0,    0,    0,    0,    0,    0],  # G
    [   0,    0,    0,   30,    0,    0,    5,    0,   15,    0,    0,    0,    0,    0],  # H
    [   0,    0,    0,    0,    0,    0,    0,   15,    0,    0,    0,   27,    0,    0],  # I
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   20,   10,    0,    0],  # J
    [   0,    0,    0,    0,   30,   15,    0,    0,    0,   20,    0,    0,    0,   25],  # K
    [   0,    0,    0,    0,    6,    0,    0,    0,   27,   10,    0,    0,   13,    0],  # L
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    6,   25],  # M
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],  # N
    [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   25,    0,   25,    0],  # O
    ])

    algorithm = STGA(cost_matrix, time_matrix)
    pareto_solutions = algorithm.run()

    print("\nFinal Pareto Optimal Solutions:")
    for i, solution in enumerate(pareto_solutions, 1):
        cost, time = algorithm.evaluate_solution(solution)
        print(f"\nSolution {i}:")
        print(f"Edges: {solution}")
        print(f"Total Cost: {cost}")
        print(f"Total Time: {time}")
```

Gambar 15. *Program Algoritma Spanning tree based genetic Algorithm bag 4*

Setelah 10 kali program dijalankan, algoritma menemukan beberapa solusi Pareto optimal yang dapat dijadikan acuan untuk pengambilan keputusan. Setiap generasi menghasilkan populasi solusi Pareto optimal yang terdiri dari 50 solusi. Stabilitas jumlah solusi Pareto di setiap generasi menunjukkan bahwa algoritma berjalan dengan baik, menemukan trade-off yang relevan antara total biaya (cost) dan total waktu (time).

Berikut merupakan grafik hasil generasi program:



Gambar 14. Grafik output total biaya dan waktu pareto optimum

IV. Analisis dan Pembahasan

Berdasarkan grafik pada gambar 14 hasil output menunjukkan solusi Pareto optimal yang dihasilkan oleh algoritma STGA (Spanning Tree-based Genetic Algorithm) untuk masalah transportasi dua kriteria, yaitu meminimalkan biaya (total cost) dan waktu (total time). Dari 10 solusi yang diberikan, terdapat beberapa solusi dengan nilai total cost dan total time yang sama, seperti Solusi 1, 4, 5, 6, 7, 8, dan 9. Semua solusi ini memiliki total cost sebesar 44,000 dan total time sebesar 219, menunjukkan bahwa algoritma menemukan beberapa konfigurasi spanning tree berbeda yang memberikan performa identik untuk kedua kriteria. Hal ini dapat memberikan fleksibilitas tambahan dalam memilih solusi yang sesuai, misalnya berdasarkan aspek lain seperti topologi jaringan atau kemudahan implementasi.

Solusi lainnya mencerminkan trade-off yang berbeda antara biaya dan waktu. Solusi 2 memiliki total cost yang lebih rendah, yaitu 39,000, tetapi membutuhkan total time yang lebih tinggi, yaitu 247. Ini menunjukkan preferensi terhadap penghematan biaya dengan konsekuensi waktu yang lebih lama. Sebaliknya, Solusi 10 memiliki total time terendah, yaitu 216, tetapi dengan total cost yang lebih tinggi, yaitu 47,000. Ini cocok untuk kasus di mana waktu adalah faktor yang sangat kritis, meskipun harus mengeluarkan biaya yang lebih besar. Solusi 3 menawarkan keseimbangan antara biaya dan waktu, dengan total cost sebesar 41,000 dan total time sebesar 224, yang menunjukkan alternatif bagi skenario yang menghendaki kompromi antara kedua tujuan.

Ragam solusi Pareto optimal ini mencerminkan kemampuan algoritma untuk mengeksplorasi berbagai konfigurasi spanning tree. Solusi yang memiliki kombinasi biaya dan waktu yang identik tetapi berbeda dalam strukturnya menunjukkan adanya redundansi solusi dengan performa sama, yang dapat dipilih sesuai dengan kebutuhan praktis. Sementara itu, variasi dalam trade-off antara biaya dan waktu pada solusi lainnya memberikan berbagai opsi bagi pengguna untuk memilih solusi yang sesuai dengan prioritas dan batasan spesifik yang dihadapi dalam masalah transportasi tersebut.

V. Kesimpulan

Berdasarkan analisis dan pembahasan serta grafik pada gambar 14, Algoritma Spanning tree based genetic algorithm

berhasil dalam menentukan berbagai Solusi pareto optimal dengan mempertimbangkan dua kriteria atau bicriteria transportation problem yaitu waktu dan biaya. Solusi yang dihasilkan menunjukkan trade-off yang beragam antara waktu dan biaya mulai dari solusi dengan biaya rendah tetapi waktu lebih lama hingga solusi dengan waktu tercepat tetapi biaya lebih tinggi. Selain itu, ditemukan beberapa solusi dengan total cost dan total time yang sama namun memiliki konfigurasi spanning tree berbeda, memberikan fleksibilitas dalam pemilihan solusi berdasarkan kebutuhan tambahan, seperti struktur jaringan atau implementasi praktis. Hasil ini menunjukkan efektivitas algoritma dalam mengeksplorasi ruang solusi dan menghasilkan opsi optimal yang dapat disesuaikan dengan preferensi atau batasan pengguna.

VII. Ucapan Terimakasih

Pertama-tama, Saya mengucapkan syukur kepada Tuhan Yang Maha Esa karena dengan rahmatNya, makalah berjudul "Optimasi Transportasi Terintegrasi di Jakarta dengan Pendekatan Spanning Tree: Minimasi Biaya dan Maksimasi Efisiensi Waktu" dapat terselesaikan dengan tepat waktu. Tidak lupa juga saya ucapkan terimakasih kepada bapak Rinaldi Munir dan bapak Arrival yang telah mengajar mata kuliah IF 2120 Matematika Diskrit dan membimbing selama satu semester ini. Terima kasih terhadap segala pihak yang tidak dapat disebutkan satu persatu sehingga turut membantu terselesaikan pembuatan makalah ini.

Referensi

[1] Susanna S. Epp (Aug 2010). *Discrete Mathematics with Applications*. Pacific Grove, CA: Brooks/Cole Publishing Co. p. 720-733. ISBN 978-0-495-39132-6.

[2] Gen, M., & Li, Y.-Z. (1998). Spanning tree-based genetic algorithm for bicriteria transportation problem. *Computers & Industrial Engineering*, 35(3–4), 531-534. https://doi.org/10.1016/S0360-8352(98)00151-X

[3] Gen, M. and Y.Z. Li, Solving Multi-objective Transportation Problem by Spanning Tree-based Genetic Algorithm, Adaptive Computing in Design and Manuj~ctu~~ [I. Parmee, ed.), pp.95-108, Springer-Verlag, 1998.

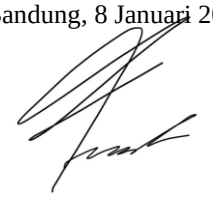
[4] R. Munir, "Graf Bagian 1," Bahan kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2023, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/20-Graf-Bagian1-2024.pdf, diakses 3 Januari 2025.

[5] R. Munir, "Pohon Bagian 1," Bahan kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2023, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/23-Pohon-Bag1-2024.pdf, diakses 4 Januari 2025.

Pernyataan

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

Bandung, 8 Januari 2025



Muhammad Farrel Wibowo 13523153