BAB I
PENDAHULUAN

1.1. Latar Belakang

Traveling Salesman Problem (TSP) pertama kali dikenalkan pada tahun 1800 oleh dua orang matematikawan dari Irlandia dan Inggris yang bernama Wiliam Roman Hamilton dan Thomas Penyngton. Pada tahun 1930, seorang matematikawan bernama Karl Menger mengawali mempelajari TSP di Vienna dan Harvard yang kemudian dipublikasikan oleh Hassler Whitney dan Merrill Flood di Princenton. Pada tahun 1985, Hoffman dan Wolfe mengganggap TSP sebagai teka-teki sederhana mengenai rute seorang *salesman* yang harus mengunjungi sejumlah kota dengan jarak terpendek. Seiring berjalannya waktu, masalah ini menjadi salah satu topik optimasi kombinatorial yang paling banyak dipelajari, karena relevansi dan aplikasinya yang luas dalam berbagai bidang seperti logistik, transportasi, perencanaan jaringan komputer, manufaktur, hingga pariwisata (Ramadhania & Rani, 2021). TSP telah menjadi salah satu masalah optimasi kombinatorial yang paling banyak dipelajari karena banyaknya aplikasi di dunia nyata seperti dalam sektor logistik dan transportasi serta manufaktur yang memerlukan rute optimal untuk efisiensi operasional (Taha, 2017).

Masalah yang sering dihadapi adalah ketidakmampuan untuk mengoptimalkan rute perjalanan yang mengakibatkan waktu dan biaya operasional yang lebih tinggi. Dengan meningkatkan jumlah lokasi yang harus dikunjungi, kompleksitas perhitungan rute meningkat secara eksponensial. Menurut Ishaya et al. (2019), metode eksak seperti *Branch and Bound* memberikan solusi optimal, namun membutuhkan waktu komputasi yang lama, terutama pada dataset yang besar, sehingga kurang praktis untuk aplikasi nyata. Di sisi lain, metode heuristik seperti *Nearest Neighbor* dan *Cheapest Insertion* dapat menghasilkan solusi dengan lebih cepat, meskipun hasilnya tidak selalu optimal (Winston, 2004).

Dalam mengatasi keterbatasan tersebut, metode metaheuristik seperti *Genetic Algorithm* (GA) dan *Ant Colony Optimization* (ACO) dikembangkan sebagai pendekatan alternatif yang lebih efisien. *Genetic Algorithm* dan *Ant Colony Optimization* dirancang untuk menemukan solusi yang mendekati optimal dalam waktu relatif singkat, sehingga lebih cocok untuk masalah berskala besar yang membutuhkan efisiensi waktu (Santosa & Ai, 2017).

Penelitian ini bertujuan untuk membandingkan efektivitas beberapa metode penyelesaian TSP yaitu metode eksak seperti *Branch and Bound* dan *Cutting Plane*, metode heuristik seperti *Nearest Neighbor* dan *Cheapest Insertion* dan metode metaheuristik seperti *Genetic Algorithm* dan *Ant Colony Optimization*. Fokus penelitian pada dataset simetris dengan harapan dapat menemukan metode yang unggul dalam hal kecepatan komputasi dan kualitas solusi yang dihasilkan. Selain itu, diharapkan dapat memberikan wawasan yang bermanfaat dalam memilih metode yang tepat untuk mengoptimalkan rute distribusi sehingga menekan biaya operasional dan meningkatkan efisiensi. Memahami kelebihan dan kekurangan

masing – masing metode akan membantu menentukan strategi paling sesuai dengan kebutuhan operasional.

Berdasarkan uraian diatas, maka akan dilakukan penelitian dengan judul **"Perbandingan Metode Eksak, Heuristik dan Metaheuristik pada Penyelesaian Traveling Salesman Problem"**

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan sebelumnya, permasalahan yang akan dibahas dalam penelitian ini meliputi:

a. Bagaimana performa dari metode eksak, heuristik dan metaheuristik dalam penyelesaian TSP pada berbagai ukuran dataset?
b. Seberapa optimal solusi yang dihasilkan tiap metode untuk setiap ukuran dataset?
c. Bagaimana waktu komputasi yang dihasilkan dari masing – masing metode?

1.3. Batasan Masalah

Batasan masalah yang terdapat dalam penelitian ini adalah sebagai berikut:

a. Penelitian ini menggunakan data simetris pada tiga kategori dataset: kecil (5 dan 10 kota), dataset sedang (50 kota) dan dataset besar (100 kota).
b. Metode yang digunakan:
   - Metode eksak: *Branch and Bound* dan *Cutting Plane* yang hanya diterapkan pada dataset kecil.
   - Metode heursitik dan metaheuristik: *Nearest Neighbor, Cheapest Insertion, Genetic Algorithm* dan *Ant Colony Optimization* yang diterapkan pada semua ukuran dataset dari kecil hingga besar.
c. Evaluasi hasil dibatasi pada dua parameter utama, yaitu waktu komputasi yaitu durasi yang dibutuhkan setiap metode untuk menghitung rute dan kualitas solusi, yaitu total jarak rute yang dihasilkan serta konsistensi metode dalam menghasilkan jarak minimum di setiap pengujian.
d. Penelitian ini tidak mencakup optimasi parameter secara mendalam pada metode metaheuristik, tetapi menggunakan konfigurasi standar atau umum dari masing – masing algoritma.

1.4. Tujuan Penelitian

Tujuan dari penelitian ini antara lain:

a. Membandingkan performa metode eksak, heuristik dan metaheuristik dalam menyelesaikan TSP pada dataset dengan ukuran kecil (5 dan 10 kota), sedang (50 kota) dan besar (100 kota).
b. Mengetahui efektivitas setiap metode dalam menyelesaikan TSP pada berbagai ukuran dataset.

1.5. Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini antara lain:

a. Memberikan panduan mengenai metode yang efektif untuk berbagai ukuran dataset dalam TSP.
b. Menjadi referensi untuk memilih metode penyelesaian TSP di bidang logistik dan optimasi rute.
c. Menambah literatur mengenai perbandingan metode dalam TSP.

1.6. Landasan Teori

Pada sub bab ini, akan dijelaskan mengenai konsep dari *Traveling Salesman Problem* (TSP) dan bagaimana permasalahan ini diselesaikan menggunakan berbagai pendekatan. Penjelasan akan mencakup deskripsi TSP sebagai masalah optimasi kombinatorial, karakteristik data yang digunakan (simetris atau asimetris), serta tujuan utama dari TSP itu sendiri.

Di samping itu, akan dibahas proses penerapan masing – masing metode mulai dari metode eksak, heuristik dan metaheuristik, yang akan dijelaskan berdasarkan cara algoritma bekerja dan langkah – langkah penyelesaiannya. Dengan demikian, pembaca akan mendapatkan gambaran yang lebih jelas tentang bagaimana masing – masing pendekatan bekerja dalam menyelesaikan masalah TSP.

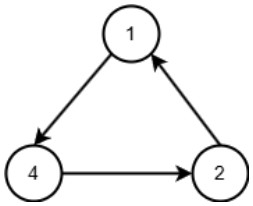
1.6.1. Traveling Salesman Problem (TSP)

Traveling Salesman Problem adalah masalah optimasi kombinatorial yang bertujuan menemukan rute terpendek yang memungkinkan seorang *salesman* mengunjungi setiap kota tepat satu kali sebelum akhirnya kembali ke titik awal. Istilah *Traveling Salesman* berasal dari seorang legendaris yang harus melakukan perjalanan mengelilingi kota – kota dengan jarak yang paling efisien. TSP memiliki dua variasi utama, yaitu TSP simetris dan TSP asimetris (Rardin, 2017).

TSP simetris adalah jenis masalah di mana jarak antara dua kota adalah sama, terlepas dari arah perjalanan. Artinya, jika jarak dari kota A ke kota B adalah $d_{A,B}$ maka jarak dari kota B ke kota A juga sama yaitu $d_{B,A}$. Formulasi matematis untuk TSP simetris didefenisikan sebagai berikut:

$$x_{ij} = \begin{cases} 1 & \text{jika dari kota } i \text{ ke kota } j \\ 0 & \text{jika tidak} \end{cases}$$

Didefenisikan sebagai x_{ij} di mana $i < j$. Tujuannya adalah meminimalkan total biaya perjalanan, sambil memastikan bahwa setiap kota dikunjungi hanya sekali dan membentuk suatu rute *(tour)* tertutup tanpa adanya *subtour* (rute yang dimulai dan berakhir di kota yang sama dan membentuk siklus). Sebagai ilustrasi, bentuk *subtour* dapat dilihat pada **Gambar 1.**



Gambar 1. Contoh *Subtour.*

Dengan demikian, panjang total rute dapat dinyatakan dalam bentuk linear yang lebih sederhana yaitu:

$$min \sum_{i} \sum_{j>i} d_{i,j} x_{i,j} \quad (1)$$

TSP yang menggunakan pendekatan *Integer Linear Programming* (ILP) menjadi kompleks karena adanya berbagai kendala. Pada kasus simetris, setiap titik i harus memiliki tepat dua variabel x yang bernilai 1 dalam solusi yang layak. Satu variabel x menghubungkan titik i ke kota sebelumnya dalam rute, dan variabel x lainnya menghubungkan titik i ke kota berikutnya. Hal ini memastikan setiap kota dikunjungi sekali dan rute membentuk siklus tertutup. Model matematis yang menggambarkan kendala yang tersebut adalah sebagai berikut:

$$min \sum_{j<i} x_{j,i} + \sum_{j>i} x_{i,j} = 2, \text{untuk semua } i \quad (2)$$

Sementara itu, TSP asimetris adalah jarak antara dua kota berbeda tergantung pada arah perjalanan. Artinya, jarak kota A ke kota B yang dinyatakan sebagai $d_{A,B}$ tidak selalu sama dengan jarak dari kota B ke kota A, yaitu $d_{B,A}$ sehingga jarak perjalanan dapat bervariasi. Formulasi matematis dari TSP asimetris didefenisikan sebagai berikut:

$$x_{ij} = \begin{cases} 1 & \text{jika dari kota } i \text{ ke kota } j \\ 0 & \text{jika tidak} \end{cases}$$

Model TSP asimetris memiliki dua batasan tambahan yang menggantikan batasan yang berlaku pada model simetris, yaitu:

1. Untuk setiap kota i harus ada satu rute yang masuk *(enter)*

$$\sum_{j} x_{j,i} = 1 \text{ untuk semua } i \quad (3)$$

2. Untuk setiap kota i harus ada satu rute yang keluar *(leave)*

$$\sum_{j} x_{i,j} = 1 \text{ untuk semua } i \quad (4)$$

Selain itu, setiap rute harus melewati setiap *subset* S dari titik-titik yang ada, baik untuk masuk maupun keluar. Oleh karena itu, *subtour* dapat dihindari dengan memastikan bahwa *tour* meninggalkan setiap *subset* S setidaknya satu kali.

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \text{ untuk semua himpunan bagian titik yang tepat } S \quad (5)$$

Tabel 1. Contoh TSP Simetris.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	0	3	4	2	7
	B	3	0	4	6	3
	C	4	4	0	5	8
	D	2	6	5	0	6
	E	7	3	8	6	0

Tabel 2. Contoh TSP Asimetris.

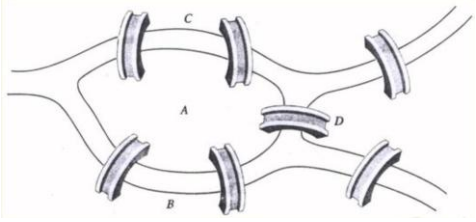
		Ke Kota				
		A	B	C	D	E
Dari Kota	A	0	3	4	2	7
	B	3	0	3	5	1
	C	2	4	0	6	4
	D	5	6	5	0	3
	E	4	3	2	6	0

di mana:

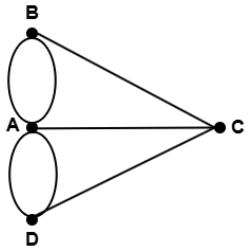
- $d_{i,j}$: Jarak dari kota i ke kota j
- i : Kota awal
- j : Kota tujuan

1.6.2. Graf

Teori graf muncul pada tahun 1736 saat seorang tokoh matematikawan dari Swiss bernama Leonhard Euler memecahkan masalah Jembatan Königsberg yang melibatkan penentuan terkait perjalanan melewati semua jembatan dalam satu perjalanan tanpa kembali ke jembatan yang sama. Dari masalah tersebut, ia memodelkan masalahnya ke dalam bentuk graf (Munir, 2010).



(a)



(b)

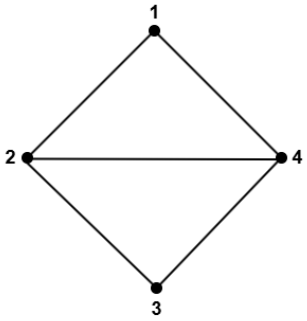
Gambar 2. Jembatan Königsberg dan Contoh Graf.

Defenisi dari graf G adalah pasangan himpunan (V, E) yang dinotasikan dengan $G = (V, E)$ yang dalam hal ini V adalah himpunan titik tidak kosong dari simpul – simpul *(vertices* atau *node)* dan E adalah himpunan sisi *(edges* atau busur*)* yang menghubungkan sepasang simpul. Sebuah graf dapat saja tidak memiliki satu pun sisi, namum setidaknya harus terdapat satu simpul. Penamaan simpul dalam graf dapat diberi label dengan huruf seperti a, b, c, …, z atau dengan bilangan asli 1, 2, 3, …, atau kombinasi dari keduanya. Kemudian untuk sisi yang menghubungkan simpul u dengan simpul v dinyatakan dengan pasangan (u, v) atau dapat dituliskan dengan $e_1, e_2, ...$ Maka, jika e sisi yang menghubungkan simpul u atau v, dapat ditulis sebagai berikut:

$$e = (u, v) \tag{6}$$

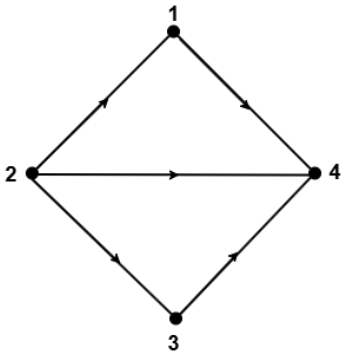
Misalkan pada **Gambar 2b**, yang disebut dengan himpunan *vertex* atau simpul yaitu $v = \{A, B, C, D\}$ dan himpunan *edges* atau sisi $e = \{(B, C), (A, C), (C, D), (A, B), (A, D)\}$. Graf mempunyai orientasi arah pada sisi yang secara umum dibedakan atas 2 jenis yaitu:

a. Graf tidak berarah *(undirected graph)*
   Pada graf tidak berarah yang dapat dilihat pada **Gambar 3**, sisi – sisi yang menghubungkan simpul – simpul tidak memiliki arah. Artinya, urutan pasangan simpul yang dihubungkan oleh sisi tidak dipertimbangkan. Contohnya, jika ada sisi yang menghubungkan simpul u dan v, sisi tersebut tidak membedakan arah mana yang datang lebih dulu. Misalnya $(u, v) = (1,2)$ sama saja dengan $(v, u) = (2,1)$.



Gambar 3. Graf Tidak Berarah.

b. Graf berarah *(directed graph* atau *digraph)*
Pada graf berarah, sisi – sisi yang menghubungkan simpul diberi arah, yang disebut dengan busur atau *arc* yang berarti arah dari sisi tersebut penting dan berbeda dari satu arah dengan arah sebaliknya. Contohnya, busur yang menghubungkan simpul u dan v berbeda dengan busur yang menghubungkan simpul v ke u. Misalnya $(u, v) = (1,4)$ tidak sama dengan $(v, u) = (4,1)$.



Gambar 4. Graf Berarah.

TSP adalah masalah klasik dalam teori graf dan optimasi kombinatorial. Masalah ini bertujuan untuk menemukan rute terpendek dalam mengunjungi serangkaian kota (simpul) tepat satu kali dan kembali ke titik awal. TSP dapat dimodelkan menggunakan graf berbobot tidak berarah, di mana:

- Simpul *(vertices)* mewakili kota
- Sisi *(edges)* merepresentasikan jalur antar kota, dengan bobot yang menunjukkan jarak atau biaya perjalanan di antara keduanya.

Pada graf tidak berarah, hubungan antar kota bersifat simetris. Sebaliknya, pada graf berarah, arah perjalanan menjadi penting, karena biaya atau jarak dari kota A ke B mungkin berbeda dari kota B ke A, tergantung pada jenis graf yang digunakan pada TSP (Brucato, 2010).

TSP dimodelkan sebagai pencarian siklus Hamiltonian dalam graf lengkap. Siklus Hamiltonian *(Hamiltonian Cycle)* merupakan siklus atau jalur yang mengunjungi setiap simpul tepat sekali dan kembali ke simpul awal (Lam & Newman, 2005). Graf yang memiliki lintasan Hamilton dapat dilihat pada **Gambar 4** yang lintasannya adalah 1 – 2 – 3 – 4 – 1.

1.6.3. Metode Eksak

Metode eksak pada TSP berfokus pada pencarian solusi optimal dengan menggunakan algoritma yang menjamin hasil terbaik. Dua metode eksak yang umum digunakan yaitu *Branch and Bound* dan *Cutting Plane* (Winston, 2004).

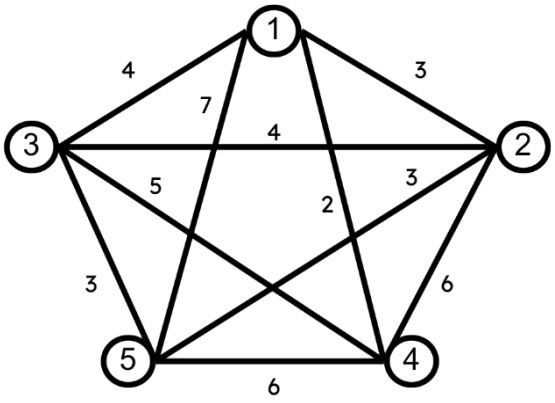
1.6.3.1 *Branch and Bound*

Branch and Bound adalah teknik yang mengembangkan pohon solusi dengan cara mengeksplorasi semua kemungkinan rute secara sistematis. Pada

setiap simpul pohon, algoritma ini menghitung batas bawah dari biaya rute yang mungkin dan membuang cabang – cabang yang tidak menjanjikan jika biaya tersebut lebih tinggi dari solusi terbaik yang ditemukan sejauh ini (Rajarajeswari & Maheswari, 2020). Pendekatan ini memungkinkan untuk masalah kecil tetapi dapat mengalami peningkatan waktu pemrosesan yang signifikan ketika diterapkan pada masalah yang lebih besar bahkan tidak praktis ketika digunakan (Winston, 2004). Adapun langkah – langkah dari algoritma ini antara lain:

a. Inisialisasi: Menentukan matriks jarak antar kota.
b. Lakukan reduksi matriks (pengurangan pada matriks dengan mengurangi nilai minimum dari baris dan kolom).
c. Pencarian simpul: Gunakan teknik *Breadth First Search* (BFS) untuk mengeksplorasi simpul – simpul dalam pohon pencarian.
d. Hitung batas bawah untuk setiap simpul dan bandingkan dengan solusi optimal yang diperoleh saat ini.
e. Hapus simpul – simpul yang tidak mungkin menghasilkan solusi lebih baik dari solusi optimal saat ini.
f. Setelah semua simpul dieksplorasi, hasil akhir adalah rute terpendek yang ditemukan.

Contoh**:** Misalkan jarak antar 5 kota di representasikan dalam bentuk graf tidak berarah. Tentukan rute optimal dengan titik awal rute dimulai dari kota 1, sebagaimana disajikan sebagai berikut.



Gambar 5. Graf Jarak Lima Kota.

Langkah 1: Representasikan graf pada **Gambar 5** tersebut dalam bentuk matriks. Misalkan matriks tersebut disimbolkan dengan D_{ij}, di mana D_{ij} menunjukkan jarak dari kota i ke kota j. Elemen pada diagonal utama (D_{ii}) diberi tanda "–", alih – alih nol, untuk menunjukkan bahwa perjalanan dari suatu kota ke dirinya sendiri tidak valid atau tidak relevan.

Tabel 3. Jarak Antar Kota (km).

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	–	3	4	2	7
	B	3	–	4	6	3
	C	4	4	–	5	8
	D	2	6	5	–	6
	E	7	3	8	6	–

Langkah 2: Lakukan reduksi matriks D_{ij} dengan mengurangi elemen terkecil dari setiap baris (r_i) dan kolom (c_i). Jika dua kota tidak terhubung, isi elemen matriks tersebut dengan simbol tak hingga (∞).

Tabel 4. Reduksi Baris (r_i).

		Ke Kota					r_i
		A	B	C	D	E	
Dari Kota	A	∞	3	4	2	7	2
	B	3	∞	4	6	3	3
	C	4	4	∞	5	8	4
	D	2	6	5	∞	6	2
	E	7	3	8	6	∞	3
$\sum_{i=1}^{n} r_i$							14

Kemudian setiap baris pada matriks (D_{ij}) pada **Tabel 4** dikurangi dengan r_i (nilai elemen minimum dari setiap kolom), maka diperoleh hasil yang disajikan pada **Tabel 5.**

Tabel 5. Hasil Reduksi Baris (r_i).

		Ke Kota					
		A	B	C	D	E	
Dari Kota	A	∞	1	2	0	5	$\sum_{j=1}^{n} c_j$
	B	0	∞	1	3	0	
	C	0	0	∞	1	4	
	D	0	4	3	∞	4	
	5	4	0	5	3	∞	
c_j		0	0	1	0	0	1

Selanjutnya, setiap kolom matriks (D_{ij}) pada **Tabel 5** dikurangkan dengan c_j (nilai elemen minimum dari setiap kolom), maka diperoleh hasil yang disajikan pada **Tabel 6**.

Tabel 6. Hasil Reduksi Kolom (c_j).

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	∞	1	1	0	5
	B	0	∞	0	3	0
	C	0	0	∞	1	4
	D	0	4	2	∞	4
	E	4	0	4	3	∞

Langkah 3: Menghitung batas bawah awal *(initial lower bound)* bertujuan untuk menghindari perhitungan tidak perlu pada tahap selanjutnya dengan menggunakan formula berikut:

$$LB\ node\ A = \sum_{i=1}^{n} r_i + \sum_{j=1}^{n} c_j = 14 + 1 = 15$$

Hasil dari batas bawah ini juga merupakan nilai pada besar jarak pada node A atau kota A.

Langkah 4: Selanjutnya, lakukan ekspansi node dengan memilih kota yang belum dikunjungi. Kota yang belum dikunjungi yaitu kota B, C, D dan E. Misalnya dimulai dari kota B, maka pada node A → B, bisa dilihat pada hasil reduksi matriks pada langkah 3, nilai $v(A, B) = 1$. Selanjutnya,

nilai pada baris 1 dan kolom 2 diganti menjadi ∞ begitu pula $v(B,A) = v(2,1) = \infty$, sehingga matriks (D_{ij}) yang diperoleh adalah

Tabel 7. Ekspansi Node A → B.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	∞	∞	∞	∞	∞
	B	∞	∞	0	3	0
	C	0	∞	∞	1	4
	D	0	∞	2	∞	4
	E	4	∞	4	3	∞

Selanjutnya, dengan menggunakan cara yang sama yaitu mereduksi setiap baris dan kolom, diperoleh matriks (D_{ij}) sebagai berikut.

Tabel 8. Reduksi Baris dan Kolom Node A → B.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	∞	∞	∞	∞	∞
	B	∞	∞	0	2	0
	C	0	∞	∞	0	0
	D	0	∞	2	∞	4
	E	1	∞	1	0	∞

Diperoleh nilai reduksi matriks dari node B sebesar 2. Dengan demikian, jarak untuk node B adalah

$$biaya\ node\ B = biaya\ node\ A + v(A,B) + reduksi\ matriks\ D_{AB}$$
$$= 15 + 1 + 3$$
$$= 19$$

Dengan cara yang sama, akan diperoleh:

- Biaya node A → C = 17 (node 3)
- **Biaya node A → D** = 15 (node 4)
- Biaya node A → E = 21 (node 5)

Terlihat bahwa jarak antara node A → D = 15 yang merupakan biaya terkecil diantara node B, C, dan E. Oleh karena itu, dari node 4 dibentuk percabangan baru dengan node yang belum dikunjungi. Selanjutnya akan dilakukan reduksi matriks untuk rute

A → D → B dengan membuat baris A dan D serta kolom B menjadi ∞, sehingga $v(D,A) = v(B,A) = \infty$. Dengan demikian, reduksi matriks yang diperoleh adalah

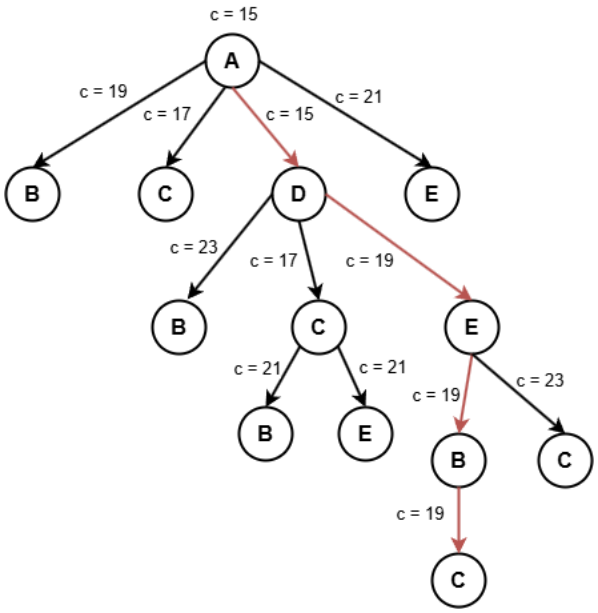
Tabel 9. Reduksi Baris dan Kolom Node A → D → B.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	∞	∞	∞	∞	∞
	B	∞	∞	0	∞	0
	C	0	∞	∞	∞	4
	D	∞	∞	∞	∞	∞
	E	4	∞	4	∞	∞

Diperoleh nilai reduksi matriks untuk node A sebesar 4. Dengan demikian, jarak yang diperoleh untuk node B pada percabangan dari A → D adalah

$$\begin{aligned} biaya\ node\ B &= biaya\ node\ D + v(D,B) + reduksi\ matriks\ D_{DB} \\ &= 15 + 4 + 4 \\ &= 23 \end{aligned}$$

Lakukan pencarian dengan cara yang sama menggunakan reduksi matriks untuk setiap cabang sehingga diperoleh nilai batas bawah untuk setiap node. Dengan demikian, solusi optimal yang diperoleh adalah A → D → E → B → C → A dengan jarak total sebesar 19 km. Representasi percabangan dari metode *Branch and Bound* dapat dilihat pada **Gambar 6**.



Gambar 6. Hasil Percabangan *Branch and Bound.*

Jika diimplementasikan menggunakan MATLAB diperoleh hasil sebesar 19 km dan waktu komputasi sebesar $1{,}681 \times 10^{-1}$ detik.

```
Solusi terbaik dengan biaya minimum: 19.00
Rute terbaik: 1  4  5  2  3  1
Waktu komputasi: 0.1681 detik
fx >>
```

Gambar 7. Hasil Implementasi *Branch and Bound* pada MATLAB.

1.6.3.2 *Cutting plane*

Cutting plane adalah metode lain yang efektif untuk menyelesaikan TSP. Metode ini dimulai dengan menyelesaikan relaksasi linear dari masalah dan kemudian menambahkan batasan *(cut)* untuk menghilangkan solusi tidak valid dengan tetap mendapatkan solusi optimal integer (Applegate et al., 2006). Adapun langkah – langkah dari algoritma ini antara lain:

a. Formulasi masalah TSP sebagai model program linear dengan mendefenisikan variabel keputusan yang menunjukkan apakah rute tertentu diambil atau tidak (0 atau 1). Misalkan x_{ij} adalah variabel biner yang menunjukkan apakah rute dari kota i ke kota j diambil.
b. Modelkan TSP dalam bentuk fungsi tujuan yang dapat dinyatakan sebagai berikut:

$$min \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \qquad (7)$$

di mana d_{ij} adalah jarak antara kota i dan kota j.
c. Selesaikan LP relaksasi menggunakan metode optimasi (misalnya menggunakan metode simpleks).
d. Identifikasi *subtour* dengan memeriksa apakah solusi yang diperoleh bersifat fraksional. Jika ya, tambahkan batasan untuk menghindari *subtour* menggunakan *Subtour Elimination Constraints* (SEC) dengan persamaan berikut:

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \quad \forall S \subset N, \quad |S| \geq 2 \qquad (8)$$

di mana:
   - S adalah himpunan kota yang membentuk *subtour.*
   - N adalah himpunan semua kota yang terlibat dalam masalah TSP (misalnya, $N = \{1,2, \dots \dots n\}$.
   - $|S|$ adalah jumlah dalam kota yang terdapat dalam *subset* S, yaitu ukuran dari *subset* tersebut (jumlah kota dalam *subtour*).
   - x_{ij} adalah variabel biner yang menunjukkan apakah ada rute atau perjalanan langsung dari kota i ke kota j.

e. Setelah semua *cut* ditambahkan dan tidak ada lagi yang membentuk *subtour* maka diperoleh solusi akhir.

Contoh**:** Dari data jarak sebelumnya yang terdapat pada **Tabel 3**, akan dicari rute optimal menggunakan metode *Cutting Plane.*

Langkah 1: Dari matriks diatas, ubah kedalam model *Linear Programming* (LP) dan akan dihitung LP *Relaxation*-nya dengan mengabaikan kendala integer.

$$Minimize\ 3x_{12} + 4x_{13} + 2x_{14} + 7x_{15} + 4x_{23} + 6x_{24} + 3x_{25} + 5x_{34} + 8x_{35} + 6x_{45}$$

Dengan kendala,

$$x_{12} + x_{13} + x_{14} + x_{15} = 2$$

$$x_{12} + x_{23} + x_{24} + x_{25} = 2$$

$$x_{13} + x_{23} + x_{34} + x_{35} = 2$$

$$x_{14} + x_{24} + x_{34} + x_{45} = 2$$

$$x_{15} + x_{25} + x_{35} + x_{45} = 2$$

Langkah 2: Dengan menggunakan bantuan *software* LINDO, diperoleh LP *Relaxation*-nya. Terlihat dari variabel $x_{13} = x_{14} = x_{34} = 1$ dan $x_{25} = x_{52} = 1$ jika dibentuk dalam bentuk *tour* yaitu $1 \rightarrow 3 \rightarrow 4 \rightarrow 1$ dan $2 \rightarrow 5 \rightarrow 2$. Terlihat masih membentuk *subtour* dan tidak membentuk rute secara lengkap. Nilai fungsi objektif diperoleh sebesar $z = 17$.

```
LP OPTIMUM FOUND AT STEP      4

        OBJECTIVE FUNCTION VALUE

        1)       17.00000

 VARIABLE         VALUE          REDUCED COST
      X12          0.000000          2.500000
      X13          1.000000          0.000000
      X14          1.000000          0.000000
      X15          0.000000          3.500000
      X23          0.000000          0.500000
      X24          0.000000          4.500000
      X25          2.000000          0.000000
      X34          1.000000          0.000000
      X35          0.000000          1.500000
      X45          0.000000          1.500000


      ROW    SLACK OR SURPLUS      DUAL PRICES
       2)          0.000000         -0.500000
       3)          0.000000          0.000000
       4)          0.000000         -3.500000
       5)          0.000000         -1.500000
       6)          0.000000         -3.000000

NO. ITERATIONS=        4
```

Gambar 8. Hasil LP *Relaxation* I.

Langkah 3: Tambahkan SEC untuk menghapus *subtour.* Penambahan SEC berasal dari *subtour* sebelumnya yaitu $1 \rightarrow 3 \rightarrow 4 \rightarrow 1$ dan $2 \rightarrow 5 \rightarrow 2$. Sehingga fungsi kendala sekarang yaitu:

$$x_{12} + x_{13} + x_{14} + x_{15} = 2$$

$$x_{12} + x_{23} + x_{24} + x_{25} = 2$$

$$x_{13} + x_{23} + x_{34} + x_{35} = 2$$

$$x_{14} + x_{24} + x_{34} + x_{45} = 2$$

$$x_{15} + x_{25} + x_{35} + x_{45} = 2$$

$$\boldsymbol{x_{13} + x_{34} + x_{14} \leq 2}$$

$$\boldsymbol{x_{25} \leq 1}$$

dengan bantuan menggunakan *software* LINDO, diperoleh hasil LP-*Relaxation* dengan kendala yang baru yaitu sebagai berikut.

```
LP OPTIMUM FOUND AT STEP      8

        OBJECTIVE FUNCTION VALUE

        1)       19.00000

 VARIABLE         VALUE          REDUCED COST
      X12          0.000000          0.000000
      X13          1.000000          0.000000
      X14          1.000000          0.000000
      X15          0.000000          0.000000
      X23          1.000000          0.000000
      X24          0.000000          4.000000
      X25          1.000000          0.000000
      X34          0.000000          2.000000
      X35          0.000000          0.000000
      X45          1.000000          0.000000


      ROW    SLACK OR SURPLUS      DUAL PRICES
        2)          0.000000         -1.500000
        3)          0.000000         -1.500000
        4)          0.000000         -2.500000
        5)          0.000000         -0.500000
        6)          0.000000         -5.500000
        7)          0.000000          0.000000
        8)          0.000000          4.000000

NO. ITERATIONS=        8
```

Gambar 9. Hasil LP *Relaxation* II.

Langkah 4: Cek apakah rute tersebut sudah tidak mengandung *subtour*. Jika tidak, maka rute tersebut telah membentuk rute lengkap dan optimal. Terlihat variabel yang diperoleh adalah $x_{13} = x_{14} = x_{23} = x_{25} = x_{45} = 1$. Sehingga, hasil *tour* didapatkan $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 1$ atau $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$. Dengan fungsi objektif (total jarak) sebesar 19 km.

1.6.4. Metode Heuristik

Metode Heuristik merupakan pendekatan untuk mencari solusi dalam masalah optimasi secara cepat tanpa memerlukan pembuktian analitik yang rumit. Metode ini memungkinkan penyelesaian masalah dengan menggunakan langkah – langkah yang sistematis, biasanya menghasilkan solusi yang mendekati optimal. Karena tidak berbasis teori optimasi mendalam, pendekatan ini sering kali berfokus pada efisiensi komputasi daripada keakuratan hasil.

Kelebihan dari heuristik adalah kecepatannya dalam memberikan solusi yang cukup baik dengan walaupun solusinya terkadang kurang optimal dibandingkan dengan metode eksak. Heuristik sering menjadi pilihan ketika fokus utamanya adalah waktu, karena metode ini tidak membutuhkan perhitungan yang kompleks, sehingga menghasilkan solusi dalam waktu singkat (Winston, 2004). Sedangkan, kekurangan dari heuristik adalah sifatnya yang tidak universal, di mana metode ini dirancang khusus untuk jenis masalah tertentu dan tidak bisa langsung diterapkan pada masalah lain. Karena pendekatannya yang spesifik, heuristik lebih sering digunakan pada kasus – kasus metode analitik atau eksak yang kurang efisien atau sulit diterapkan. Contoh heuristik yang umum digunakan adalah *Nearest Neighbor Heuristic* dan *Cheapest Insertion Heuristic* (Santosa & Ai, 2017).

1.6.4.1 *Nearest Neighbor Heuristic* (NNH)

Nearest Neighbor Heuristic adalah algoritma sederhana yang memulai dari sebuah kota terdekat dan secara iteratif akan mengunjungi kota terdekat berikutnya hingga semua kota dikunjungi (Winston, 2004). Urutan dalam algoritma ini adalah sebagai berikut:

a. Memilih kota pertama dan mengunjungi kota berikutnya dengan jarak yang terdekat.
b. Selanjutnya, mengunjungi kota terdekat berikutnya yang belum dikunjungi dengan jarak yang paling dekat dengan kota yang terakhir dikunjungi.
c. Ulangi langah (b) sampai seluruh kota telah terlewati, tidak membentuk *subtour* dan membentuk satu rute secarah lengkap.

Meskipun mudah untuk diimplementasikan, hasil yang diperoleh terkadang terjebak pada solusi lokal yang kurang optimal di mana hanya memilih berdasarkan jarak terdekat tanpa mempertimbangkan jarak pada keseluruhan rute. Contoh: Dari data jarak sebelumnya yang terdapat pada **Tabel 3**, akan dicari rute optimal menggunakan metode *Nearest Neighbor Heuristic.*

Langkah 1: Jika dimulai dari kota A, maka cek baris pada kota A yang memiliki nilai terkecil, kemudian tandai kota tersebut.

Tabel 10. Memilih kota terdekat.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	–	3	4	2	7
	B	3	–	4	6	3
	C	4	4	–	5	8
	D	2	6	5	–	6
	E	7	3	8	6	–

Langkah 2: Terlihat kota D memiliki nilai terkecil sehingga A → D, kemudian temukan kota selanjutnya yang belum dikunjungi dari kota D dengan mengabaikan kota A. Sehingga diperoleh A → D → C.

Tabel 11. Memilih kota terdekat selanjutnya.

		Ke Kota				
		A	B	C	D	E
Dari Kota	A	–	3	4	2	7
	B	3	–	4	6	3
	C	4	4	–	5	8
	D	2	6	5	–	6
	E	7	3	8	6	–

Langkah 3: Lakukan cara yang sama sampai semua kota telah dikunjungi dan kembali pada kota awal untuk menyelesaikan *tour*. Sehingga diperoleh *tour* secara lengkap yaitu $\mathrm{A} \rightarrow \mathrm{D} \rightarrow \mathrm{C} \rightarrow \mathrm{B} \rightarrow \mathrm{E} \rightarrow \mathrm{A}$ dengan total jarak $2 + 5 + 4 + 3 + 7 = 21$ km. Jika diimplementasikan menggunakan MATLAB dihasilkan jarak sebesar 21 km dan waktu komputasi sebesar $6{,}9 \times 10^{-3}$ detik.

```
bestTour =

     1     4     3     2     5     1


totalDistance =

    21


computationTime =

    0.0069
```

Gambar 10. Implementasi Hasil *Nearest Neighbor* pada MATLAB.

1.6.4.2 *Cheapest Insertion Heurisic* (CIH)

Metode *Cheapest Insertion Heuristic* adalah menyisipkan kota yang belum dilewati dengan tambahan jarak minimum sehingga seluruh kota terlewati untuk mendapatkan solusi optimal. Sebuah *tour* dimulai dari simpul awal 1 menuju ke seluruh simpul (2, 3,…n) dan akan kembali ke simpul awal 1 dengan catatan tidak mengunjungi lebih dari satu kali dan memperhitungkan tambahan jarak minimum ketika satu simpul disisipkan ke dalam sebagian *tour* (Meliantari et al., 2018). Urutan dalam algoritma ini adalah sebagai berikut:

a. Memulai dengan kota pertama terdekat dan dihubungkan dengan sebuah kota terakhir.
b. Hubungkan kedua kota tersebut dengan membentuk *subtour* misalnya $(1,4) \rightarrow (4,3) \rightarrow (3,1)$.
c. Subtitusikan salah satu arah *(arc)* dari dua kota dengan kombinasi dua *arcs* yaitu (i,j) dengan (i,k) dan (k,j) dengan k adalah kota sisipan dengan tambahan jarak terkecil yang dihitung menggunakan persamaan:

$$c_{jk} = c_{ik} + c_{kj} - c_{ij} \quad (9)$$

di mana:

- c_{jk} : jarak dari kota j ke kota k
- c_{ik} : jarak dari kota i ke kota k
- c_{kj} : jarak dari kota k ke kota j
- c_{ij} : jarak dari kota i ke kota j

d. Ulangi langkah (c) sampai seluruh kota telah terlewati dan tidak membentuk *subtour* dan membentuk satu rute secarah lengkap.

Metode ini sering kali menghasilkan rute yang lebih efisien dibandingkan *Nearest Neighbor* karena mempertimbangkan dampak keseluruhan dari setiap penambahan kota (Winston, 2004).

Contoh**:** Dari data jarak sebelumnya yang terdapat pada **Tabel 3**, akan dicari rute optimal menggunakan metode *Cheapest Insertion Heuristic*.

Langkah 1: Inisiasi *tour* dimulai dengan memilih pasangan kota terdekar. Dari matriks tersebut, jika titik awal kota adalah kota A, maka kota terdekat yang dipilih adalah kota D. Kota D kemudian dipasangkan dengan kota awal, membentuk sub*tour* terdekat yaitu (A, D) → (D, A).

Tabel 12. Pilih kota pasangan terdekat.

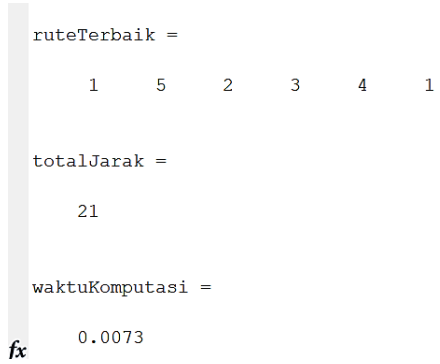
		Ke Kota				
		A	B	C	D	E
Dari Kota	A	–	3	4	2	7
	B	3	–	4	6	3
	C	4	4	–	5	8
	D	2	6	5	–	6
	E	7	3	8	6	–

Langkah 2: Buatlah tabel untuk mencatat titik-titik kota yang dapat disisipkan ke dalam *subtour*. Kemudian, periksa kota – kota yang akan disisipkan. Kota yang belum dikunjungi adalah Kota B, C dan E. Misalkan k adalah titik kota yang belum dikunjungi. Contohnya pada **Tabel 13**.

Tabel 13. Daftar Kandidat Kota untuk Penyisipan.

Arc	*Arc* yang akan dimasukkan pada *subtour*	Sisipan = $C_{ik} + C_{kj} - C_{ij}$
(A, D)	(A, B) – (B, D)	$C_{AB} + C_{BD} - C_{AD} = 7$
(A, D)	(A, C) – (C, D)	$C_{AC} + C_{CD} - C_{AD} = 7$
(A, D)	(A, E) – (E, D)	$C_{AE} + C_{ED} - C_{AD} = 11$
(D, A)	(D, B) – (B, A)	$C_{DB} + C_{BA} - C_{DA} = 7$
(D, A)	(D, C) – (C, A)	$C_{DC} + C_{CA} - C_{DA} = 7$
(D, A)	(D, E) – (E, A)	$C_{DE} + C_{EA} - C_{DA} = 11$

Dari **Tabel 13** di atas, diperoleh sisipan terkecil sementara sebesar 7. Jika terdapat dua rute dengan jarak yang sama, pilih salah satu rute tersebut. Misalnya *arc* (A, D) dengan *arc* yang akan disisipkan pada *subtour* adalah (A, B) – (B, D). Dengan demikian, *arc terbaru* yang terbentuk menjadi (A, B) – (B, D) – (D, A). Lanjutkan dengan cara yang hingga semua kota telah dikunjungi dan kembali ke kota awal. Rute akhir yang diperoleh (A, E) – (E, B) – (B, C) – (C, D) – (D, A) dengan total jarak sebesar 21 km. Implementasi pada MATLAB menghasilkan rute sebagai berikut dengan waktu komputasi sebesar $7{,}3 \times 10^{-3}$ detik.



Gambar 11. Implementasi Hasil *Cheapest Insertion* pada MATLAB.

1.6.5. Metode Metaheuristik

Metaheuristik berkembang sebagai lanjutan dari heuristik untuk menjawab terkait fleksibilitas dari metode ini dalam menyelesikan masalah optimasi. Berbeda dengan heuristik, metaheuristik tidak dirancang spesifik untuk satu jenis masalah. Metode ini menggunakan prinsip iterasi dan pendekatan stokastik (berbasis acak) yang terinspirasi dari fenomena alam seperti evolusi dan perilaku hewan.

Metaheuristik memungkinkan pencarian solusi optimal atau mendekati optimal di ruang solusi yang luas dengan mempertimbangkan kesetimbangan antara eksplorasi (memaksimalkan solusi yang sudah baik). Teknik ini dikenal mampu

menghindari jebakan suboptimal *(local optima)* dan mencari solusi di ruang yang lebih luas, meskipun tidak selalu menjamin solusi optimal. Contoh dari metode metaheuristik adalah *Genetic Algorithm* dan *Ant Colony Optimization*. Kelemahan dari metaheuristik antara lain adalah kompleksitas dalam pengaturan parameter dan efisiensi yang kurang untuk masalah berukuran kecil yang dapat diselesaikan oleh metode eksak (Santosa & Ai, 2017).

1.6.5.1 *Genetic Algorithm* (GA)

Genetic Algorithm meniru proses evolusi biologis dengan menghasilkan populasi awal dari rute acak lalu melalui proses seleksi, *crossover* (kawin silang), dan mutasi untuk menghasilkan generasi baru. Proses diulang sampai suatu kriteria terpenuhi dan hasil yang diperoleh akan mendekati. Metode ini mempertahankan variasi solusi dalam populasi, sehingga memungkinkan algoritma ini untuk terus berkembang menuju solusi yang lebih baik. Proses seleksi memastikan bahwa solusi-solusi terbaik terus maju ke tahap berikutnya, sementara *crossover* dan mutasi menciptakan solusi-solusi baru yang lebih bervariasi. Pendekatan ini sangat cocok untuk masalah optimasi yang membutuhkan pencarian solusi dalam ruang yang besar dan kompleks (Santosa & Ai, 2017).

Contoh**:** Dari data jarak sebelumnya yang terdapat pada **Tabel 3**, akan dicari rute optimal menggunakan metode *Genetic Algorithm* dengan bantuan *software* MATLAB. Misalkan parameter yang digunakan, yaitu jumlah kromosom $N = 20$, probabilitas *crossover* = 0,8, probabilitas mutasi = 0,01, *threshold fitness* = 0,0001 (nilai ambang untuk menghentikan proses iterasi) dan maksimal iterasi sebesar 10.

Langkah 1: Inisialisasi kromosom yang dibuat secara acak dalam membangkitkan populasi (himpunan solusi baru yang terdiri dari urutan gen). Kromosom mengandung informasi terkait solusi dari sekian banyaknya kemungkinan solusi masalah (Lukas et al., 2005). Sehingga diperoleh hasil yang dapat dilihat pada **Gambar 12**.

```
Populasi =

     1     2     3     5     4     1
     1     2     4     5     3     1
     1     5     2     4     3     1
     1     4     2     5     3     1
     1     4     3     5     2     1
     1     4     3     5     2     1
     1     3     4     2     5     1
     1     3     4     5     2     1
     1     3     2     4     5     1
     1     4     2     3     5     1
     1     4     2     5     3     1
     1     3     5     4     2     1
     1     4     2     5     3     1
     1     2     4     3     5     1
     1     4     2     3     5     1
     1     4     2     3     5     1
     1     3     2     5     4     1
     1     2     3     4     5     1
     1     4     5     3     2     1
     1     3     4     5     2     1
```

Gambar 12. Inisialisasi Populasi.

Langkah 2: Selanjutnya mengevaluasi kromosom dengan menghitung nilai *fitness* pada masing-masing kromosom. Nilai *fitness* adalah ukuran yang digunakan untuk mengevaluasi seberapa baik suatu solusi (kromosom) dalam menyelesaikan masalah. Semakin pendek total jarak, semakin tinggi nilai *fitness* dari kromosom tersebut. Pada contoh ini, diperoleh nilai *fitness* terbesar yaitu $5{,}26 \times 10^{-2}$ yang terdapat pada kromosom ke 17 yaitu $1-3-2-5-4-1$.

```
Fitness =

  Columns 1 through 7

    0.0435    0.0370    0.0400    0.0435    0.0476    0.0476    0.0400

  Columns 8 through 14

    0.0476    0.0370    0.0370    0.0435    0.0370    0.0435    0.0345

  Columns 15 through 20

    0.0370    0.0370    0.0526    0.0400    0.0435    0.0476
```

Gambar 13. Nilai *Fitness* Kromosom.

Setelah itu, dilakukan proses seleksi kromosom dengan menggunakan proses *elitism* yang dilakukan dengan menyalin kromosom terbaik. Dalam hal ini, jika jumlah populasi genap, akan disalin sebanyak 4 kromosom. Jika jumlah ganjil akan disalin sebanyak 3 kromosom (Santosa & Ai, 2017). Karena jumlah populasi genap, maka disalin sebanyak 4 kromosom sehingga terbentuk populasi sementara sebagai berikut:

```
Populasi_s =

     1     3     2     5     4     1
     1     3     2     5     4     1
     1     3     2     5     4     1
     1     3     2     5     4     1
     1     4     3     5     2     1
     1     4     3     5     2     1
     1     3     4     2     5     1
     1     3     4     5     2     1
     1     3     2     4     5     1
     1     4     2     3     5     1
     1     4     2     5     3     1
     1     3     5     4     2     1
     1     4     2     5     3     1
     1     2     4     3     5     1
     1     4     2     3     5     1
     1     4     2     3     5     1
     1     3     2     5     4     1
     1     2     3     4     5     1
     1     4     5     3     2     1
     1     3     4     5     2     1
```

Gambar 14. Populasi Sementara.

Langkah 3: Dilakukan perkawinan silang *(crossover)*. *Crossover* akan menghasilkan dua kromosom dengan cara menukar informasi dari dua induk kromosom. Namun sebelumnya, tentukan individu yang akan menjadi induk yang akan dikawinkan dari populasi menggunakan *Roulette Wheel Selection* (Santosa & Ai, 2017). Diperoleh *parents* yaitu kromosom bapak berada dikromosom ke – 7 dan kromosom ibu berada dikromosom ke –18. Kemudian proses perkawinan silang dilakukan secara acak dan akan menghasilkan kromosom anak. Diperoleh hasil anak ke – 1 dan anak ke – 2.

```
Bapak =

     7


Ibu =

    18

anak1 =

     1     5     3     4     2     1


anak2 =

     1     5     4     2     3     1
```

Gambar 15. Hasil Perkawinan silang *(crossover).*

Kemudian hasil *crossover* dimasukkan pada populasi sementara. Proses diulang sebanyak jumlah N.

```
Populasi_s =

     1     3     2     5     4     1
     1     3     2     5     4     1
     1     3     2     5     4     1
     1     3     2     5     4     1
     1     5     3     4     2     1
     1     5     4     2     3     1
     1     4     2     5     3     1
     1     3     4     5     2     1
     1     4     2     5     3     1
     1     3     4     5     2     1
     1     3     5     4     2     1
     1     3     4     5     2     1
     1     3     4     2     5     1
     1     4     3     5     2     1
     1     4     2     5     3     1
     1     3     5     4     2     1
     1     4     2     3     5     1
     1     3     2     4     5     1
     1     3     4     2     5     1
     1     2     4     5     3     1
```

Gambar 16. Populasi Sementara Setelah *Crossover.*

Langkah 4: Selanjutnya dilakukan proses mutasi. Proses mutasi yaitu melakukan modifikasi satu atau lebih gen dalam individu sehingga hasil dari mutasi tersebut akan menghasilkan individu baru. Dari hasil mutasi ini akan membuat populasi menjadi bervariasi dengan menukar gen yang hilang dari populasi selama proses seleksi serta menyimpan gen yang tidak tersimpan pada populasi awal (Priandani & Mahmudy, 2015). Hasil tersebut dimasukkan kembali ke populasi sementara menggantikan populasi sebelumnya.

```
Mutcrom =

     5     3     4     2
```

Gambar 17. Hasil Mutasi.

Langkah ini akan kembali diulang sampai seluruh kriteria tercapai. Setelah kriteria terpenuhi, diperoleh rute terbaik yaitu $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ sebesar 19 km dan waktu komputasinya $2{,}031 \times 10^{-1}$ detik.

```
t =

    0.2031


RuteTerbaik =

     1     4     5     2     3     1


jarak =

    19
```

Gambar 18. Hasil Rute Terbaik *Genetic Algorithm*.

1.6.5.2 *Ant Colony Optimization* (ACO)

Ant Colony Optimization pertama kali diperkenalkan oleh Mayson dan Manderick kemudian dikembangkan oleh Marco Dorigo (Karjono et al., 2016). Metode ini terinspirasi dari perilaku koloni semut dalam mencari jalan terpendek menuju sumber makanan. Dalam metode ini, semut buatan secara kolektif mengeksplorasi berbagai kemungkinan rute dan meninggalkan jejak *pheromone* sebagai indikator kualitas rute yang dilalui.

Pheromone dalam konteks optimasi berfungsi sebagai penanda yang meniru zat kimia yang dilepaskan oleh semut di alam untuk berkomunikasi dan mengidentifikasi rute terbaik menuju sumber makanan. Selain itu, *pheromone* mengalami evaporasi (penguapan) untuk mencegah terjebak dalam solusi suboptimal *(local optimum)*, memungkinkan algoritma tetap eksploratif dan mencari solusi yang lebih baik secara global (Wei, 2014). Pembaruan *pheromone* dapat dihitung dengan rumus sebagai berikut:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij} \quad (10)$$

di mana:

- $\Delta\tau_{ij}$: *pheromone* tambahan yang ditambahkan oleh semut
- $\tau_{ij}(t)$ adalah nilai *pheromone* pada jalur antara kota i dan j pada iterasi t

Rute dengan jejak *pheromone* yang lebih kuat cenderung lebih sering dipilih oleh semut – semut berikutnya, sehingga secara bertahap memperkuat jalur rute terbaik. *Ant Colony Optimization* sangat efektif untuk optimasi kombinatorial seperti TSP karena karakter adaptifnya memungkinkan algoritma secara bertahap menemukan solusi optimal atau mendekati optimal dengan memperkuat jalur terbaik dari waktu ke waktu. Proses ini mendukung eksplorasi yang efektif terhadap berbagai solusi, serta kemampuan untuk terus belajar dan memperbaiki solusi secara iteratif (Santosa & Ai, 2017).

Contoh**:** Berdasarkan data jarak yang terdapat pada **Tabel 3**, akan dicari rute optimal menggunakan metode *Ant Colony Optimization* dengan bantuan *software* MATLAB. Misalkan $\rho = 0{,}5$ (koefisien penguapan/evaporasi), $\alpha = 1$ (pangkat *visibility*) dan $\beta = 2$ (pangkat *pheromone*) dan maksimal iterasi 10.

Langkah 1: Misalkan jumlah koloni semut $m = 10$. Setiap semut memulai perjalanannya dari kota awal dan mengunjungi kota – kota lain berdasarkan probabilitas yang dihitung dari *pheromone* dan jarak antar kota. Kumpulan semut ini berfungsi mengeksplorasi berbagai rute yang mungkin. Selanjutnya, *visibility* (h) antar kota dihitung sebagai invers jarak antar kota $\frac{1}{d_{ij}}$. *Visibility* adalah salah satu parameter yang mempengaruhi keputusan semut dalam menentukan jalur selama pencarian solusi (Karjono et al., 2016). Kemudian akan diberikan nilai *pheromone* awal $\tau_0 = 1$. Diperoleh sebagai berikut.

```
h =

         0    0.3333    0.2500    0.5000    0.1429
    0.3333         0    0.2500    0.1667    0.3333
    0.2500    0.2500         0    0.2000    0.1250
    0.5000    0.1667    0.2000         0    0.1667
    0.1429    0.3333    0.1250    0.1667         0

tho =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Gambar 19. Hasil *Visibility* (h) dan *Pheromone* Awal (τ_0).

Langkah 2: Menetukan nilai peluang dari satu kota ke kota lainnya. Dalam hal ini, kota 1 sebagai kota pemberangkatan dan akan kembali ke kota 1

sebagai kota terakhir. Karena kota 1 sebagai kota awal, maka kota 1 menjadi tabu untuk dikunjungi kembali. Sehingga tingkat *visibility* kota 1 dijadikan 0.

```
mh =

     0    0.3333    0.2500    0.5000    0.1429
     0         0    0.2500    0.1667    0.3333
     0    0.2500         0    0.2000    0.1250
     0    0.1667    0.2000         0    0.1667
     0    0.3333    0.1250    0.1667         0
```

Gambar 20. Nilai *Visibility* ($\boldsymbol{h}$) Kota 1.

Langkah 3: Hitung peluang mengunjungi kota lain dari kota awal dengan bobot *visibility* = 1 dan *pheromone* = 2. Akan dihitung probabilitas perpindahan dari kota awal ke kota lainnya dari semut pertama. Dengan menggunakan persamaan probabilitaas transisi (Santosa & Ai, 2017). Hasil diperoleh pada **Gambar 21** berikut:

$$p_k(r,s) = \frac{\tau(r,s)^{\alpha}\eta(r,s)^{\beta}}{\sum_{u\epsilon M_k}\tau(r,s)^{\alpha}\eta(r,s)^{\beta}} \qquad (11)$$

di mana:

- $p_k(r,s)$ adalah peluang semut k berpindah dari kota r ke kota s.
- $\tau(r,s)$ adalah nilai *pheromone* antara kota r dan s.
- $\eta(r,s)$ adalah *visibility* antara kota r dan s (invers dari jarak).
- M_k adalah kumpulan kota yang belum dikunjungi oleh semut k.
- α dan β adalah parameter yang mengontrol pengaruh *pheromone* dan *visibility.*

```
p =

     0    0.2718    0.2039    0.4078    0.1165
```

Gambar 21. Hasil Probabilitas I.

Peluang dari kota 1 ke kota 2 oleh semut pertama = $2{,}718 \times 10^{-1}$
Peluang dari kota 1 ke kota 3 oleh semut pertama = $2{,}039 \times 10^{-1}$
Peluang dari kota 1 ke kota 4 oleh semut pertama = $4{,}078 \times 10^{-1}$
Peluang dari kota 1 ke kota 5 oleh semut pertama = $1{,}165 \times 10^{-1}$

Langkah 4: Bangkitkan bilangan acak r untuk menentukan kota yang akan dikunjungi berikutnya dengan membandingkan nilai peluang kumulatif. Misalnya, jika diperoleh bilangan acak $r = 3{,}967 \times 10^{-1}$, maka nilai ini akan dibandingkan dengan peluang kumulatif sebelumnya. Bilangan r terletak dibawah nilai peluang pada kota keempat yaitu $4{,}078 \times 10^{-1}$, maka dari kota 1 akan dipilih kota 4 yang akan dikunjungi karena nilai

kumulatif peluang kota 4 terletak dibawah nilai bilangan acak r. Sehingga matriks *visibility* untuk kolom 4 dibuat menjadi 0.

```
mh =

         0    0.3333    0.2500         0    0.1429
         0         0    0.2500         0    0.3333
         0    0.2500         0         0    0.1250
         0    0.1667    0.2000         0    0.1667
         0    0.3333    0.1250         0         0
```

Gambar 22. Nilai *Visibility* (h) Kota 4.

Kemudian dengan cara yang sama, hitung kembali peluang dari kota 4 ke semua kota yang belum dikunjungi kecuali kota 1. Diperoleh sebagai berikut.

```
p =

         0    0.3125    0.3750         0    0.3125
```

Gambar 23. Hasil Probabilitas II.

Proses berlanjut seperti sebelumnya sampai seluruh kota telah dikunjungi. Sehingga diperoleh hasil rute $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 1$ dengan jarak sebesar 21 km dan waktu komputasinya adalah $1{,}865 \times 10^{-1}$ detik.

```
besttour =

     1     4     3     2     5     1


mincost =

    21


compTime =

    0.1865
```

Gambar 24. Implementasi *Ant Colony Optimization* pada MATLAB.

1.6.6. Parameter Metaheuristik

Dalam penerapan algoritma metaheuristik seperti *Genetic Algorithm* dan *Ant Colony Optimization*, pengaturan parameter merupakan aspek penting untuk mencapai hasil yang optimal. Pengaturan ini dapat bervariasi tergantung ukuran dataset yang digunakan.

1.6.6.1. *Genetic Algorithm* (GA)

Genetic Algorithm merupakan metode yang terinspirasi dari proses evolusi biologis. Beberapa parameter kunci dalam metode ini meliputi:

a. Jumlah kromosom: Representasi dari solusi dalam populasi. Jumlah kromosom yang besar dapat meningkatkan peluang menemukan solusi optimal, tetapi juga memperpanjang waktu komputasi (Nababan et al., 2018).
b. Populasi: Jumlah keseluruhan kromosom dalam satu generasi atau iterasi. Populasi yang lebih besar memungkinkan eksplorasi ruang solusi yang lebih luas, tetapi juga meningkatkan waktu komputasi (De Jong & Spears, 1990).
c. Generasi: Merujuk pada jumlah iterasi yang dilakukan pada algoritma. Semakin banyak generasi, memungkinkan solusi optimal ditemukan namun dengan waktu komputasi yang meningkat (Nababan et al., 2018).
d. *Crossover*: Menggabungkan dua kromosom untuk menghasilkan keturunan baru. Nilai *crossover* yang tinggi meningkatkan keragaman genetik, namun menyebabkan kehilangan informasi genetik dari orang tua *(parents)*. Parameter ini berkisar antara 0,6 hingga 0,9 (Nababan et al., 2018).
e. Mutasi: Mengubah satu atau lebih gen dalam kromosom untuk menjaga keragaman genetik. Parameter ini biasanya berkisar antara 0,01 hingga 0,1. Mutasi yang terlalu tinggi dapat mengganggu konvergensi (Herdiana et al., 2022).
f. Nilai *fitness*: Ukuran seberapa baik solusi (kromosom) dalam memenuhi kriteria optimasi. Semakin tinggi nilai *fitness*, semakin baik solusi yang dihasilkan (Santosa & Ai, 2017).

1.6.6.2. *Ant Colony Optimization* (ACO)

Ant Colony Optimization merupakan metode yang terinspirasi dari proses perilaku semut dalam mencari makanan. Beberapa parameter kunci dalam metode ini meliputi:

a. Jumlah semut: Jumlah semut digunakan untuk mengeksplorasi solusi. Semut yang lebih banyak dapat meningkatkan eksplorasi ruang solusi, namun meningkatkan waktu komputasi juga. Jumlah semut biasanya berkisar antara 10 hingga 100 (Joelianto & Wiranto, 2011).
b. *Pheromone*: Zat kimia yang dikeluarkan oleh semut. Tingkat *pheromone* (τ) menentukan seberapa menariknya jalur tersebut untuk dilalui.

Pheromone diperbaharui setelah setiap iterasi berdasarkan kualitas solusi yang ditemukan; jalur dengan solusi yang lebih baik akan mendapatkan lebih banyak *pheromone* (Santosa & Ai, 2017).

c. Pangkat *pheromone* (α): Mengontrol pengaruh *pheromone* terhadap keputusan semut saat memilih jalur. Nilai α biasanya berkisar 1 hingga 5, di mana nilai yang lebih tinggi menunjukkan pengaruh *pheromone* yang lebih besar dibandingkan dengan faktor lain (Khoswara et al., 2023).

d. Pangkat visibilitas (β): Mengukur seberapa baik jalur tertentu berdasarkan jarak. Nilai β diatur antara 1 sampai 5. Semakin besar jarak, semakin tinggi visibilitasnya (Joelianto & Wiranto, 2011).

e. Koefisien evaporasi (ρ): Parameter yang menentukan seberapa cepat *pheromone* menguap dari jalur yang telah dilalui semut. Nilai ρ yang lebih tinggi menunjukkan penguapan yang cepat, sehingga daya Tarik jalur kurang dilalui, sementara nilai yang lebih rendah memperlambat penguapan dan meningkatkan daya tarik jalur tersebut (Nugraha et al., 2020).

f. Kriteria terminasi: Kriteria ini berupa jumlah iterasi maksimum atau ketika tidak ada perbaikan signifikan dalam solusi selama beberapa kali iterasi. Untuk dataset kecil, kriteria terminasi biasanya lebih ketat dibandingkan dengan dataset besar yang memerlukan banyak iterasi sebelum mencapai konvergensi (Lusweti et al., 2022).

1.6.7. Standar Deviasi

Standar deviasi merupakan ukuran yang menggambarkan seberapa besar penyimpangan nilai – nilai dalam suatu dataset terhadap rata – rata *(mean)* dataset tersebut (Montgomery, 2013). Dalam konteks TSP, standar deviasi digunakan untuk mengukur variasi hasil solusi dan membantu menilai konsistensi serta stabilitas rute yang ditemukan oleh algoritma. Perhitungan standar deviasi pada TSP dapat dilakukan dengan menggunakan persamaan berikut:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{12}$$

di mana:

- n : jumlah *run* atau percobaan
- x_i : jarak kota pada *run* ke – i
- $\bar{x}$: rata – rata jarak total dari seluruh *run*

Dalam penelitian ini, standar deviasi digunakan untuk mengevaluasi stabilitas dan konsistensi solusi yang dihasilkan oleh algoritma metaheuristik. Metaheuristik mengandung elemen acak, yang menyebabkan hasil solusi bervariasi setiap kali dijalankan. Oleh karena itu, uji coba dilakukan sebanyak 10 kali *run* menggunakan MATLAB, dan standar deviasi dihitung untuk mengetahui sejauh mana hasil tersebut bervariasi dari rata – rata solusi yang diperoleh.

1.6.8. Koefisien Variasi

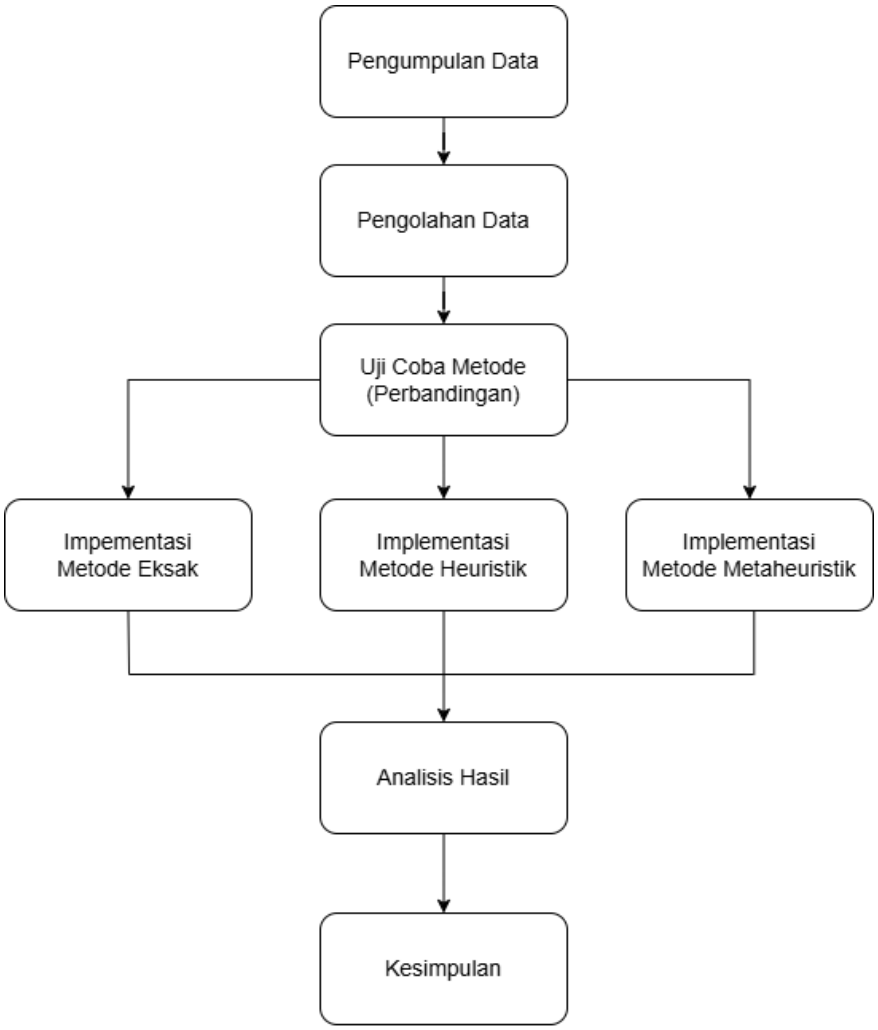
Koefisien variasi atau *Coefficient of Variation* (CV) adalah ukuran statistik yang digunakan untuk menilai seberapa besar variasi atau fluktuasi relatif terhadap nilai rata – rata suatu dataset. CV memberikan gambaran yang mudah dipahami karena hasilnya yang dinyatakan dalam bentuk presentase dari rata – rata (Siagian & Sugiarto, 2000). Adapun persamaannya adalah sebagai berikut:

$$CV = \left(\frac{Standar\ Deviasi}{Rata - rata}\right) \times 100\% \qquad (13)$$

Dalam penelitian ini, CV digunakan untuk menganalisis variabilitas solusi yang dihasilkan oleh metode metaheuristik sehingga memungkinkan untuk mengukur fluktuasi relatif terhadap hasil yang diperoleh, memberikan gambaran yang jelas tentang seberapa stabil atau bervariasinya solusi yang dihasilkan oleh masing – masing metode.

BAB II
METODE PENELITIAN

Penelitian ini dirancang sebagai studi komparatif yang bertujuan untuk menganalisis dan membandingkan tiga pendekatan utama dalam menyelesaikan TSP. Perbandingan dilakukan untuk mengevaluasi efektivitas masing – masing metode dalam menghasilkan solusi optimal dan mempertimbangkan waktu komputasi pada berbagai ukuran dataset. Adapun tahapan dalam penelitian ini ditunjukkan pada diagram alur yang diperlihatkan pada gambar **Gambar 25**.



Gambar 25. Alur Penelitian.

2.1. Pengumpulan dan Pengolahan Data

Pada penelitian ini, penulis menggunakan data sekunder dengan ukuran data berukuran kecil (5 dan 10 kota), data berukuran sedang (50 kota) dan data

berukuran besar (100 kota) yang diperoleh dari *Google Maps*. Kota – kota yang dimaksud berasa dari Jawa Barat, Brunei Darussalam, New Zealand dan Jerman. Adapun teknik pengumpulan data yang digunakan pada penelitian ini sebagai berikut:

1. Menentukan jumlah kota yang akan digunakan berdasarkan kebutuhan penelitian, dengan mempertimbangkan kriteria seperti ketersediaan data jarak dan representasi dataset kecil, sedang dan besar.
2. Mencari jarak kota melalui *Google Maps* yang menyediakan data akurat mengenai jarak tempuh dan waktu perjalanan dan mencatat jarak dalam satuan kilometer.
3. Data jarak yang diperoleh kemudian disusun dalam bentuk matriks di mana setiap baris dan kolom mewakili kota – kota yang dipilih, dan setiap sel berisi jarak antara kota tersebut.
4. Dengan data yang telah terkumpul dan disusun, selanjutnya dilakukan penelitian dengan menerapkan berbagai metode penyelesaian TSP.

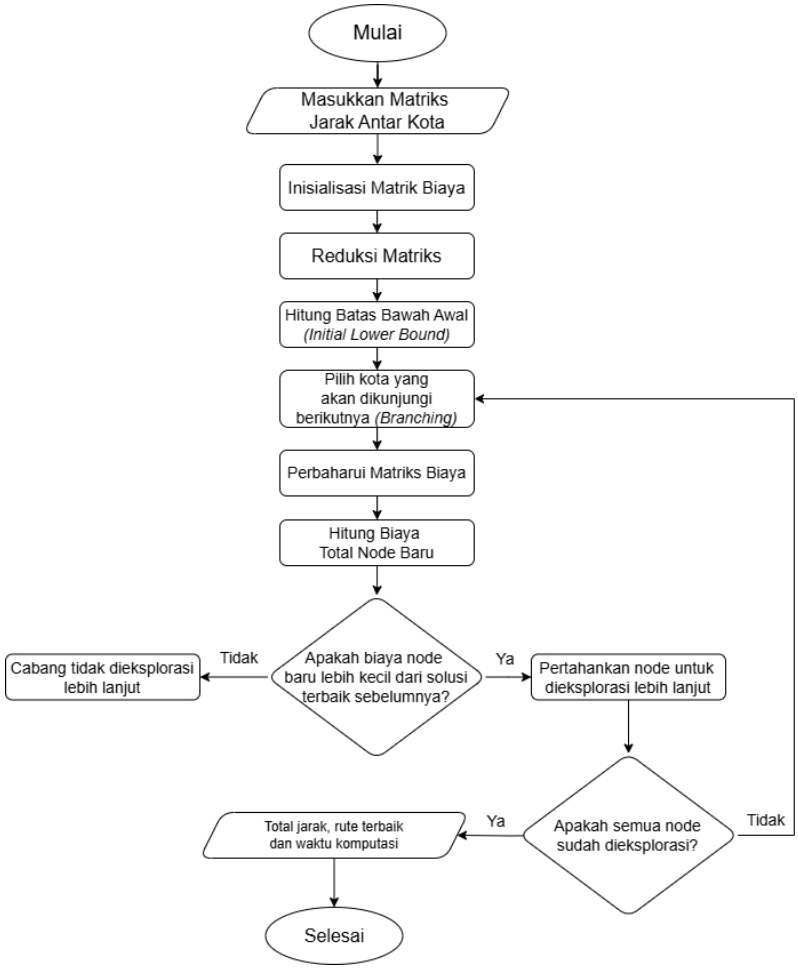
2.2. Metode Analisis

2.2.1. Penyelesaian Masalah TSP dengan Algoritma *Branch and Bound*

Pada **Gambar 26** menunjukkan diagram alur dari metode *Branch and Bound*. Dalam penerapan algoritma ini, matriks jarak antar kota direduksi di setiap langkah percabangan untuk memperbaharui batas bawah *(lower bound)* dari node tersebut. Metode ini menghasilkan solusi optimal dengan mengurangi matriks dan melakukan percabangan (Rajarajeswari & Maheswari, 2020). Langkah – langkah algoritma metode ini adalah sebagai berikut:

1. Memasukkan data matriks TSP ke dalam program.
2. Inisialisasi: Buat matriks jarak antar kota dengan nilai tak hingga ∞ untuk kota yang tidak terhubung, lalu lakukan reduksi baris dan kolom untuk menghitung batas bawah awal *(initial lower bound)*.
2. Percabangan *(Branching)*: Pilih kota berikutnya, kemudian perbarui matriks dengan menghapus baris dan kolom yang dikunjungi kemudian lakukan ulang reduksi matriks untuk mendapatkan nilai batas bawah baru.
3. Pembatasan *(Bounding)*: Hitung total biaya dari setiap node baru. Jika total biaya lebih kecil dari solusi terbaik saat ini, node tersebut dieksplorasi lebih lanjut. Jika tidak, node tersebut dipangkas.
4. Penyelesaian: Proses berlanjut hingga semua jalur telah dieksplorasi atau dipangkas dan jalur dengan biaya terendah menjadi solusi optimal.
5. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.

Metode ini memprioritaskan simpul dengan batas bawah terendah dan memangkas cabang yang tidak menjanjikan solusi lebih baik.

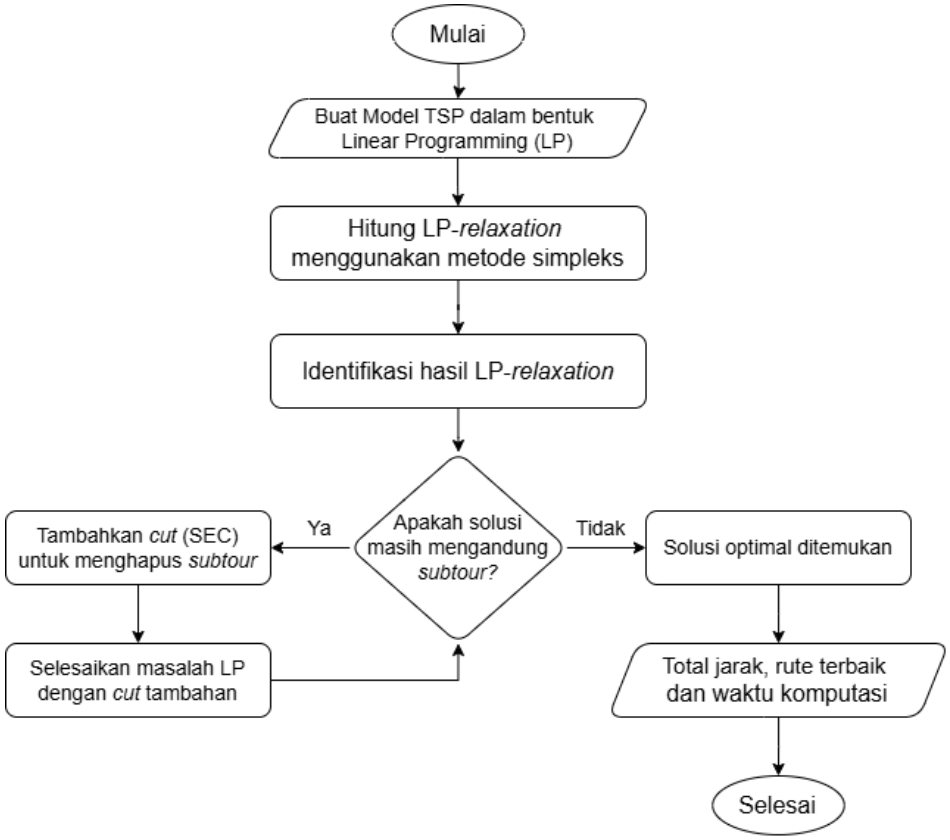


Gambar 26. Diagram Alur Algoritma *Branch and Bound.*

2.2.2. Penyelesaian Masalah TSP dengan Algoritma *Cutting Plane*

Pada **Gambar 27** menunjukkan diagram alur dari metode *Cutting Plane.* Teknik ini bekerja dengan menambahkan batasan tambahan *(constraints)* atau *cuts* pada *Linnear Programming* (LP) untuk mendekati solusi integer optimal (Applegate et al., 2003). Langkah – langkah algoritma metode ini adalah sebagai berikut:

1. Inisialisasi model TSP ke dalam bentuk LP dengan variabel biner x_{ij} untuk setiap pasang kota i dan j dengan LP relaksasi awal dan mengabaikan batasan integer $x_{ij} \in \{0,1\}$ sehingga $0 \leq x_{ij} \leq 1$.
2. Hitung LP relaksasi dengan menggunakan metode optimasi seperti simpleks untuk mendapat solusi sementara x_{ij}.
3. Identifikasi jika solusi menghasilkan *subtour* tambahkan *Subtour Elimination Constraint* (SEC) dengan menggunakan persamaan (8).
4. Proses ini berlanjut hingga solusi yang diperoleh adalah integer dan membentuk rute lengkap yang valid.
5. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.

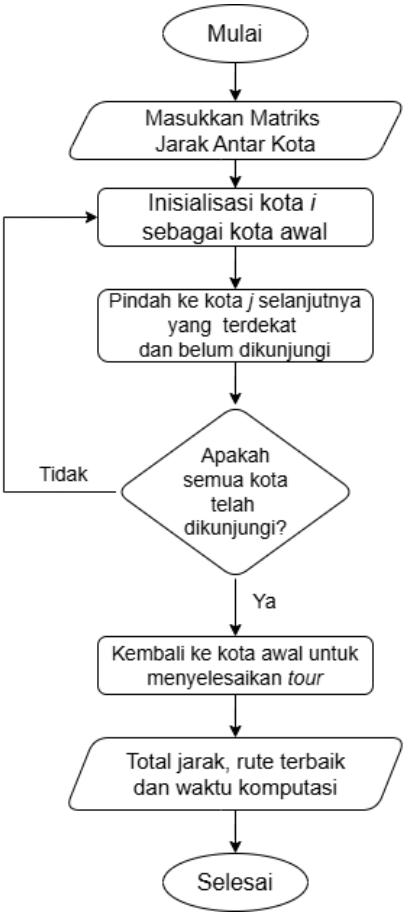


Gambar 27. Alur Diagram Algortima *Cutting Plane.*

2.2.3. Penyelesaian masalah TSP dengan Algoritma *Nearest Neighbor*

Pada **Gambar 28** menunjukkan diagram alur dari metode *Nearest Neighbor Heuristic.* Metode ini merupakan metode heuristik sederhana dan cepat, sering digunakan untuk mencari solusi mendekati optimal pada TSP (Winston, 2004). Langkah – langkah algoritma metode ini adalah sebagai berikut:

1. Memasukkan data matriks TSP ke dalam program.
2. Inisialisasi dengan memilih kota i sebagi kota awal.
3. Pindah kota *j* terdekat selanjutnya yang belum dikunjungi.
4. Jika belum ada kota yang belum dikunjungi kembali ke langkah 3 sampai seluruh kota telah dikunjungi.
5. Setelah semua kota dikunjungi, akhiri *tour* dengan kembali ke kota awal.
6. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.



Gambar 28. Diagram Alur Algoritma *Nearest Neighbor Heuristic.*

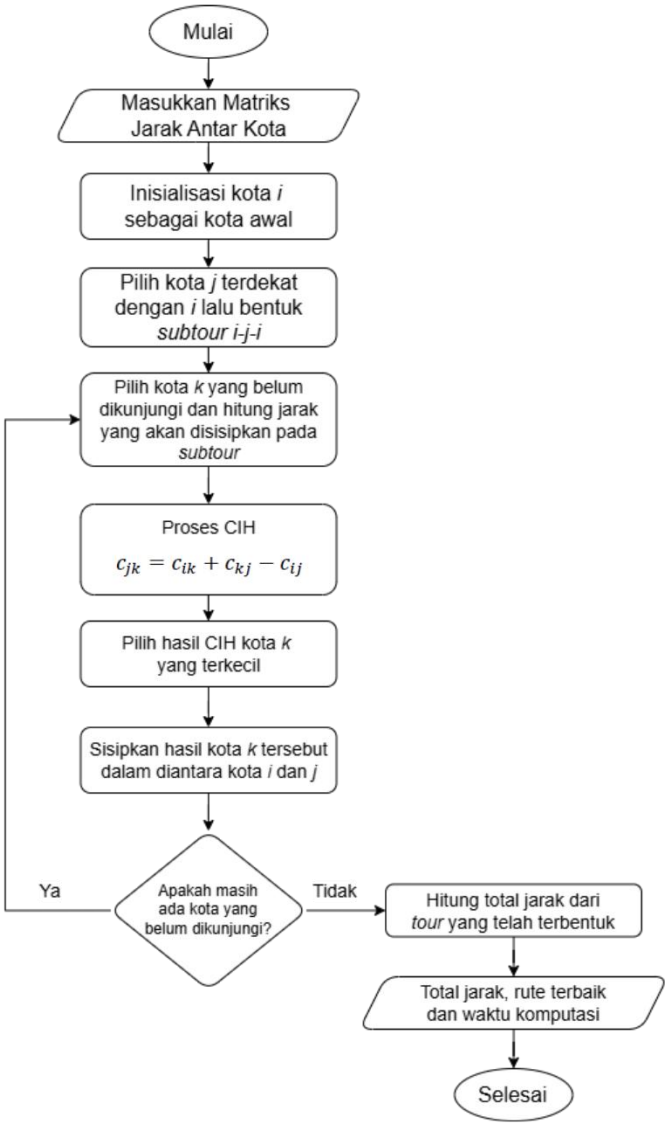
2.2.4. Penyelesaian masalah TSP dengan Algoritma *Cheapest Insertion*

Pada **Gambar 29** menunjukkan diagram alur dari metode *Cheapest Insertion Heuristic.* Metode ini sering kali lebih efektif daripada *Nearest Neighbour Heuristic*, namun keduanya memiliki keterbatasan dalam hal optimalitas. Dalam beberapa kasus, solusi yang dihasilkan oleh kedua metode ini dapat cukup jauh dari solusi optimal. Oleh karena itu, penting untuk mempertimbangkan kombinasi metode atau pendekatan lain guna meningkatkan akurasi hasil (Winston, 2004).

Langkah – langkah algoritma ini adalah sebagai berikut:

1. Memasukkan data matriks TSP ke dalam program.
2. Inisialisasi dengan memilih kota i sebagi kota awal dan kota *j* terdekat.
3. Bentuk *subtour* menjadi kota $i \rightarrow j \rightarrow i$.
4. Untuk setiap kota yang belum dimasukkan, hitung biayanya yang akan ditambahkan ke *tour* yang ada.
5. Pilih kota k yang belum dikunjungi dan hitung jarak yang akan disisipkan pada *subtour* saat ini. Kemudian proses perhitungan dilakukan menggunakan persamaan (9).

6. Pilih hasil perhitungan dengan kota k yang terkecil dan sisipkan pada *subtour* diantara kota i dan j.
7. Ulangi langkah 5 sampai 6 hingga seluruh kota sudah disisipkan dan membentuk rute secara lengkap.
8. Setelah semua kota dikunjungi, akhiri *tour* dengan kembali ke kota awal.
9. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.

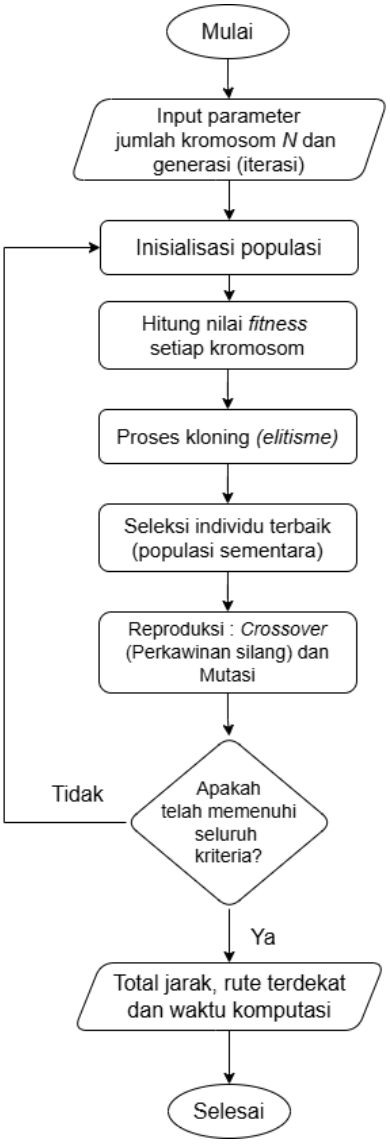


Gambar 29. Diagram Alur Algoritma *Cheapest Insertion Heuristic.*

2.2.5. Penyelesaian masalah TSP dengan Algoritma *Genetic Algorithm*

Pada **Gambar 30** menunjukkan diagram alur dari metode *Genetic Algorithm.* Metode ini meniru mekanisme evolusi dalam perkembangan makhluk hidup.

Algoritma ini membentuk prosedur yang pada akhirnya menghasilkan solusi untuk masalah optimasi. Dalam metode ini, pencarian solusi hanya didasarkan pada nilai fungsi tujuan, tanpa memanfaatkan gradien atau teknik kalkulus lainnya (Santosa & Ai, 2017).



Gambar 30. Diagram Alur *Genetic Algorithm.*

Langkah – langkah algoritma ini adalah sebagai berikut:

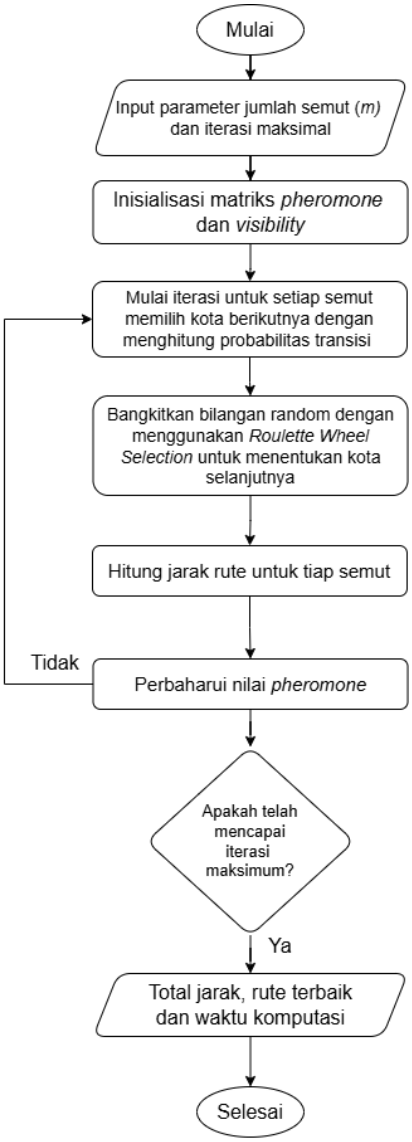
1. Memasukkan data matriks TSP ke dalam program.
2. Memasukkan parameter awal seperti jumlah kromosom N dan iterasi maksimal.
3. Inisialisasi populasi awal dari solusi yang mungkin.

4. Memilih kromosom terbaik untuk reproduksi berdasarkan nilai *fitness* – nya serta melakukan proses *elitisme* dilakukan dengan menyalin kromosom terbaik.
5. Menggabungkan dua individu untuk menghasilkan keturunan baru *(crossover)*.
6. Seleksi individu terbaik dengan mengubah sebagian dari individu untuk menjaga keragaman genetik dalam populasi.
7. Menggantikan individu untuk menjaga keragaman genetik dalam populasi.
8. Menggantikan individu terburuk dengan keturunan baru untuk membentuk generasi berikutnya.
9. Mengulangi proses hingga kriteria penghentian terpenuhi seperti mencapai jumlah generasi (iterasi) tertentu.
10. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.

2.2.6 Penyelesaian masalah TSP dengan Algoritma *Ant Colony Optimization*

Pada **Gambar 31** menunjukkan diagram alur dari metode *Ant Colony Optimization*. Metode ini yang terinspirasi oleh perilaku koloni semut dalam mencari sumber makanan dari sarang dan kembali ke sarang yang secara alami menghasilkan rute terpendek. Semut yang melewati suatu lintasan meninggalkan jejak *pheromone,* yang dapat dicium oleh semut – semut berikutnya untuk mengetahui apakah jalur tersebut telah dilewati atau belum (Santosa & Ai, 2017). Langkah – langkah algoritma ini adalah sebagai berikut:

1. Memasukkan data matriks TSP ke dalam program.
2. Memasukkan parameter awal seperti jumlah koloni semut (m), jumlah iterasi maksimum dan tingkat penguapan *pheromone*.
3. Inisialisas matriks dengan menyimpan informasi *pheromone* dan *visibility.*
4. Pembuatan jalur setiap semut berdasarkan probabilitas transisi yang dipengaruhi oleh jumlah *pheromone.*
5. Membangkitkan nilai *random* dalam hal ini menggunakan *Roulette Wheel Selection* untuk menentukan rute selanjutnya.
6. Menghitung rute jarak untuk tiap semut dan kembali memperbaharui nilai *pheromone.*
7. Ulangi proses dari langkah pembuatan jalur sampai kriteria penghentian terpenuhi.
8. Hasil yang diperoleh yaitu total jarak, rute terbaik dan waktu komputasi.



Gambar 31. Diagram Alur Algoritma *Ant Colony Optimization*.

2.3. Parameter Metaheuristik

Penelitian ini menggunakan parameter standar pada algoritma metaheuristik untuk menjaga keseimbangan antara kualitas solusi dan waktu komputasi. Adapun parameter yang digunakan untuk metode *Genetic Algorithm* dan *Ant Colony Optimization* merujuk pada parameter yang telah digunakan sebelumnya yaitu Santosa & Ai (2017), Khoswara et al. (2023) dan Dorigo & Gambardella (1997). Penggunaan parameter untuk kedua metode adalah sebagai berikut:

Tabel 14. Parameter *Genetic Algorithm*.

Dataset	Jumlah Kromosom	Maks. Generasi	Probabilitas *Crossover*	Probabilitas Mutasi
5 Kota	10	50	0,8	0,01
10 Kota	20	100		
50 Kota	100	200		
100 Kota	200	500		

Tabel 15. Parameter *Ant Colony Optimization*.

Dataset	Jumlah Semut	Maks. Iterasi	Koefisien Evaporasi (ρ)	Pangkat *Pheromone* (α)	Pangkat *Visibility* (β)
5 Kota	10	50	0,5	1	2
10 Kota	20	100			
50 Kota	30	200			
100 Kota	50	500			

2.4. Pengujian

Pengujian dilakukan dengan membandingkan hasil solusi optimal dari metode eksak (sebagai *benchmark*), heuristik dan metaheuristik dalam penyelesaian TSP. Kemudian, dilakukan analisis untuk melihat efektivitas dari masing – masing metode dalam segi kualitas solusi dan waktu komputasi yang dihasilkan.