LAPORAN PROYEK
Traveling Salesman Problem with Genetic Algorithm

Disusun oleh:

1. 12S17001 – Krista Lumbantoruan
2. 12S17026 – Mika L. V. Manurung
3. 12S17033 – Martin H Sinaga
4. 12S17049 – Paddy T Silitonga

PROGRAM STUDI SARJANA SISTEM INFORMASI
FAKULTAS TEKNIK INFORMATIKA DAN ELEKTRO
INSTITUT TEKNOLOGI DEL
Januari 2019

DAFTAR ISI

1. Pendahuluan

1.1.Latar Belakang

Perkembangan teknologi komputer dan ilmu pengetahuan saat ini mengalami perkembangan yang sangat pesat terutama dalam hal proses komputasi yang memungkinkan sebuah masalah dapat diselesaikan dengan bantuan komputer sebagai solusinya. Seiring dengan itu pula peningkatan ilmu pengetahuan telah melahirkan begitu banyak algoritma-algoritma yang sangat membantu bagi pekerjaan manusia.

Travelling Salesman Problem (TSP) adalah sebuah masalah dimana seorang *salesman* mempunyai tugas untuk mengirimkan pesanan barang ke rumah *client* yang berada di sejumlah tempat yang berbeda di sebuah kota. *Salesman* tersebut mempunyai masalah dalam hal menentukan tempat mana yang terlebih dahulu dikunjungi sedemikian sehingga total jarak dan waktu bepergian diperkecil dan setiap kota hanya boleh dilalui sekali dalam satu perjalanan, dan perjalanan berakhir pada kota awal dimana seorang sales memulai perjalananya.

Luasnya daerah, banyaknya persimpangan jalan dan banyaknya jalan raya adalah faktor yang membuat kesulitan bagi seorang *Salesman* untuk mencari rute optimum, baik dari jarak maupun biaya yang diperlukan untuk pengantaran barang dari satu tempat ke tempat lainnya. Rute optimum bertujuan untuk mendapatkan jarak yang optimal, biaya yang optimal untuk menempuh suatu perjalanan dari titik asal ke titik tujuan.

Permasalahan yang timbul yaitu seorang sales harus bisa menentukan rute terpendek dengan waktu yang lebih efektif sehingga suatu tempat tidak terlewati lebih dari satu kali. Algoritma genetik adalah algoritma pencarian heuristik yang bekerja berdasarkan mekanisme seleksi alam dan genetika alam. Ide utama dibalik algoritma ini adalah memilih individu-individu terbaik dari sebuah populasi 2 individu dan melakukan rekombinasi antar individu untuk membangkitkan individu baru yang diharapkan lebih baik dari individu sebelumnya. Jadi, solusi yang dapat kami berikan untuk menyelesaikan permasalahan tersebut adalah membangun sebuah sistem yang dapat melakukan pencarian alamat yang memiliki rute perjalanan terpendek. *Salesman* dapat memilih alamat yang akan dikunjungi melalui sistem ini, kemudian sistem akan memberikan daftar alamat beserta rute perjalanan terpendek yang dipilihnya.

Proses pencarian pada algoritma genetika dimulai dengan memilih himpunan penyelesaian, digambarkan dengan kromosom yang disebut dengan populasi. Solusi dari satu populasi diambil untuk membentuk populasi baru, dimana pemilihannya tergantung dari *fitness* terbaiknya. Hal ini dimotivasi dengan harapan bahwa populasi yang baru akan lebih baik dibandingkan populasi terdahulu. Proses ini dilakukan berulang-ulang hingga kondisi tertentu terpenuhi.

1.2. Tujuan

Tujuan dari sistem *travelling salesman problem with genetic algorithm* adalah :

1. Memenuhi persyaratan lulus mata kuliah Kecerdasan Buatan
2. Untuk mengetahui cara kerja algoritma genetik dalam memecahkan masalah *Travelling Salesman Problem*.
3. Melakukan analisis hasil eksekusi dari perangkat lunak yang dibangun terhadap beberapa contoh kasus dalam menyelesaikan *Travelling salesman problem*. Analisis bertujuan untuk mengetahui perbandingan performansi kedua algoritma tersebut.

2. Landasan Teori

2.1. Graph

Teori Graf merupakan suatu diagram yang memuat informasi tertentu jika diinterpretasikan secara tepat. Dalam kehidupan sehari-hari graf digunakan untuk mengambarkan berbagai macam struktur yang ada. Tujuannya adalah sebagai visualisasi objek-objek agar lebih mudah dimengerti. Beberapa contoh graf yang sering dijumpai, antara lain struktur organisasi, bagan alir, pengambilan mata kuliah, peta, rangkaian listrik, dan lain-lain.

Graf G didefinisikan sebagai pasangan himpunan (V,E), ditulis dengan notasi G=(V,E), yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul dan E adalah himpunan sisi yang menghubungkan sepasang simpul.

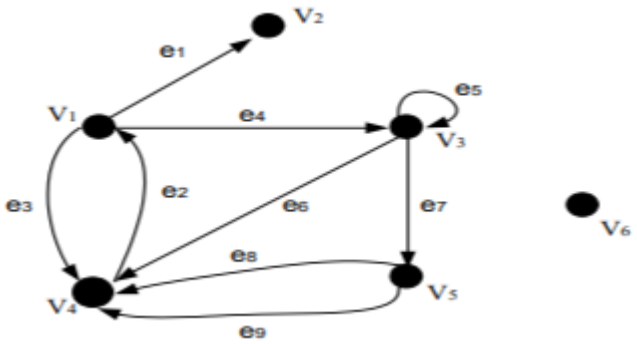
Garis yang hanya berhubungan dengan satu titik ujung disebut loop. Dua garis berbeda yang menghubungkan titik yang sama disebut garis paralel. Dua titik dikatakan berhubungan (adjacent) jika garis menghubungkan keduanya. Titik yang tidak memiliki garis yang berhubungan dengannya disebut titik terasing (isolating point). Graf yang tidak memiliki titik (sehingga tidak mewakili garis) disebut garis kosong.

Jika semua garisnya berarah, maka grafnya disebut graf berarah (directed graph), atau sering disingkat digraph. Jika semua garisnya tidak berarah, maka grafnya disebut graf tak berarah (undirected graph). Sehinga dapat ditinjau dari arahnya, graf dapat dibagi menjadi dua yaitu graf berarah dan graf tidak berarah.

2.1.1 Graf Berarah (Directed Graf = Digraph)

Pada graf berarah, arah sisi/urutan ikut diperhatikan. Dalam suatu graf, lintasan (path) adalah urutan simpul, atau sisi yang dibentuk untuk bergerak dari satu simpul ke simpul yang lain. Dalam graf berarah, titik akhir dari sebuah busur

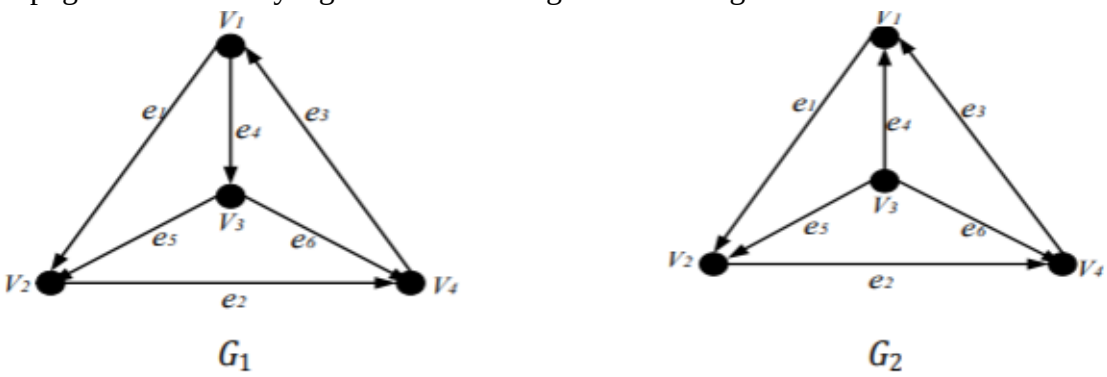
akan menjadi titik awal dari busur berikutnya. Sirkuit adalah lintasan yang memiliki simpul awal dan akhir yang sama. Panjang lintasan adalah banyaknya sisi yang dilalui lintasan tersebut.



Gambar 1. Graf berarah

2.1.2 Graf berarah terhubung

Suatu graf tak berarah disebut terhubung jika ada walk yang menghubungkan setiap dua titiknya. Pengertian itu berlaku juga bagi graf berarah. Berdasarkan arah garisnya, dalam graf berarah dikenal dua jenis keterhubungan, yaitu terhubung kuat dan terhubung lemah. Misalkan G adalah suatu graf berarah dan v, w adalah sembarang 2 titik dalam G. G disebut terhubung kuat jika ada path berarah dari v ke w. G disebut terhubung lemah, jika G tidak terhubung kuat, tetapi graf tak berarah yang bersesuaian dengan G terhbung.



Gambar 2. Graf berarah terhubung

2.1.3 Graf tak berarah

Suatu graf G terdiri dari dua himpunan yang berhingga, yaitu himpunan simpul-simpul tak kosong (V(G)) dan himpunan jalur-jalur (E(G)). Jika semua jalurnya tidak berarah, maka grafnya disebut graf tak berarah.

2.2.Lintasan

Lintasan *Euler* adalah lintasan yang melalui masing-masing sisi di dalam graf tepat satu kali. bila lintasan tersebut kembali ke simpul asal, sehingga membentuk lintasan tertutup maka disebut sirkuit euler.Sirkuit Euler ialah sirkuit yang melewati masing-masing sisi di dalam graf tepat satu kali [5]. *Sirkuit Euler* adalah sirkuit dimana setiap titik dalam graf G muncul paling sedikit satu kali dan setiap garis muncul tepat satu kali.Sebuah perjalanan Euler (*Euler cycle*) pada *graph* G adalah sebuah *cycle* sederhana yang melalui setiap *edge* di G hanya sekali.

- Graf yang mempunyai lintasan Euler disebut juga graf semi-Euler (*semi-Euleriangraph*).
- Graf yang mempunyai sirkuit Euler disebut juga graf Euler (*Eulerian graph*).

2.3.Travelling Salesman Problem (TSP)

Persoalan TSP merupakan salah satu persoalan optimasi *kombinatorial* (kombinasi permasalahan). Banyak permasalahan yang dapat direpresentasikan dalam bentuk TSP. Persoalan ini sendiri menggunakan representasi graf untuk memodelkan persoalan yang diwakili sehingga lebih memudahkan penyelesaiannya. Diantara permasalahan yang dapat direpresentasikan dengan TSP adalah pencarian rute bis sekolah untuk mengantarkan siswa, pengambilan tagihan telepon, efisiensi pengiriman surat atau barang, perancangan pemasangan pipa saluran dan lain-lain. Persoalan yang muncul adalah bagaimana cara mengunjungi node (simpul) pada graf dari titik awal ke setiap titik-titik lainnya dengan bobot minimum dan kembali lagi ke asal node. Bobot ini sendiri dapat mewakili berbagai hal, seperti berapa biaya minimum, jarak minimum, waktu minimum, dan lain-lain.

2.4.Local Search

Algoritma pencarian dirancang untuk menjelajahi ruang pencarian secara sistematis. metode heuristik untuk menyelesaikan masalah optimisasi keras komputasi. Pencarian lokal dapat digunakan pada masalah yang dapat dirumuskan sebagai menemukan solusi memaksimalkan kriteria di antara sejumlah solusi kandidat . Algoritme pencarian lokal bergerak dari solusi ke solusi dalam ruang kandidat solusi (ruang pencarian) dengan menerapkan perubahan lokal, hingga solusi yang dianggap optimal ditemukan atau batasan waktu telah berlalu. Algoritma pencarian lokal secara luas diterapkan pada banyak masalah komputasi yang sulit, termasuk masalah dari ilmu komputer (khususnya kecerdasan buatan), matematika , riset operasi , teknik , dan bioinformatika . Contoh dari algoritma pencarian lokal adalah WalkSAT ,

algoritma 2-opt untuk Traveling Salesman Problem dan algoritma Metropolis-Hastings .

Beberapa masalah yang menerapkan local search :

1. Masalah penutup simpul, dimana solusi adalah penutup simpul dari grafik dan targetnya adalah untuk menemukan solusi dengan jumlah simpul mini mal.
2. Masalah kepuasan boolean , di mana solusi kandidat adalah penugasan kebenaran, dan targetnya adalah untuk memaksimalkan jumlah klausa yang dipenuhi oleh penugasan; dalam hal ini, solusi final hanya digunakan jika memenuhi *semua* klausa.
3. Traveling Salesman Problem dimana solusi adalh siklus yang berisi semua node grafik dan targetnya adalah untuk meminimalkan total panjanng siklus

Algoritma pencarian lokal biasanya merupakan pendekatan atau algoritma yang tidak lengkap , karena pencarian dapat berhenti bahkan jika solusi terbaik yang ditemukan oleh algoritma tidak optimal. Ini dapat terjadi bahkan jika penghentian disebabkan oleh ketidakmungkinan meningkatkan solusi, karena solusi optimal dapat terletak jauh dari lingkungan solusi yang dilintasi oleh algoritma.

2.5. Genetic Algorithm

Genetic algorithm merupakan suatu metode optomasi untuk mencari solusi yang optimal dari suatu permasalahan. Algoritma ini memanfaatkan proses seleksi alamiah yang dikenal dengan proses evolusi. Dalam proses evolusi, individu secara terus-menerus mengalami perubahan gen untuk menyesuaikan dengan lingkungan hidupnya.

Sebuah solusi yang dibangkitkan dalam algoritma genetika disebut sebagai koromosom, sedangkan kumpulan kromosom yang berupa individu tersebut disebut sebuah populasi.

Sebuah kromosom dibentuk dari komponen-komponen penyusun yang disebut sebagai gen dan nilainya dapat berupa bilangan numerik, biner, simbol ataupun karakter tergantung dari permasalahan yang ingin diselesaikan. Kromosom-kromosom tersebut akan berevolusi secara berkelanjutan yang disebut dengan generasi. Pada setiap generasi kromosom-kromosom tersebut dievaluasi tingkat keberhasilan nilai solusinya terhadap masalah yang ingin diselesaikan menggunakan ukuran kebugaran yang disebut *fitness*.

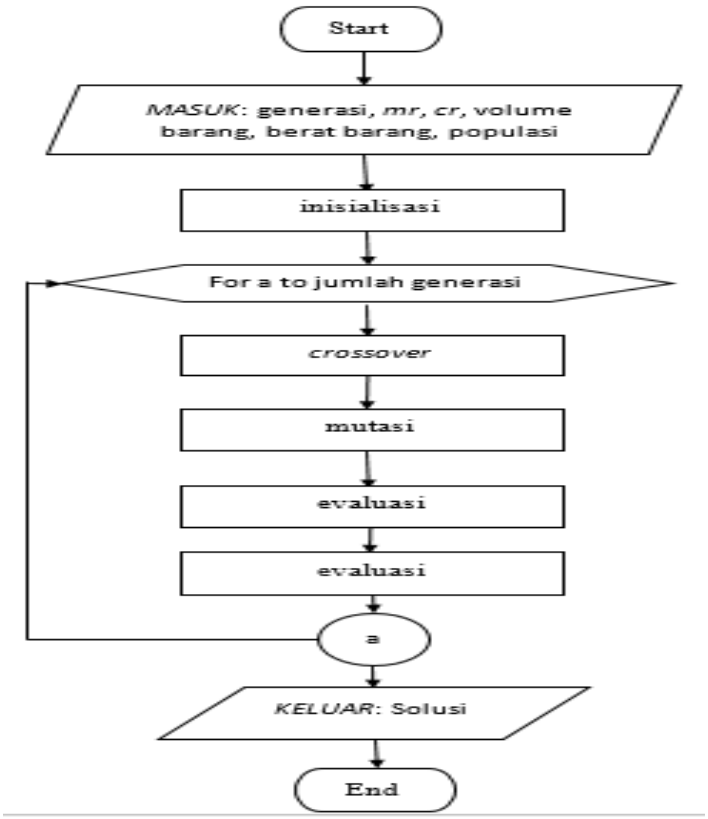
Jika nilai *fitness* semakin besar, maka sistem yang dihasilkan akan semakin baik. Funghsi *fitness* ini ditentukan dengan menggunakan metode heuristic. Algoritma genetika mempunyai metodologi optimasi sederhana sebagai berikut :

1. Menentukan populasi solusi sejumlah tertentu.
2. Menghitung nilai *fitness function* semua solusi yang ada dalam populasi.
3. Memilih beberapa solusi dengan nilai *fitness function* yang paling tinggi.
4. Melakukan optimasi dengan cara mutasi dan *crossover* sebanyak yang diperlukan.
5. Menentukan solusi terbaik sebagai solusi terhadap permasalahan yang dioptimasi.

3. Desain

3.1. Cara Kerja TSP

Optimasi pemilihan rute merupakan masalah yang banyak dibahas pada penelitian ilmu komputer. Antar jemput laundry dengan pelanggan yang memiliki waktu khusus untuk menerima barang adalah salah satu contoh kasus pemilihan rute. Penghitungan rute tercepat memegang peranan penting karena harus tepat waktu dan semua pelanggan dapat dilayani. Berbeda dengan traveling salesman problem (TSP) konvensional yang bertujuan untuk meminimalkan jarak, kasus ini juga harus dipertimbangkan waktu ketersediaan setiap pelanggan. Pencarian solusi untuk permasalahannya adalah dengan mengkombinasikan solusi-solusi (kromosom) untuk menghasilkan solusi baru dengan menggunakan operator genetika (seleksi, crossover dan mutasi). Untuk mencari solusi terbaik digunakan beberapa kombinasi probabilitas crossover dan mutasi serta ukuran populasi dan ukuran generasi. Dari hasil pengujian kombinasi probabilitas crossover yang terbaik adalah 0,4 dan mutasi adalah 0,6 sedangkan untuk ukuran generasi optimal adalah 2000. Dari nilai-nilai parameter ini didapatkan solusi yang memungkinkan untuk melayani semua pelanggan dengan time window masing - masing



Gambar 3. Cara kerja sistem

3.1.1. Tampilan (Antarmuka Pengguna)

Tidak ada

3.1.2. Deskripsi dan Format Masukan

Masukan untuk tampilan fungsi input titik koordinat kota berupa float sedangkan untuk input nama kota berupa string. User memasukkan *source* yaitu list nama kota yang akan dikunjungi oleh *salesman* dan titik koordinat setiap kota.

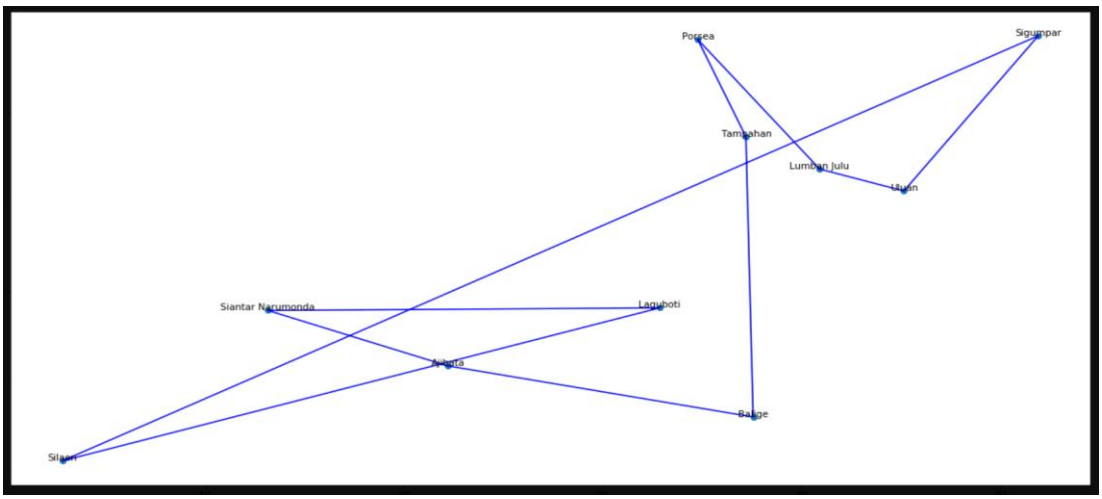
```
1 cityList = [[77.580643,12.972442],[72.88261,19.07283],[77.216721,28.644800],[42.856255,10.516726]
2             ,[85.158875,25.612677],[80.9231262,26.8392792],[74.797371,34.083656],[53.19384,18.92832]
3             ,[91.912832,34.293363],[62.239233,15.82732]]
4 val = distance_between_cities(cityList).values()
5
```

Gambar 4. Inputan titik koordinat

```python
import numpy as np
city_names = ['Balige', 'Laguboti', 'Tampahan', 'Silaen','Uluan','Lumban Julu',
              'Porsea','Siantar Narumonda','Sigumpar','Ajibata']
def plot_pop(cities):
    plt.figure(figsize=(20,10))
    x = [i[0] for i in cities]
    y = [i[1] for i in cities]
    x1=[x[0],x[-1]]
    y1=[y[0],y[-1]]
    plt.plot(x, y, 'b', x1, y1, 'b')
    plt.scatter (x, y)
    j = [77.580643,72.88261,77.216721,42.856255,85.158875,80.9231262,74.797371,53.19384,91.912832,62.239233]
    k = [12.972442,19.07283,28.644800,10.516726,25.612677,26.8392792,34.083656,18.92832,34.293363,15.82732]


    for i, txt in enumerate(city_names):
        plt.annotate(txt, (j[i], k[i]),horizontalalignment='center',
            #verticalalignment='bottom',
                     )
    plt.show()
    return
```

Gambar 5. Inputan nama kota



Gambar 6. Graph antar kota

3.1.3. Deskripsi Proses

Implementasi *Genetic Algorithm* diawali dengan mendaftarkan semua populasi dan di *generate* berdasarkan titik koordinat kota untuk mendapatkan graph yang paling optimum.

```python
def initialPopulation(cities, populationSize):
    population = [generatePath(cities) for i in range(0, populationSize)]
    return population
population = initialPopulation(cityList,20)
```

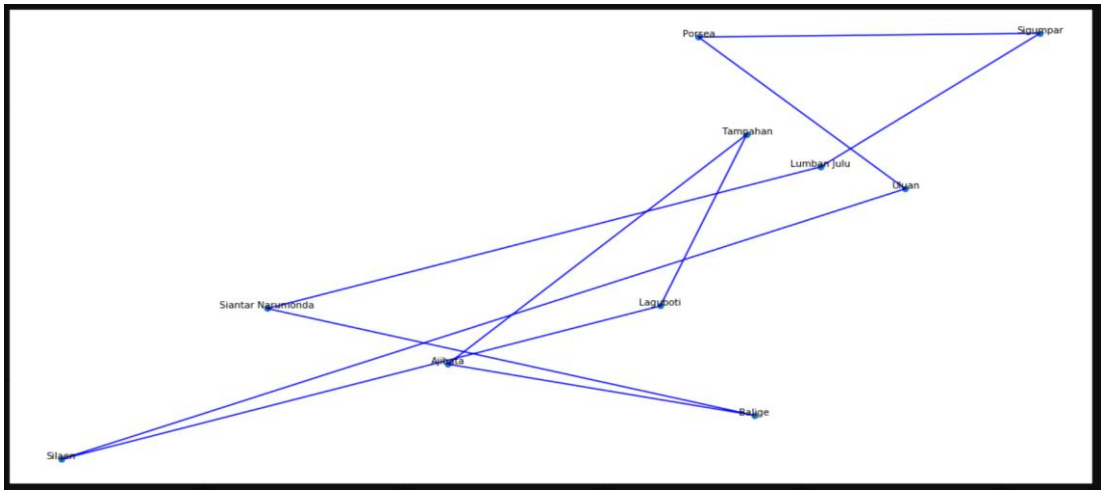
Gambar 7. Code Inisial Populasi

```
for idx, pop_plot in enumerate (population):
    print('Initial Population '+ str(idx),pop_plot)
```

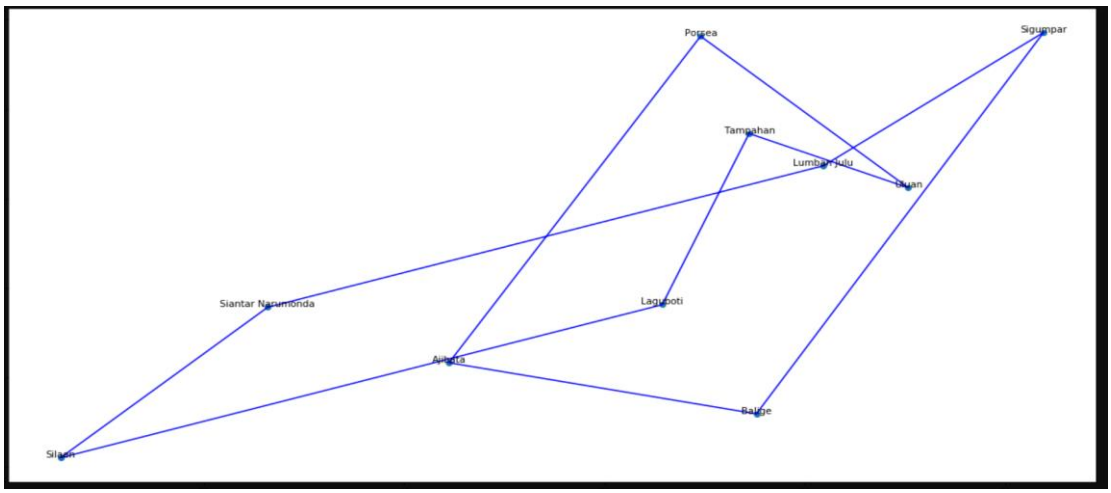
```
Initial Population 0 [[72.88261, 19.07283], [42.856255, 10.516726], [85.158875, 25.612677], [74.797371, 34.083656], [91.91283
2, 34.293363], [80.9231262, 26.8392792], [53.19384, 18.92832], [77.580643, 12.972442], [62.239233, 15.82732], [77.216721, 28.
6448]]
Initial Population 1 [[77.580643, 12.972442], [62.239233, 15.82732], [74.797371, 34.083656], [85.158875, 25.612677], [77.2167
21, 28.6448], [72.88261, 19.07283], [42.856255, 10.516726], [53.19384, 18.92832], [80.9231262, 26.8392792], [91.912832, 34.29
3363]]
Initial Population 2 [[85.158875, 25.612677], [53.19384, 18.92832], [72.88261, 19.07283], [42.856255, 10.516726], [77.580643,
12.972442], [74.797371, 34.083656], [91.912832, 34.293363], [80.9231262, 26.8392792], [62.239233, 15.82732], [77.216721, 28.6
448]]
Initial Population 3 [[42.856255, 10.516726], [77.580643, 12.972442], [62.239233, 15.82732], [53.19384, 18.92832], [85.15887
5, 25.612677], [91.912832, 34.293363], [80.9231262, 26.8392792], [72.88261, 19.07283], [77.216721, 28.6448], [74.797371, 34.0
83656]]
Initial Population 4 [[53.19384, 18.92832], [80.9231262, 26.8392792], [91.912832, 34.293363], [77.216721, 28.6448], [62.23923
3, 15.82732], [72.88261, 19.07283], [85.158875, 25.612677], [77.580643, 12.972442], [42.856255, 10.516726], [74.797371, 34.08
3656]]
Initial Population 5 [[72.88261, 19.07283], [80.9231262, 26.8392792], [77.580643, 12.972442], [74.797371, 34.083656], [53.193
84, 18.92832], [85.158875, 25.612677], [77.216721, 28.6448], [42.856255, 10.516726], [62.239233, 15.82732], [91.912832, 34.29
3363]]
Initial Population 6 [[53.19384, 18.92832], [62.239233, 15.82732], [42.856255, 10.516726], [74.797371, 34.083656], [91.91283
2, 34.293363], [72.88261, 19.07283], [77.216721, 28.6448], [80.9231262, 26.8392792], [85.158875, 25.612677], [77.580643, 12.9
72442]]
Initial Population 7 [[53.19384, 18.92832], [77.216721, 28.6448], [77.580643, 12.972442], [80.9231262, 26.8392792], [85.15887
5, 25.612677], [74.797371, 34.083656], [62.239233, 15.82732], [91.912832, 34.293363], [72.88261, 19.07283], [42.856255, 10.51
6726]]
Initial Population 8 [[77.580643, 12.972442], [77.216721, 28.6448], [72.88261, 19.07283], [62.239233, 15.82732], [91.912832,
34.293363], [42.856255, 10.516726], [85.158875, 25.612677], [74.797371, 34.083656], [53.19384, 18.92832], [80.9231262, 26.839
2792]]
Initial Population 9 [[91.912832, 34.293363], [72.88261, 19.07283], [80.9231262, 26.8392792], [53.19384, 18.92832], [42.85625
5, 10.516726], [74.797371, 34.083656], [77.580643, 12.972442], [85.158875, 25.612677], [62.239233, 15.82732], [77.216721, 28.
6448]]
Initial Population 10 [[77.580643, 12.972442], [42.856255, 10.516726], [91.912832, 34.293363], [62.239233, 15.82732], [77.216
721, 28.6448], [53.19384, 18.92832], [72.88261, 19.07283], [74.797371, 34.083656], [85.158875, 25.612677], [80.9231262, 26.83
92792]]
Initial Population 11 [[77.580643, 12.972442], [62.239233, 15.82732], [53.19384, 18.92832], [72.88261, 19.07283], [80.923126
2, 26.8392792], [85.158875, 25.612677], [91.912832, 34.293363], [74.797371, 34.083656], [42.856255, 10.516726], [77.216721, 2
8.6448]]
Initial Population 12 [[91.912832, 34.293363], [62.239233, 15.82732], [72.88261, 19.07283], [80.9231262, 26.8392792], [74.797
371, 34.083656], [85.158875, 25.612677], [77.216721, 28.6448], [42.856255, 10.516726], [53.19384, 18.92832], [77.580643, 12.9
72442]]
Initial Population 13 [[77.580643, 12.972442], [53.19384, 18.92832], [72.88261, 19.07283], [74.797371, 34.083656], [77.21672
1, 28.6448], [91.912832, 34.293363], [85.158875, 25.612677], [62.239233, 15.82732], [80.9231262, 26.8392792], [42.856255, 10.
516726]]
Initial Population 14 [[80.9231262, 26.8392792], [74.797371, 34.083656], [85.158875, 25.612677], [62.239233, 15.82732], [77.2
16721, 28.6448], [72.88261, 19.07283], [91.912832, 34.293363], [77.580643, 12.972442], [53.19384, 18.92832], [42.856255, 10.5
16726]]
Initial Population 15 [[42.856255, 10.516726], [72.88261, 19.07283], [74.797371, 34.083656], [85.158875, 25.612677], [77.5806
43, 12.972442], [77.216721, 28.6448], [53.19384, 18.92832], [91.912832, 34.293363], [80.9231262, 26.8392792], [62.239233, 15.
82732]]
Initial Population 16 [[85.158875, 25.612677], [74.797371, 34.083656], [77.216721, 28.6448], [72.88261, 19.07283], [53.19384,
18.92832], [62.239233, 15.82732], [91.912832, 34.293363], [77.580643, 12.972442], [42.856255, 10.516726], [80.9231262, 26.839
2792]]
Initial Population 17 [[42.856255, 10.516726], [74.797371, 34.083656], [72.88261, 19.07283], [62.239233, 15.82732], [53.1938
4, 18.92832], [91.912832, 34.293363], [85.158875, 25.612677], [77.216721, 28.6448], [77.580643, 12.972442], [80.9231262, 26.8
392792]]
Initial Population 18 [[77.216721, 28.6448], [91.912832, 34.293363], [74.797371, 34.083656], [77.580643, 12.972442], [80.9231
262, 26.8392792], [42.856255, 10.516726], [85.158875, 25.612677], [53.19384, 18.92832], [72.88261, 19.07283], [62.239233, 15.
82732]]
Initial Population 19 [[72.88261, 19.07283], [77.216721, 28.6448], [77.580643, 12.972442], [91.912832, 34.293363], [53.19384,
18.92832], [85.158875, 25.612677], [42.856255, 10.516726], [62.239233, 15.82732], [80.9231262, 26.8392792], [74.797371, 34.08
3656]]
```

Gambar 8. Inisial populasi

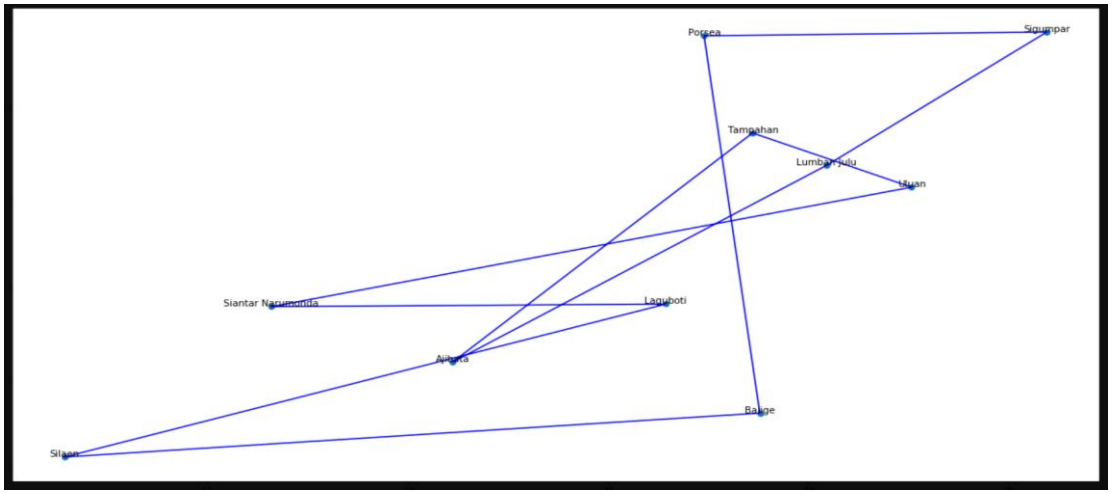
Berikut merupakan iterasi untuk setiap populasi untuk mendapatkan *path* yang optimum.



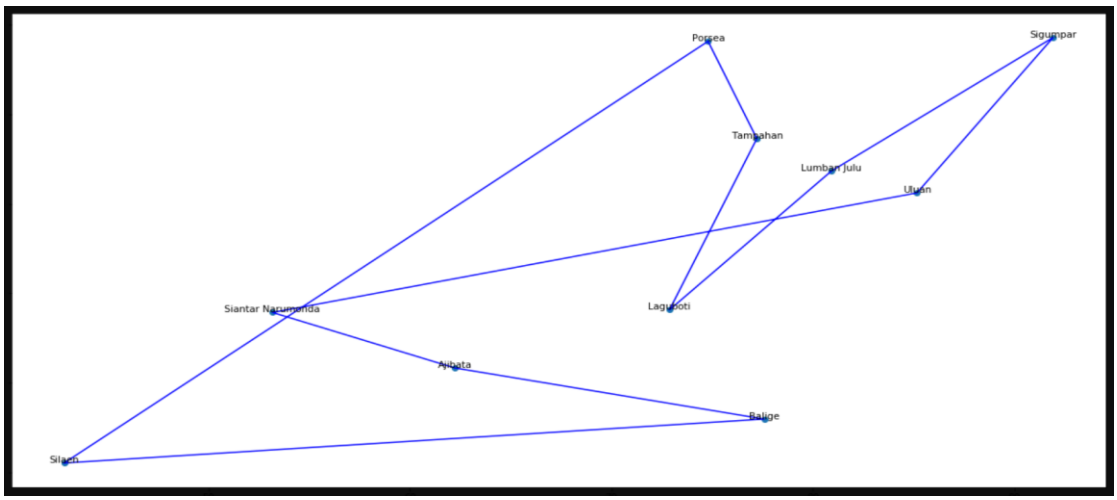
Gambar 9. Iterasi 1



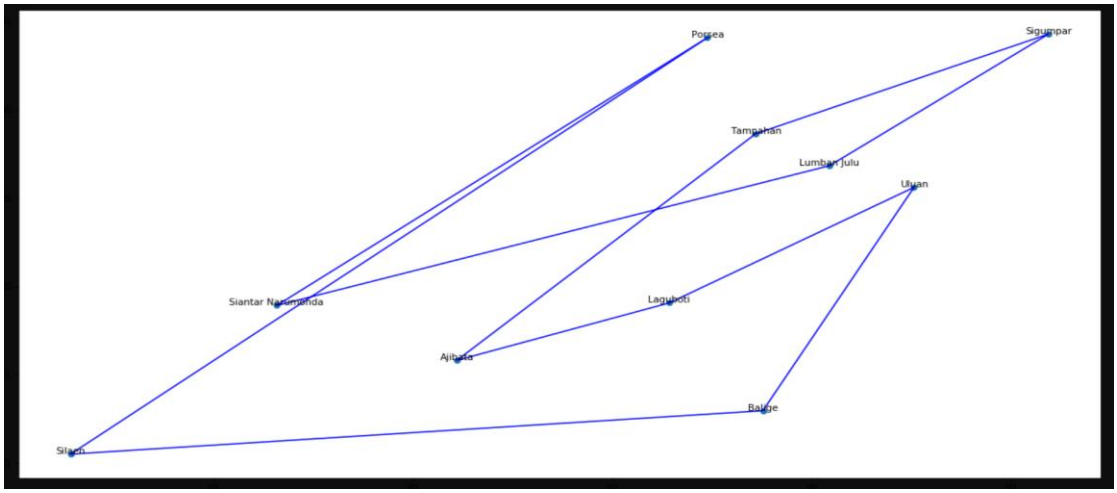
Gambar 10. Iterasi 2



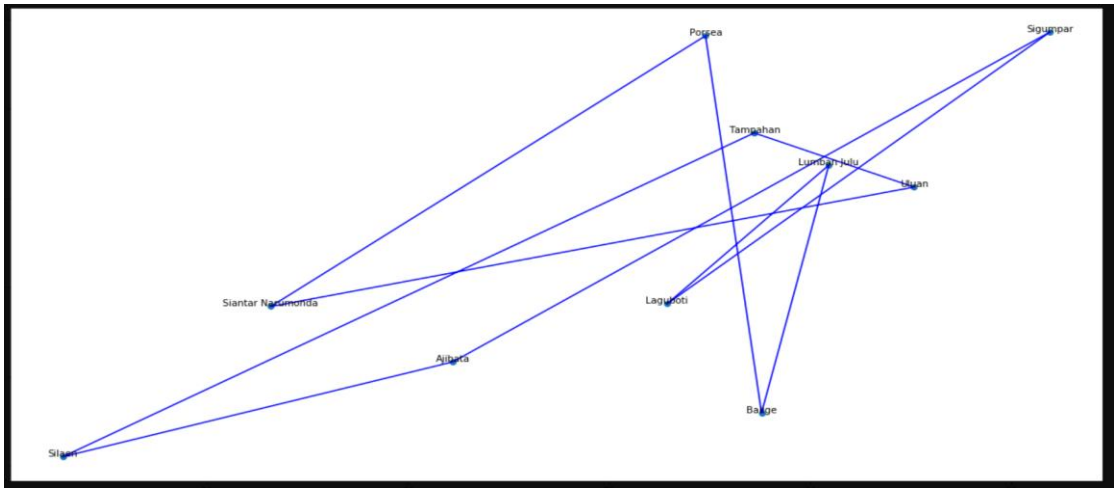
Gambar 11. Iterasi 3



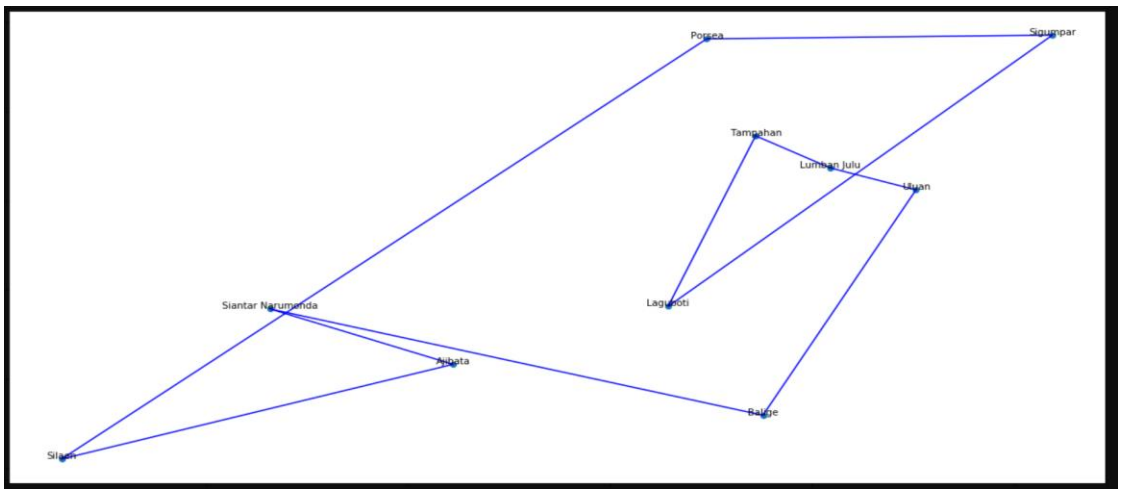
Gambar 12. Iterasi 4



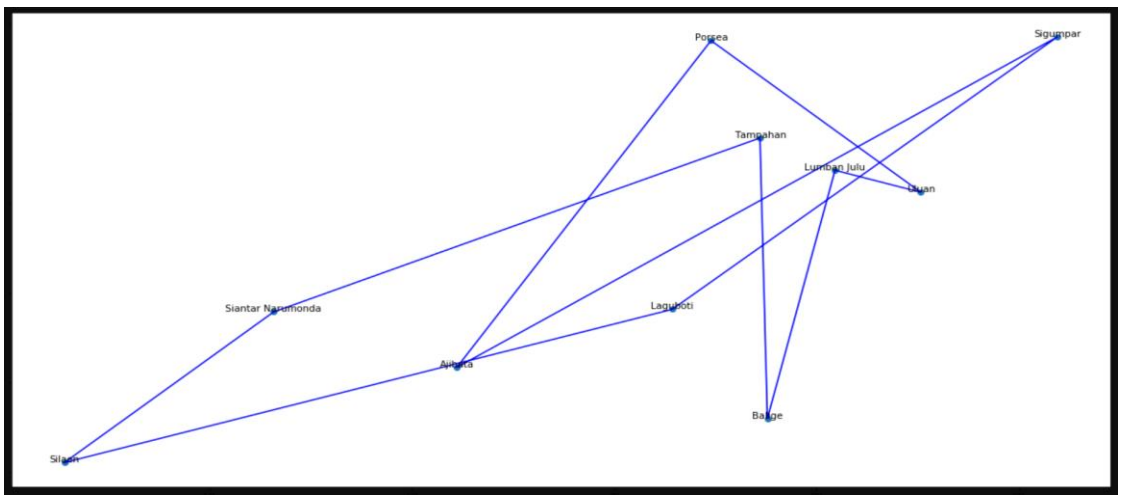
Gambar 13. Iterasi 5



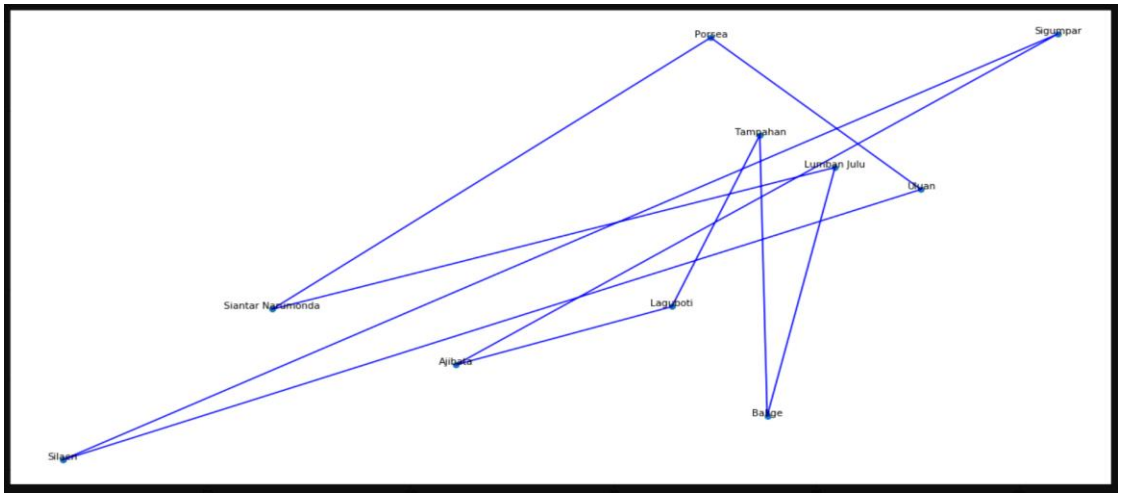
Gambar 14. Iterasi 6



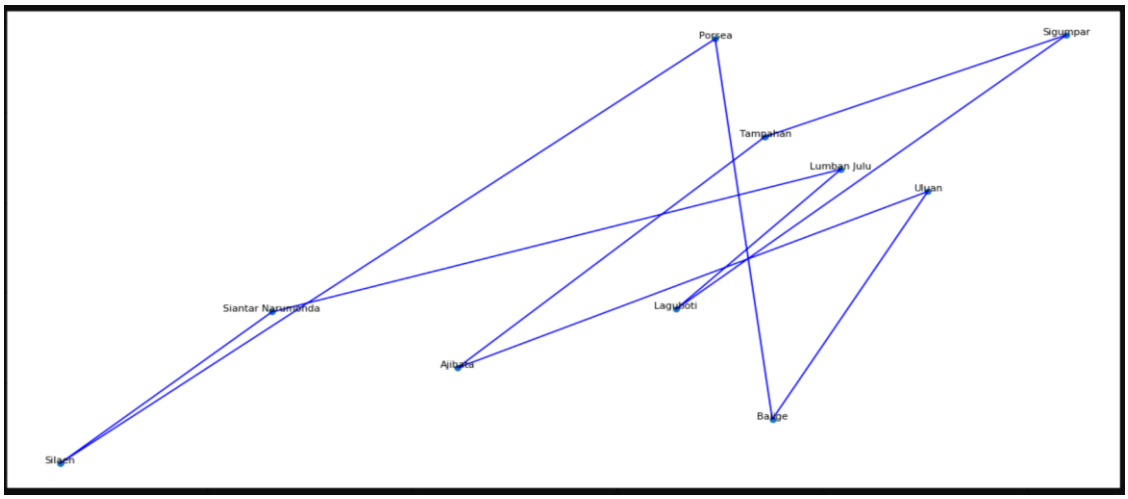
Gambar 15. Iterasi 7



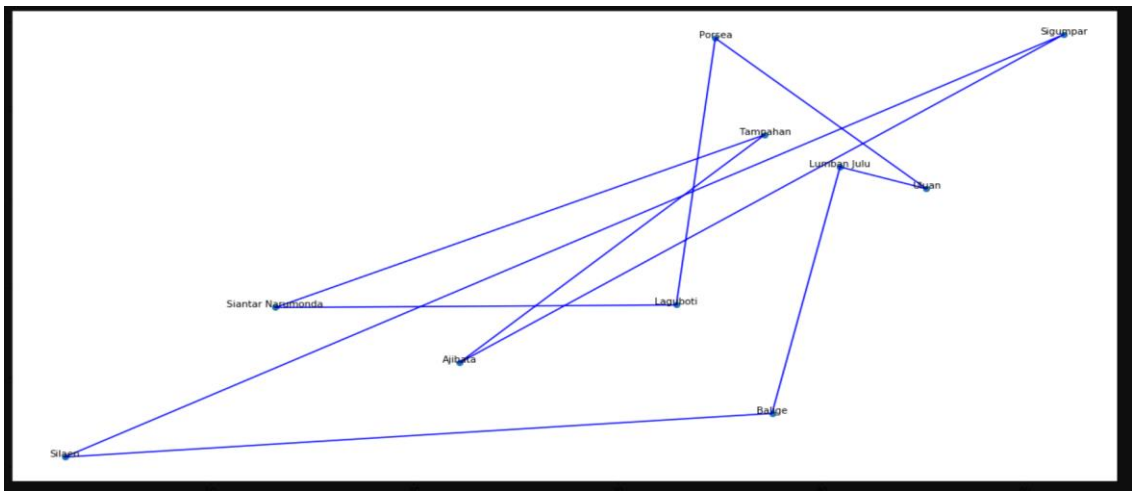
Gambar 16. Iterasi 8



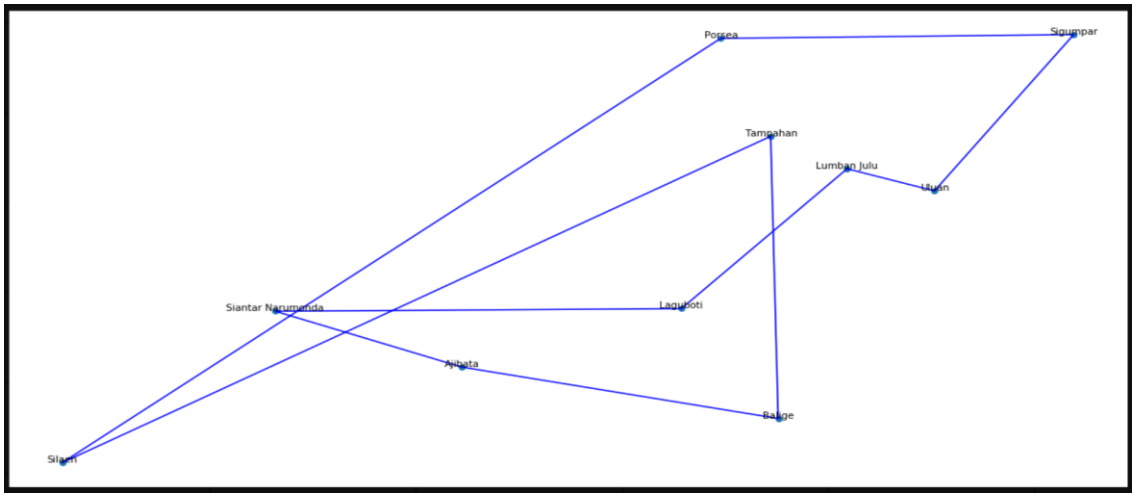
Gambar 17. Iterasi 9



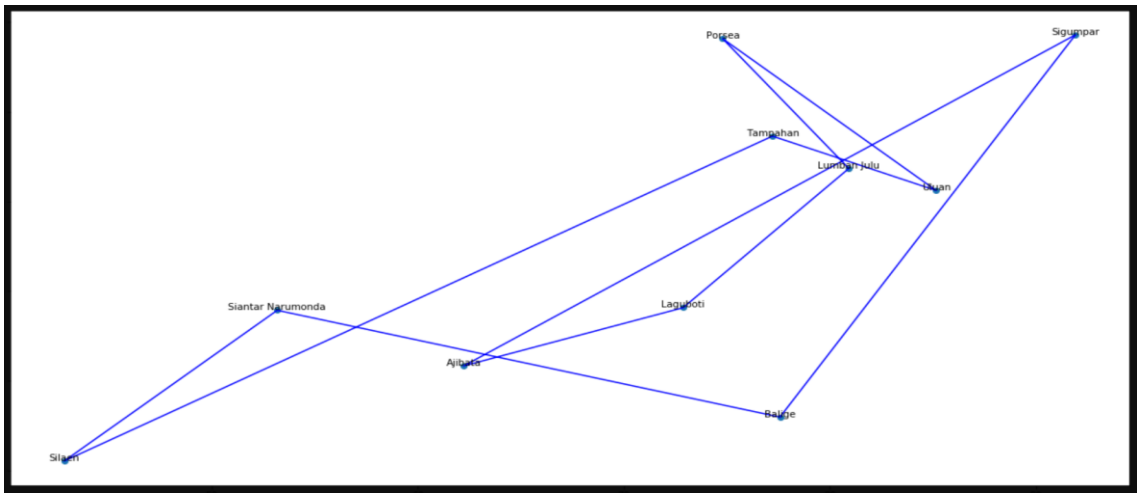
Gambar 18. Iterasi 10



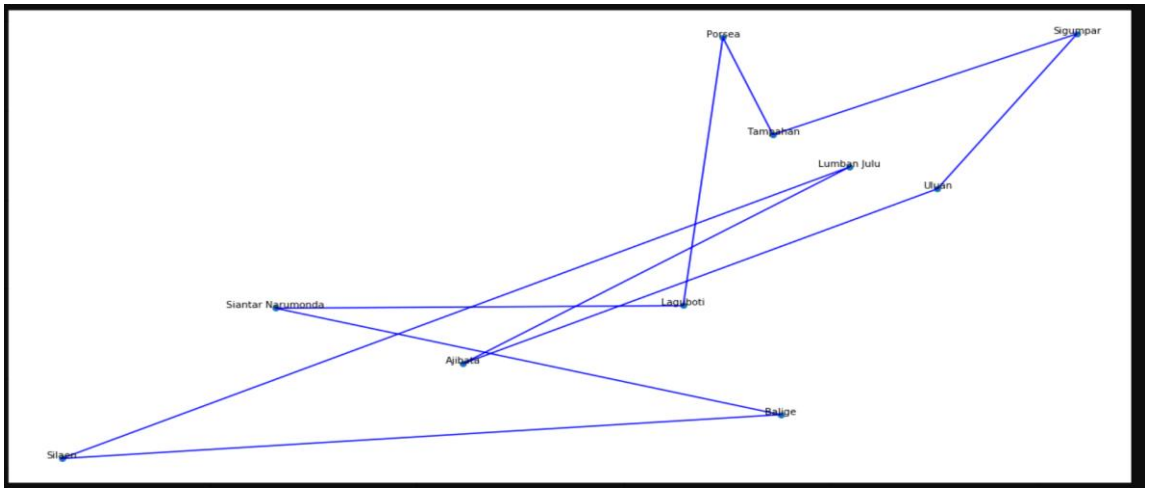
Gambar 19. Iterasi 11



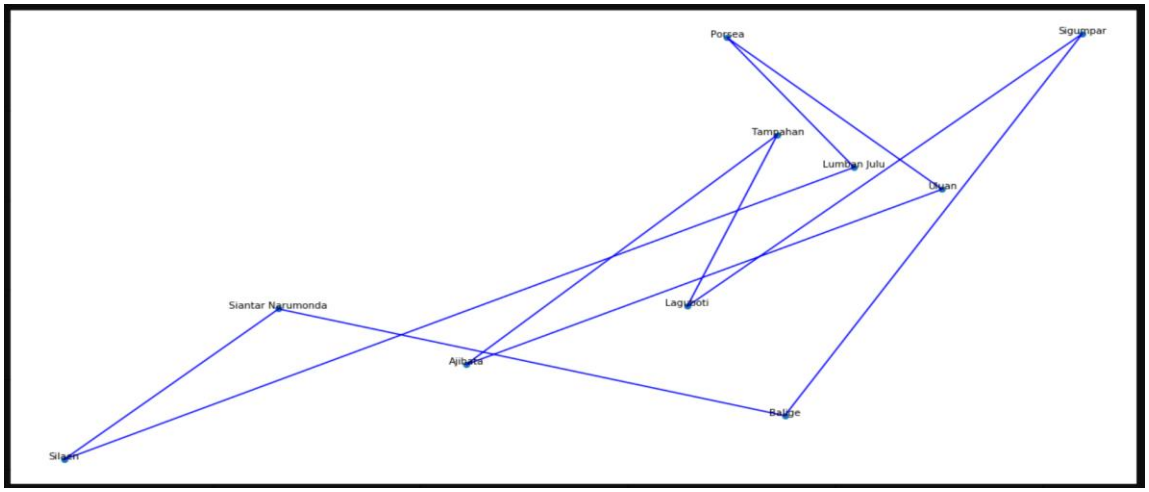
Gambar 20. Iterasi 12



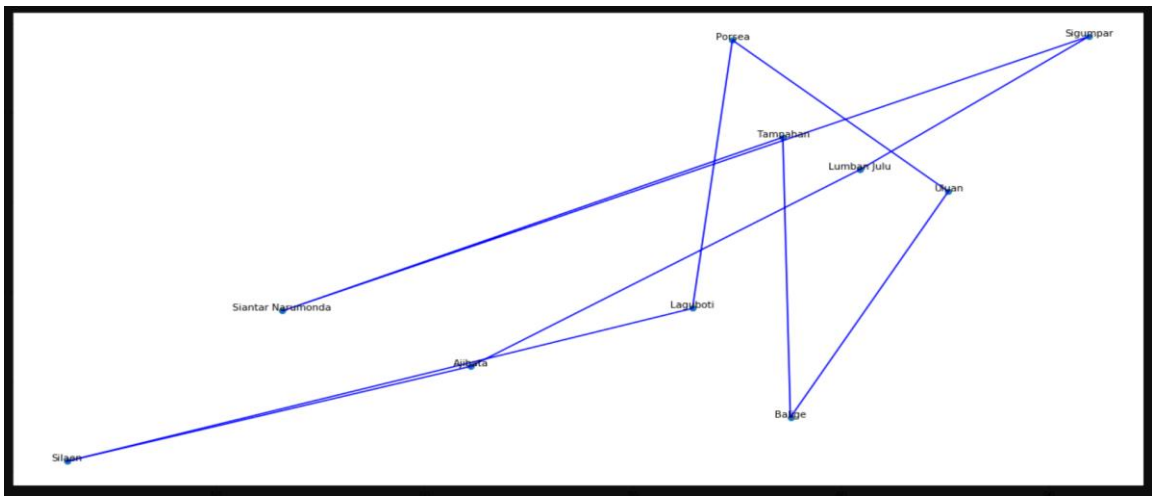
Gambar 21. Iterasi 13



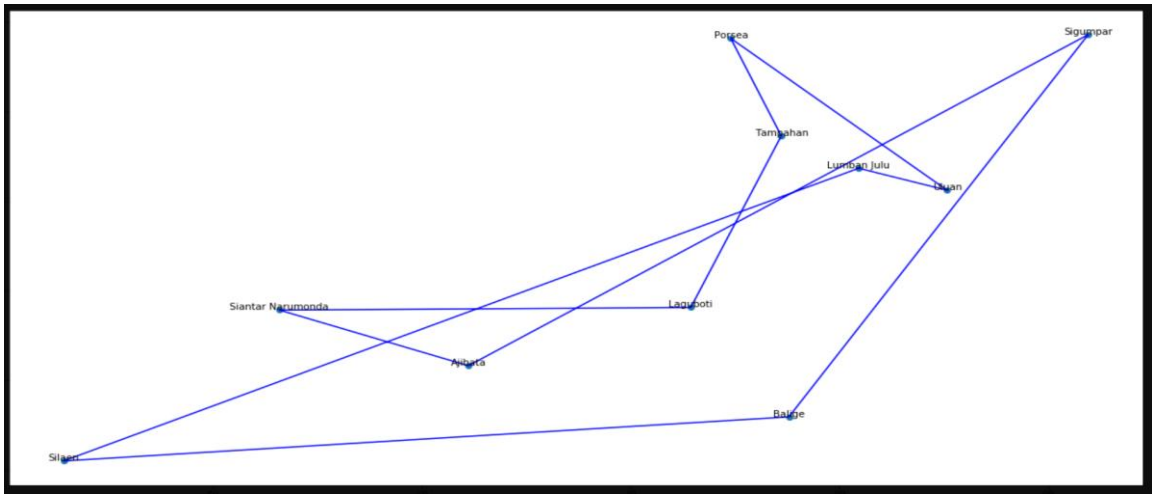
Gambar 22. Iterasi 14



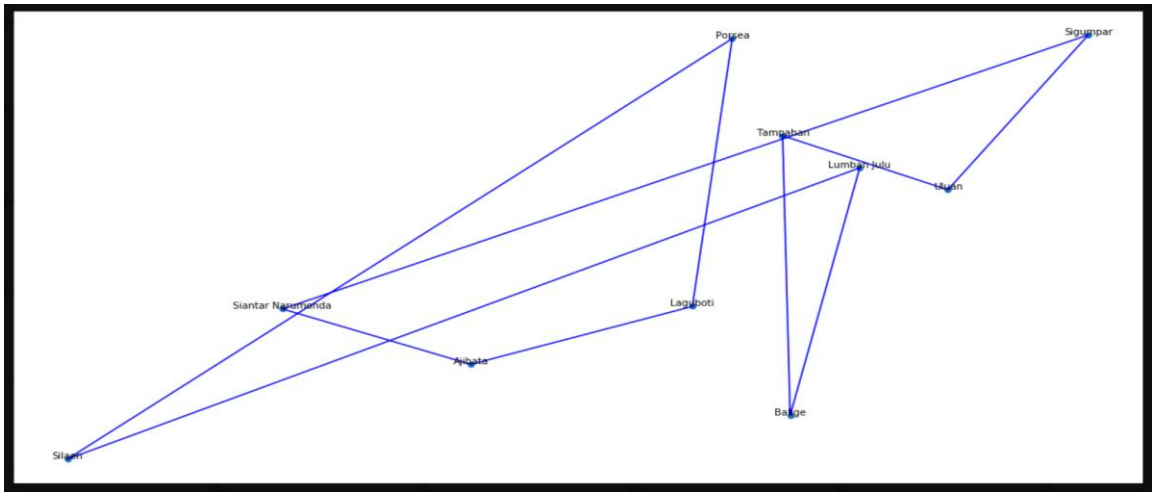
Gambar 23. Iterasi 15



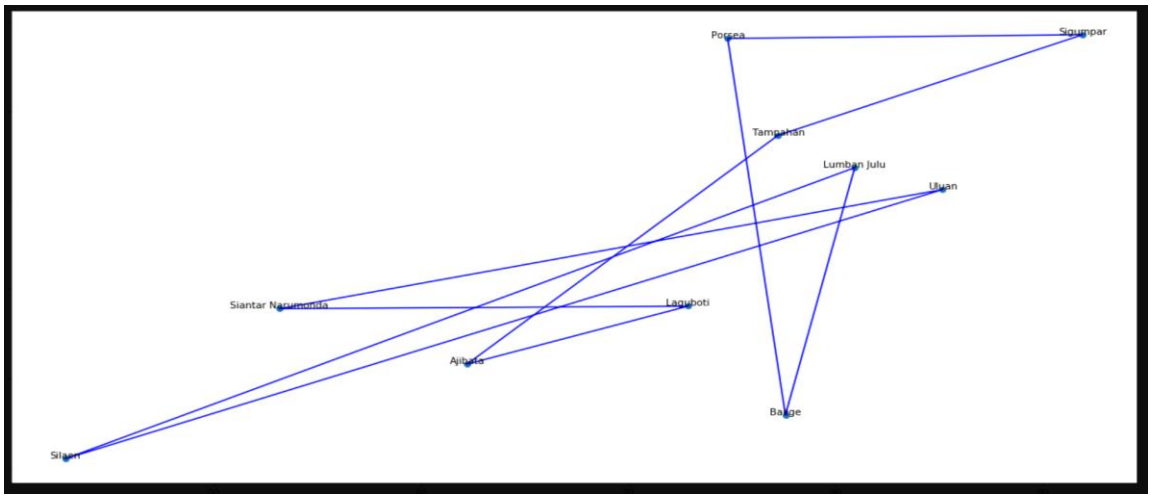
Gambar 24. Iterasi 16



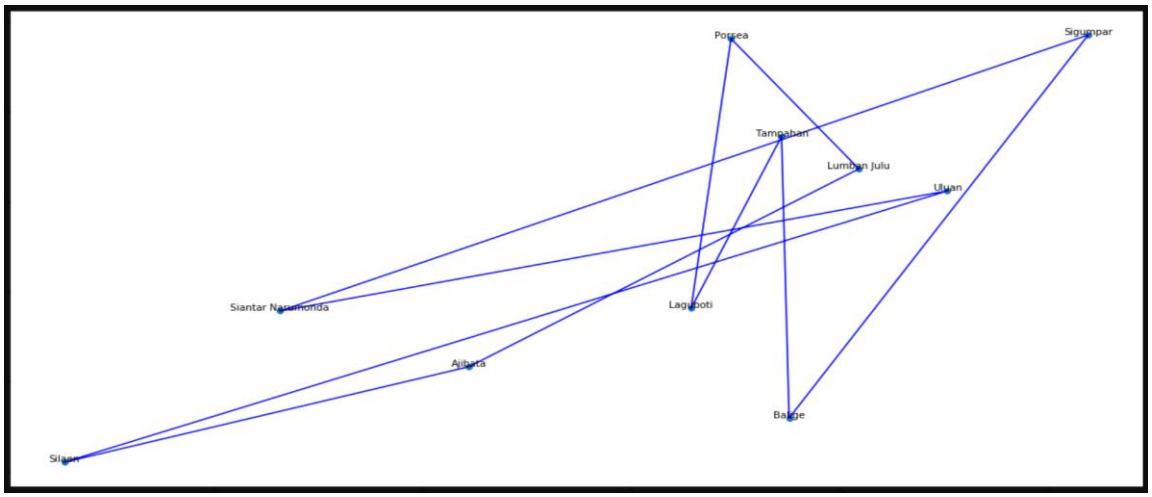
Gambar 25. Iterasi 17



Gambar 26. Iterasi 18



Gambar 27. Iterasi 19



Gambar 28. Iterasi 20

3.1.4. Deskripsi & Format Keluaran

Pada rancangan yang dibuat akan ditampilkan sebuah sistem informasi geografis penentuan rute terdekat pada jasa pengiriman barang yang berada diwilayah pencarian. Implementasi penerapan Algoritma GA pada sistem, terletak pada proses pencarian rute terdekat pada jasa pengiriman barang di wilayah pencarian. Sistem ini menggunakan titik-titik persimpangan jalan besar wilayah pencarian untuk melakukan perhitungan pencarian rute terdekat menggunakan algortima GA. Setiap titik-titik yang ditampilkan pada *map* merupakan data koordinat yang akan dihitung dengan menggunakan algoritma GA dimana koordinat tersebut sudah ditentukan sebelumnya.

Adapun proses pertama kali yang dilakukan dalam menentukan rute optimal yaitu menentukan titik posisi yang akan dikunjungi.. Di mana titik awal tersebut akan diambil berdasarkan koordinat dari gps *smartphone Salesman*. Kemudian algoritma GA akan bekerja mencari rute dengan akumulasi bobot terendah, setelah itu dijumlahkan dengan biaya perkiraan jalan yang memiliki bobot terendah, rute yang memiliki titik tujuan dengan nilai yang paling kecil akan dikembangkan terlebih dahulu. Kemudian apabila *node* yang menjadi titik tujuan telah ditemukan, maka program akan melakukan *backtrack* ke *parent* dari tiap *node* untuk mendapatkan rangkaian *node* yang membentuk rute yang paling optimal.

Setelah input diproses dengan menggunakan algoritma ini, maka *output* yang dihasilkan dalam bentuk *graph.* Program akan menampilkan rute optimal yang dapat dilalui sebanyak sekali oleh *Salesman.* Berikut adalah potongan *code* untuk menampilkan *output :*

```python
def GA(city_names,cities, population_size, eliteSize, mutat_rate, generations):
    population = initialPopulation(cities,population_size)
    #print(population_)
    print("Incipient distance: " + str(1 / rankPathes(population)[0][1]))
    for i in range(generations):
        population = get_following_gen(population, eliteSize, mutat_rate)
        #print(population)

    print("Eventual distance: " + str(1 / rankPathes(population)[0][1]))
    optimal_route_id = rankPathes(population)[0][0]
    optimal_route = population[optimal_route_id]
    ordered_cities = get_names(optimal_route,cities,city_names)
    print([(indx,val) for indx,val in enumerate(ordered_cities)])
    plot_pop(optimal_route)
    return optimal_route

result_lst = GA(city_names,cityList, population_size=100,
                eliteSize=5, mutat_rate=0.01,
                generations=500)
```

```
Incipient distance: 146.24922254078797
Eventual distance: 127.73224066652291
[(0, 'Siantar Narumonda'), (1, 'Silaen'), (2, 'Ajibata'), (3, 'Laguboti'), (4, 'Balige'), (5, 'Uluan'), (6, 'Sigumpar'), (7, 'L
umban Julu'), (8, 'Tampahan'), (9, 'Porsea')]
```

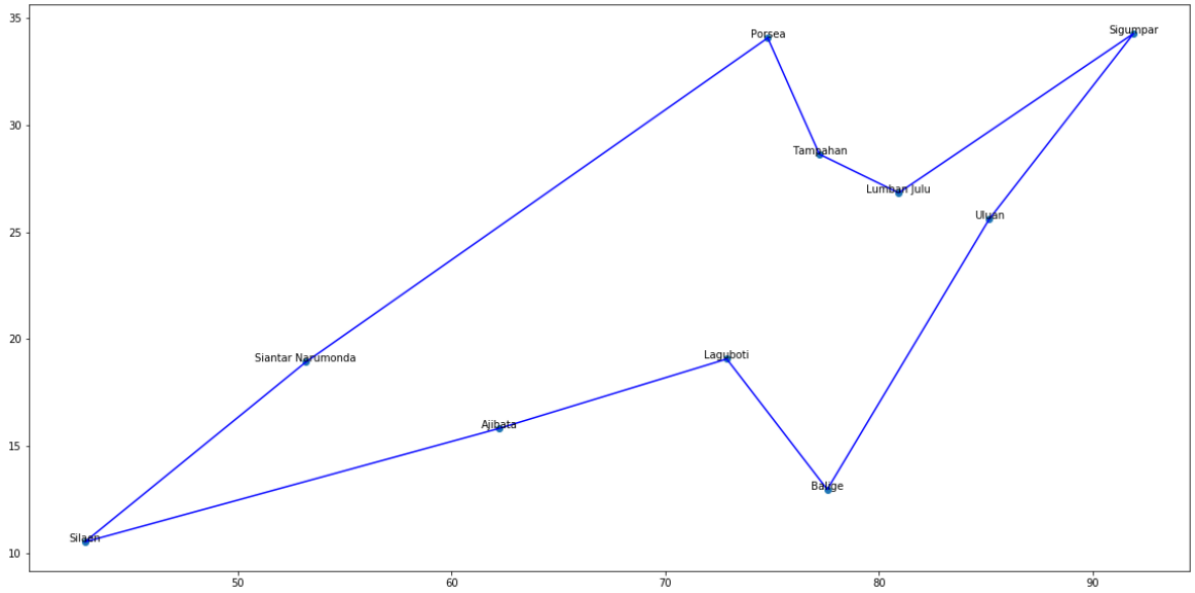
Gambar 29. Kode program menampilkan *output*

4. Hasil dan Pembahasan

Bagian ini berisi hasil proyek dan pembahasan hasil proyek.

4.1.Hasil

Berikut adalah keluaran yang diperoleh dari program diatas :



Gambar 30. Keluaran

Pada tampilan tersebut, kami menggunakan sistem ini untuk mencari rute yang paling optimum untuk menuju semua lokasi yang telah di masukan, dimana pada grafik ini yang kami jadikan sebagai titik sampel adalah Laguboti, Balige, Muara, Silaen, Tarutung , Siborongborong, Dan Porsea, dengan jarak masing-masing yang telah diinput juga. Sistem ini aknan mulai mengeksekusi dan mencari jarak yang paling optimum untuk dilalui *Salesman* , agar *salesman* dapat menjangkau semua tempat itu tepat waktu dan memastikan bahwa tempat yang

dikunjungi tidak lebih dari sekali. Maka dapat dilihat pada hasil yang telah ditampilkan, GA memberikan hasil pencarian rute yang paling optimum.

4.2.Pembahasan

Dengan hasil pencarian rute yang diberikan, dapat dikatakan bahwa algoritma GA memiliki keakuratan yang baik dalam mencari rute tercepat yang akan dilaui *salesman.* Dengan adanya sistem ini, akan semakin mudah bagi *salesman* untuk mencari rute tercepat dalam pengiriman barang. Dengan memasukkan titik awal dan titik tujuan, sistem akan memberikan hasil keluaran yang menunjukkan jalan atau rute yang optimal yang dapat memudahkan *Salesman* dalam menempuh perjalanan ke berbagai lokasi yang telah diinput.

5. Kesimpulan dan Saran

5.1.Kesimpulan

Setelah dilakukan implementasi GA pada Travel Salesman Problem , maka adapat diambil kesimpulan sebgai berikut:

a. Proses pembangunan rute dilakukan dengann melakukaknnya kedalam kasus TSP, sehingga jalan rute langsung tertuju pada posisi lokasi yang akan dikunjungi.
b. Berdasarkan hasil pengujian, dipastikan bahwa rute yang yang telah diperoleh adalah yang paling optimal.
c. Penerapan GA sangat membantu dalam mencari solusi rute optimal pada kegiatan apapun yang berkaitan dengan TSP.

5.2.Saran

Adapun saran untuk penelitian selanjutnya tentang penerapan algoritma genetika pada pemodelan kasus TSP adalah sebagai berikut:

a. Penelitian yang dilakukan hanya sebatas mencari rute terpendek. Akan lebih baik apabila dalam membangun rute juga memperhatikan kapasitas angkut kendaraan, estimasi bahan bakar kendaraan, dan waktu operasional yang diperbolehkan.
b. Penginputan data uji masih dilakukan secara manual, yakni dengan membuat file .tsp terlebih dahulu. Diharapkan untuk pengembangan sistem selanjutnya, proses input data dapat dilakukan secara langsung, yakni di dalam sistem sudah terdapat peta untuk menentukan koordinat titik-titik lokasi.
c. Diharapkan setelah ini peneliti selanjutnya dapat menerapkan algoritma genetika dalam membangun suatu sistem yang digunakan untuk pengelolaan rute kendaraan atau untuk kasus yang identik.

6. Referensi

[1] Amri, Faisal., Nababan, Erna Budhiarti., dan Syahputra, Mohammad Fadly. 2012. Artificial Bee Colony Algorithm untuk Menyelesaikan Travelling Salesman Problem. Jurnal Dunia Teknologi Informasi, Vol. 1, No. 1, pp. 8-13.

[2] Basuki, Achmad. 2003a. Algoritma Genetika: Suatu Alternatif Penyelesaian Permasalahan Searching, Optimasi, dan Machine Learning. http://budi.blog.undip.ac.id/files/2009/06/algoritma_ge netika.pdf. Diakses Juli 2014.

[3] https://pdfs.semanticscholar.org/e4fa/805704e093fec79bfc7a5ad00c61f53c79ae.pdf

[4] Manggolo, Inu., Marzuki, Marza Ihsan., dan Alaydrus, Mudrik. 2011. Optimalisasi Perencanaan Jaringan Akses Serat Optik Fiber To The Home Menggunakan Algoritma Genetika. InComTech, Jurnal Telekomunikasi dan Komputer, Vol. 2, No. 2, pp. 21- 36.

[5] Rosa, A.S. dan Shalahuddin, M. 2011. Modul Pembelajaran Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek). Bandung: Modula. [11] Suarga. 2012. Algoritma dan Pemrograman. Yogyakarta: Penerbit ANDI.

[6] Yusuf, Akhmad dan Soesanto, Oni. 2012. Algoritma Genetika pada Penyelesaian Akar Persamaan Sebuah Fungsi. Jurnal Matematika Murni dan Terapan, Vol. 6, No. 2, pp. 47-56

[7] Carwoto. 2007. Implementasi Algoritma Genetika untuk Optimasi Penempatan Kapasitor Shunt pada Penyulang Distribusi Tenaga Listrik. Jurnal Teknologi Informasi DINAMIK, Vol. 12, No. 2, pp. 122-130.

[8] Deep, Kusum dan Mebrahtu, Hadush. 2011. Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman Problem. International Journal of Combinatorial Optimization Problems and Informatics, Vol. 2, No. 3, pp. 1-23.

[9] Dewi, Kania Evita. 2012. Perbandingan Metode Newton-Raphson dan Algoritma Genetik pada Penentuan Implied Volatility Saham. Jurnal Ilmiah Komputer dan Informatika (KOMPUTA), Vol. 1, No. 2, pp. 9-16.

[10] Fox, Roland dan Tseng, Steven. 1997. Traveling Salesman Problem by Genetic Algorithm and Simulated Annealing. http://www.cs.nthu.edu.tw/~jang /courses/cs5611/project/2/. Diakses Mei 2014