PENYELESAIAN TRAVELLING SALESMAN PROBLEM MENGGUNAKAN GENETIC ALGORITHM PADA KASUS PENGIRIMAN BARANG PADA SALON DI KOTA MAKASSAR

KELOMPOK 2

ANGGOTA KELOMPOK:

ARVIN AZMI SAVA	(5026211097)
NIKOLAUS VICO CRISTIANTO	(5026211107)
I GUSTI AGUNG JAWA HISWARA	(5026211122)

KELAS

KOMPUTASI LUNAK (A)

DEPARTEMEN SISTEM INFORMASI

FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SEMESTER GENAP 2024

LATAR BELAKANG

Indonesia merupakan negara yang memiliki banyak pulau, provinsi dan kota, dan salah satu kota terbesar di Indonesia adalah kota Makassar yang berada di Sulawesi Selatan. Dalam kota yang besar akan terdapat banyak hal-hal yang membantu manusia untuk melakukan pekerjaan sehari-hari, salah satunya adalah toko rias atau salon. Terdapat banyak sekali salon di kota Makassar, dan hal ini menimbulkan suatu permasalahan dan peluang bisnis. Butuhnya salon akan bahan baku kecantikan atau perawatan rambut, menjadi suatu potensi bisnis yang dapat dilakukan oleh para pelaku bisnis retail di dunia kecantikan/perawatan rambut tersebut. Mereka akan dengan senang hati untuk melakukan transaksi dengan salon-salon ini, namun efisiensi tetap menjadi kunci mereka dalam bertransaksi.

Traveling Salesman Problem (TSP) merupakan sebuah permasalah optimasi yang dapat diterapkan pada berbagai kegiatan seperti pendistribusian barang, pengambilan tagihan listrik dan pedagang keliling. Masalah optimasi pada TSP sangat terkenal dan telah menjadi standar untuk mencoba algoritma yang komputasional. Pokok dari permasalahan TSP adalah bagaimana seorang salesman harus dapat mengunjungi sejumlah kota atau lokasi yang telah diketahui jarak lokasi satu dengan yang lainnya. (Erdiwansyah, 2017)

Salah satu cara untuk menyelesaikan TSP ini adalah dengan menggunakan Genetic Algorithm (GA). Genetic Algorithm (GA) adalah bagian dari Evolutionary Algorithm yaitu suatu algoritma yang mencontoh proses evolusi alami dimana konsep utamanya adalah individu-individu yang paling unggul akan bertahan hidup, sedangkan individu-individu yang lemah akan punah (Engelbrecht). Genetic Algorithm ini merupakan salah satu cara yang umum digunakan dalam penyelesaian TSP ini

Kajian Teori

Adapun kajian teor sederhanai yang kami gunakan dalam mengerjakan problem ini adalah sebagai berikut

Traveling Salesman Problems

Traveling Salesman Problem (TSP) merupakan sebuah permasalah optimasi yang dapat diterapkan pada berbagai kegiatan seperti pendistribusian barang, pengambilan tagihan listrik dan pedagang keliling. Masalah optimasi pada TSP sangat terkenal dan telah menjadi standar untuk mencoba algoritma yang komputasional. Pokok dari permasalahan TSP adalah bagaimana seorang salesman harus dapat mengunjungi sejumlah kota atau lokasi yang telah diketahui jarak lokasi satu dengan yang lainnya.

TSP pertama kali diperkenalkan pada tahun 1930-an oleh Karl Menger seorang ahli matematika dan ekonomi. Menger menyebutnya sebagai “Messenger Problem” yakni masalah yang dihadapi oleh pengirim surat dan banyak pengelana. TSP ini merupakan masalah yang sangat kompleks, dikarenakan jika kita memiliki banyak sekali tujuan, maka hal ini akan membuat adanya banyak sekali kemungkinan rute yang dapat ditempuh, belum lagi rute yang ditempuh harus disaring agar menghasilkan cost atau target variabel yang diinginkan. Pada kasus ini target yang diinginkan adalah Minimasi Jarak, jadi TSP yang harus diselesaikan adalah terdapat berbagai lokasi dimana kami harus mencari rute yang menghasilkan jarak terkecil

Genetic Algorithm

Genetic Algorithm (GA) adalah bagian dari Evolutionary Algorithm yaitu suatu algoritma yang mencontoh proses evolusi alami dimana konsep utamanya adalah individu-individu yang paling unggul akan bertahan hidup, sedangkan individu-individu yang lemah akan punah[1]. Keunggulan individu-individu ini diuji melalui suatu fungsi yang dikenal sebagai fitness function. Fitness dalam GA didefinisikan sebagai gambaran kelayakan suatu solusi terhadap suatu permasalahan. Fitness Function akan menghasilkan suatu nilai fitness value yang akan menjadi referensi untuk proses GA selanjutnya. (Binus, 2018)

Proses GA dimulai dengan menentukan populasi awal initial population yang terdiri dari beberapa kromosom yang disusun oleh beberapa gen yang merupakan representasi dari kandidat-kandidat solusi dari suatu masalah. Kandidat-kandidat terbaik akan dipilih melalui proses selection, berdasarkan fitness value yang telah dihitung untuk setiap kromosom dalam populasi. Kandidat – kandidat terpilih dari proses ini adalah individu-individu yang akan mengisi mating pool yaitu suatu set dimana dua parents akan dibentuk dari sini. Dalam Evolutionary Algorithm prinsip bertahan muncul karena adanya proses reproduksi. Turunan offspring yang dihasilkan akan membawa sifat gen orangtuanya (parents) , oleh sebab itu parents dipilih dari mating pool yang merupakan kumpulan kandidat-kandidat terbaik dari suatu populasi. Dengan demikian turunan yang dihasilkan adalah turunan yang memiliki sifat unggul dari kedua orang tuanya.

Overview Studi Kasus

Pada tugas ETS kali ini, kami mengambil studi kasus sebuah perusahaan penyedia bahan baku untuk salon kecantikan. Perusahaan ini memiliki beberapa cabang *warehouse* di Indonesia. Agar biaya pengiriman menjadi lebih efektif dan efisien, perusahaan akan mengirim barang ke customer melalui cabang *warehouse* terdekatnya. Untuk kepentingan tugas ETS ini, kelompok kami mengambil cabang *warehouse* di Makassar. Warehouse ini memiliki beberapa customer,

tetapi untuk tugas ini kami hanya menggunakan data 20 *customer* saja. Di sini, kami seolah-olah ditugaskan untuk membantu perusahaan menentukan bagaimana rute pengiriman barang terdekat kepada 20 *customer* tersebut dengan aturan bahwa dalam satu kali proses pengiriman harus mengunjungi semua *customer* tersebut. Data masukan yang digunakan untuk menyelesaikan permasalahan tersebut adalah data *geo latitude* dan *geo longitude* yang merepresentasikan alamat *customer*. Geo Latitude (garis lintang) adalah jarak suatu lokasi dari garis khatulistiwa, diukur dalam derajat utara atau selatan dari khatulistiwa. Sedangkan, Geo Longitude (garis bujur) adalah jarak suatu lokasi dari garis nol bujur yang berada di Greenwich, Inggris, diukur dalam derajat timur atau barat dari Greenwich. Kombinasi dari *geo latitude* dan *longitude* memberikan koordinat geografis yang unik untuk setiap lokasi di bumi. Berikut ini merupakan tampilan data awal yang akan digunakan untuk tugas ini

Cust Code	Customer	City	Kecamatan	Kelurahan	Geo Latitude	Geo Longitude	Customer/Customer Category/Display Name
1026045	ASSYIFA SALON,MKS	Kota Makassar	0	0	'-5.16992	119.44766	Professional / Salon & Spa / Salon
1029398	INNER V SALON,MKS	Kota Makassar	0	0	'-5.18299	119.44767	Professional / Salon & Spa / Salon
1025029	MAHA SURYA TOKO	Kota Makassar	0	0	'-5.18717	119.45603	General Trade / Retailer / Provision
1038997	NONA COSMETIK	Kota Makassar	Tamalate	Parang Tan	'-5.19008	119.41641	General Trade / Retailer / Cosmetic Store
1029136	RAHRA SALON,MKS	Kota Makassar	0	0	'-5.1724	119.44828	Professional / Salon & Spa / Salon
1028585	RIAS TOKO,MKS	Kota Makassar	Mariso	Mario	'-5.15648	119.41471	General Trade / Retailer / Cosmetic Store
1029045	WATNIS SALON, TODDOPULI 6, MKS	Kota Makassar	0	0	'-5.16799	119.45341	Professional / Salon & Spa / Salon
1025122	YASMIN SALON,MKS	Kota Makassar	0	0	'-5.17053	119.44264	Professional / Salon & Spa / Salon
1029919	ADHEL SALON,MKS	Kota Makassar	0	0	'-5.17876	119.43459	Professional / Salon & Spa / Salon
1025477	ANI SALON,MKS	Kota Makassar	0	0	'-5.17487	119.40671	Professional / Salon & Spa / Salon
1025568	BARY 2 GOWA SALON,MKS	Kabupaten Gowa	0	0	'-5.17774	119.42831	Professional / Salon & Spa / Salon
1024981	BEAUTY 8 SALON	Kota Makassar	Tamalate	Bongaya	'-5.16838	119.41802	Professional / Salon & Spa / Salon
1025499	BEN ART MANURUKI SALON,MKS	Kota Makassar	Tamalate	Mannuruki	'-5.17758	119.42829	Professional / Salon & Spa / Salon
1031183	BULAN SALON GOWA,MKS	Gowa	0	0	'-5.20561	119.44873	Professional / Salon & Spa / Salon
1028843	EGHY SALON,MKS	Kota Makassar	Tamalate	Maccini Sor	'-5.17863	119.40675	Professional / Salon & Spa / Salon
1031561	ELHA SALON, MAMOA RAYA, MKS	Kota Makassar	0	0	'-5.18121	119.43237	Professional / Salon & Spa / Salon
1026193	HERMAN DG.TATA SALON,MKS	Kota Makassar	Tamalate	Parang Tan	'-5.18557	119.42406	Professional / Salon & Spa / Salon
1036038	HERMAN SALON MAPPAODDANG	Kota Makassar	Tamalate	Jongaya	'-5.17063	119.41714	Professional / Salon & Spa / Salon
1028531	IWAN KUMALA SALON,MKS	Kota Makassar	0	0	'-5.16852	119.41937	Professional / Salon & Spa / Salon
1025440	NADIA PURNAMA SALON,MKS	Kota Makassar	0	0	'-5.17323	119.41585	Professional / Salon & Spa / Salon

Pembahasan Penerapan Algoritma Genetika

1. Mendefinisikan data input dan parameter

   Pada langkah ini, kami mendefinisikan data input berupa data koordinat sejumlah dua data input yaitu data geo longitude dan geo latitude. Data input tersebut merepresentasikan alamat customer pada koordinat geografis. Kemudian, akan ditentukan parameter-parameter algoritma genetika seperti ukuran populasi, persentase crossover, persentase mutasi, dan jumlah generasi. Berikut ini parameter yang digunakan untuk menyelesaikan permasalahan ini:

   - Ukuran populasi 200

- Probabilitas crossover 0,8
- Probabilitas mutasi 0,2
- Generasi 700

Data

Cities Coordinates and Algorithm Parameters

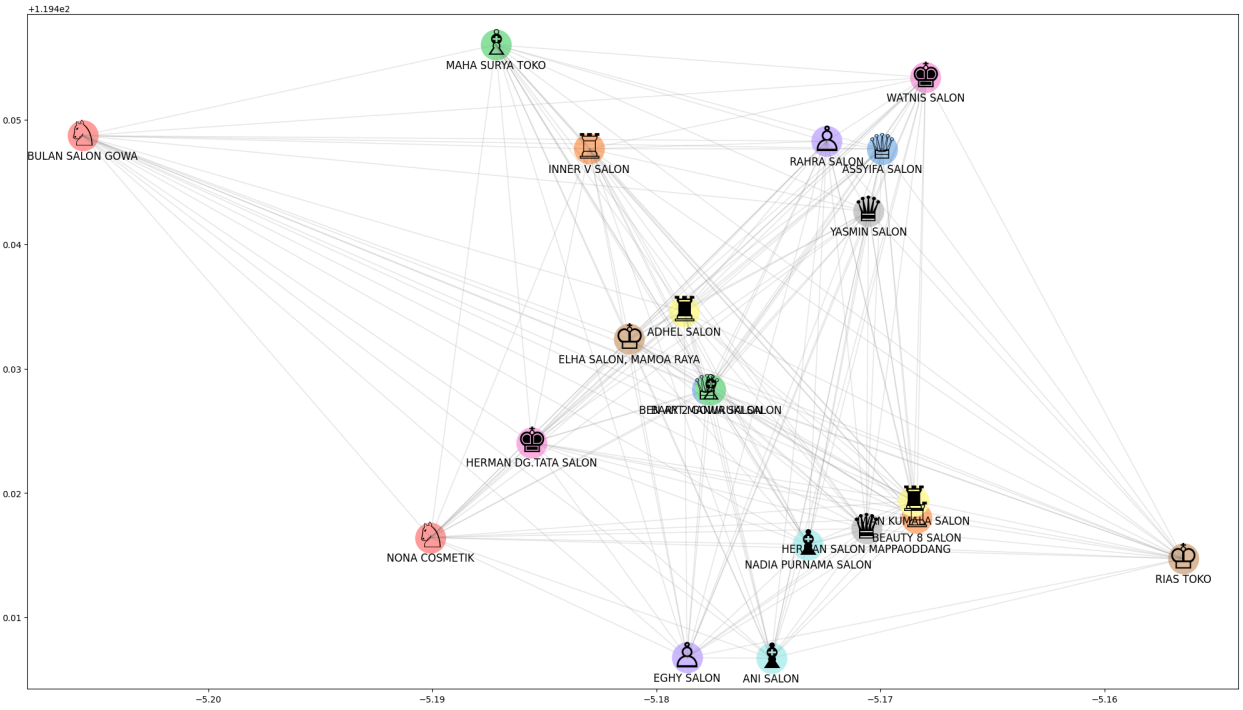
```
x = [-5.16992,-5.18299,-5.18717,-5.19008,-5.1724,-5.15648,-5.16799,-5.17053,-5.17876,-5.17487,-5.17774,-5.16838,-5.17758,-5.20561,-5.17863,-5.18121,-5.18557,-5.17063, -5.16852,-5.
y = [119.44766,119.44767,119.45603,119.41641,119.44828,119.41471,119.45341,119.44264,119.43459,119.40671,119.42831,119.41802,119.42829,119.44873,119.40675,119.43237,119.42406,119.
]
cities_names = ["ASSYIFA SALON", "INNER V SALON", "MAHA SURYA TOKO", "NONA COSMETIK", "RAHRA SALON", "RIAS TOKO", "WATNIS SALON", "YASMIN SALON", "ADHEL SALON", "ANI SALON","BARY
city_coords = dict(zip(cities_names, zip(x, y)))
n_population = 200
crossover_per = 0.8
mutation_per = 0.2
n_generations = 700
```

2. Memvisualisasikan Alamat Customer

Langkah berikutnya, kami mencoba untuk memvisualisasikan data dari alamat customer ke dalam *scatterplot*. Tujuan dari visualisasi tahap awal ini akan membantu dalam memahami letak relatif antar customer satu dengan yang lain.

Visualising Cities

```
# Pastel Pallete
colors = sns.color_palette("pastel", len(cities_names))

# City Icons
city_icons = {
    "ASSYIFA SALON": "🏛", "INNER V SALON": "🏛","MAHA SURYA TOKO": "🏛", "NONA COSMETIK": "🏛", "RAHRA SALON": "🏛",
    "RIAS TOKO": "🏛", "WATNIS SALON": "🏛", "YASMIN SALON": "🏛", "ADHEL SALON": "🏛","ANI SALON": "🏛", "BARY 2 GOWA SALON": "🏛" ,"BEAUTY 8 SALON": "🏛",
    "BEN ART MANURUKI SALON": "🏛", "BULAN SALON GOWA": "🏛","EGHY SALON": "🏛","ELHA SALON, MAMOA RAYA": "🏛","HERMAN DG.TATA SALON": "🏛",
    "HERMAN SALON MAPPAODDANG": "🏛","IWAN KUMALA SALON": "🏛","NADIA PURNAMA SALON": "🏛",
}

fig, ax = plt.subplots()

ax.grid(False)  # Grid

for i, (city, (city_x, city_y)) in enumerate(city_coords.items()):
    color = colors[i]
    icon = city_icons[city]
    ax.scatter(city_x, city_y, c=[color], s=1200, zorder=1)
    ax.annotate(icon, (city_x, city_y), fontsize=30, ha='center', va='center', zorder=3)
    ax.annotate(city, (city_x, city_y), fontsize=12, ha='center', va='bottom', xytext=(0, -30),
                textcoords='offset points')

    # Connect cities with opaque lines
    for j, (other_city, (other_x, other_y)) in enumerate(city_coords.items()):
        if i != j:
            ax.plot([city_x, other_x], [city_y, other_y], color='gray', linestyle='-', linewidth=1, alpha=0.1)

fig.set_size_inches(25, 14)
plt.show()
```



3. Inisialisasi Populasi Awal

Selanjutnya, akan dibuat fungsi menggunakan np.random.permutation dari NumPy yang bertujuan untuk menghasilkan populasi awal. Populasi awal ini akan digunakan sebagai titik awal dalam algoritma genetika untuk menemukan solusi terbaik. Populasi awal ini dibentuk dengan cara memilih secara acak urutan customer dari semua kemungkinan permutasi alamat customer yang diberikan. Variabel "cities_names"merepresentasikan list yang berisi nama-nama customer yang akan dijadikan sebagai populasi awal. Sedangkan, variabel "n_population" adalah jumlah individu (kemungkinan jalur) dalam populasi.

Population Function (Random Initialization)

```python
def initial_population(cities_names, n_population):
    """
    Generating initial population of cities randomly selected from all possible permutations
    of the given cities.

    Args:
    - cities_names (list): List of city names
    - n_population (int): Number of individuals in the population

    Returns:
    - population_perms (list): List of initial population of cities
    """
    cities_list = list(cities_names)
    population_perms = []

    for _ in range(n_population):
        perm = list(np.random.permutation(cities_list))
        population_perms.append(perm)

    return population_perms
```

4. Menghitung jarak antar dua customer

Pada tahap ini, kami menghitung jarak (dalam kilometer) antara dua titik koordinat geografis (latitude dan longitude) menggunakan rumus Haversine. Rumus ini berguna untuk menghitung jarak antara dua titik pada permukaan bulat seperti bumi. Fungsi ini menerima empat parameter: lat1 dan lon1 adalah koordinat latitude dan longitude dari titik pertama, sedangkan lat2 dan lon2 adalah koordinat latitude dan longitude dari titik kedua.

Distance between two cities

```python
def haversine_distance(lat1, lon1, lat2, lon2):
    # Convert latitude and longitude from degrees to radians
    lat1_rad = np.radians(lat1)
    lon1_rad = np.radians(lon1)
    lat2_rad = np.radians(lat2)
    lon2_rad = np.radians(lon2)

    # Haversine formula
    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad
    a = np.sin(dlat / 2)**2 + np.cos(lat1_rad) * np.cos(lat2_rad) * np.sin(dlon / 2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance = 6371 * c  # Radius of the Earth in kilometers
    return distance
```

```python
[29]
def dist_two_cities(city1, city2):
    # Get coordinates of the two cities
    lat1, lon1 = city_coords[city1]
    lat2, lon2 = city_coords[city2]

    # Calculate the distance between the two cities
    distance = haversine_distance(lat1, lon1, lat2, lon2)
    return distance

```

Lalu, kami membuat sebuah matriks jarak antara setiap pasang customer. Matriks ini akan digunakan sebagai masukan dalam algoritma genetika untuk menyelesaikan *Traveling Salesman Problem* (TSP) dengan mencari rute terpendek yang melintasi setiap kota tepat sekali dan kembali ke kota awal.

```python
# Create a matrix to store distances
distance_matrix = [[0 for _ in range(len(x))] for _ in range(len(x))]

# Calculate distances between each pair of coordinates
for i in range(len(x)):
    for j in range(len(x)):
        lat1, lon1 = city_coords[cities_names[i]]
        lat2, lon2 = city_coords[cities_names[j]]
        distance_matrix[i][j] = haversine_distance(lat1, lon1, lat2, lon2)

# Convert distances in the matrix to kilometers
for i in range(len(x)):
    for j in range(len(x)):
        distance_matrix[i][j] = round(distance_matrix[i][j], 2)  # Round to 2 decimal places

# Print the distance matrix
for i in range(len(x)):
    for j in range(len(x)):
        print(f"Distance from {cities_names[i]} to {cities_names[j]}:", distance_matrix[i][j], "km")
```

```
Distance from ASSYIFA SALON to ASSYIFA SALON: 0.0 km
Distance from ASSYIFA SALON to INNER V SALON: 1.45 km
Distance from ASSYIFA SALON to MAHA SURYA TOKO: 2.13 km
Distance from ASSYIFA SALON to NONA COSMETIK: 4.12 km
Distance from ASSYIFA SALON to RAHRA SALON: 0.28 km
Distance from ASSYIFA SALON to RIAS TOKO: 3.94 km
Distance from ASSYIFA SALON to WATNIS SALON: 0.67 km
Distance from ASSYIFA SALON to YASMIN SALON: 0.56 km
Distance from ASSYIFA SALON to ADHEL SALON: 1.75 km
Distance from ASSYIFA SALON to ANI SALON: 4.57 km
Distance from ASSYIFA SALON to BARY 2 GOWA SALON: 2.31 km
Distance from ASSYIFA SALON to BEAUTY 8 SALON: 3.29 km
Distance from ASSYIFA SALON to BEN ART MANURUKI SALON: 2.31 km
```

5. Menghitung Total Jarak yang ditempuh Individu

Langkah ini digunakan untuk menghitung total jarak yang ditempuh oleh seorang individu dalam populasi. Seorang individu direpresentasikan sebagai daftar yang berisi urutan customer yang akan dikunjungi. Fungsi ini melakukan perulangan melalui daftar customer dalam individu dan mengakumulasi jarak antara setiap pasang customer yang dikunjungi, termasuk jarak dari customer terakhir kembali ke customer awal. Jumlah total jarak ini kemudian dikembalikan sebagai output dari fungsi. Total jarak ini digunakan sebagai nilai fitness dalam algoritma genetika untuk mengevaluasi seberapa baik suatu solusi dalam populasi. Solusi dengan total jarak terpendek dianggap sebagai solusi yang optimal dalam *Traveling Salesman Problem* (TSP).

Total distance traveled by individual

```python
def total_dist_individual(individual):

    """
    Calculating the total distance traveled by individual,
    one individual means one possible solution (1 permutation)
    Input:
    1- Individual list of cities
    Output:
    Total distance traveled
    """

    total_dist = 0
    for i in range(0, len(individual)):
        if(i == len(individual) - 1):
            total_dist += dist_two_cities(individual[i], individual[0])
        else:
            total_dist += dist_two_cities(individual[i], individual[i+1])
    return total_dist
```

6. Menghitung *Fitness Probability*

Langkah selanjutnya yaitu menghitung *fitness probability* dari setiap individu dalam populasi. Proses ini melibatkan perhitungan total jarak yang ditempuh oleh setiap individu dalam populasi. Fungsi ini mengubah total jarak yang ditempuh oleh individu dalam populasi menjadi sebuah probabilitas untuk menentukan individu mana yang akan dipilih sebagai *parent* untuk reproduksi.

Fitness probability function

```python
def fitness_prob(population):
    """
    Calculating the fitness probability
    Input:
    1- Population
    Output:
    Population fitness probability
    """
    total_dist_all_individuals = []
    for i in range(0, len(population)):
        total_dist_all_individuals.append(total_dist_individual(population[i]))

    max_population_cost = max(total_dist_all_individuals)
    population_fitness = max_population_cost - total_dist_all_individuals
    population_fitness_sum = sum(population_fitness)
    population_fitness_probs = population_fitness / population_fitness_sum
    return population_fitness_probs
```

7. Pemilihan Parent

Setelah menghitung *fitness probability* tiap individu. Selanjutnya akan dilakukan proses pemilihan parent dengan mengimplementasikan strategi seleksi menggunakan fungsi ‘roulette_wheel’. Fungsi ini menentukan individu mana yang akan dipilih sebagai orangtua untuk berkembang biak berikutnya dengan cara menghitung kumulatif jumlah

fitness probability untuk setiap individu, kemudian menggunakan nilai acak antara 0 dan 1 untuk menentukan posisi "roulette_wheel" yang menentukan individu yang dipilih. Hasilnya, fungsi ini memilih satu individu dari populasi sebagai output berdasarkan *fitness probability* mereka, dengan individu yang memiliki *fitness probability* lebih tinggi memiliki kemungkinan yang lebih besar untuk dipilih.

Roulette wheel

```
[33]  1 def roulette_wheel(population, fitness_probs):
      2     """
      3     Implement selection strategy based on roulette wheel proportionate selection.
      4     Input:
      5     1- population
      6     2- fitness probabilities
      7     Output:
      8     selected individual
      9     """
     10     population_fitness_probs_cumsum = fitness_probs.cumsum()
     11     bool_prob_array = population_fitness_probs_cumsum < np.random.uniform(0,1,1)
     12     selected_individual_index = len(bool_prob_array[bool_prob_array == True]) - 1
     13     return population[selected_individual_index]
```

8. Melakukan Crossover

Langkah berikutnya yaitu melakukan perkawinan silang (crossover) antar induk dengan menggunakan fungsi. Proses ini bertujuan untuk menghasilkan keturunan (offspring) yang menggabungkan informasi dari kedua induk. Langkah-langkahnya mencakup pemilihan titik pemotongan secara acak, pembuatan dua keturunan dengan membagi dan menggabungkan urutan customer dari kedua induk, dan mengembalikan keturunan sebagai output. Dengan menggunakan strategi ini, keturunan memiliki potensi untuk mewarisi sifat-sifat baik dari kedua induk, sehingga meningkatkan kemungkinan untuk menemukan solusi yang optimal dalam algoritma genetika.

Crossover

```
      1 def crossover(parent_1, parent_2):
      2     """
      3     Implement mating strategy using simple crossover between 2 parents
      4     Input:
      5     1- parent 1
      6     2- parent 2
      7     Output:
      8     1- offspring 1
      9     2- offspring 2
     10     """
     11     n_cities_cut = len(cities_names) - 1
     12     cut = round(random.uniform(1, n_cities_cut))
     13     offspring_1 = []
     14     offspring_2 = []
     15
     16     offspring_1 = parent_1[0:cut]
     17     offspring_1 += [city for city in parent_2 if city not in offspring_1]
     18
     19
     20     offspring_2 = parent_2[0:cut]
     21     offspring_2 += [city for city in parent_1 if city not in offspring_2]
     22
     23
     24     return offspring_1, offspring_2
     25
```

9. Mutasi

Selanjutnya yaitu terjadi proses mutasi pada satu individu keturunan (offspring). Pada proses ini dua indeks acak dipilih untuk merepresentasikan dua customer yang akan ditukar posisinya dalam urutan kunjungan customer individu. Proses mutasi dilakukan

dengan menukar posisi customer pada indeks yang dipilih sehingga menghasilkan individu keturunan yang telah dimutasi. Mutasi ini bertujuan untuk memperkenalkan variasi genetik ke dalam populasi.

Mutation

```
[35]  1 def mutation(offspring):
      2     """
      3     Implement mutation strategy in a single offspring
      4     Input:
      5     1- offspring individual
      6     Output:
      7     1- mutated offspring individual
      8     """
      9     n_cities_cut = len(cities_names) - 1
     10     index_1 = round(random.uniform(0,n_cities_cut))
     11     index_2 = round(random.uniform(0,n_cities_cut))
     12
     13     temp = offspring[index_1]
     14     offspring[index_1] = offspring[index_2]
     15     offspring[index_2] = temp
     16     return(offspring)
     17
```

10. Implementasi Algoritma Genetika

Proses dimulai dengan inisialisasi populasi awal. Kemudian, dilakukan seleksi orangtua (parents) berdasarkan *fitness probability* dengan metode *roulette wheel*. Setelah itu, dilakukan *crossover* antara induk dan mutasi untuk menghasilkan keturunan (offspring). Keturunan dan orangtua digabungkan menjadi satu populasi. Lalu, seleksi dilakukan kembali untuk memilih individu dengan *fitness probability* terbaik. Proses ini diulang sebanyak n_generations kali. Pada akhirnya, solusi terbaik yang ditemukan akan dihitung total jaraknya untuk evaluasi performa algoritma. Hasil akhir dari algoritma ini adalah solusi terbaik yang ditemukan mengenai urutan kunjungan *customer* yang memberikan total jarak tempuh terpendek.

Algorithm

```
def run_ga(cities_names, n_population, n_generations, crossover_per, mutation_per):

    population = initial_population(cities_names, n_population)
    fitness_probs = fitness_prob(population)

    parents_list = []
    for i in range(0, int(crossover_per * n_population)):
        parents_list.append(roulette_wheel(population, fitness_probs))

    offspring_list = []
    for i in range(0,len(parents_list), 2):
        offspring_1, offspring_2 = crossover(parents_list[i], parents_list[i+1])

    #     print(offspring_1)
    #     print(offspring_2)
    #     print()

        mutate_threashold = random.random()
        if(mutate_threashold > (1-mutation_per)):
            offspring_1 = mutation(offspring_1)
    #         print("Offspring 1 mutated", offspring_1)

        mutate_threashold = random.random()
        if(mutate_threashold > (1-mutation_per)):
            offspring_2 = mutation(offspring_2)
    #         print("Offspring 2 mutated", offspring_2)


        offspring_list.append(offspring_1)
        offspring_list.append(offspring_2)

    mixed_offspring = parents_list + offspring_list
```

```
    fitness_probs = fitness_prob(mixed_offspring)
    sorted_fitness_indices = np.argsort(fitness_probs)[::-1]
    best_fitness_indices = sorted_fitness_indices[0:n_population]
    best_mixed_offsrping = []
    for i in best_fitness_indices:
        best_mixed_offsrping.append(mixed_offspring[i])

    for i in range(0, n_generations):
        # if (i%10 == 0):
            # print("Generation: ", i)

        fitness_probs = fitness_prob(best_mixed_offsrping)
        parents_list = []
        for i in range(0, int(crossover_per * n_population)):
            parents_list.append(roulette_wheel(best_mixed_offsrping, fitness_probs))

        offspring_list = []
        for i in range(0,len(parents_list), 2):
            offspring_1, offspring_2 = crossover(parents_list[i], parents_list[i+1])

            mutate_threashold = random.random()
            if(mutate_threashold > (1-mutation_per)):
                offspring_1 = mutation(offspring_1)

            mutate_threashold = random.random()
            if(mutate_threashold > (1-mutation_per)):
                offspring_2 = mutation(offspring_2)

            offspring_list.append(offspring_1)
            offspring_list.append(offspring_2)

        mixed_offspring = parents_list + offspring_list
        fitness_probs = fitness_prob(mixed_offspring)
        sorted_fitness_indices = np.argsort(fitness_probs)[::-1]
        best_fitness_indices = sorted_fitness_indices[0:int(0.8*n_population)]

        best_mixed_offsrping = []
```

```
                offspring_1 = mutation(offspring_1)

            mutate_threashold = random.random()
            if(mutate_threashold > (1-mutation_per)):
                offspring_2 = mutation(offspring_2)

            offspring_list.append(offspring_1)
            offspring_list.append(offspring_2)

        mixed_offspring = parents_list + offspring_list
        fitness_probs = fitness_prob(mixed_offspring)
        sorted_fitness_indices = np.argsort(fitness_probs)[::-1]
        best_fitness_indices = sorted_fitness_indices[0:int(0.8*n_population)]

        best_mixed_offsrping = []
        for i in best_fitness_indices:
            best_mixed_offsrping.append(mixed_offspring[i])

        old_population_indices = [random.randint(0, (n_population - 1)) for j in range(int(0.2*n_population))]
        for i in old_population_indices:
#           print(i)
            best_mixed_offsrping.append(population[i])

        random.shuffle(best_mixed_offsrping)

    return best_mixed_offsrping
```

```
[15] best_mixed_offsrping = run_ga(cities_names, n_population, n_generations, crossover_per, mutation_per)
```

```
[16] total_dist_all_individuals = []
     for i in range(0, n_population):
         total_dist_all_individuals.append(total_dist_individual(best_mixed_offsrping[i]))
```

```
[17] index_minimum = np.argmin(total_dist_all_individuals)
```

11. Output Solusi Algoritma Genetika

Kami akan menggunakan 3 skenario parameter yaitu skenario Rendah, Sedang, dan Tinggi. Parameter yang kami ubah adalah n_populasi dan n_generasi, untuk mutation rate dan crossover rate tidak kami ubah karena nilai tersebut sudah merupakan best practice berdasarkan hasil penelitian. Untuk parameter n_populasi kami akan menggunakan nilai 40, 100, dan 200 sedangkan untuk parameter n_generasi kami akan menggunakan nilai 200, 450, dan 700. Kami tetap menggunakan nilai generasi lebih rendah daripada populasi karena hal itu juga sudah sesuai teori yang ada.

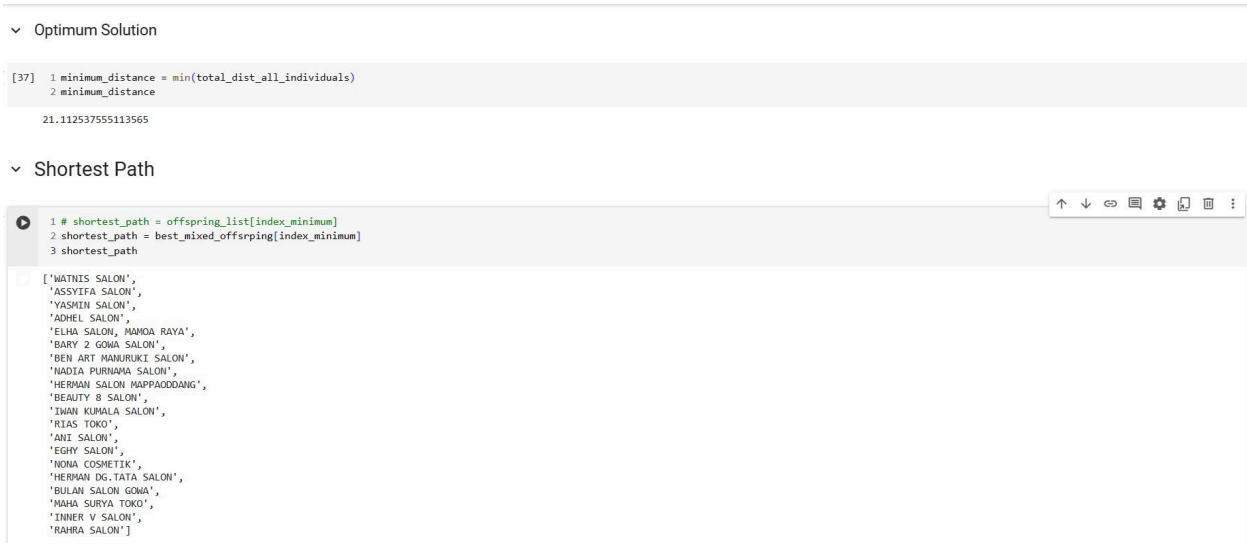
n_populasi	n_generasi	Cost/DIstance (KM)
40	200	28.724

40	450	25.43
40	700	27.689
100	200	23.56
100	450	23.928
100	700	23.464
200	200	22.163
200	450	22.679
200	700	21.11

Setelah menerapkan algoritma genetika, diperoleh sebuah solusi berupa jarak rute terdekat yang harus ditempuh untuk mengunjungi semua customer yaitu **21,11 km**. Selain itu, didapatkan juga urutan customer yang harus dikunjungi yang menghasilkan jarak terpendek tersebut.

Berikut ini daftar urutan customer yang harus dikunjungi untuk menghasilkan rute terpendek:

Rute Terpendek Nya: WATNIS SALON → ASSYIFA SALON → YASMIN SALON → ADHEL SALON → ELHA SALON, MAMOA RAYA → BARY 2 GOWA SALON → BEN ART MANURUKI SALON → NADIA PURNAMA SALON → HERMAN SALON MAPPAODDANG → BEAUTY 8 SALON → IWAN KUMALA SALON → RIAS TOKO → ANI SALON → EGHY SALON → NONA COSMETIK → HERMAN DG.TATA SALON → BULAN SALON GOWA → MAHA SURYA TOKO → INNER V SALON → RAHRA SALON



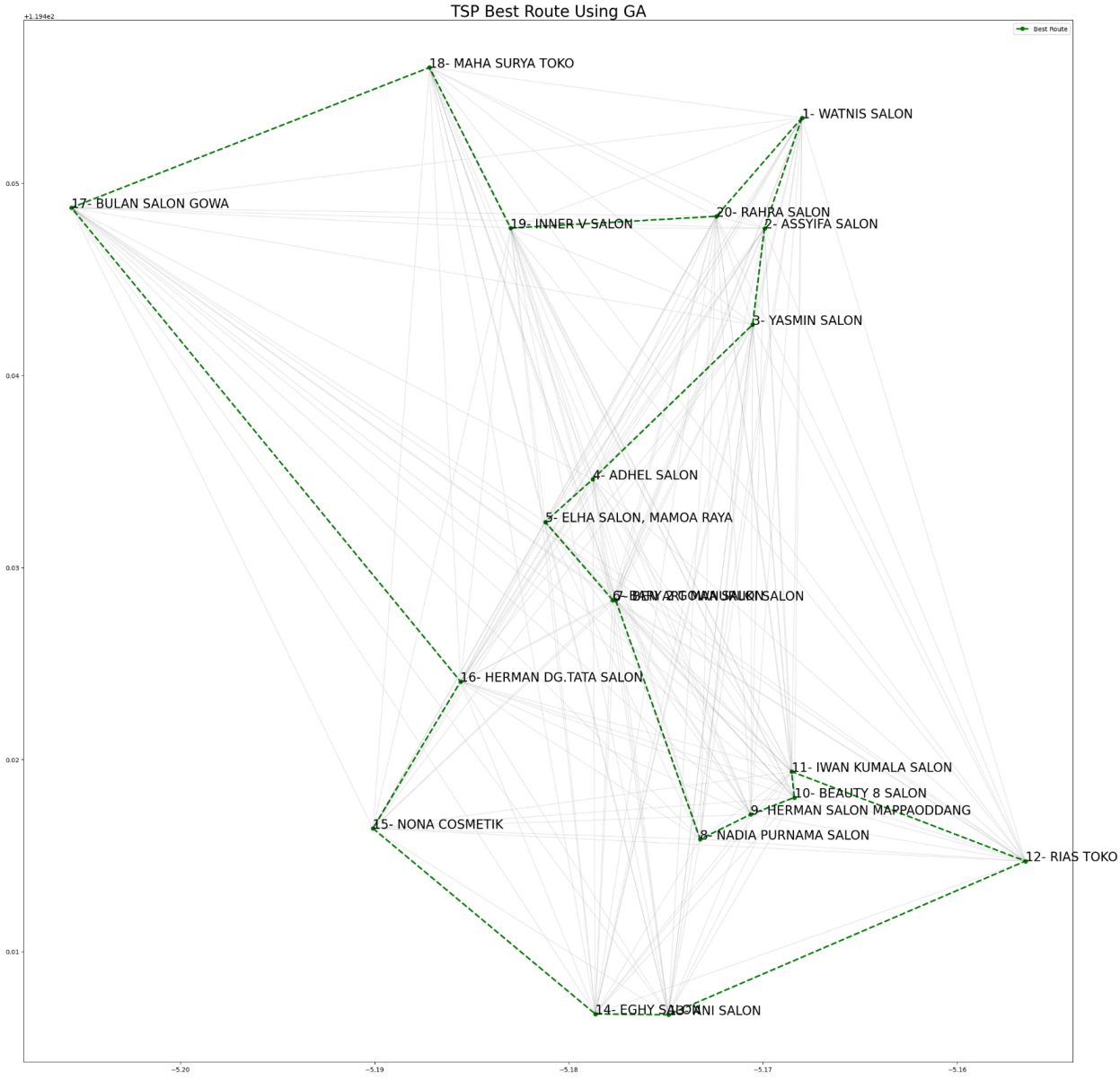
12. Visualiasi Solusi

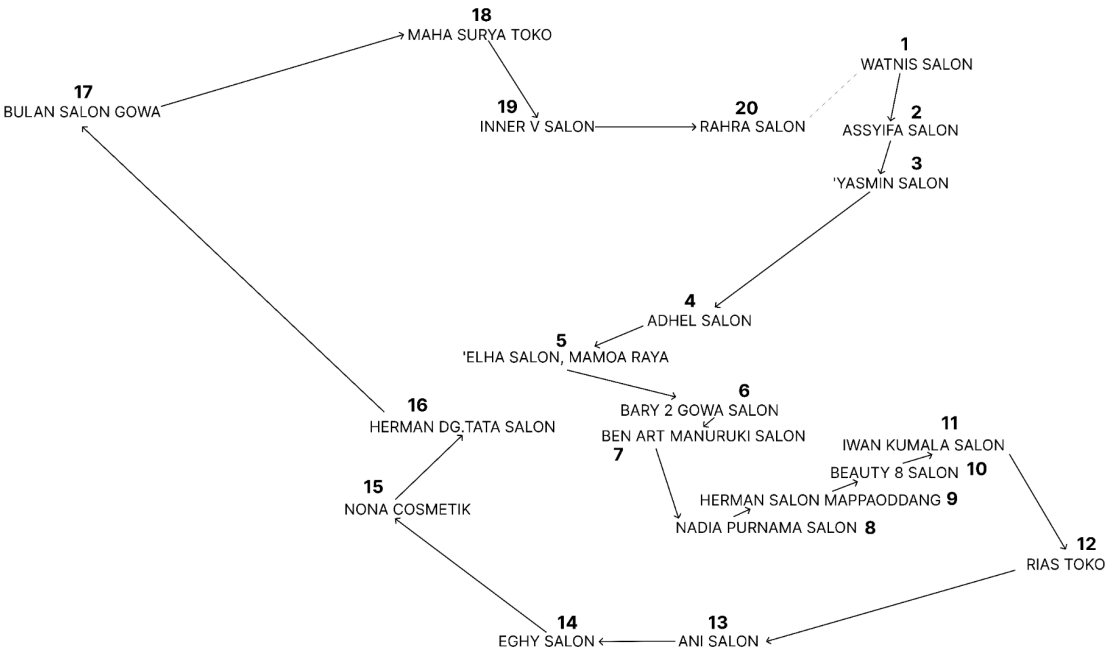
Berikut ini merupakan proses visualisasi solusi akhir yang dihasilkan. Melalui visualisasi ini, kita dapat dengan jelas melihat rute terpendek yang telah dihasilkan oleh algoritma genetika untuk mengatasi masalah pengiriman ke customer. Berdasarkan visualisasi tersebut, customer pertama yang harus dikunjungi yaitu WATNIS SALON dan customer terakhir yaitu RAHRA SALON.

[22]
```
x_shortest = []
y_shortest = []
for city in shortest_path:
    x_value, y_value = city_coords[city]
    x_shortest.append(x_value)
    y_shortest.append(y_value)

x_shortest.append(x_shortest[0])
y_shortest.append(y_shortest[0])
```

[23]
```
fig, ax = plt.subplots()
ax.plot(x_shortest, y_shortest, '--go', label='Best Route', linewidth=2.5)
plt.legend()

for i in range(len(x)):
    for j in range(i + 1, len(x)):
        ax.plot([x[i], x[j]], [y[i], y[j]], 'k-', alpha=0.09, linewidth=1)

plt.title(label="TSP Best Route Using GA",
          fontsize=25,
          color="k")

str_params = '\n'+str(n_generations)+' Generations\n'+str(n_population)+' Population Size\n'+str(crossover_per)+' Crossover\n'+str(mutation_per)+' Mutation'
plt.suptitle("Total Distance Travelled: "+
             str(round(minimum_distance, 3)) + " KM" +
             str_params, fontsize=18, y = 1.047)

for i, txt in enumerate(shortest_path):
    ax.annotate(str(i+1)+ "- " + txt, (x_shortest[i], y_shortest[i]), fontsize= 20)

fig.set_size_inches(30, 30)
# plt.grid(color='k', linestyle='dotted')
plt.savefig('solution.png')
plt.show()
```

Total Distance Travelled: 21.113 KM
700 Generations
200 Population Size
0.8 Crossover
0.2 Mutation





Kesimpulan

Tugas ini bertujuan untuk menyelesaikan permasalahan distribusi barang ke salon kecantikan di Makassar dengan menggunakan Algoritma Genetika. Algoritma Genetika dapat membantu perusahaan menghemat biaya pengirimanntarkan barang ke semua customer, sehingga dapat meminimalkan biaya pengiriman.

Algoritma Genetika meniru teori evolusi untuk menemukan solusi terbaik dalam menyelesaikan permasalahan *Traveling Salesman Problem* (TSP). Algoritma ini dioptimalkan untuk kasus ini dengan parameter yang sesuai. Hasilnya, Algoritma Genetika berhasil menemukan rute terpendek dengan total jarak tempuh 21,11 km. Dengan urutan customer yang harus dikunjungi diawali dari WATNIS SALON hingga terakhir adalah RAHRA SALON. Dapat disimpulkan bahwa Algoritma Genetika terbukti efektif untuk menyelesaikan masalah TSP dalam kasus distribusi barang ke salon kecantikan di Makassar

Daftar Pustaka

1. Engelbrecht, A. P. (2005, December 16). *Fundamentals of Computational Swarm Intelligence*. Wiley. http://books.google.ie/books?id=3xhrQgAACAAJ&dq=Fundamentals+of+Computational+Swarm+Intelligence&hl=&cd=1&source=gbs_api
2. Erdiwansyah, et. al. (2017, Agustus 4). Analisis Hibridisasi Pencarian Lokal Dengan Populasi Dalam Travelling Salesman Problem (TSP)
3. *Genetic Algorithm*. (2018, December 8). School of Computer Science. https://socs.binus.ac.id/2018/12/08/genetic-algorithm/