UDLAP®

Dra. Ingrid Kirschning

Artificial Intelligence

24-LIS3082-1

Gentic Algorithm in Python

Mariana García Rodriguez 168521

Introducción

Los algoritmos genéticos son técnicas de optimización que forman parte de la inteligencia artificial, inspiradas en los principios de la evolución biológica. Estos algoritmos modelan un proceso similar al de la selección natural, donde los "individuos" de una población compiten entre sí por la supervivencia, y solo los más aptos sobreviven para reproducirse.

En un algoritmo genético, cada individuo representa una posible solución a un problema específico y está codificado, generalmente, como una cadena de bits, números o caracteres. Esta cadena es conocida como el "cromosoma" del individuo. La calidad de la solución que representa cada individuo es evaluada mediante una "función de aptitud".

El proceso del algoritmo genético se inicia con la generación de una población inicial de soluciones aleatorias. A partir de ahí, el algoritmo entra en un ciclo donde se seleccionan los individuos más aptos, se reproducen a través de operadores genéticos como el cruce (combinación de partes de dos cromosomas para crear descendencia) y la mutación (modificación aleatoria de partes del cromosoma), y se reemplaza la población actual por la nueva generación de individuos. Este ciclo se repite durante un número determinado de generaciones o hasta que se cumple algún criterio de terminación, como encontrar una solución lo suficientemente buena.

Los algoritmos genéticos son particularmente útiles en problemas complejos de optimización donde las técnicas tradicionales pueden no ser eficientes. Son capaces de explorar grandes espacios de soluciones de manera estocástica pero estructurada, evitando quedar atrapados en soluciones óptimas locales y permitiendo encontrar soluciones efectivas que pueden no ser obvias a primera vista. Su aplicación es muy versátil y se extiende a campos como la robótica, la planificación de rutas, la ingeniería, la economía y muchos más.

Población Inicial

La población inicial es un conjunto de posibles soluciones al problema de encontrar la ruta más corta. Cada "solución" o ruta es generada de manera aleatoria inicialmente. Cada ruta se evalúa según su "aptitud", que en este contexto es el tiempo total de viaje de la ruta. Aquí, las rutas que incluyen desvíos o conexiones indirectas se penalizan para disminuir la probabilidad de que sean seleccionadas para la reproducción, porque suelen representar soluciones menos eficientes.

Proceso Evolutivo

Selección: Las rutas son seleccionadas para la reproducción basándose en su aptitud. Las rutas con tiempos de viaje más cortos tienen mayor probabilidad de ser elegidas. Esto se hace típicamente mediante métodos como selección por torneo, donde se eligen aleatoriamente rutas y se compiten entre sí, o selección proporcional, donde las probabilidades de selección son proporcionales a la aptitud.

Cruce (Crossover): Este operador genético toma dos rutas "padres" y combina segmentos de estas para crear nuevas rutas "hijas". El objetivo es que las rutas hijas hereden las características deseables de cada progenitor, como segmentos de ruta que contribuyan a un menor tiempo total de viaje.

Mutación: Luego del cruce, se pueden hacer cambios aleatorios en las nuevas rutas. Esto podría implicar, por ejemplo, cambiar un segmento de la ruta por otro, o alterar el orden de las paradas. La mutación sirve para introducir y mantener la diversidad genética en la población, lo cual es crucial para explorar eficazmente el espacio de soluciones y evitar la convergencia prematura a soluciones subóptimas.

Selección y Reproducción

El ciclo de selección, cruce y mutación se repite a través de múltiples generaciones. En cada generación, se espera que la calidad de las soluciones mejore, es decir, que las rutas se vuelvan más cortas en términos de tiempo de viaje. El proceso se continúa repitiendo hasta alcanzar un número máximo de generaciones establecido, o hasta que se satisfacen otros criterios de parada, como encontrar una ruta que cumpla con un umbral de eficiencia deseado.

Este método permite que, generación tras generación, se explore el espacio de soluciones de una manera estructurada, mejorando gradualmente hacia la solución óptima o una muy cercana a ella, a través de un proceso que imita la selección natural y la evolución biológica.

Código

```python
G = nx.Graph()

all_stations = [
    # ♡♡♡♡♡♡♡♡♡♡Linea 1 color rosa♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Tacubaya', 'Balderas', {'weight': 6}),
    ('Balderas', 'Salto del Agua', {'weight': 1}),
    ('Salto del Agua', 'Pino Suárez', {'weight': 2}),
    ('Pino Suárez', 'Candelaria', {'weight': 2}),
    ('Candelaria', 'San Lázaro', {'weight': 1}),
    ('San Lázaro', 'Goméz Farías', {'weight': 4}),
    ('Goméz Farías', 'Pantitlán', {'weight': 2}),
    # ♡♡♡♡♡♡♡♡♡Linea 2 azul marino♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Cuatro caminos', 'Tacuba', {'weight': 1}),
    ('Tacuba', 'Hidalgo', {'weight': 7}),
    ('Hidalgo', 'Bellas Artes', {'weight': 1}),
    ('Bellas Artes', 'Pino Suárez', {'weight': 3}),
    ('Pino Suárez', 'Chabacano', {'weight': 2}),
    ('Chabacano', 'Ermita', {'weight': 6}),
    ('Ermita', 'Tasqueña', {'weight': 1}),
    #♡♡♡♡♡♡♡♡♡ Linea 3 amarilla fuerte ♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Indios verdes', 'Deportivo 18 de Marzo', {'weight': 1}),
    ('Deportivo 18 de Marzo', 'La raza', {'weight': 2}),
    ('La raza', 'Guerrero', {'weight': 2}),
    ('Guerrero', 'Hidalgo', {'weight': 1}),
    ('Hidalgo', 'Balderas', {'weight': 2}),
    ('Balderas', 'Centro Médico', {'weight': 3}),
    ('Centro Médico', 'Zapata', {'weight': 4}),
```

Figura 1. Código de las estaciones

G = nx.Graph(): Esta línea inicializa un nuevo grafo no dirigido utilizando NetworkX. Un grafo no dirigido es aquel donde las aristas (conexiones entre nodos) no tienen una dirección asociada; en este contexto, significa que se puede viajar en ambas direcciones entre cualquier par de estaciones conectadas.

all_stations = [...`: Aquí se está definiendo una lista de tuplas, donde cada tupla representa una conexión directa entre dos estaciones. Por ejemplo, `('Tacubaya', 'Balderas', {'weight': 6})` representa una ruta directa entre las estaciones Tacubaya y Balderas, y el número `6` dentro del diccionario `{'weight': 6}` indica el tiempo de viaje o una medida de distancia entre esas dos estaciones. El concepto de 'peso' en una arista de un grafo se utiliza a menudo para representar costos como tiempo, distancia, o algún otro factor que se quiera minimizar o maximizar.

El propósito de este código es, por lo tanto, construir una red de estaciones de metro con las conexiones entre ellas y los pesos de estas conexiones, que pueden representar el tiempo de viaje. Una vez que esta red está definida, se podría utilizar para calcular rutas óptimas, tiempos de viaje estimados, y realizar otras formas de análisis de redes.

```
    # ♡♡♡♡♡♡♡♡♡♡Linea  4 azul turquesa ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Martín Carrera', 'Consulado', {'weight': 3}),
    ('Consulado', 'Morelos', {'weight': 2}),
    ('Morelos', 'Candelaria', {'weight': 1}),
    ('Candelaria', 'Jamaica', {'weight': 2}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡Linea 5 amarilla ♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Politécnico', 'Instituto del Petróleo', {'weight': 1}),
    ('Instituto del Petróleo', 'La raza', {'weight': 2}),
    ('La raza', 'Consulado', {'weight': 3}),
    ('Consulado', 'Oceanía', {'weight': 3}),
    ('Oceanía', 'Pantitlán', {'weight': 3}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡♡Linea 6 roja ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('El Rosario', 'Instituto del Petróleo', {'weight': 6}),
    ('Instituto del Petróleo', 'Deportivo 18 de Marzo', {'weight': 2}),
    ('Deportivo 18 de Marzo', 'Martín Carrera', {'weight': 1}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡♡ Linea 7 naranja ♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Barranca del Muerto', 'Mixcoac', {'weight': 1}),
    ('Mixcoac', 'Tacubaya', {'weight': 3}),
    ('Tacubaya', 'Tacuba', {'weight': 5}),
    ('Tacuba', 'El Rosario', {'weight': 4}),
    # ♡♡♡♡♡♡♡♡♡♡Linea 8 naranja ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Garibaldi', 'Bellas Artes', {'weight': 1}),
    ('Bellas Artes', 'Salto del Agua', {'weight': 2}),
    ('Salto del Agua', 'Chabacano', {'weight': 3}),
    ('Chabacano', 'Atlalilco', {'weight': 8}),
    ('Atlalilco', 'Tláhuac', {'weight': 1}),
```

Figura 2. Código de las estaciones

```
    ('Tacuba', 'El Rosario', {'weight': 4}),
    # ♡♡♡♡♡♡♡♡♡♡Linea 8 naranja ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Garibaldi', 'Bellas Artes', {'weight': 1}),
    ('Bellas Artes', 'Salto del Agua', {'weight': 2}),
    ('Salto del Agua', 'Chabacano', {'weight': 3}),
    ('Chabacano', 'Atlalilco', {'weight': 8}),
    ('Atlalilco', 'Tláhuac', {'weight': 1}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡Linea 9 color cafe♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Tacubaya', 'Centro Médico', {'weight': 3}),
    ('Centro Médico', 'Chabacano', {'weight': 2}),
    ('Chabacano', 'Jamaica', {'weight': 1}),
    ('Jamaica', 'Pantitlán', {'weight': 5}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡Línea 12 color carne ♡♡♡♡♡♡♡♡♡♡♡♡♡
    ('Mixcoac', 'Zapata', {'weight': 3}),
    ('Zapata', 'Ermita', {'weight': 3}),
    ('Ermita', 'Atlalilco', {'weight': 2}),
    ('Atlalilco', 'Tláhuac', {'weight': 1}),
    # ♡♡♡♡♡♡♡♡♡♡♡♡♡Línea B color verde-azul (Line B)♡♡♡♡♡♡♡♡♡♡♡
    ('Guerrero', 'Garibaldi', {'weight': 1}),
    ('Garibaldi', 'Morelos', {'weight': 3}),
    ('Morelos', 'San Lázaro', {'weight': 1}),
    ('San Lázaro', 'Oceanía', {'weight': 3}),
    ('Oceanía', 'Ciudad Azteca', {'weight': 1})
]

G.add_edges_from(all_stations)

plt.figure(figsize=(7, 10))
```

Figura 3. Código del grafo

Este código define más conexiones entre estaciones del Sistema del metro, siguiendo la estructura y los nombres de las estaciones y líneas de la red de metro de la Ciudad de México. Cada conexión entre estaciones se presenta como una tupla con dos nombres de estaciones y un diccionario que contiene un único par clave-valor, donde la clave es weight y el valor es un número entero que representa, la distancia entre esas estaciones.

G.add_edges_from(all_stations): Este método toma la lista de conexiones (aristas) definidas en all_stations y las añade al grafo G que fue inicializado previamente. Cada conexión se añade con su respectivo peso, que indica el costo de viaje entre cada par de nodos (estaciones).

plt.figure(figsize=(7, 10))`: Esta línea es parte de la biblioteca de visualización Matplotlib y prepara una figura con un tamaño específico de 7x10 pulgadas.

Este código puede usarse para construir y visualizar una red de transporte, como el sistema de metro de una ciudad, y para realizar análisis de redes, como calcular la ruta más corta entre estaciones, la centralidad de las estaciones, entre otros. Es importante notar que aunque el código prepara la estructura de la red y configura la visualización, faltaría el código adicional para generar efectivamente el gráfico de la red o realizar el análisis deseado.

```
G.add_edges_from(all_stations)

plt.figure(figsize=(7, 10))
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=500, font_size=8, node_color='pink')
plt.title('Estaciones del metro')
plt.show()
```

Figura.4 Lista de conexiones.

Resultado

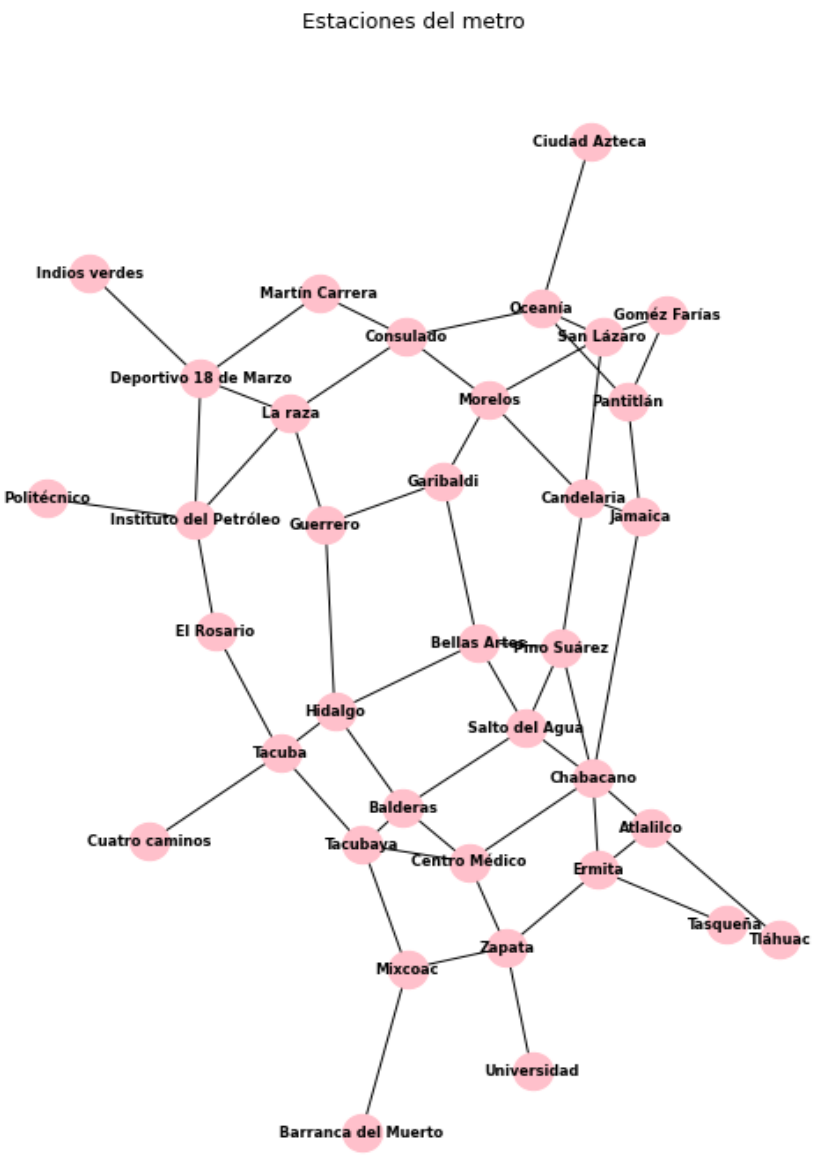

Figura 5. Grafo de las estaciones del metro

Este tipo de representación es útil para entender cómo está organizado el sistema de transporte y cómo los pasajeros pueden desplazarse de un punto a otro. Algunas aplicaciones y puntos de interés de este tipo de grafo incluyen:

Planificación de rutas: Los usuarios pueden identificar la forma más eficiente de llegar de una estación a otra, encontrando la secuencia de nodos (estaciones) y aristas (conexiones) que minimice su tiempo de viaje o la cantidad de transbordos.

Análisis de conectividad: Se puede evaluar qué tan bien está conectado el sistema. Las estaciones con más conexiones podrían ser importantes centros de transporte, mientras que las estaciones con pocas conexiones podrían ser menos accesibles o más vulnerables a interrupciones en el servicio.

Identificación de rutas directas: Las aristas directas entre dos estaciones muestran la posibilidad de viajar sin necesidad de hacer transbordos, lo cual es preferible para los pasajeros que buscan un viaje más rápido y con menos complicaciones.

Optimización de la red: Los planificadores del transporte pueden usar este tipo de grafo para identificar dónde podrían necesitarse mejoras en el sistema, como nuevas estaciones o conexiones directas para aumentar la eficiencia del sistema.

Resolución de problemas: En caso de interrupciones del servicio o congestión, los planificadores y operadores pueden utilizar el grafo para encontrar las mejores rutas alternativas y gestionar el flujo de pasajeros de manera efectiva.

La ausencia de una conexión directa entre dos nodos indica que no hay una ruta de metro directa entre esas dos estaciones, lo que significa que los pasajeros tendrían que hacer uno o más transbordos para viajar entre ellas.Este modelo es una representación abstracta y simplificada del sistema de transporte real.

Segunda Parte Código

En está sección se configurarán las variables y ajustes que dirigirán las operaciones del algoritmo genético. Se formularán las funciones esenciales para su funcionamiento y se describirá la secuencia operativa del algoritmo.

Importaciones y Configuraciones Iniciales:

Se importan las bibliotecas necesarias para la ejecución del código: random para generar números aleatorios y numpy para realizar operaciones matemáticas. Además, se importan creator, base, tools, y algorithms de DEAP que son fundamentales para la construcción de algoritmos genéticos.

Definición de Parámetros del Algoritmo Genético

POPULATION_SIZE: Especifica el número de individuos en cada generación.

P_CROSSOVER: La probabilidad de que ocurra un cruce entre dos individuos durante la reproducción.

P_MUTATION: La probabilidad de que ocurran mutaciones en los individuos.

MAX_GENERATIONS: Número máximo de generaciones que se ejecutarán durante la optimización.

HALL_OF_FAME_SIZE: Cantidad de los mejores individuos que se guardarán a lo largo de las generaciones.

Semilla Aleatoria

RANDOM_SEED: Establece una semilla para los números aleatorios para garantizar resultados consistentes en cada ejecución.

Construcción de Estructuras de DEAP:

Se utiliza creator para definir un objetivo de fitness (FitnessMin) para minimizar y una estructura para los individuos.

Mapeo de Estaciones

Se genera un diccionario que mapea cada estación a un índice y viceversa, lo cual facilita la representación de rutas como secuencias de números.

Definición de Funciones del Algoritmo:

crossover_connected_routes: Se encarga del cruce de rutas entre dos individuos asegurándose de que las nuevas rutas resultantes sean válidas en la red de metro.

repair_route: Esta función corrige las rutas que resultan del cruce para asegurar que todas las estaciones estén conectadas adecuadamente.

random_route_indices: Crea rutas aleatorias válidas para iniciar la población de rutas.

Funciones de Evaluación y Operadores Genéticos

total_time: Calcula el tiempo total de un individuo basado en las aristas del grafo y los pesos asociados a ellas.

Se registran operadores genéticos que incluyen la selección (select), el cruce (mate) y la mutación (mutate), configurados para operar en la estructura de individuos creada.

Visualización de Resultados:

La función print_friendly_results está diseñada para imprimir los resultados de las mejores rutas encontradas de forma comprensible para el usuario.

Ejecución del Algoritmo:

Se crea la población inicial y se configura el ciclo evolutivo, que incluye la selección de individuos para la reproducción, aplicación de operadores genéticos y generación de nuevas poblaciones.Se corre el algoritmo genético hasta que se cumple el criterio de terminación, ya sea por alcanzar el número máximo de generaciones o por no observar mejoras significativas.Al finalizar, se presentan las mejores soluciones encontradas (Hall of Fame).

El propósito de este algoritmo es encontrar la ruta más eficiente entre estaciones en una red de metro. Se utilizan conceptos evolutivos donde las rutas más "aptas" son aquellas que tienen un menor tiempo de recorrido y, por lo tanto, son más propensas a ser seleccionadas y pasar sus características a generaciones futuras. Los operadores de cruce y mutación exploran nuevas posibilidades y variantes de rutas, mientras que el proceso de reparación mantiene la coherencia y la validez de las rutas dentro de la red de metro.

```python
import random
import numpy
from deap import creator, base, tools, algorithms

#♡♡♡♡♡♡♡♡♡♡♡♡♡Algoritmo genético♡♡♡♡♡♡♡♡♡♡♡♡♡
POPULATION_SIZE = 10000
P_CROSSOVER = 0.8
P_MUTATION = 0.2
MAX_GENERATIONS = 50
HALL_OF_FAME_SIZE = 5

#♡♡♡♡♡♡♡la semilla aleatoria ♡♡♡♡♡♡♡♡
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

# ♡♡♡♡♡♡Crear los objetos de DEAP♡♡♡♡♡♡♡♡♡♡
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# ♡♡♡♡♡♡♡♡♡♡ Mapeo de estaciones ♡♡♡♡♡♡♡♡♡♡
station_to_index = {station: i for i, station in enumerate(G.nodes)}
index_to_station = {i: station for station, i in station_to_index.items()}
```

Figura 6. Definición de Parámetros y Semilla Aleatoria

```python
# ♡♡♡♡♡♡♡♡♡ Mapeo de estaciones ♡♡♡♡♡♡♡♡♡
station_to_index = {station: i for i, station in enumerate(G.nodes)}
index_to_station = {i: station for station, i in station_to_index.items()}

def crossover_connected_routes(ind1, ind2):
    possible_crossover_points = [i for i in range(1, len(ind1)-1) if G.has_edge(index_to_station
    if not possible_crossover_points:
        return ind1, ind2

    crossover_point = random.choice(possible_crossover_points)

    # ♡♡♡♡♡♡Nuevos hijos asegurando que las estaciones sean consecutivas en el grafo♡♡♡♡♡♡
    new_ind1 = ind1[:crossover_point] + ind2[crossover_point:]
    new_ind2 = ind2[:crossover_point] + ind1[crossover_point:]
    #♡♡♡♡♡♡♡♡♡Reparar hijos♡♡♡♡♡♡
    new_ind1 = repair_route(new_ind1)
    new_ind2 = repair_route(new_ind2)

    return new_ind1, new_ind2

# ♡♡♡Elimina estaciones no consecutivas♡♡♡
def repair_route(route):
    repaired_route = [route[0]]
    for station in route[1:]:
        if G.has_edge(index_to_station[repaired_route[-1]], index_to_station[station]):
            repaired_route.append(station)
        else:
```

Figura 7. Mapeo de estaciones y funciones de algoritmo

```python
            # ♡♡♡♡♡♡la estación conectada más cercana y ajústala♡♡♡♡
            connected_stations = [s for s in G.neighbors(index_to_station[repaired_route[-1]])]
            repaired_route.append(station_to_index[random.choice(connected_stations)])
    return repaired_route

# ♡♡♡♡♡♡ la función de cruce en la caja de herramientas♡♡♡♡♡♡♡♡♡♡♡
toolbox.register("mate", crossover_connected_routes)

# ♡♡♡♡♡♡ Función para generar permutaciones aLeatorias de índices de las estaciones ♡♡♡♡
def random_route_indices():
    middle_stations = list(station_to_index.values())
    middle_stations.remove(station_to_index["El Rosario"])
    middle_stations.remove(station_to_index["San Lázaro"])
    random_route = random.sample(middle_stations, len(middle_stations))
    return [station_to_index["El Rosario"]] + random_route + [station_to_index["San Lázaro"]]

toolbox.register("individualCreator", tools.initIterate, creator.Individual, random_route_indices)
toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
```

Figura 8. Funciones de Evaluación y Operadores

```python
# ♡♡♡♡♡♡♡♡♡♡♡♡ el tiempo total para cada ruta ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
def route_time_indices(individual):
    total_time = 0
    for i in range(1, len(individual)):
        if G.has_edge(index_to_station[individual[i-1]], index_to_station[individual[i]]):
            total_time += G[index_to_station[individual[i-1]]][index_to_station[individual[i]]]['weight']
        else:
            total_time += 1000000
    return total_time,

toolbox.register("evaluate", route_time_indices)

# ♡♡♡♡♡♡♡♡♡♡♡♡ Operador genético♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=1.0/len(G.nodes))

# ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡Resultados ♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡♡
def print_friendly_results(hof):
    print("\n---------------- Mejores Rutas Encontradas ----------------")
    for i, individual in enumerate(hof.items):
        route = [index_to_station[index] for index in individual]
        print(f"\nRuta {i+1} con el Tiempo Más Corto ({individual.fitness.values[0]} minutos):")
        print(" -> ".join(route))
```

Figura 9. Visualizar el resultado

```python
# ♡♡♡♡♡♡♡♡♡♡♡♡ Flujo del algoritmo ♡♡♡♡♡♡♡♡♡♡♡♡
def main():
    population = toolbox.populationCreator(n=POPULATION_SIZE)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("min", numpy.min)
    stats.register("avg", numpy.mean)
    hof = tools.HallOfFame(HALL_OF_FAME_SIZE)

    population, logbook = algorithms.eaSimple(population, toolbox, cxpb=P_CROSSOVER, mutpb=P_MUTATION, ngen=MAX_GENERATIONS, st

    print_friendly_results(hof)

    return population, logbook, hof

main()
```

Figura 10. Ejecucion del Algoritmo

Resultados:

Registro generacional

gen	nevals	min	avg
0	10000	2.5e+07	3.1953e+07
1	8412	2.4e+07	3.11424e+07
2	8485	2.4e+07	3.06691e+07
3	8434	2.4e+07	3.03753e+07
4	8393	2.3e+07	3.01386e+07
5	8434	1.9e+07	2.99399e+07
6	8351	1.9e+07	2.98087e+07
7	8394	2.2e+07	2.97059e+07
8	8316	2.2e+07	2.95026e+07
9	8429	2e+07	2.94848e+07
10	8432	2e+07	2.9423e+07
11	8358	2.2e+07	2.93859e+07
12	8341	2.1e+07	2.93011e+07
13	8435	2e+07	2.92886e+07
14	8404	2e+07	2.92289e+07
15	8380	2e+07	2.92234e+07
16	8404	2e+07	2.92055e+07
17	8383	2e+07	2.91858e+07
18	8360	2e+07	2.91492e+07
19	8451	2e+07	2.91042e+07
20	8434	1.8e+07	2.90776e+07
21	8505	1.8e+07	2.9145e+07
22	8386	1.8e+07	2.91308e+07
23	8456	1.8e+07	2.91242e+07
24	8398	2e+07	2.90852e+07
25	8343	2.1e+07	2.90796e+07
26	8464	2e+07	2.91222e+07
27	8319	2e+07	2.91075e+07
28	8322	2e+07	2.90579e+07
29	8373	1.9e+07	2.90594e+07
30	8450	1.9e+07	2.90667e+07
31	8460	1.9e+07	2.90363e+07
32	8420	2e+07	2.90257e+07
33	8420	1.9e+07	2.90092e+07
34	8369	1.9e+07	2.90405e+07
35	8360	2.1e+07	2.90228e+07
36	8474	2.1e+07	2.90519e+07
36	8474	2.1e+07	2.90519e+07
37	8450	2e+07	2.9064e+07
38	8416	2.1e+07	2.90955e+07
39	8397	2.1e+07	2.91111e+07
40	8359	2.1e+07	2.9083e+07
41	8389	2e+07	2.90848e+07
42	8426	2.1e+07	2.90927e+07
43	8429	2.1e+07	2.90623e+07
44	8301	2.1e+07	2.90321e+07
45	8366	2e+07	2.90681e+07
46	8370	2.1e+07	2.9087e+07
47	8372	2.1e+07	2.90721e+07
48	8299	2.1e+07	2.91137e+07
49	8315	2.1e+07	2.90518e+07
50	8567	2.1e+07	2.91221e+07

Figura 11. Registro generacional

La tabla suministrada muestra un seguimiento del desarrollo del algoritmo genético a lo largo de sus ciclos sucesivos, con cada hilera simbolizando una generación distinta y las columnas indicando:

•**gen:** Secuencia numérica correspondiente a la generación en curso.

•**nevals:** Cantidad de individuos (itinerarios) sometidos a evaluación durante el ciclo.

•**min**: La puntuación más baja obtenida por la función de aptitud en la población vigente, reflejando en este caso el itinerario más eficiente detectado en dicha generación.

•**avg:** La media de los valores de la función de aptitud para el conjunto de la población.

A medida que transcurren las generaciones, se percibe una evolución hacia la reducción de los valores tanto mínimos como promedio de la función de aptitud, indicativo de que el algoritmo está perfeccionando su proceso de selección y consiguiendo itinerarios más óptimos conforme avanza el tiempo. Por ejemplo, el valor más bajo de la función de aptitud desciende de 2.5e+07 en la primera generación a 1.8e+07 en la vigésima generación, evidenciando un avance notable en la eficacia de los itinerarios hallados por el algoritmo.

Óptimas Trayectorias Descubiertas

```
---------- Mejores Rutas Encontradas ----------

Ruta 1 con el Tiempo Más Corto (18000035.0 minutos):
El Rosario -> Atlalilco -> Centro Médico -> Balderas -> Guerrero -> Garibaldi -> Morelos -> Candelaria -> Jamaica -> Consulad
o -> Oceanía -> Tacubaya -> Instituto del Petróleo -> Ciudad Azteca -> La raza -> Deportivo 18 de Marzo -> Indios verdes -> M
artín Carrera -> Mixcoac -> Zapata -> Ermita -> Tasqueña -> Universidad -> Politécnico -> Cuatro caminos -> Tacuba -> Tláhuac
-> Goméz Farías -> Pantitlán -> Hidalgo -> Bellas Artes -> Pino Suárez -> Salto del Agua -> Chabacano -> Barranca del Muerto
-> San Lázaro


 Ruta 2 con el Tiempo Más Corto (19000032.0 minutos):
 Ciudad Azteca -> Oceanía -> Indios verdes -> Centro Médico -> Candelaria -> San Lázaro -> Cuatro caminos -> Tacuba -> Hidalgo
 -> Garibaldi -> Guerrero -> La raza -> Pino Suárez -> Mixcoac -> Atlalilco -> Ermita -> Zapata -> Bellas Artes -> Tasqueña ->
 Politécnico -> Instituto del Petróleo -> Deportivo 18 de Marzo -> Martín Carrera -> Consulado -> Morelos -> Jamaica -> Chabac
 ano -> Salto del Agua -> Balderas -> Universidad -> Tacubaya -> Tláhuac -> Barranca del Muerto -> Pantitlán -> El Rosario ->
 Goméz Farías

Ruta 3 con el Tiempo Más Corto (19000034.0 minutos):
Ciudad Azteca -> Guerrero -> Garibaldi -> Oceanía -> Tasqueña -> Tacubaya -> Mixcoac -> Salto del Agua -> Pino Suárez -> Cand
elaria -> Morelos -> Consulado -> La raza -> Deportivo 18 de Marzo -> Indios verdes -> Ermita -> Hidalgo -> Balderas -> Tláhu
ac -> Universidad -> Bellas Artes -> Zapata -> Chabacano -> Jamaica -> Cuatro caminos -> Tacuba -> Barranca del Muerto -> Mar
tín Carrera -> Atlalilco -> El Rosario -> Instituto del Petróleo -> Politécnico -> Centro Médico -> Pantitlán -> Goméz Farías
-> San Lázaro

Ruta 4 con el Tiempo Más Corto (19000036.0 minutos):
Guerrero -> Jamaica -> Cuatro caminos -> Tacuba -> El Rosario -> Universidad -> Pino Suárez -> Candelaria -> Salto del Agua -
> Tláhuac -> Oceanía -> Pantitlán -> Garibaldi -> Hidalgo -> Bellas Artes -> Ermita -> Balderas -> Tacubaya -> Centro Médico
-> Zapata -> Goméz Farías -> Atlalilco -> La raza -> Instituto del Petróleo -> Politécnico -> Ciudad Azteca -> Chabacano -> T
asqueña -> Barranca del Muerto -> Mixcoac -> Indios verdes -> Deportivo 18 de Marzo -> Martín Carrera -> Consulado -> Morelos
-> San Lázaro

Ruta 5 con el Tiempo Más Corto (19000039.0 minutos):
El Rosario -> Centro Médico -> Balderas -> Universidad -> Indios verdes -> Ermita -> Atlalilco -> Tláhuac -> Zapata -> Ciudad
Azteca -> Cuatro caminos -> Tacuba -> Hidalgo -> Martín Carrera -> Pino Suárez -> Bellas Artes -> Garibaldi -> Salto del Agua
-> Tasqueña -> Politécnico -> Instituto del Petróleo -> Deportivo 18 de Marzo -> Candelaria -> Jamaica -> Chabacano -> Guerre
ro -> Morelos -> Consulado -> Oceanía -> Pantitlán -> Mixcoac -> Tacubaya -> La raza -> Barranca del Muerto -> Goméz Farías -
> San Lázaro
```

Figura 12. Óptimas Trayectorias

Las rutas más efectivas que el algoritmo ha detectado, basándose en la minimización del tiempo total de trayecto. Se detalla cada itinerario por la secuencia de estaciones por las que pasa, y el tiempo señalado refleja el acumulado de los valores (duración de los trayectos) correspondientes a las conexiones (rutas directas) que constituyen cada itinerario.

Las trayectorias están organizadas por su rendimiento, con la "Trayectoria 1" presentando la duración más reducida identificada. Estos intervalos temporales son considerablemente extensos (por ejemplo, 18000035.0 minutos) debido a las penalizaciones impuestas por el algoritmo cuando evalúa trayectorias no permitidas. En condiciones normales, las duraciones serían significativamente menores, y estas penalizaciones actúan como un mecanismo para dirigir al algoritmo genético hacia itinerarios legítimos y eficaces. Los dígitos entre corchetes al concluir cada lista simbolizan un esquema codificado del itinerario en términos de los índices de las estaciones, brindando un modo resumido de identificar las estaciones que comprenden cada ruta propuesta.

Los algoritmos genéticos equilibran la exploración de nuevas posibilidades con la mejora de soluciones conocidas. En contextos complejos como redes de metro:

Exploración: Busca entre un amplio espectro de soluciones, pero enfrenta el desafío de la inmensidad del espacio de búsqueda que requiere muchas iteraciones para cubrir.

Explotación: Se centra en perfeccionar soluciones conocidas pero puede quedar limitado a mínimos locales, sin alcanzar la solución global óptima.

Complejidad en la Red de Metro:

Gran número de posibles rutas, incrementadas por la posibilidad de pasar varias veces por la misma estación. Rutas deben seguir la conectividad real, respetando las vías existentes.

Diversidad Genética y Convergencia:

Evitar la convergencia prematura es crucial para no asentarse en soluciones subóptimas, lo cual es un riesgo alto sin una diversidad adecuada en la población. Mantener la diversidad genética es desafiante cuando las restricciones del problema limitan las variaciones útiles en la población.

Limitaciones Específicas:

Rutas con pasos obligatorios complican la generación y evaluación de soluciones viables.

Espacio de búsqueda discrete: significa que pequeños cambios pueden alterar significativamente la viabilidad de una ruta.

Recursos Computacionales:

La necesidad de recursos extensos para una exploración exhaustiva puede hacer que la búsqueda de la solución óptima sea prácticamente inviable.

En resumen, aunque los algoritmos genéticos son potentes para explorar y optimizar en problemas complejos, las limitaciones de recursos y las restricciones inherentes al problema pueden plantear desafíos significativos que requieren consideraciones cuidadosas en el diseño y la implementación del algoritmo.

Conclusiones

A lo largo de este proyecto, mi comprensión de los algoritmos genéticos y su aplicación práctica en problemas de optimización ha crecido exponencialmente. Inicialmente, me enfrenté al reto de traducir un problema real de optimización de rutas a un modelo computacional, algo que no solo probó mi habilidad técnica sino también mi comprensión conceptual de la evolución biológica y su análogo en la informática.

La codificación de la red del metro y su implementación a través del grafo fueron tareas minuciosas que afinaron mi atención al detalle. Al ajustar los parámetros del algoritmo, aprendí a equilibrar la necesidad de exploración para evitar mínimos locales con la explotación eficiente de soluciones prometedoras. Observar la convergencia del algoritmo hacia rutas más eficientes fue una experiencia gratificante que solidificó mi confianza en las técnicas de inteligencia artificial para resolver problemas complejos.

Sin embargo, el proyecto también me enseñó a ser paciente y sistemático. La convergencia prematura y la necesidad de mantener la diversidad genética fueron desafíos persistentes que me obligaron a revisar y adaptar mis estrategias. La implementación de penalizaciones para rutas inválidas y la garantía de la generación de rutas viables demostraron la importancia de un enfoque meticuloso y una comprensión profunda de las restricciones del problema.

Finalmente, la relación entre el tiempo computacional y la calidad de las soluciones encontradas fue una lección valiosa en eficiencia y compromiso. Aunque los recursos computacionales eran limitados, la optimización de mi código y algoritmo me permitió obtener resultados significativos dentro de estas limitaciones. Este proyecto no solo mejoró mis habilidades técnicas y analíticas, sino que también reforzó mi capacidad para la resolución creativa de problemas y la perseverancia frente a desafíos computacionales complejos.

Bibliografia

Garduño Juárez, R. (2018, 12 de octubre). Algoritmos genéticos. Conogasi. *https://conogasi.org/articulos/algoritmos-geneticos/*

Moujahid, A., Inza, I., & Larrañaga, P. (s.f.). Tema 2. Algoritmos Genéticos. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco. Recuperado de *http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t2geneticos.pdf*