Tarea validación algoritmo genético

Instituto Tecnológico de México Campus Culiacán

Tópicos avanzados de I.A.

Zuriel Dathan Mora Félix

Integrantes de equipo:

Sánchez Arévalo José Antonio – 21170474

Félix Avendaño Mateo – 21170314

08/11/2025

Prueba de Ejecución

Configuraciones del Algoritmo Genético:

Ciudades:

```
ciudades = [
    Municipio("Madrid", 40.4168, -3.7038),
    Municipio("Barcelona", 41.3784, 2.1925),
    Municipio("Valencia", 39.4699, -0.3763),
    Municipio("Sevilla", 37.3886, -5.9953),
    Municipio("Bilbao", 43.2630, -2.9350),
    Municipio("Zaragoza", 41.6488, -0.8891),
    Municipio("Malaga", 36.7213, -4.4214),
    Municipio("Murcia", 37.9847, -1.1287),
    Municipio("Palma", 39.5696, 2.6502),
    Municipio("Las Palmas", 28.1235, -15.4363)
]
```

Población: Ciudades

Tamaño de Población: 100

Individuos seleccionados: 20

Razón Mutación: 0.01

Generaciones: 500

```
PS C:\Users\jose.sanchez\Documents\TopicosIA\Unidad3_TareaValidacion> & C:\U
sanchez/Documents/TopicosIA/Unidad3_TareaValidacion/main.py
Distancia Inicial: 59.84628000017788
Distancia Final: 56.54268122567097
Mejor Ruta Encontrada:
Madrid (40.4168,-3.7038)
Zaragoza (41.6488,-0.8891)
Bilbao (43.263,-2.935)
Palma (39.5696,2.6502)
Barcelona (41.3784,2.1925)
Murcia (37.9847,-1.1287)
Valencia (39.4699,-0.3763)
Malaga (36.7213,-4.4214)
Las Palmas (28.1235,-15.4363)
Sevilla (37.3886,-5.9953)
```

Pruebas para el Algoritmo Genético

Prueba de Aptitud

```python
# Creacion de una ruta conocida par aprobar la Aptitud
# Ruta: A -> B -> C -> D
#Se crea una ruta conocida que forma un cuadrado de lado 1.
#La distancia total esperada es 4.0 y la aptitud esperada es 0.25.
#Esta prueba valida que la función de aptitud calcula correctamente la distancia y la aptitud de la ruta.

munis = [
    Municipio('A', 0, 0),
    Municipio('B', 0, 1),
    Municipio('C', 1, 1),
    Municipio('D', 1, 0)
]
apt = Aptitud(munis)
print("Distancia esperada: 4.0, calculada:", apt.distanciaRuta())
print("Aptitud esperada: 0.25, calculada:", apt.rutaApta())
```

```
Distancia esperada: 4.0, calculada: 4.0
Aptitud esperada: 0.25, calculada: 0.25
```

Prueba de Selección

```python
# Prueba de selección
# Se crea una población inicial con cuatro rutas distintas:
# - munis: ruta original (A -> B -> C -> D)
# - munis[::-1]: ruta invertida (D -> C -> B -> A)
# - munis[1:] + munis[:1]: ruta comenzando en B (B -> C -> D -> A)
# - munis[2:] + munis[:2]: ruta comenzando en C (C -> D -> A -> B)
# Se verifica que el método seleccionRutas elige rutas (individuos) de la población
# priorizando aquellas con mejor aptitud. Al imprimir los índices seleccionados,
# se puede observar que los individuos con mayor aptitud son seleccionados más frecuentemente.
# Lo que indica que el proceso de seleccion funciona correctamente.
poblacion = [munis, munis[::-1], munis[1:] + munis[:1], munis[2:] + munis[:2]]
popRanked = Ruta.clasificacionRutas(poblacion)
seleccionados = Ruta.seleccionRutas(popRanked, 2)
print("Índices seleccionados:", seleccionados)
```

```
Índices seleccionados: [0, 1, 3]
```

Prueba de Cruce

```python
# Prueba de cruce
# Se verifica que el método de cruce genera un hijo combinando dos rutas (padres) válidas.
# El hijo resultante debe contener todos los municipios sin repetir y en una nueva combinación,
# asegurando que la descendencia sea válida.

rp = Reproduccion()
padre1 = munis
padre2 = munis[::-1]
hijo = rp.reproduccion(padre1, padre2)
print("Hijo generado:", hijo)
#Ordena ambas listas por coordenadas (x,y) para comparar
#si el hijo contiene los mismos municipios que los padres
print("¿Es válido?", sorted(hijo, key=lambda m: (m.x, m.y)) == sorted(munis, key=lambda m: (m.x, m.y)))
```

```
Hijo generado: [A (0,0), B (0,1), C (1,1), D (1,0)]
¿Es válido? True
```

Prueba de Mutación

```python
# Prueba de mutación
# Se verifica que el método de mutación modifica el orden de los municipios en la ruta,
# pero mantiene todos los municipios originales sin repetir ni omitir ninguno.
mu = Mutacion()
individuo = munis[:]
mutado = mu.mutacion(individuo, 0.1)
print("Individuo original:", [m.nombre for m in munis])
print("Individuo mutado:", [m.nombre for m in mutado])
#Valida si el individuo mutado contiene los mismos municipios que el original
print("¿Es válido?", sorted(mutado, key=lambda m: (m.x, m.y)) == sorted(munis, key=lambda m: (m.x, m.y)))
```

```
Individuo original: ['A', 'B', 'C', 'D']
Individuo mutado: ['D', 'B', 'C', 'A']
¿Es válido? True
```

Enlace de repositorio GitHub del código reconstruido y README: https://github.com/JSarevalo25/TareaValidacionU3AG