Tecnológico Nacional de México
Campus Culiacán

Trabajo de Validación Algoritmo Genético

Tópicos de Inteligencia Artificial

Profesor:
Zuriel Dathan Mora Felix

Integrantes:
Aguilar Recio Jesús Octavio
Echeagaray Aceves Astrid Monserrath

Repositorio GitHub:
https://github.com/OctavioAR/TOPICOS-IA.git

Índice

Índice de Imágenes

1. Introducción

El presente trabajo tiene como objetivo la reconstrucción y mejora de un algoritmo genético aplicado al problema del agente viajero. A partir de una versión inicial donde toda la lógica se encontraba en un solo archivo, se desarrolló una nueva arquitectura modular y orientada a objetos, dividiendo el código en clases y módulos independientes como municipio, aptitud, operadorGenetico, etc.

El algoritmo genético busca encontrar la ruta más corta que permita visitar todos los municipios y volver al punto de inicio, optimizando la distancia total recorrida. Para ello, se implementaron operadores de selección, cruce y mutación inspirados en la evolución natural, donde las rutas más eficientes tienen mayor probabilidad de ser preservadas y combinadas.

Durante el proceso de reconstrucción se aplicaron mejoras en la estructura del código y la nomenclatura de funciones, haciendo el programa más legible, reutilizable y fácil de mantener. Además, se añadió el atributo "nombre" a la clase municipio, lo cual permitió una mejor identificación y visualización de los nodos en los resultados. Estas modificaciones no solo optimizaron la lógica del algoritmo, sino también la comprensión y el flujo de ejecución.

2. Desarrollo

2.1. Clase Municipio

La clase "municipiorepresenta un nodo dentro de la ruta del agente viajero. Dicho objeto municipio con un nombre y las coordenadas X (latitud) y Y(Longitud). Además, la clase cuenta con el método "distancia()" que calcula la distancia euclidiana a otro objeto municipio utilizando la librería "numpy.sqrt" para la raíz cuadrada. Y el método –repr— que devuelve una cadena del municipio, la cual incluye nombre y coordenadas, para poder así visualizar mejor la ruta.

2.2. Clase Aptitud

En la clase Aptitud se calcula el valor de aptitud para una ruta dada en el TSP. La clase inicializa los atributos distancia y f-aptitud a cero. El método principal "distanciaRuta()", itera sobre la lista de municipios, sumando la distancia entre cada par de municipios consecutivos. Dicha función también se encarga de cerrar automáticamente el ciclo de ruta, al calcular la distancia entre el último municipio de la lista y el primero. Por último, tenemos el método "rutaApta()" el cual calcula el valor de aptitud como el inverso de la distancia total (1 / distancia), convirtiendo el problema de minimización de distancia en un problema de maximización de aptitud.

2.3. Módulo de Población inicial y Clasificación

Este grupo de funciones se encarga de inicializar la población para el proceso del algoritmo genético, la función "crearRuta()" genera un individuo inicial seleccionando aleatoriamente de todos los municipios disponibles. Luego el método "poblacionInicial()" utiliza la anterior para crear el conjunto inicial de rutas. El método "clasificacionRutas()" evalúa la aptitud de todas las rutas de la población y las clasifica en orden descendente, identificando así la mejor ruta.

2.4. Módulo de Selección y Apareamiento

El proceso de selección determina que rutas pasarán a la siguiente generación o serán utilizadas para el cruce. El método "seleccionRutas()" implementa una estrategia que combina seleccionando un número definido de individuos de mejor clasificación y como segundo seleccionando el resto de los individuos utilizando el método de la ruleta rusa, la cual se basa en la aptitud acumulada, lo que les brinda a las mejores rutas una mayor probabilidad de ser elegidas. La función "grupoApareamiento()" solamente obtiene las rutas seleccionadas por sus índices para formar los padres.

2.5. Módulo de Cruce y Mutación

En este módulo aplica los operadores genéticos para generar la nueva población. El método "cruce()" selecciona un segmento aleatorio del progenitor 1 y luego rellena los genes restantes con los municipios del progenitor 2, manteniendo así un orden. Luego la función "crucePoblacion()" aplica este operador asegurando que los individuos élite se preserven sin cruce antes de generar los nuevos hijos para completar

el tamaño de la población. Por último, el método “mutación()” aplica una mutación por intercambio, si un numero aleatorio es menor que la “tasaMutacion” selecciona dos genes aleatorios dentro de la ruta y los intercambia de posición.

2.6. Módulo de Pruebas Unitarias

En el módulo de “pruebas” se encarga de validar los componentes importantes del AG mediante distintos métodos de prueba con resultados esperados. Utilizando datos de prueba simples como un cuadrado de cuatro municipios. Las funciones de prueba mostrarón lo siguiente 1, estas verifican por ejemplo la distancia mediante el método “pruebaDistancia()” el cual confirma que el calculo de la distancia euclidiana es correcto para movimientos en el eje X, el eje Y y en diagonal (triangulo 3-4-5). Para comprobar la aptitud se realiza mediante el método ”pruebaAptitud()” el cual valida que “distanciaRuta()” cierre correctamente el ciclo y que “rutaApta()” calcule el valor correcto de aptitud. Y por último el método “pruebaClasificacionRutas()” confirma que la ruta con la menor distancia se coloque en el primer lugar.

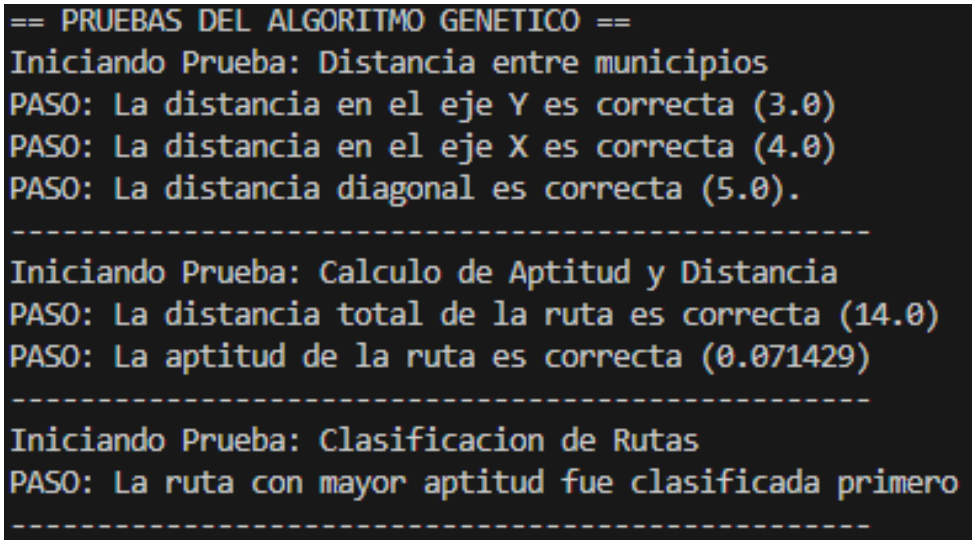

Figura 1: Resultado de las pruebas

2.7. Cambios

Cuadro 1: Coambios entre Reconstrucción y Lógica AG

	Reconstrucción	**Lógica AG**	**Resumen**
Arquitectura	Modularidad, POO. La lógica se separa por arhivos (municipio, Aptitud, operadorGenetico, etc.).	Todo el código (clases, funciones y ejecución) está contenido en un solo archivo.	Un enfoque orientado a POO facilita la reutilización, el mantenimiento y las pruebas.
Clase municipio	Contiene nombre, x, y. El nombre permite identificar los nodos para la visualización.	Solo contiene x, y. La lógica original no contiene nombre, lo que reduce la claridad de los datos.	El poner nombre como otro atributo de la clase municipio permite más claridad en resultados.
Nomenclatura	Utiliza `cruce` y `crucePoblacion`.	Renombra a `reproduccion` y `reproduccionPoblacion`.	Se utiliza una nomenclatura más acorde a los términos del algoritmo genético.

2.8. Resultados

Este resultado nos muestra que el algoritmo genético es eficaz en la minimización de la distancia, El proceso comenzó con una distancia inicial de 823.33, mostrando la aleatoriedad de la población inicial. El AG encontró de forma rápida una mejora logrando un descenso a 538.50 en las primeras 60 generaciones, gracias por la selección de las elite rutas y cruces. Después el proceso se volvió lento hasta que alrededor de la generación 140 se encontró el valor final óptimo 537.27. El algoritmo se estabilizó en esta solución hasta el final de las 500 generaciones, confirmando la ruta encontrada Salvador – ... – Angostura la cual es la mejor reduciendo así un 35 % la distancia con respecto al inicio.

```
Distancia Inicial:  823.33
Generacion   10  Distancia: 588.90
Generacion   20  Distancia: 553.34
Generacion   30  Distancia: 539.85
Generacion   40  Distancia: 539.69
Generacion   50  Distancia: 539.49
Generacion   60  Distancia: 538.50
Generacion   70  Distancia: 538.50
Generacion   80  Distancia: 538.06
Generacion   90  Distancia: 538.06
Generacion  100  Distancia: 538.06
Generacion  110  Distancia: 538.06
Generacion  120  Distancia: 538.06
Generacion  130  Distancia: 538.06
Generacion  140  Distancia: 537.27
```

Figura 2: Resultado consola parte 1

```
Generacion  480  Distancia: 537.27
Generacion  490  Distancia: 537.27
Generacion  500  Distancia: 537.27

--- Resultados Finales ---
Distancia Final: 537.27
Mejor Ruta (Ciclo Cerrado):
   1: Salvador Alvarado (45.4642,9.19)
   2: Guasave (48.8566,2.3522)
   3: Los Mochis (51.5074,-0.1278)
   4: Sinaloa (43.6532,-79.3832)
   5: Badiraguato (37.7749,-122.4194)
   6: Mazatlan (34.0522,-118.2437)
   7: Culiacan (40.4168,-3.7038)
   8: El Fuerte (52.52,13.405)
   9: Ahome (55.7558,37.6173)
  10: Mocorito (39.9042,116.4074)
  11: Navolato (35.6895,139.6917)
  12: Elota (35.0116,135.7681)
  13: Concordia (34.6937,135.5023)
  14: Escuinapa (31.2304,121.4737)
  15: Angostura (41.9028,12.4964)
  16: Regresa a Salvador Alvarado (45.4642,9.19)

Distancia Optima: 537.27
```

Figura 3: Resultado consola parte 2

3. Conclusión

La implementación del algoritmo genético para resolver el problema del agente viajero permitió comprender cómo los principios de la evolución pueden aplicarse a la optimización computacional. A través de las clases y funciones, fue posible observar cómo la población de rutas evoluciona gradualmente hacia soluciones más eficientes, reduciendo la distancia total recorrida.

Durante la reconstrucción del algoritmo genético se aplicaron mejoras en la estructura del código y la nomenclatura de funciones, haciendo el programa más legible, reutilizable y fácil de mantener, en donde cada clase cumple un propósito específico dentro del algoritmo: desde la representación de los municipios y el cálculo de distancias, hasta la evaluación de la aptitud, selección, cruce y mutación de rutas. Además, la realización de pruebas unitarias contribuyó a reforzar la importancia de validar el correcto funcionamiento del código antes de ejecutar procesos complejos de optimización.

En conclusión, este trabajo permitió comprender cómo los algoritmos genéticos pueden utilizarse para resolver problemas reales de búsqueda y optimización.