INSTITUTO TECNOLÓGICO DE CULIACÁN

Ingeniería en Sistemas Computacionales

Tópicos de Inteligencia Artificial

Hora: 12:00 – 01:00 PM

Tarea de validación 3:

Algoritmo Genético para TSP

Equipo:

Peña López Miguel Ángel

Robles Rios Jacquelin

Docente:

Mora Félix Zuriel Dathan

Culiacán, Sinaloa

08/11/2025

1 Tabla de contenido

2 Implementación de Algoritmo Genético para el Problema del Viajante (TSP)

2.1 1.1 El Problema del Viajante (TSP)

El Problema del Viajante (Traveling Salesperson Problem o TSP) es un problema de optimización combinatoria. El objetivo es encontrar el camino más corto posible que conecta un conjunto de N puntos (ciudades), visita cada uno exactamente una vez y regresa al punto de partida.

Naturaleza y Complejidad:

El TSP es un problema NP-Hard. El número de posibles rutas crece factorialmente con el número de ciudades: $\left(\frac{(N-1)!}{2}\right)$.

Para 10 ciudades: $181{,}440\ rutas\ posibles$.

Para 20 ciudades: $1.2\ x\ 10^{12}\ Rutas\ posibles$.

Debido a esta explosión combinatoria, los algoritmos exactos (como la programación dinámica o *Branch and Bound*) se vuelven intratables para un gran número de ciudades. Los Algoritmos Genéticos (AG) se clasifican como **heurísticas** o **metaheurísticas**, diseñadas para encontrar soluciones **subóptimas** de alta calidad en un tiempo razonable.

3 Descripción del algoritmo genético

El algoritmo g**enético** es una técnica de optimización inspirada en el proceso de selección natural y la genética. Los pasos para seguir el AG son:

3.1 Construcción del Algoritmo Genético

3.1.1 Población Inicial:

Este es un conjunto de individuos que se llama población. Cada individuo es una solución que se desea resolver. Un individuo se caracteriza por un conjunto de parámetros, variables, conocidos como genes. Los genes se unen en una cadena para formar un cromosoma, solución.

- **Representación:** Una lista (o arreglo) de objetos Ciudad.
- **Gen:** Cada ciudad en la lista.
- **Cromosoma:** La secuencia completa de ciudades (la ruta).

Requisito Clave: Dado que cada ciudad debe visitarse una sola vez, la representación es una **permutación** de las ciudades disponibles.

3.1.2 Función de Aptitud (Fitness Function)

Es este paso, el algoritmo capaz de determinar que es lo que hace una solución sea más adecuada que otra. Este determina por la función fitness o de aptitud.

El objetivo de la función es evaluar la viabilidad genética de las soluciones dentro de la población, colocando a aquellos con los rasgos genéticos más viables, favorables y superiores a la cabeza de la lista. Esta función se somete a ligeras modificaciones. Es el diferenciador en la población que separa a los individuos más fuerte de los débiles

$$Aptitud: \frac{1}{Distancia\ Total\ de\ la\ Ruta}$$

$$\boldsymbol{Distancia\ Total\ de\ la\ Ruta}$$

Calculada como la suma de las distancias euclidianas entre ciudades consecutivas, más la distancia de la última ciudad a la primera (cerrando el ciclo).

$$Distancia = \sum_{i=1}^{N+1} Distancia(C_1\ C_{i+1})$$

Justificación: El TSP es un problema de **minimización** (minimizar distancia). Los AG están diseñados para la **maximización**. Al usar el inverso de la distancia, una ruta más corta produce un valor de aptitud más alto, lo que permite que el AG la seleccione con mayor probabilidad.

3.1.3 Selección

Durante cada generación sucesiva, una proporción de la población existente es seleccionada para criar una nueva generación

Elitismo y s**elección por ruleta:**

- **Elitismo:** Garantiza la convergencia al copiar los mejores individuos a la siguiente generación, impidiendo que el mejor resultado se pierda.
- **Ruleta:** Asegura la diversidad al dar oportunidad de reproducción a individuos con aptitud alta, pero no necesariamente los mejores, evitando un mínimo local.

3.1.4 Cruce

Reproducción de los cromosomas padres que producirán a los descendientes:

Es el cruce estándar para problemas basados en permutaciones. Mantiene el orden relativo de las ciudades en los padres, crucial para el TSP, y asegura que el hijo sea una permutación válida sin duplicados ni omisiones.

3.1.5 Mutación:

Después de la selección y cruce, se tiene una nueva población llena de individuos, algunos serán iguales y otros se producen por el cruce, aquí se agrega una posibilidad de mutación para asegurar que no sean iguales. Se recorre los alelos de los genes y el seleccionado puede cambiarlo por una pequeña cantidad, suelen ser decimas de porcentaje.

En nuestro código: Es simple y eficaz. Con una baja probabilidad, se intercambian dos ciudades aleatorias en la ruta. Esto introduce pequeñas variaciones y previene que la población se estanque en un mínimo local (convergencia prematura).

3.2 Justificación de Parámetros y sus Valores

Los parámetros del Algoritmo Genético son cruciales para el equilibrio entre la e**xploración** (búsqueda de nuevas áreas de solución) y la e**xplotación** (refinamiento de soluciones existentes).

Parámetro	Rango idóneo	Valor seleccionado
tamano_poblacion	50-500	100
generaciones	50-5000	100
Indiv_seleccionados	1% - 30%	30
razón_mutacion	0.001 - 0.1	0.1

- **tamano_poblacion:** Una población mayor aumenta la diversidad (mejor exploración), pero ralentiza la ejecución por generación. 100 es un buen equilibrio entre diversidad y velocidad para problemas de tamaño pequeño a mediano (cantidad de cromosomas en el AG).
- **generaciones:** Un valor bajo para una demostración y pruebas iniciales. Es suficiente para observar la convergencia del algoritmo. Para la solución final de un problema real, se necesitarían >= 500 generaciones. (cantidad de veces que se generarán cromosomas)
- **Indiv_seleccionados(Elitismo):** Un porcentaje de élite relativamente alto garantiza una fuerte **explotación** de los mejores resultados, forzando una convergencia rápida. Si fuera demasiado bajo, la mejor solución podría ser eliminada por malos cruces.
- **Razón_mutacion:** Un valor bajo es vital. Una tasa de mutación alta destruye las buenas soluciones encontradas por el cruce, convirtiendo el AG en una búsqueda aleatoria. Una tasa baja (1%) asegura una mínima **exploración** para escapar de mínimos locales sin sacrificar la convergencia.

4 Pruebas y compilaciones del algoritmo

En esta sección se detalla el conjunto de pruebas diseñadas para verificar el correcto funcionamiento de la lógica en las funciones implementadas para el algoritmo genético.

4.1 Prueba de aptitud (Fitness)

Para verificar que la funcion **`distanciaRuta`** (dentro de la clase Aptitud), se creo un ejemplo simple de 3 ciudades configuradas de la siguiente forma:

```
a = Ciudad(0, 0)
b = Ciudad(3, 0)
c = Ciudad(0, 4)
```

Resultando en una **`ruta[(0,0),(3,0),(0,4)]`**, donde la distancia total es sencilla de calcular, resultando en A->B) + 5 (B->C) + 4 (C->A) = 12.

Resultado obtenido

```
--- 1. Prueba de Aptitud (Fitness) ---
Ruta de prueba: [(0,0), (3,0), (0,4)]
Distancia calculada: 12.0
Distancia esperada (manual): 12.0
La función de distancia funciona.
```

4.2 Prueba de cruce

Para asegurar que la función **`reproducción`** (cruce OX1) produzca un hijo válido, se debe producir una ruta que contenga todas las ciudades exactamente una vez. Para ello se debe verificar que el hijo generado tenga la misma longitud que los padres y que no contenga ciudades duplicadas ni le falte ninguna.

Método seguido:

1. Definimos dos padres de ejemplo, estos con un numero de ciudades pequeño, haciendolos mas fáciles de manejar y comprender.
2. Mandamos llamar la función **`reproduccion(P1, P2)`** para generar un descendiente (Hijo). La prueba consiste en una validación de integridad del individuo hijo resultante cumpliendo lo anteriormente mencionado.

No verificar esto correctamente resultaria en la contaminación de la población y el algoritmo fallaría sin encontrar una soluación válida.

Resultado obtenido

```
--- 2. Prueba de Cruce (reproduccion) ---
Progenitor 1: [(0,0), (1,1), (2,2), (3,3), (4,4)]
Progenitor 2: [(2,2), (4,4), (1,1), (0,0), (3,3)]
Hijo generado: [(1,1), (0,0), (3,3), (2,2), (4,4)]
Longitud correcta (debe ser 5): True
Contenido válido (sin duplicados/faltantes): True
El cruce OX1 produce hijos válidos.
```

4.3 Prueba de mutación

Para la verificacion de la función de mutación (mutación por intercambio o swap) se debe validar que se esta produciendo un individuo válido, para ello se realizo lo siguiente:

- Se creo un individuo sencillo de comprender y manejar, al cual se le aplicó la función de mutacion forzandola con un 1.0 (100%) de probabilidad para asegurar que ocurra al menos una mutación y que la función sea probada.

  Para ello al igual que en la prueba de cruce, al individuo mutado debe mantener su validez cumpliendo las condiciones de longitud e integridad de ciudades.

Una correcta mutación es clave para introducir diversidad y evitar la convergencia prematura en óptimos locales. Sin embargo, esta exploración no debe hacerse a costa de cambiar la estructura de la solución. El operador debe modificar el *orden* de la ruta, no el *contenido*.

Resultado obtenido

```
--- 3. Prueba de Mutación (mutacion) ---
Individuo original: [(0,0), (1,1), (2,2), (3,3), (4,4)]
Individuo mutado: [(2,2), (1,1), (4,4), (0,0), (3,3)]
Longitud correcta: True
Contenido válido: True
La mutación produce individuos válidos.
```

4.4 Prueba de selección

La validación del mecanismo de selecciónRutas se combina el elitismo y la selección por ruleta, para asi confirmar la selección hacia los individuos más aptos. Para realizar esta prueba se hizo lo siguiente:

- Se construyó una población falsa ya clasificada, donde el individuo con indice 0 posee una aptitud alta (100.0) y el resto una aptitud baja (1.0), posteriormente se ejecuto el proceso de selección varias veces y se contabilizó la frecuencia de selección de cada individuo.
  Para garantizar el éxito los resultados obtenidos deben de mostrar una distribución de selección fuerte hacia el individuo con mayor aptitud, para así demostrar que la selección por ruleta lo favoreció.

Esta prueba valida el principio fundamental de la "supervivencia del más apto". Si la selección no favoreciera clara y consistentemente a los mejores, el algoritmo no tendría dirección y seria incapaz de mejorar la calidad de las soluciones a través de las generaciones.

Resultado obtenido:

```
--- 4. Prueba de Selección (seleccionRutas) ---
Resultados de selección después de 100 ejecuciones:
Total de selecciones por individuo (de 400 posibles):
{0: 387, 1: 2, 2: 5, 3: 6}
El individuo más apto (0) fue seleccionado con mucha más frecuencia.
```