Prácticas de Búsqueda. Curso 2023-2024. Sesión 3. Algoritmos genéticos en aima-python.

Ramiro Varela, Francisco J. Gil-Gala. Irene Díaz, Juan Luis Mateo-Cerdán, Noelia Rico. Sistemas Inteligentes. Grado en Ingeniería Informática. EII. Universidad de Oviedo. Campus de los Catalanes. Oviedo

Resumen. En esta sesión, se trata de conocer la implementación de un algoritmo genético. Para ello se utilizará la demo y la implementación que incluye aima-python. En primer lugar, haremos algunos experimentos con un problema muy simple que nos ayudarán a entender la dinámica de estos algoritmos, y luego haremos las modificaciones necesarias para resolver un problema más difícil como es el TSP, y comparar los resultados con los que se obtienen con el algoritmo A*.

Palabras claves: Algoritmos Genéticos, TSP, codificación con permutaciones, cruce en dos puntos, fitness, elitismo.

1 Introducción

Los algoritmos genéticos son una clase de algoritmos evolutivos que basan su estrategia en las leyes de la evolución natural, principalmente la herencia genética y la adaptación al entorno, incluyendo la supervivencia del más fuerte. Pertenecen a una clase de algoritmos de búsqueda denominados metaheurísticas (heurísticas que organizan la aplicación de otras heurísticas) que se describen muy bien en el texto de E. G. Talbi [1], por ejemplo.

El proyecto aima-python incluye la implementación de un algoritmo genético que utiliza una codificación basada en listas de elementos de un alfabeto, con operadores de cruce y mutación simples, y una estrategia evolutiva también muy simple. La implementación se proporciona en el notebook `search3e-GeneticAlgorithm_2023_2024.ypynb`. Por otro lado, en el notebook `search3e-GeneticAlgorithm_2023_2024-ConPermutaciones.ypynb`. se proporciona una implementación basada en permutaciones sin repeticiones.

2 Ejercicios para esta práctica

En primer lugar, vamos a hacer algunos experimentos con el notebook search3e-`GeneticAlgorithm_2023_2024.ypynb`. Después, haremos experimentos con A* para el TSP en el notebook `search3e-GeneticAlgorithm_2023_2024-ConPermutaciones.ypynb`

Ejercicio 1. Visualizar detenidamente el contenido del notebook `GeneticAlgorithm_2023_2024.ypynb` para entender los detalles de la implementación de todos los elementos generales del algoritmo genético, en particular los detalles concretos de la implementación correspondientes a los tres problemas que se consideran:

a) Generación de cadenas
b) Coloreamiento de grafos
c) N-reinas

Ejercicio 2. Visualizar detenidamente el contenido del notebook `search3e-GeneticAlgorithm_2023_2024-ConPermutaciones.ypynb` para entender los detalles de la implementación correspondientes a las permutaciones sin repeticiones. Realiza algún experimento con el problema de las N-reinas y compáralos con los obtenidos en el notebook anterior. Observa detenidamente la implementación de los operadores `uniform_crossover` y `init_population`.

Ejercicio 3. La codificación basada en permutaciones permite adaptar el algoritmo genético a muchos problemas sin más que redefinir la función fitness, entre otros al TSP y al problema de las N-reinas. Para el segundo, cualquier permutación de 0,…,N-1 es un cromosoma válido, sin embargo para el TSP los cromosomas deberían ser permutaciones de N-1 ciudades, por ejemplo 1,…,N, tomando el 0 como ciudad de partida que se conecta a la primera y última de la permutación. En este ejercicio hay que hacer las modificaciones oportunas para que los cromosomas sean permutaciones de 1,..,N.

Ejercicio 4. En este ejercicio, se trata de resolver el TSP. Para ello, utilizaremos la versión con permutaciones de 1,…,N, e introduciremos el grafo de conexiones para representar los datos del problema. Como en el caso de A*, los datos se obtendrán de un fichero con la estructura del fichero `gr17.tsp.gz` (que se puede descargar del repositorio TSPLIB). Para esto es de utilidad la clase Grafo de la implementación del TSP con A*. Implementaremos la función fitness_fn para que calcule el coste que representa un cromosoma (teniendo en cuenta que la ciudad 0 está conectada a la primera y a la última del cromosoma).

Ejercicio 5. Hacer algunas mejoras en el algoritmo: introducir elitismo y escalado del fitness. Incluir un operador de cruce que permita mantener mejor las características que los padres transmiten a sus descendientes. En concreto, implementar un operador de cruce el OX (`order_crossover`) de las diapositivas de teoría.

Ejercicio 6. Hacer un estudio experimental, considerando ejemplos como el anterior, con 17 ciudades, y alguno de mayor tamaño, y comparar los resultados que se obtienen con los dos operadores de cruce, `uniform_crossover` y `order_crossover`; y con los que se obtuvieron con la implementación del algoritmo A*.

Bibliografía

1. E. G. Talbi. Metaheuristics: from design to implementation. John Wiley & Sons. 2009.
2. TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/)