PRÁCTICA 5

Algoritmos geneticos

Inteligencia Artificial para la Ciencia de Datos

2º Curso de Ciencia e Ingeniería de Datos
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Aris Vazdekis Soria y Alejandra Ruiz de Adana Fleitas

MEMORIA SOBRE EL PROYECTO DE ALGORITMOS GENETICOS EN PYTHON

El proyecto se centró en la optimización de la organización de la cola de trenes en una estación ferroviaria de mercancías mediante el uso de algoritmos genéticos. El objetivo principal fue minimizar el tiempo de espera de los trenes en la cola, considerando tres tipos de operaciones (Carbón, Gas y Contenedores) y tres muelles especializados.

La implementación se realizó utilizando la biblioteca DEAP (Distributed Evolutionary Algorithms in Python) para algoritmos evolutivos en Python. Se definieron diversas clases y funciones para representar los elementos clave del problema y los operadores genéticos.

Se diseñaron clases como Tren y Cola_de_trenes para representar los trenes y la cola de trenes, respectivamente. Se definieron funciones como producir_lista_trenes_aleatoria para generar una lista aleatoria de trenes y crear_cola_de_trenes para reorganizar aleatoriamente la lista de trenes, simulando su llegada a la estación.

Se implementó la función fitness para evaluar la aptitud de cada individuo, calculando el tiempo total requerido para la organización de la cola de trenes.

El algoritmo genético se ejecutó utilizando la función eaSimple, donde se configuraron parámetros como la población inicial, las probabilidades de cruce y mutación, el número de generaciones, entre otros.

Finalmente, se obtuvo el mejor individuo encontrado, junto con su tiempo total de organización, y se graficó la evolución de la aptitud a lo largo de las generaciones para analizar el rendimiento del algoritmo.

ÍNDICE

Clases y funciones

En esta sección se definen las clases y funciones necesarias para el desarrollo del algoritmo genético y la simulación del problema de la cola de trenes.

- **class Tren**

  Define un objeto que contiene información sobre los vagones, operación y matrícula de un tren. La función __str__ permite imprimir de manera legible la información de un objeto Tren.

- **class Cola_de_trenes**

  Representa una lista especializada para trenes. Al crear un objeto Cola_de_trenes y imprimirlo, se muestra una representación legible de la cola de trenes

- **def producir_lista_trenes_aleatoria**

  Genera una lista de trenes con características aleatorias, como el número de vagones y el tipo de carga.

- **def crear_colo_de_trenes**

  Crea una cola de trenes reorganizando aleatoriamente la lista de trenes original

- **def fitness**

  Simula el proceso de carga/descarga de trenes y calcula el tiempo total requerido

Definición y configuración del algoritmo

- **FitnessMin**

  Es una clase que define el tipo de aptitud utilizada en el algoritmo genético. En este caso, estamos tratando de minimizar una función objetivo, por lo que se utiliza *FitnessMin* para indicar que valores más bajos de la función de aptitud son mejores

- **Individual**

  Es una clase que representa a un individuo en la población del algoritmo genético. En este caso, un individuo es una lista de elementos, que en

nuestro caso específico representan la secuencia en la que los trenes están organizados en la cola

- **indices**

  Es una función que genera una permutación aleatoria de los índices que representan los trenes en la cola. Esta función se utiliza para inicializar individuos de manera aleatoria.

- **individual**

  Es una función que utiliza tools.initIterate para inicializar un individuo utilizando la clase Individual creada anteriormente y los índices generados por la función indices

- **population**

  Es una función que utiliza tools.initRepeat como implementación para crear una población de individuos. Toma como argumentos el tipo de datos para representar la población (en este caso, una lista) y otra función registrada llamada individual que se utiliza para inicializar cada individuo de la población

- **evaluate**

  Es una función que toma un individuo como argumento y calcula su aptitud utilizando la función fitness definida anteriormente. Esta función se utiliza para evaluar la aptitud de cada individuo en la población durante el proceso de evolución

- **mate**

  Es una función que define el operador de cruce utilizado durante la evolución del algoritmo genético. En este caso, se utiliza tools.cxOrdered, que durante la operación de cruce selecciona un segmento aleatorio de los elementos de uno de los padres y se copia en el hijo resultante, manteniendo el orden relativo de los elementos en ese segmento. Luego, se rellenan los elementos faltantes en el hijo con los elementos restantes del otro padre, en el orden en que aparecen.

- **mutate**

  Es una función que define el operador de mutación utilizado durante la evolución del algoritmo genético. Aquí se utiliza tools.mutShuffleIndexes, que durante la operación de mutación selecciona uno o más índices aleatorios en el individuo y permuta los elementos en esos índices. Esto implica cambiar el orden de los elementos en la representación del individuo.

- **select**

  Es una función que define el operador de selección utilizado durante la evolución del algoritmo genético. En este caso, se utiliza tools.selTournament, que durante la operación de selección selecciona varios individuos de la población de manera aleatoria y los enfrenta en torneos. En cada torneo, se comparan los individuos y se elige el mejor de ellos para ser seleccionado como uno de los padres para la reproducción. El argumento tournsize=3 indica el tamaño del torneo, es decir, cuántos individuos se seleccionan para participar en cada torneo.

Ejecución del algoritmo

En la ejecución del algoritmo genético, primero creamos una población inicial de 50 individuos utilizando la función population de la caja de herramientas. Luego, establecemos un objeto HallOfFame para mantener un registro del mejor individuo encontrado durante la evolución. A continuación, creamos un objeto Statistics para realizar un seguimiento de las estadísticas de la población, incluyendo el promedio, mínimo y máximo de la aptitud en cada generación. Después, ejecutamos el algoritmo genético utilizando eaSimple, donde especificamos la población inicial, la caja de herramientas, las probabilidades de cruce y mutación, el número de generaciones, así como los objetos Statistics y HallOfFame para mantener un registro de los mejores individuos. Finalmente, imprimimos el mejor individuo encontrado junto con su aptitud (tiempo total).

Gráficos

En la sección de gráficos, primero creamos una lista de generaciones y luego extraemos las estadísticas de aptitud de la población en cada generación. Utilizamos estas estadísticas para trazar líneas que representan el promedio, mínimo y máximo de la aptitud en cada generación. Etiquetamos los ejes “x” e e “y” añadimos un título al gráfico. Finalmente, mostramos el gráfico.

EXPERIMENTACIÓN

Durante la experimentación del algoritmo genético para optimizar la organización de la cola de trenes en la estación ferroviaria de mercancías, se realizaron múltiples ejecuciones con diferentes configuraciones de parámetros. Se observó que con 500 generaciones, el mejor individuo encontrado tenía una disposición representada por la lista [67, 37, 63, ..., 94, 49, 23, 13, 56, 18, 70, 20, 81, 5, 38, 29, 52], con un tiempo total de espera en la cola de 625 minutos. Al aumentar el número de generaciones a 800, el mejor individuo encontrado cambió a [1, 78, 21, ..., 20, 88], con un tiempo total de espera reducido a 595 minutos. Sin embargo, al extender la ejecución a 1500 generaciones, el mejor individuo encontrado fue [61, 25, 47, ..., 79], con un tiempo total de espera en la cola de 627 minutos. Estos resultados sugieren que, aunque se puede observar una mejora en el tiempo de espera promedio con un mayor número de generaciones, no garantiza necesariamente una solución óptima en cada ejecución.

CONCLUSIONES

El uso de algoritmos genéticos para abordar el problema de optimización en la organización de la cola de trenes ha demostrado ser efectivo. Sin embargo, es importante tener en cuenta que un aumento en el número de generaciones no necesariamente conduce a una mejora significativa en la solución. Es fundamental realizar un análisis exhaustivo de los resultados y ajustar adecuadamente los parámetros del algoritmo para obtener una solución óptima en cada ejecución.

RECURSOS UTILIZADOS

El proyecto de Algoritmos Genéticos se llevó a cabo íntegramente en Google Colab. Los recursos utilizados incluyeron la biblioteca DEAP para implementar los algoritmos genéticos en Python, junto con la capacidad de búsqueda en línea para obtener información relevante.