Patrón de fondo

Descripción generada automáticamente

Alan Ramírez Rivera

21089241

Doc. Ernesto Ríos Willors

Algoritmo

Genético

**Encabezado**

INTRODUCCIÓN

Los algoritmos genéticos son una clase de algoritmos de optimización y búsqueda inspirados en los principios de la evolución biológica. Estos algoritmos son útiles para resolver una amplia gama de problemas de optimización, desde la ingeniería y la ciencia de datos hasta la robótica y la economía. La naturaleza flexible y adaptable de los algoritmos genéticos los convierte en una herramienta poderosa para abordar problemas complejos en los que otras técnicas de optimización pueden no ser efectivas.

En este reporte, se presenta la implementación de un algoritmo genético en el lenguaje de programación Python. El objetivo de este proyecto es resolver un problema específico mediante la aplicación de técnicas de optimización basadas en la evolución biológica. Se proporciona una descripción detallada del problema, así como una explicación exhaustiva del algoritmo genético implementado y sus componentes clave.

COMPONENTES DEL ALGORITMO

El algoritmo genético consta de varios componentes clave, cada uno de los cuales desempeña un papel fundamental en el proceso de optimización:

Inicialización de la Población: Se genera una población inicial de soluciones candidatas de manera aleatoria o mediante algún criterio específico.

Evaluación de las Soluciones: Cada solución candidata se evalúa utilizando una función de aptitud que cuantifica su calidad en relación con el problema dado.

Selección: Se seleccionan las mejores soluciones de la población actual para pasar a la siguiente generación.

Recombinación: Las soluciones seleccionadas se combinan para generar nuevos individuos, utilizando operadores de recombinación como el cruce de un solo punto o el cruce uniforme.

Mutación: Se introduce variabilidad en la población mediante la mutación, que altera aleatoriamente ciertos componentes de las soluciones.

Reemplazo: Los nuevos individuos reemplazan a los menos aptos en la población actual, asegurando que la población evolucione hacia soluciones más óptimas.

EXPLICACIÓN DEL CÓDIGO

1. Importación de la biblioteca ‘random’: se utiliza para generar números aleatorios, que son necesarios para la mutación y otros procesos aleatorios del algoritmo genético.

**import** random

1. Función obtener\_palabras(n): Esta función solicita al usuario ingresar un número n de palabras y devuelve una lista con esas palabras.

**def** obtener\_palabras(**n**)**:**

    palabras **=** []

**for** i **in** range(n)**:**

        palabra **=** input(f"Ingrese la palabra {i**+**1}: ")

        palabras**.**append(palabra)

**return** palabras

1. Función crear\_matriz(palabras): Esta función recibe una lista de palabras y crea una matriz donde cada palabra se coloca en una fila, y las filas se rellenan con guiones para que todas tengan la misma longitud.

**def** crear\_matriz(**palabras**)**:**

    longitud\_maxima **=** max(len(palabra) **for** palabra **in** palabras)

    matriz **=** []

**for** palabra **in** palabras**:**

        fila **=** list(palabra**.**ljust(longitud\_maxima**,** '-'))

        matriz**.**append(fila)

**return** matriz

1. Función clonar\_matrices (matriz, num\_clones): Esta función clona la matriz original varias veces para crear una población inicial de soluciones candidatas.

**def** clonar\_matrices(**matriz,** **num\_clones**)**:**

    matrices\_clonadas **=** [matriz[**:**] **for** \_ **in** range(num\_clones)]

**return** matrices\_clonadas

1. Función mutar(matrices): Esta función muta las matrices introduciendo variabilidad. Inserta guiones en posiciones aleatorias de las palabras.

**def** mutar(**matrices**)**:**

**for** matriz **in** matrices**:**

        longitud\_maxima **=** max(len(fila) **for** fila **in** matriz)

        posiciones\_aleatorias **=** [random**.**randint(0**,** longitud\_maxima) **for** \_ **in** range(len(matriz))]

        matriz\_expandida **=** []

**for** i**,** fila **in** enumerate(matriz)**:**

            nueva\_fila **=** []

**for** j **in** range(longitud\_maxima)**:**

**if** j **<** len(fila)**:**

                    nueva\_fila**.**append(fila[j])

**else:**

                    nueva\_fila**.**append(' ')

**if** j **==** posiciones\_aleatorias[i]**:**

                    nueva\_fila**.**insert(j**,** '-')

            matriz\_expandida**.**append(nueva\_fila)

        matriz[**:**] **=** matriz\_expandida

1. Función imprimir\_matriz(matriz): Esta función imprime una matriz en la consola.

**def** imprimir\_matriz(**matriz**)**:**

**for** fila **in** matriz**:**

        print(fila)

1. Texto

   Descripción generada automáticamenteFunción calcular\_puntaje\_matriz (matriz, caracteres\_a\_ignorar=None): Esta función calcula el puntaje de una matriz en función de ciertos criterios. Ignora los caracteres especificados en caracteres\_a\_ignorar.
2. Texto

   Descripción generada automáticamenteFunción cruzar\_matrices(mejores\_matrices): Esta función recibe las mejores matrices de una generación y produce nuevos individuos al combinar segmentos de los padres.
3. Texto

   Descripción generada automáticamenteFunción eliminar\_columnas\_de\_guiones(matriz): Esta función elimina las columnas de guiones de una matriz.
4. Obtención de palabras y creación de la matriz inicial: El programa solicita al usuario ingresar el número de palabras y las propias palabras. Luego, crea la matriz inicial.

*# Obtener palabras*

n **=** int(input("Ingrese el número de palabras: "))

palabras **=** obtener\_palabras(n)

*# Crear matriz*

matriz\_original **=** crear\_matriz(palabras)

1. Clonación de la matriz original: Se clona la matriz original varias veces para crear una población inicial de soluciones candidatas.

*# Pedir al usuario el número de clonaciones de la matriz original*

num\_clones **=** int(input("Ingrese el número de clonaciones de la matriz original: "))

*# Clonar la matriz original*

matrices\_clonadas **=** clonar\_matrices(matriz\_original**,** num\_clones)

1. Definición del número de generaciones: Se define el número de generaciones que el algoritmo genético llevará a cabo.

*# Definir el número de generaciones*

num\_generaciones **=** 50 *# cambiar esto al número deseado de generaciones*

1. Bucle principal de generaciones: En este bucle, se ejecutan las siguientes acciones en cada generación:

Se muta la matriz original y las matrices clonadas.

Se eliminan las columnas de guiones de la matriz original y las matrices clonadas.

Se calculan los puntajes y se seleccionan las mejores matrices.

Se cruzan las cuatro mejores matrices para generar nuevos individuos.

Se imprime la información relevante, como los puntajes de las mejores matrices y los hijos generados.

Se actualiza la matriz original para la siguiente generación.

*#EL CICLO IMPORTANTE*

**for** generacion **in** range(num\_generaciones)**:**

    print(f"Generación {generacion **+** 1}:")

*# Mutar la matriz original y las matrices clonadas*

    mutar([matriz\_original] **+** matrices\_clonadas)

*# Eliminar columnas de guiones de la matriz original y las matrices clonadas*

    matriz\_original **=** eliminar\_columnas\_de\_guiones(matriz\_original)

**for** i**,** matriz **in** enumerate(matrices\_clonadas)**:**

        matrices\_clonadas[i] **=** eliminar\_columnas\_de\_guiones(matriz)

*# Calcular puntajes y seleccionar las mejores matrices*

    puntajes **=** [calcular\_puntaje\_matriz(matriz) **for** matriz **in** [matriz\_original] **+** matrices\_clonadas]

    matrices\_ordenadas **=** sorted(zip([matriz\_original] **+** matrices\_clonadas**,** puntajes)**,** **key=**lambda **x:** x[1]**,** **reverse=**True)

    mejores\_matrices **=** [matriz **for** matriz**,** \_ **in** matrices\_ordenadas][**:**4]

*# Cruzar las cuatro mejores matrices*

    hijos **=** cruzar\_matrices(mejores\_matrices)

*# Imprimir los puntajes de las mejores matrices en esta generación*

    print("Puntajes de las mejores matrices:")

**for** i**,** (\_**,** puntaje) **in** enumerate(matrices\_ordenadas[**:**4])**:**

        print(f"Matriz {i **+** 1}: Puntaje = {puntaje}")

*# Imprimir los hijos generados en esta generación*

    print("Hijos generados:")

**for** i**,** hijo **in** enumerate(hijos)**:**

        print(f"Hijo {i **+** 1}:")

        imprimir\_matriz(hijo)

        print()

*# Actualizar la matriz original para la siguiente generación*

    matriz\_original **=** mejores\_matrices[0]  *# Podrías elegir otra estrategia de selección aquí*

    print("\n")

Imagen que contiene Calendario

Descripción generada automáticamentePRUEBAS

Imagen que contiene Texto

Descripción generada automáticamente

Como podemos ver el código se ejecuto correctamente, en este caso se hicieron 50 Generaciones con las palabras: MURCIELAGO, CAPACITACION Y SOLIDARIDAD. Como se puede ver en los puntajes aumentaron muy bien para las generaciones que pasaron y el número de clonaciones inicial. En la primera generación se tenia un total de 3,2,2,1 que es igual a 7 puntos y en la generación 50 se obtiene 3,3,3,2 que es igual a 10 puntos.

CONCLUSION

En resumen, la implementación del algoritmo genético en Python proporciona una solución efectiva para el problema planteado. La flexibilidad y adaptabilidad de los algoritmos genéticos los convierten en una herramienta poderosa para abordar una amplia gama de problemas de optimización en diversas áreas. La implementación en Python permite una fácil comprensión, modificación y extensión del algoritmo para adaptarse a diferentes problemas y requisitos específicos.