Algoritmos Genéticos

Dante Irarrázabal

1. Introducción

Los algoritmos genéticos son una técnica de optimización que se basa en la evolución biológica. Fueron propuestos por John Holland en la década de 1960 y desde entonces han sido ampliamente utilizados en diversas aplicaciones de optimización y búsqueda. Se utilizan para encontrar soluciones aproximadas a problemas complejos, como la optimización, la planificación y la selección.

2. Conceptos claves

- Población: Una población en un algoritmo genético es un conjunto de soluciones candidatas (individuos). Cada individuo representa una posible solución al problema.
- Cromosoma: Un cromosoma es una representación de un individuo en un AG. Puede ser una cadena de bits, números reales, cadenas de texto, o cualquier otra representación adecuada para el problema.
- Gen: Un gen es una parte de un cromosoma que codifica una característica o propiedad específica. Los genes pueden ser números, letras, o cualquier otro valor que tenga sentido en el contexto del problema.
- Función Fitness: La función fitness (o función de adaptación) evalúa qué tan buena es una solución candidata. Cuanto mayor sea el valor de la función fitness, mejor será la solución. La función fitness es específica para el problema que se está abordando y guía la búsqueda hacia soluciones óptimas.
- Selección: La selección es el proceso mediante el cual se eligen individuos de la población para reproducirse y crear una nueva generación. Los individuos con un mejor valor de fitness tienen una mayor probabilidad de ser seleccionados.
- Cruzamiento (Crossover): El cruzamiento implica tomar dos cromosomas padres y combinarlos para crear uno o más cromosomas descendientes. Dependiendo de cómo se implemente, esto puede implicar el intercambio de segmentos de los padres para producir descendientes con características de ambos.
- Mutación: La mutación es un proceso que introduce pequeñas modificaciones aleatorias en los cromosomas de la población. Esto ayuda a introducir diversidad en la población y evita que el algoritmo se estanque en una solución subóptima.
- Elitismo: El elitismo es una estrategia que implica mantener a los mejores individuos de una generación en la siguiente generación sin cambios. Esto garantiza que las soluciones de alta calidad no se pierdan a lo largo de las generaciones.
- Generaciones: Los AG funcionan en ciclos de generaciones. En cada generación, se seleccionan, cruzan y mutan individuos para crear la siguiente generación. El proceso se repite hasta que se alcanza un criterio de parada, como un número máximo de generaciones o una solución suficientemente buena.

- Convergencia: La convergencia se refiere al proceso en el que la población de individuos tiende a mejorar gradualmente y converger hacia una solución óptima a medida que avanza el algoritmo.
- Parámetros del AG: Los AG tienen varios parámetros clave, como el tamaño de la población, la tasa de mutación, la tasa de cruzamiento, entre otros. Estos parámetros afectan el rendimiento del algoritmo y deben ajustarse según el problema en cuestión.
- Representación del problema: La elección de la representación de un problema (cromosomas y genes) es crucial para el éxito de un AG. Debe ser adecuada para el tipo de problema que se está resolviendo.

3. Función Fitness

La función fitness es uno de los elementos más críticos en un algoritmo genético. Es una función que evalúa cuán buena o mala es una solución candidata (un cromosoma) en relación a la optimización del problema que estás tratando de resolver. La función fitness asigna un valor numérico a cada solución candidata, donde un valor más alto generalmente indica una mejor solución.

Aquí hay algunos aspectos clave relacionados con la función fitness:

- Evaluación de Soluciones: La función fitness evalúa cuán cerca está una solución candidata de ser una solución óptima o deseada para el problema en cuestión. La evaluación puede ser tan simple o tan compleja como lo requiera el problema.
- Valor Objetivo: La función fitness se diseña para maximizar o minimizar un valor objetivo. En algunos problemas, se busca maximizar el valor fitness (como en la maximización de beneficios), mientras que en otros se busca minimizarlo (como en la minimización de costos o distancias).
- Función Objetivo: La función fitness es también conocida como función objetivo, función de aptitud o función de evaluación. Es crucial definirla adecuadamente para que refleje de manera precisa la calidad de una solución candidata.
- Selección y Evolución: La función fitness se utiliza en la etapa de selección para determinar qué soluciones candidatas tienen una mayor probabilidad de convertirse en padres. Las soluciones con valores fitness más altos generalmente tienen una mayor probabilidad de ser seleccionadas.
- Convergencia: La función fitness influye en la convergencia del algoritmo. Si la función fitness se define de manera efectiva, el algoritmo tenderá a converger hacia soluciones óptimas o cercanas a óptimas.
- Normalización: En ocasiones, se realiza una normalización de los valores fitness para asegurarse de que todos los valores estén en la misma escala y sean comparables.

  La elección y diseño de la función fitness es fundamental en la aplicación de algoritmos genéticos, ya que afecta directamente la calidad de las soluciones encontradas. Es importante entender bien el problema que estás abordando para definir una función fitness adecuada y representativa del mismo.

4. Técnicas de selección

- Selección por Ruleta (Roulette Wheel Selection):

  En esta técnica, se asigna a cada individuo una probabilidad de selección proporcional a su valor de fitness. Los individuos con un mejor valor de fitness tienen una mayor probabilidad de ser seleccionados.

  El proceso es similar a lanzar una ruleta, donde los individuos con un valor de fitness más alto ocupan más espacio en la rueda y, por lo tanto, tienen más posibilidades de ser seleccionados.

La selección por ruleta es simple de implementar y eficaz cuando el rango de valores de fitness es conocido y no cambia significativamente a lo largo de las generaciones.

- Selección por Torneo (Tournament Selection):

  En esta técnica, se seleccionan al azar un número fijo de individuos de la población y se compara su valor de fitness. El individuo con el mejor valor de fitness en el torneo es seleccionado como padre.

  La ventaja de la selección por torneo es que no requiere conocer el rango de valores de fitness y es menos sensible a la escala de la función fitness. Puedes ajustar el tamaño del torneo para controlar la presión de selección. Torneos más grandes tienden a favorecer a los mejores individuos.

- Selección por Ranking (Rank-Based Selection):

  En este enfoque, los individuos se clasifican según su valor de fitness en lugar de utilizar los valores exactos. Los rangos se utilizan para determinar las probabilidades de selección. Los individuos mejor clasificados tienen una mayor probabilidad de ser seleccionados, pero esta probabilidad no depende directamente del valor de fitness. Esto puede ayudar a evitar la convergencia temprana y promover la diversidad en la población.

- Selección Proporcional a la Distancia (Stochastic Universal Sampling):

  Similar a la selección por ruleta, pero en lugar de asignar una probabilidad a cada individuo, se divide el espacio de probabilidad en sectores y se selecciona múltiples padres a la vez. Esta técnica garantiza que los padres seleccionados estén equitativamente espaciados a lo largo del rango de fitness, lo que puede ayudar a explorar más diversidad.

5. Consideraciones etapa selección

En la etapa de selección de un algoritmo genético, generalmente no se selecciona un número fijo de padres y se mantiene el tamaño de la población constante. En cambio, se seleccionan los padres y, posteriormente, se utilizan para crear la descendencia que reemplazará a la población actual.

Flujo etapa de selección:

- Se evalúa la función fitness de todos los individuos en la población actual.
- Se seleccionan los padres según la técnica de selección elegida (ruleta, torneo, etc.). Por ejemplo, puedes seleccionar un subconjunto de la población actual para ser padres.
- Los padres seleccionados se utilizan para crear la descendencia mediante operadores de cruzamiento y mutación.
- La descendencia reemplaza a la población actual.

6. Cruzamiento

El cruzamiento es el proceso en el que se combinan dos cromosomas padres para generar descendencia.

- Selección de Padres: Primero, se seleccionan dos cromosomas padres de la población actual, generalmente mediante alguna técnica de selección, como la ruleta, el torneo, o cualquier otra que prefieras.
- Punto de Cruzamiento: Luego, se elige un punto de cruzamiento en los cromosomas padres. El punto de cruzamiento es un índice que indica dónde se cortarán los padres y dónde se combinarán sus segmentos.

- Cruzamiento de un Solo Punto: En el cruzamiento de un solo punto, los segmentos de los cromosomas padres se intercambian en el punto de cruzamiento para crear dos descendientes. Por ejemplo, si tienes dos padres con los siguientes cromosomas:

  Padre 1: 10101010 Padre 2: 00110011

  Y el punto de cruzamiento se elige en el tercer índice, obtendrías los siguientes descendientes:

  Descendiente 1: 10110011 Descendiente 2: 00101010

- Cruzamiento de Dos Puntos: El cruzamiento de dos puntos es similar, pero se eligen dos puntos de cruzamiento en lugar de uno. Los segmentos entre los dos puntos se intercambian entre los padres. Esto da como resultado una mayor diversidad en la descendencia.

- Cruzamiento Uniforme: En el cruzamiento uniforme, en lugar de intercambiar segmentos enteros, se selecciona cada bit de los padres con una probabilidad del 50

- Otros Operadores de Cruzamiento: Además de los mencionados, existen varios otros operadores de cruzamiento más complejos, como el cruzamiento de múltiples puntos, el cruzamiento aritmético, el cruzamiento basado en máscaras, entre otros. La elección del operador de cruzamiento dependerá del problema que estés resolviendo y de tus objetivos de optimización.

  El objetivo del cruzamiento es combinar las características de los padres para producir descendientes que hereden lo mejor de ambos. Esto ayuda a explorar el espacio de búsqueda y encontrar soluciones prometedoras. La elección del operador de cruzamiento y la forma en que se aplica pueden influir en el rendimiento del algoritmo genético, por lo que es importante experimentar y ajustar estos aspectos.

7. Mutación en Algoritmos Genéticos

La mutación es un operador utilizado en algoritmos genéticos para introducir variabilidad genética en la población. A través de la mutación, se pueden generar cromosomas con características nuevas y diferentes a las de los padres. Aquí tienes una descripción de cómo funciona:

- Selección de Cromosomas: Primero, se selecciona un cromosoma de la población actual.

- Aplicación de la Mutación: En el proceso de mutación, uno o varios bits del cromosoma se modifican aleatoriamente. La tasa de mutación es un parámetro importante que determina con qué frecuencia se realiza la mutación. Una tasa de mutación baja significa que la mutación ocurre raramente, mientras que una tasa alta significa que ocurre con mayor frecuencia.

- Efecto de la Mutación: La mutación introduce variabilidad genética en la población. Puede ser útil para explorar regiones del espacio de búsqueda que de otra manera serían inaccesibles. Sin embargo, la mutación también debe ser aplicada con cautela, ya que una tasa de mutación muy alta puede hacer que la población evolucione demasiado aleatoriamente y se aleje de soluciones óptimas.

- Ubicación de la Mutación: Los bits que se mutan pueden seleccionarse al azar o con un patrón específico, dependiendo de cómo desees aplicar la mutación en tu algoritmo.

- Tipo de Mutación: Además de la mutación de un solo bit, hay otros tipos de mutación más complejos. Por ejemplo, la inversión de bits implica seleccionar un segmento del cromosoma y revertir el orden de los bits en ese segmento.

  La mutación es importante para evitar la convergencia temprana hacia soluciones subóptimas y promover la exploración en el espacio de búsqueda. Sin embargo, la tasa de mutación y la forma en que se aplica deben ajustarse cuidadosamente para garantizar un equilibrio adecuado entre la exploración y la explotación de la población.

8. Pseudocódigo de un Algoritmo Genético

A continuación, se muestra un pseudocódigo básico de un algoritmo genético:

Algorithm 1 Algoritmo Genético Básico

```
Inicializar población
while No se cumple el criterio de parada do
    Seleccionar padres
    Aplicar operadores de cruzamiento
    Aplicar operadores de mutación
    Evaluar la aptitud de la descendencia
    Seleccionar la nueva población
end while
```

9. Ejemplos de Aplicación

Un ejemplo común de aplicación de algoritmos genéticos es la resolución de problemas de optimización.

problemas sencillos que puedes abordar utilizando algoritmos genéticos:

- Problema de la Mochila (Knapsack Problem): Dado un conjunto de objetos con pesos y valores, el objetivo es encontrar la combinación óptima de objetos que quepa en una mochila con capacidad limitada y maximice el valor total de los objetos seleccionados.

  Supongamos que tienes una mochila con una capacidad máxima (por ejemplo, en términos de peso) y un conjunto de objetos, cada uno con un valor y un peso. El objetivo es seleccionar un subconjunto de objetos para colocar en la mochila de manera que maximices el valor total de los objetos seleccionados sin exceder la capacidad de la mochila.

- Problema del Viajante (Traveling Salesman Problem): Dado un conjunto de ciudades y las distancias entre ellas, el objetivo es encontrar la ruta más corta que visite cada ciudad exactamente una vez y regrese al punto de partida.

- Problema de la Asignación de Tareas (Job Scheduling): Dado un conjunto de trabajos y máquinas, el objetivo es encontrar la asignación óptima de trabajos a máquinas de manera que se minimice el tiempo total de procesamiento.

- Problema de la Planificación de Horarios (Timetabling): En este problema, se trata de programar eventos, clases o actividades en horarios específicos, teniendo en cuenta restricciones como disponibilidad de recursos y preferencias.

- Diseño de Circuitos Electrónicos (Circuit Design):
  El objetivo aquí es diseñar circuitos electrónicos que cumplan con ciertas especificaciones y optimicen parámetros como el consumo de energía o la latencia.

- Problema de la Secuenciación de ADN (DNA Sequencing): En bioinformática, se pueden utilizar AG para ensamblar secuencias de ADN a partir de fragmentos dados.

- Problema de Optimización de Parámetros (Parameter Tuning): Puedes aplicar AG para ajustar los parámetros de un algoritmo de aprendizaje automático o una simulación para maximizar su rendimiento.

- Diseño de Redes de Distribución (Network Design): En ingeniería, puedes utilizar AG para diseñar redes de distribución óptimas que minimicen costos o maximicen la eficiencia.

10. Abordando el problema de la mochila

- Representación de Cromosomas: Se representa un cromosoma como una cadena de bits, donde cada bit corresponde a un objeto. Un bit "1ïndica que el objeto está en la mochila, y un bit "0ïndica que no está.
- Función Fitness: La función fitness evalúa qué tan buena es una solución (conjunto de objetos en la mochila). Debe calcular el valor total de los objetos en la mochila y penalizar las soluciones que excedan la capacidad de la mochila.
- Selección: Puedes utilizar diferentes técnicas de selección, como la ruleta o el torneo, para seleccionar padres en función de su valor de fitness.
- Cruzamiento (Crossover): El cruzamiento puede ser de un solo punto, de dos puntos, o uniforme. El objetivo es combinar las características de los padres para crear descendientes que cumplan con las restricciones de capacidad.
- Mutación: La mutación implica cambiar aleatoriamente algunos bits en los cromosomas de los descendientes. Esto introduce diversidad en la población y evita soluciones subóptimas.
- Criterio de Parada: Define un criterio de parada, como un número máximo de generaciones o un valor de fitness objetivo.
- Elitismo: Puedes aplicar elitismo para mantener las mejores soluciones en cada generación sin cambios.

11. Algoritmo Genético para la Mochila (Python)

A continuación se muestra el código de un algoritmo genético en Python para resolver el problema de la mochila:

Listing 1: Algoritmo Genético para la Mochila

```
import random

# Parametros del problema

num_objetos = 10
capacidad_mochila = 50

# Valores y pesos de los objetos (valores fijos para este ejemplo)

valores = [10, 15, 7, 25, 8, 3, 20, 30, 12, 18]
pesos = [5, 10, 3, 15, 8, 2, 10, 20, 7, 10]

# Parametros del algoritmo genetico

tamanio_poblacion = 20
tasa_mutacion = 0.05
num_elitismo = 3
num_generaciones = 40

def fitness(cromosoma):
    valor_total = 0
    peso_total = 0

    for i in range(num_objetos):
```

```
            if cromosoma[i] == 1:
                valor_total += valores[i]
                peso_total += pesos[i]

        if peso_total > capacidad_mochila:
            # Penalizar soluciones
            # que exceden la capacidad de la mochila
            return 0
        else:
            return valor_total

def torneo(poblacion, fitness, tamanio_torneo):
    # Realiza una seleccion por torneo aleatorio,
    # seleccionando un subconjunto de la poblacion.
    torneo = random.sample(poblacion, tamanio_torneo)

# ----------------------------------------
    # torneo.sort(key=fitness, reverse=True)
    # return torneo[0]
# ----------------------------------------

    # Calcula el valor fitness para cada cromosoma en el torneo.
    valores_fitness = [fitness(cromosoma) for cromosoma in torneo]

    # Encuentra el indice del cromosoma con el mejor valor fitness.
    indice_mejor = valores_fitness.index(max(valores_fitness))

    # Retorna el cromosoma con el mejor valor fitness.
    return torneo[indice_mejor]

def cruzamiento(padre1, padre2):
    # Genera un punto de cruzamiento aleatorio
    # entre 1 y num_objetos - 1

    punto_cruz = random.randint(1, num_objetos - 1)

    # Combina los genes de los padres para crear dos descendientes

    descendiente1=padre1[:punto_cruz]+padre2[punto_cruz:]
    descendiente2=padre2[:punto_cruz]+padre1[punto_cruz:]

    return descendiente1, descendiente2


def mutacion(cromosoma):
    if random.random() < tasa_mutacion:
        punto_mut = random.randint(0, num_objetos - 1)
        cromosoma[punto_mut] = 1 - cromosoma[punto_mut]

def algoritmo_genetico():
    poblacion = [[random.randint(0, 1)
    for _ in range(num_objetos)]
    for _ in range(tamanio_poblacion)]

    for generacion in range(num_generaciones):
```

```
        nueva_poblacion = []

        # Elitismo

        poblacion.sort(key=lambda x: -fitness(x))
        nueva_poblacion.extend(poblacion[:num_elitismo])

        while len(nueva_poblacion) < tamanio_poblacion:
            padre1 = torneo(poblacion, fitness, 5)
            padre2 = torneo(poblacion, fitness, 5)
            descen1, descen2 =cruzamiento(padre1,padre2)
            mutacion(descen1)
            mutacion(descen2)
            nueva_poblacion.extend([descen1, descen2])

        poblacion = nueva_poblacion

    mejor_solucion = max(poblacion, key=fitness)
    mejor_valor = fitness(mejor_solucion)
    return mejor_solucion, mejor_valor

mejor_solucion, mejor_valor = algoritmo_genetico()

print("Mejor solucion encontrada:", mejor_solucion)
print("Valor total:", mejor_valor)
```

12. Conclusiones

Los algoritmos genéticos son una poderosa técnica de optimización inspirada en la evolución biológica. Pueden ser aplicados a una amplia gama de problemas de optimización y búsqueda.