Universidad Autónoma de Querétaro.
Facultad de Ingeniería.
Ingeniería Física 5to Semestre
Taller Optativo V.

Algorítmos Genéticos.

Alumnos:
José Ramón Gutiérrez Trujillo.
Juan Arturo Portillo Rodríguez.

Profesor: Marco Antonio Aceves Fernández.

22 de Septiembre del 2017.

Índice

1. Introducción.

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859). Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes "superindividuos", cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado, a cada individuo se le asigna un valor o puntuación. En la naturaleza esto equivaldra al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos (descendientes de los anteriores) los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergera hacia una solución óptima del problema.

2. Marco teórico.

2.1. Algoritmo Genético Simple.

El pseudicódigo de un algoritmo genético simple se representa de las siguiente manera:

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste la cual asigna un número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzaran generando dos hijos. El resultado de la combinación de las anteriores funciones será un conjunto de individuos, los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.

2.2. Población.

2.2.1. Tamaño de población.

Una cuestión que uno puede plantearse es la relacionada con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional.

Goldberg (1989) efectuó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para ristras de longitud l, con codificacion binaria, crece exponencialmente con el tamaño de la ristra.

Este resultado traería como consecuencia que la aplicabilidad de los Algoritmos Genéticos en problemas reales sera muy limitada, ya que resultarán no competitivos con otros métodos de optimización combinatoria. Alander (1992), basándose en evidencia empírica sugiere que un tamaño de población comprendida entre l y $2l$ es suciente para atacar con éxito los problemas considerados.

2.2.2. Población inicial.

Habitualmente la población inicial se escoge generando ristras al azar, pudiendo contener cada gen uno de los posibles valores del alfabeto con probabilidad uniforme. Qué es lo que sucederá si los individuos de la población inicial se obtuviesen como resultado de alguna técnica heurística o de optimización local. Se constata que esta inicializacion no aleatoria de la población inicial, puede acelerar la convergencia del Algoritmo Genético. Sin embargo en algunos casos la desventaja resulta ser la prematura convergencia del algoritmo, queriendo indicar con esto la convergencia hacia óptimos locales.

2.3. Función objetivo.

Dos aspectos que resultan cruciales en el comportamiento de los Algoritmos Genéticos son la determinación de una adecuada función de adaptación o función objetivo, así como la codificación utilizada.

Idealmente nos interesara construir funciones objetivo con "ciertas regularidades", es decir, funciones objetivo que veriquen que para dos individuos que se encuentren cercanos en el espacio de búsqueda, sus respectivos valores en las funciones objetivo sean similares. Por otra parte una dificultad en el comportamiento del Algoritmo Genético puede ser la existencia de gran cantidad de óptimos locales, así como el hecho de que el óptimo global se encuentre muy aislado.

La regla general para construir una buena función objetivo es que esta debe reflejar el valor del individuo de una manera "real", pero en muchos problemas de optimización combinatoria, donde existen gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos.

2.4. Selección.

La función de selección de padres más utilizada es la denominada función de selección proporcional a la función objetivo, en la cual cada individuo tiene una probabilidad de ser seleccionado como padre que es proporcional al valor de su función objetivo.

La selección por torneo, constituye un procedimiento de selección de padres muy extendido y en el cual la idea consiste en escoger al azar un número de individuos de la población, tamaño del torneo, seleccionar el mejor individuo de este grupo, y repetir el proceso hasta que el número de individuos seleccionados coincida con el tamaño de la población. Habitualmente el tamaño del torneo es 2, y en tal caso se ha utilizado una versión probabilística en la cual se permite la selección de individuos sin que necesariamente sean los mejores.

2.5. Cruce.

El Algoritmo Genético simple, utiliza el cruce basado en un punto, en el cual los dos individuos seleccionados para jugar el papel de padres, son recombinados por medio de la selección de un punto de corte, para posteriormente intercambiar las secciones que se encuentran a la derecha de dicho punto.

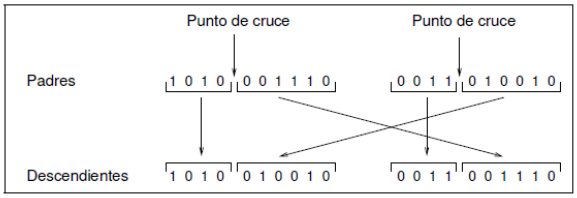

Figura 1: Operador de cruce basado en un punto.

2.6. Mutación.

La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoridad en la vecindad de los individuos de la población. Si bien se admite que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo (Davis, 1985).

Si bien en la mayoría de las implementaciones de Algoritmos Genéticos se asume que tanto la probabilidad de cruce como la de mutación permanecen constantes, algunos autores han obtenido mejores resultados experimentales modicando la probabilidad de mutación a medida que aumenta el número de iteraciones.

3. Desarrollo del algoritmo.

Se supone que los individuos, pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (referida como cromosoma).

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina fenotipo. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir, puede ser computada a partir del cromosoma, usando la función de evaluación.

La función de adaptación debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectua al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que esta basado en la ruleta sesgada. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogeran mas que de vez en cuando.

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los operadores de cruce y mutación.

El operador de cruce, coge dos padres seleccionados y corta sus ristras de cromosomas en una posición escogida al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos. Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto. Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

El operador de mutación se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, este último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de vital importancia para asegurar la convergencia de los Algoritmos Genéticos.

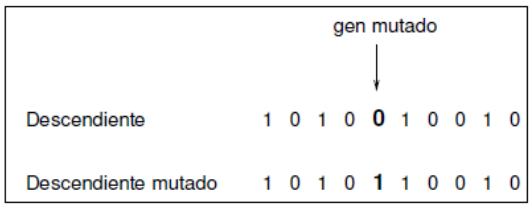

Figura 2: Operador de mutción.

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. El concepto de convergencia esta relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la poblacion comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha denición al caso en que al menos un $\beta\,\%$ de los individuos de la población hayan convergido.

4. Resultados.

Nuestro Algoritmo Genético lo realizamos Python, el código se muestra posteriormente, con nuestro código generamos una población inicial aleatoria de 50 individuos con 5 génes distintos, convertimos la población en decimal lo cual nos permite ponerla en el espacio de búsqueda que va de $[-1, 1]$, posteriormente ordenamos los valores por aptitud, selecciona los mejores individuos para posteriormente utilizar la función ruleta para cruzar los padres y así generar la segunda población, este paso se repite por 30 veces.

Consideramos que a nuestro programa aún le falta arreglos, ya que aunque las poblaciones posteriores a la original no empeoran pero tampoco mejoran, entonces nunca llegamos una convergencia en la que los individuos de la población sean mejores.

5. Código.

```
import numpy as np
import random
filas = 50
columnas = 5
def poblacion_binaria(filas, columnas):

    #esta funcion llena de 1 y 0 la matriz p
    p = np.zeros((filas, columnas))
    for i in range(0, filas):
        for j in range(0, columnas):
            p[i][j] = random.randrange(0, 2, 1)
    return

p pb = poblacion_binaria(filas, columnas)
def poblacion_decimal(pb,filas, columnas):

    #esta funcion convierte los valores de la matriz pb a decimales
    suma = 0
    d = np.zeros((filas,1))
    for i in range(0,filas):
        for j in reversed(range(0, columnas, 1)):
            suma += pb[i][j] * (2 ** ((columnas - 1) - j))
            d[i][0] = suma
            suma=0
    return d

pd = poblacion_decimal(pb, filas, columnas)
def convert(pd):


    #esta funcion convierte la poblacion decimal al espacio de busqueda que va de -1 a 1
    espacio_fase = np.zeros_like(pd)
    for i in range(0, 50):
        espacio_fase[i] = pd[i] / 31
        if espacio_fase[i] < 0.5:
            espacio_fase[i] = espacio_fase[i] - 1
    return espacio_fase

conv_estado_base = convert(pd)
def ordenamiento(conv_estado_base, pb):

    #se ordenan los valores por aptitud
    o = np.copy(conv_estado_base)
    bi = np.copy(pb)
    for i in range(len(o)):
        for j in range(len(o)-1):
            if abs(o[j, 0]) < abs(o[j+1, 0]):
              temp = o[j, 0]
```

```
            o[j, 0] = o[j+1, 0]
              o[j+1, 0] = temp
              temp1 = bi[[j],:]
              bi[[j],:] = bi[[j+1],:]
              bi[[j+1],:] = temp1
    return o, bi

ordenamiento, ordenamiento_binario = ordenamiento(conv_estado_base, pb)
ordenamiento = abs(ordenamiento)
def probabilidad_de_seleccion(ordenamiento):

    aptitud_total = 0
    for i in ordenamiento:
        aptitud_total += i
    aptitud_relativa = np.zeros_like(ordenamiento)
    for i in range(len(ordenamiento)):
        aptitud_relativa[i] = ordenamiento[i]/aptitud_total
        probabilidad = [sum(aptitud_relativa[:i+1])
        for i in range(len(aptitud_relativa))]
    return probabilidad

probabilidad = probabilidad_de_seleccion(ordenamiento)
def descendencia(pb, probabilidad, numero):

    generacion = np.zeros_like(pb)
    corte = 3
    generaciones = 0
    while generaciones < 30:
    #inicia la generacion de padres por ruleta
    for j in range(0, len(pb)-1, 2):
        padres = []
        for n in range(numero):
            r = random.random()
            for (i, individual) in enumerate(pb):
              if r <= probabilidad[i]:
                padres.append(list(individual))
                break
        padre_1 = np.asarray(padres[0:1])
        padre_2 = np.asarray(padres[1:2])
        print(padre_1)
        print(padre_2, end ='\n\n')

#se cruzan los padres


    hijo_1 = np.concatenate((padre_1[0:corte],padre_2[corte:4]), axis=0)
    hijo_2 = np.concatenate((padre_2[0:corte],padre_1[corte:4]), axis = 0)

#se llena la matriz generacion con los hijos

    generacion[j] = hijo_1
    generacion[j+1] = hijo_2
    generaciones += 1

return generacion
generacion = descendencia(pb, probabilidad, 2)
```