INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

ASIGNATURA:

Metaheurísticas

Actividad #22:

Guía Taller No.11

"Operadores de cruzamiento en Algoritmos Genéticos"

PROFESORA:

Dra. Yenny Villuendas Rey

PRESENTA:

Juan René Hernández Sánchez

Adriana Montserrat García Carrillo

Centro de Investigación en Computación
Instituto Politécnico Nacional

1. Introducción

Los algoritmos genéticos (AG) son algoritmos inspirados en la selección natural que han logrado semejar todo el proceso evolutivo de los seres vivos para resolver problemas de optimización, búsqueda y aprendizaje en las máquinas

La teoría de los algoritmos genéticos fue desarrollada por John H. Holland, sus colegas y sus estudiantes en la Universidad de Michigan, en un proyecto que buscaba explicar el proceso de adaptación natural de los sistemas y diseñar sistemas artificiales que conservaran los más importantes mecanismos de los sistemas naturales. Casi todos los conceptos de los algoritmos genéticos están basados en conceptos de biología y genética.

Las características de los seres vivos se encuentran registradas en los cromosomas y los genes, lo que hace que cada individuo sea diferente dentro de una población. En algoritmos genéticos los cromosomas se representan mediante cadenas (strings) y cada posición de la cadena representa un gen.

El primer paso desarrollar un algoritmo genético es codificar el parámetro deseado como una cadena y usar un alfabeto. La longitud de un cromosoma está determinada por el número de genes que lo conforman.

El conjunto ordenado de genes de un cromosoma se conoce como genotipo. De acuerdo con los valores que tengan los genes de una persona podremos decir que es alta o baja, de ojos azules o verdes, de cabello liso o rizado, etc. Esto se denomina fenotipo, y en los algoritmos genéticos corresponde a la decodificación de un cromosoma.

Cada cromosoma es una solución, buena o mala, del problema. A medida que el algoritmo genético se va desarrollando las soluciones se acercan cada vez más a la óptima.

Simulando los sistemas de selección natural a los que se encuentran sometidos los organismos vivos, los algoritmos genéticos utilizan tres operadores básicos para solucionar el problema: Selección, Cruzamiento y Mutación.

2. Desarrollo

Asignatura: Metaheurísticas
Actividad No.22
Guía Taller No.11
Título: Operadores de selección en Algoritmos Genéticos

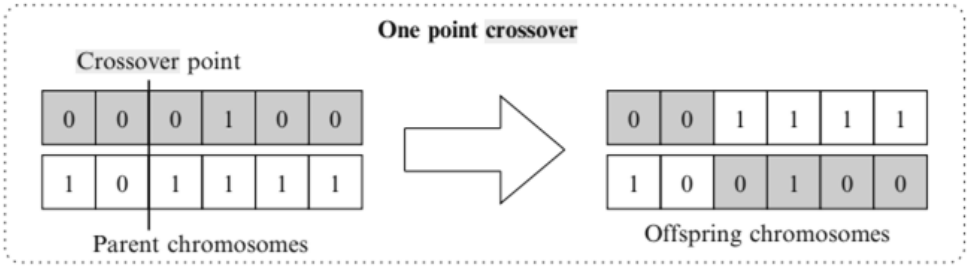
Contenido:

- Métodos heurísticos de solución de problemas.
- Algoritmos Genéticos
- Operadores de cruzamiento

Objetivo: Implementar operadores de selección para algoritmos genéticos, en lenguajes de alto nivel, para la solución de problemas de la profesión.

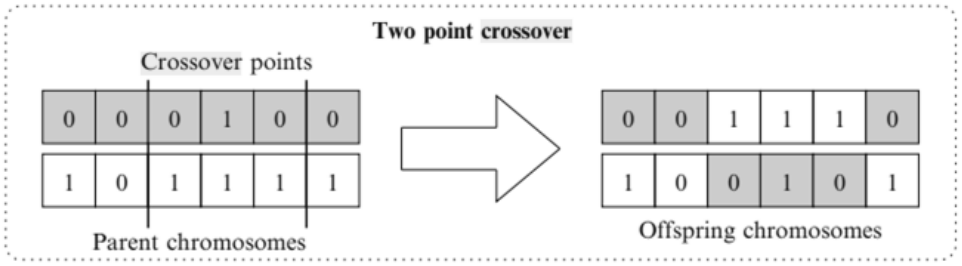
1. Explique el funcionamiento del operador de cruzamiento de un punto de corte.

Para dos individuos seleccionados para realizar el cruzamiento, se asignará un punto entre 1 y l-1 de forma aleatoria, donde l es la longitud de cromosoma. Esto significa generar un entero aleatorio en el rango [1,l-1]. Los genes después del punto son cambiados entre los padres y los cromosomas resultantes son los descendientes. Los descendientes también tienen cromosomas de longitud l.



2. Explique el funcionamiento del operador de cruzamiento de dos puntos de corte.

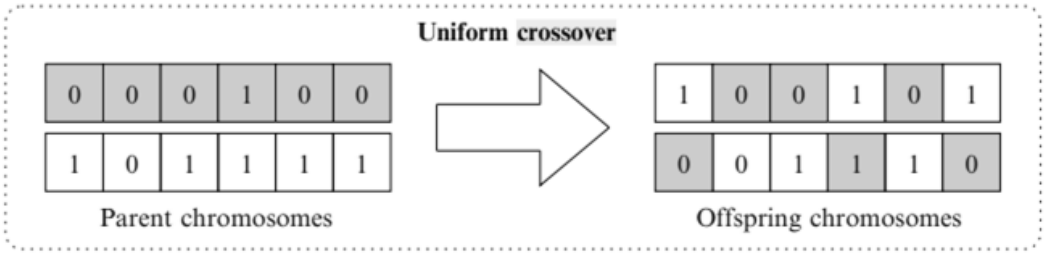
Para dos individuos seleccionados para realizar el cruzamiento de dos puntos, se asignarán dos puntos diferentes entre 1 y l-1 de forma aleatoria, donde l es la longitud de cromosoma. Esto significa generar un entero aleatorio en el rango [1,l-1]. Los genes después del punto son cambiados entre los padres y los cromosomas resultantes son los descendientes. Los descendientes también tienen cromosomas de longitud l.



3. Explique el funcionamiento del operador de cruzamiento uniforme

Un cruzamiento uniforme o también conocido como cruzamiento discreto intercambia información entre los padres de una manera diferente. Se suponen dos individuos, por ejemplo, A y B son seleccionados para realizar el cruzamiento uniforme. Cada gen tiene la probabilidad de 0.5 de heredar el gen de A, caso contrario se hereda el gen de B. Si se quiere generar dos descendientes a partir de dos padres, entonces cada gen del segundo descendiente puede ser seleccionado inversamente al correspondiente gen del primer descendiente. El cruzamiento uniforme puede utilizar múltiples padres para generar múltiples hijos.

Nota: El cruzamiento uniforme parametrizado consiste en tener dos padres y obtener un descendiente. Este cruzamiento maneja una probabilidad diferente de 0.5



4. Explique el funcionamiento del operador de cruzamiento aritmético.

Existen dos tipos de cruzamiento aritmético: completo y local.

- Cruzamiento aritmético completo

Se toman dos individuos y se representan como vectores, $x^1 = x_1^1, x_2^1, \dots, x_n^1$ y $x^2 = x_1^2, x_2^2, \dots, x_n^2$ su resultado del cruzamiento aritmético está dado por la siguiente expresión

$$y = \alpha x^1 + (1-\alpha)x^2, \text{ donde } \alpha \sim U(0,1)$$

La descendencia es igual a $y = y_1, y_2, \dots, y_n$

En caso de que se quiera tener dos descendientes de dos padres, se realiza las siguientes ecuaciones:

$$y^1 = \alpha x^1 + (1-\alpha)x^2, \text{ donde } \alpha \sim U(0,1)$$

$$y^2 = \beta x^1 + (1-\beta)x^2, \text{ donde } \beta = (1-\alpha)$$

- Cruzamiento aritmético local

α en lugar de ser un escalar es un vector y se exprese de la siguiente forma: $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$, donde $\alpha_i \sim U(0,1)$

Se realiza la siguiente ecuación para el cruzamiento aritmético local:

$$y = \alpha \cdot x^1 + (1-\alpha) \cdot x^2$$

5. Explique el funcionamiento del operador de cruzamiento blx-α

Se sugirió el BLX para expandir el rango del cruzamiento aritmético. El BLX se realiza al nivel de genes. Para los genes x_i^1 y x_i^2, se supone $x_i^1 < x_i^2$ y el gene descendiente será dado por la siguiente ecuación:

$$y_i = rand(\left(x_i^1 - \alpha(x_i^2 - x_i^1)\right), \left(x_i^2 + \alpha(x_i^2 - x_i^1)\right))$$

Donde rand(a,b) es una función que genera números aleatorios distribuidos uniformemente en el rango (a,b), donde a < b, si no se cumple esta que $x_i^1 < x_i^2$ entonces se intercambian los valores para que el rango se mantenga como a < b.

α es un parámetro definido por el usuario para controlar la extensión del rango. Los investigadores Eshelman and Schaffer reportan que $\alpha = 0.5$ es una buena elección para la mayoría de las situaciones.

6. Explique el funcionamiento del operador de cruzamiento Simulated Binary Crossover (SBX)

Lo que buscaban los investigadores Deb and Agrawal era que los hijos no fueran muy diferentes a sus padres. Ellos definieron el factor de propagación β_i con la siguiente ecuación:

$$\beta_i = \left| \frac{c_i^1 - c_i^2}{p_i^1 - p_i^2} \right|$$

Observaron que si $\beta_i > 1$ el operador se comportaba como un cruzamiento expansivo, pero si $\beta_i < 1$ entonces el operador se comportaba como un cruzamiento de contracción, por el contrario, si $\beta_i = 1$ el operador es llamado un cruzamiento estacionario.

De igual manera, ellos demostraron que se puede obtener una alta probabilidad de tener descendientes de un comportamiento estacionario con un número aleatorio u_i que sea uniformemente distribuido entre 0 y 1.

$$\beta_i = \begin{cases} (2u_i)^{\frac{1}{n+1}}, & u_i \leqslant 0.5 \\ (2(1-u_i))^{-\frac{1}{n+1}}, & u_i > 0.5 \end{cases}$$

n es un parámetro de control, entre mayor es la *n* más alta es la probabilidad de que β_i tienda a 1.

Una β_i mayor significa una mayor distancia del centroide entre los padres y los descendientes y viceversa. Con esto en mente podemos aplicar las siguientes fórmulas para obtener cada gen del descendiente.

$$c_i^1 = 0.5\left(p_i^1 + p_i^2\right) + 0.5\beta_i\left(p_i^1 - p_i^2\right)$$
$$c_i^2 = 0.5\left(p_i^1 + p_i^2\right) + 0.5\beta_i\left(p_i^2 - p_i^1\right)$$

Dicho lo anterior, el BLX aumenta la probabilidad de que los descendientes tengan el mismo centroide que sus padres en el espacio de búsqueda. El centroide se podría definir como el centro geométrico en múltiples dimensiones.

7. Explique el funcionamiento del operador de cruzamiento Uniform Order-Based Crossover

Los métodos de cruzamiento de k-puntos y uniforme descritos anteriormente no son adecuados para los problemas de búsqueda con códigos de permutación como los utilizados en el TSP. A menudo crean descendientes que representan soluciones inválidas para el problema. Por lo tanto, cuando se resuelven problemas de búsqueda con códigos de permutación, a menudo un mecanismo de reparación específico para el problema es requerido y utilizado junto con los métodos de recombinación anteriores para crear soluciones candidatas válidas.

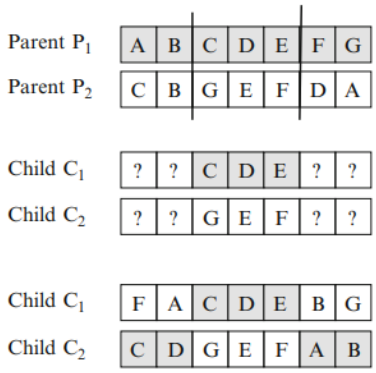
Otra alternativa es utilizar métodos de recombinación desarrollados específicamente para códigos de permutación, que siempre generan soluciones candidatas válidas como el operador de cruzamiento Uniform Order-Based.

En el Uniform Order-Based Crossover se seleccionan al azar dos padres (P1 y P2) y se genera una plantilla binaria aleatoria (véase la Fig. 4). Algunos de los genes para el descendiente C1 se rellenan tomando los genes del padre P1 donde hay un uno en la plantilla. En este punto tenemos a C1 parcialmente rellenado, pero tiene algunos "huecos". Los genes del padre P1 en las posiciones correspondientes a los ceros de la plantilla se toman y se ordenan en el mismo orden en que aparecen en el padre P2. La lista ordenada se utiliza para rellenar los huecos de C1. El descendiente C2 se crea mediante un proceso similar.

Parent P_1	A	B	C	D	E	F	G
Parent P_2	E	B	D	C	F	G	A
Template	0	1	1	0	0	1	0
Child C_1	E	B	C	D	G	F	A
Child C_2	A	B	D	C	E	G	F

8. Explique el funcionamiento del operador de cruzamiento Order-Based Crossover

El Order-Based Crossover es una variación del Uniform Order-Based Crossover en el que se seleccionan al azar dos padres y se generan dos sitios de cruce al azar (véase la Fig. 5). Los genes entre los puntos de corte se copian a los hijos. Posteriormente, empezando a partir del segundo sitio de cruce se copian los genes que no están todavía presentes en la descendencia del padre alternativo en el orden en que aparecen. Por ejemplo, para la descendencia C1, dado que los alelos C, D y E se copian del padre P1, se obtienen los alelos B, G, F y A del padre P2. Después, a partir del segundo sitio de cruce, que es el sexto gen, se copian los alelos B y G como sexto y séptimo gen respectivamente. A continuación, se copian los alelos F y A como primer y segundo gen.



9. Proponga las estructuras de datos necesarias para la implementación de los operadores de cruzamiento.

- Listas y arreglos

10. Implemente al menos tres de los operadores antes mencionados, considerando su aplicación en problemas de optimización con representaciones binarias, reales y de orden.

- **Operador de cruzamiento de un punto de corte**

```
import random

tam_cromosoma = 10

cromosomaA = []
cromosomaB = []

for j in range(tam_cromosoma):
    cromosomaA.append(random.randint(0, 1))
    cromosomaB.append(random.randint(0, 1))

punto_corte = random.randint(1, tam_cromosoma-1)

descendienteA = cromosomaA[:punto_corte] + cromosomaB[punto_corte:]
descendienteB = cromosomaB[:punto_corte] + cromosomaA[punto_corte:]

print("Padre A: ", cromosomaA)
print("Padre B: ", cromosomaB)
print("Punto de corte:", punto_corte)
print("Hijo A: ", descendienteA)
print("Hijo B: ", descendienteB)
```

- **Operador de cruzamiento uniforme**

```
import random

tam_cromosoma = 10

cromosomaA = []
cromosomaB = []

for j in range(tam_cromosoma):
    cromosomaA.append(random.randint(11, 20))
    cromosomaB.append(random.randint(21, 30))

descendienteA = []
descendienteB = []
mascara_indices = []

for j in range(tam_cromosoma):
    if random.random() < 0.5:
        descendienteA.append(cromosomaA[j])
        descendienteB.append(cromosomaB[j])
        mascara_indices.append('A')
    else:
        descendienteA.append(cromosomaB[j])
        descendienteB.append(cromosomaA[j])
        mascara_indices.append('B')

print("Padre A: ", cromosomaA)
print("Padre B: ", cromosomaB)
print("Mascara: ", mascara_indices)
print("Hijo A: ", descendienteA)
print("Hijo B: ", descendienteB)
```

- **Operador de cruzamiento Uniform Order-Based Crossover**

```
import random
import numpy as np

tam_cromosoma = 10

# Inicializacion de cromosomas Padres
cromosomaA = np.zeros((tam_cromosoma), int)
cromosomaB = np.zeros((tam_cromosoma), int)
```

```
# Damos Valores a cromosomas Padres
for i in range(tam_cromosoma):
    cromosomaA[i] = i+1
    cromosomaB[i] = i+1
# Revolvemos cromosomas Padres de forma aleatoria
np.random.shuffle(cromosomaA)
np.random.shuffle(cromosomaB)

# Inicializacion Plantilla binaria aleatoria
template_binario = np.zeros((tam_cromosoma), int)
# Rellenamos Plantilla binaria aleatoria
for i in range(tam_cromosoma):
    if random.random() < 0.5:
        template_binario[i] = 1

# Inicializacion de cromosomas Hijos
descendienteA = np.zeros((tam_cromosoma), int)
descendienteB = np.zeros((tam_cromosoma), int)

# Rellenamos Hijos con Plantilla binaria aleatoria
for i in range(tam_cromosoma):
    if template_binario[i] == 1:
        descendienteA[i] = cromosomaA[i]
        descendienteB[i] = cromosomaB[i]

buffer_hijo_A = []
buffer_hijo_B = []

# creando Buffer de genes restantes del Hijo A
for i in range(tam_cromosoma):
    if not (cromosomaB[i] in descendienteA):
        buffer_hijo_A.append(cromosomaB[i])

# creando Buffer de genes restantes del Hijo B
for i in range(tam_cromosoma):
    if not (cromosomaA[i] in descendienteB):
        buffer_hijo_B.append(cromosomaA[i])

# Rellenar los espacios restantes del Hijo A con los valores en orden del
Padre B
padre = 0
for i in range(tam_cromosoma):
    if descendienteA[i] == 0:
        descendienteA[i] = buffer_hijo_A[padre]
        padre += 1
```

```
# Rellenar los espacios restantes del Hijo B con los valores en orden del
Padre A
padre = 0
for i in range(tam_cromosoma):
    if descendienteB[i] == 0:
        descendienteB[i] = buffer_hijo_B[padre]
        padre += 1

print("Padre A: ", cromosomaA)
print("Padre B: ", cromosomaB)
print("Plantilla: ", template_binario)
print("Hijo A: ", descendienteA)
print("Hijo B: ", descendienteB)
```

3. Conclusiones

Los algoritmos genéticos presentan importantes ventajas sobre los algoritmos tradicionales, como son: su capacidad para trabajar con varios puntos simultáneamente, abarcando así un espacio mayor de búsqueda (paralelismo), además no requieren manipular directamente los parámetros del problema y poseen una gran facilidad para adaptarse a diferentes tipos de problemas.

A su vez, los algoritmos genéticos también presentan algunas desventajas como son: la dificultad para encontrar una representación apropiada de los parámetros del problema o para hallar una función objetivo adecuada.

Los AG algoritmos no aseguran encontrar la solución óptima, pero en muchos casos pueden proporcionar soluciones bastante adecuadas y que por otros métodos hubiera sido imposible encontrar. Dicho lo anterior, estos algoritmos representan, por todas sus características, una opción que debe ser tenida en cuenta cuando se busca resolver un problema con algún grado de complejidad.

4. Referencias

[1] Yu & Gen. Introduction to Evolutionary Algorithms. 2010. Capítulos 1 al 3.

[2] Burke & Kendall. Search Metodologies. 2005 Capítulo 4.

[3] Russell & Norving. Artificial Intelligence - A Modern Approach – 1995. Capítulo 4.