INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

ASIGNATURA:

Metaheurísticas

Actividad #23:

Guía Taller No.12

"Estrategias de reemplazo en Algoritmos Genéticos"

PROFESORA:

Dra. Yenny Villuendas Rey

PRESENTA:

Juan René Hernández Sánchez

Adriana Montserrat García Carrillo

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.23
Guía Taller No.12
Título: Estrategias de reemplazo 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. Enuncie las ventajas y desventajas de los Algoritmos Genéticos

- **Ventajas:**

a) Paralelización: Los algoritmos evolutivos son por naturaleza paralelos. Independientemente de cómo se realice la paralelización, la idea clave es distribuir la carga computacional en varios procesadores y así acelerar la ejecución general del AG.
b) Hibridación: puede ser una forma muy eficaz de mejorar el rendimiento de los AG. La forma más común de hibridación es acoplar los AG con técnicas de búsqueda local e incorporar conocimientos específicos del dominio en el proceso de búsqueda.
c) Continuación en el tiempo: las capacidades de la mutación y la recombinación se utilizan de forma óptima para obtener una solución de la mayor calidad posible con recursos computacionales limitados. La utilización del tiempo (o la continuación) explota la compensación entre la búsqueda de soluciones con grandes poblaciones y una única época de convergencia o utilizar una pequeña población con múltiples épocas de convergencia.
d) Relajación de la evaluación: una evaluación del fitness precisa, pero computacionalmente costosa es sustituida por una estimación del fitness menos precisa, pero de bajo costo computacional.

- **Desventajas:**

a) Los AG tienen muchos parámetros a configurar.
b) Por lo general mejoran la calidad de la solución, pero se produce un aumento del tiempo de cálculo.
c) Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen, el tamaño de la población, el número de generaciones, entre otros.
d) Pueden converger prematuramente. La convergencia prematura y el utilizar poblaciones muy pequeñas podría incluso extinguir a dichas poblaciones o mermar su diversidad.

2. Enuncie las semejanzas y diferencias en los modelos generacional y estacionario.

- **Semejanzas:**

a) Tanto en el modelo generacional como estacionario se procura conservar el mismo tamaño de población.

- **Diferencias:**

a) Modelo generacional: Durante cada iteración se crea una población completa con nuevos individuos. La nueva población reemplaza directamente a la antigua.
b) Modelo estacionario: Durante cada iteración se escogen dos padres de la población (diferentes mecanismos de muestreo) y se les aplican los operadores genéticos. El/los descendiente/s reemplaza/n a uno/dos cromosoma/s de la población inicial.

3. Explique el funcionamiento del reemplazo aleatorio

En los Algoritmos Evolutivos las reglas de reemplazo pueden dividirse generalmente en tres categorías:

a) Reemplazo aleatorio: Los individuos que se reemplazan se seleccionan al azar.
b) Reemplazo basado en el fitness: Los individuos peores tienen una mayor probabilidad de ser reemplazados.
c) Reemplazo basado en la edad: Los individuos de mayor edad tienen una mayor probabilidad de ser reemplazados.

Nota: Las 3 reglas de reemplazo pueden combinarse entre sí. La regla de reemplazo basado en la edad y la regla de reemplazo aleatorio son similares desde el punto de vista estadístico, ambas tienen una presión selectiva más débil.

4. Explique el funcionamiento del reemplazo elitista

Existen muchas otras técnicas para ajustar la regla de reemplazo. Una de ellas es el reemplazo elitista. La cual consiste en que, si el mejor individuo es seleccionado para ser sustituido, según la regla de reemplazo, el reemplazo no se producirá. Así que el elitismo aumenta la presión selectiva. El elitismo puede combinarse con las reglas de reemplazo: aleatorio, basado en el fitness y basado en la edad.

5. Explique el funcionamiento del reemplazo por Crowding determinístico

Paso 1. Se seleccionan al azar dos padres p1 y p2 con reemplazo de la población actual.

Paso 2. Generar dos descendientes c1 y c2 utilizando los operadores de variación.

Paso 3. Evaluar el fitness de los descendientes, f(c1) y f(c2), y calcular sus distancias respecto a los padres, es decir, d(p1, c1), d(p1,c2), d(p2, c1) y d(p2, c2).

Paso 4. Identificar un par de competencia cercano. Si [d(p1, c1) + d(p2,c2)] ≤ [d(p1, c2) + d(p2, c1)], la competencia es entre p1 ⇔ c1 y p2 ⇔ c2. En caso contrario, la competencia es entre p1 ⇔ c2 y p2 ⇔ c1.

Paso 5. Determinar al ganador. Los individuos con fitness más alto ganan la competencia y permanecen en la población. Los perdedores se descartan.

Nota: Repetir los pasos anteriores hasta que se cumplan los criterios de parada.

6. Explique el funcionamiento del reemplazo por torneo restringido entre semejantes.

Paso 1. Se seleccionan al azar dos padres p1 y p2 con reemplazo de la población actual.

Paso 2. Generar dos descendientes c1 y c2 utilizando los operadores de variación.

Paso 3. Evaluar el fitness de los descendientes f(c1) y f(c2). Seleccionar aleatoriamente w (tamaño de la ventana) individuos de la población actual con reemplazo como el conjunto de comparación.

Paso 4. Obtener d1 y d2, es decir, obtener los individuos más cercanos en el conjunto de comparación respecto a c1 y c2, respectivamente.

Paso 5. Determinar al ganador. Si f(c1) > f(d1), reemplazar d1 por c1, en caso contrario, se mantiene d1 y se descarta c1. La misma operación se realiza entre d2 y c2.

Nota: Repetir los pasos anteriores hasta que se cumplan los criterios de parada.

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

- Listas

8. Implemente tres de las estrategias antes mencionadas en representaciones binarias, reales y de orden.

- **Reemplazo aleatorio**

```
import random

tam_cromosoma = 10
tam_poblacion = 10

class Gen:
    cromosoma = []
    aptitud = int
    generacion = int

    def __init__(self, generacion, cromosoma):
        self.cromosoma = cromosoma
        self.aptitud = self.calcular_aptitud()
        self.generacion = generacion

    def calcular_aptitud(self):
        return sum(self.cromosoma) * random.randint(2, 10)

    def __str__(self):
```

```
        return "< Crom: " + str(self.cromosoma) + " Apt: " +
str(self.aptitud) + " Gen:" + str(self.generacion) + " >"

# Con Valores Binarios
def generar_cromosoma():
    cromosoma = [random.randint(0, 1) for _ in range(tam_cromosoma)]
    return cromosoma

poblacion = []

for _ in range(tam_poblacion):
    poblacion.append(Gen(1, generar_cromosoma()))

print("\n\nPoblacion inicial:")
for i in poblacion:
    print(i)

reemplazo = random.randint(0, tam_poblacion-1)

poblacion.pop(reemplazo)

poblacion.append(Gen(2, [5 for _ in range(tam_cromosoma)]))

print("\n\nPoblacion Final:")
for i in poblacion:
    print(i)
```

- **Reemplazo elitista**

```
import random

tam_cromosoma = 10
tam_poblacion = 10

class Gen:
    cromosoma = []
    aptitud = int
    generacion = int

    def __init__(self, generacion, cromosoma):
        self.cromosoma = cromosoma
        self.aptitud = self.calcular_aptitud()
        self.generacion = generacion
```

```
    def calcular_aptitud(self):
        return sum(self.cromosoma) * random.randint(2, 10)

    def __str__(self):
        return "< Crom: " + str(self.cromosoma) + " Apt: " +
str(self.aptitud) + " Gen:" + str(self.generacion) + " >"

# Con Valores de Representaciones de Orden
def generar_cromosoma():
    cromosoma = [i+1 for i in range(tam_cromosoma)]
    random.shuffle(cromosoma)
    return cromosoma

poblacion = []

for _ in range(tam_poblacion):
    poblacion.append(Gen(1, generar_cromosoma()))

print("\n\nPoblacion inicial:")
for i in poblacion:
    print(i)

elite = max(poblacion, key=lambda x: x.aptitud)

reemplazo = random.randint(0, tam_poblacion-1)
bandera = True

if poblacion[reemplazo] != elite:
    bandera = False
    poblacion.pop(reemplazo)
    poblacion.append(Gen(2, generar_cromosoma()))

print("\n\n elite generacional:")
print(elite)

print("\n\nPoblacion Final:")
for i in poblacion:
    print(i)

if bandera:
    print("\n<< Se intento reemplazar al elite >>")
```

- **Reemplazo por Crowding determinístico**

```
import random
import math

tam_cromosoma = 10
tam_poblacion = 10

class Gen:
    cromosoma = []
    aptitud = int
    generacion = int

    def __init__(self, generacion, cromosoma):
        self.cromosoma = cromosoma
        self.aptitud = self.calcular_aptitud()
        self.generacion = generacion

    def calcular_aptitud(self):
        return sum(self.cromosoma) * random.randint(2, 5)

    def __str__(self):
        cromosoma_str = [round(x, 2) for x in self.cromosoma]
        return "< Crom: " + str(cromosoma_str) + " Apt: " +
str(round(self.aptitud, 2)) + " Gen:" + str(self.generacion) + " >"

# Con valores Reales
def generar_cromosoma():
    cromosoma = [random.uniform(-10.0, 10.0) for _ in range(tam_cromosoma)]
    return cromosoma

def distancia_cromosoma(gen1, gen2):
    distancia = 0
    for i in range(tam_cromosoma):
        distancia += (gen1.cromosoma[i] - gen2.cromosoma[i])**2
    return math.sqrt(distancia)

poblacion = []

for _ in range(tam_poblacion):
    poblacion.append(Gen(1, generar_cromosoma()))

print("\n\nPoblacion Inicial:")
for i in poblacion:
    print(i)
```

```
descendiete1 = Gen(2, generar_cromosoma())
descendiete2 = Gen(2, generar_cromosoma())

pos1 = random.randint(0, tam_poblacion-1)
padre1 = poblacion.pop(pos1)
pos2 = random.randint(0, tam_poblacion-2)
padre2 = poblacion.pop(pos2)

dis_11 = distancia_cromosoma(padre1, descendiete1)
dis_12 = distancia_cromosoma(padre1, descendiete2)

dis_21 = distancia_cromosoma(padre2, descendiete1)
dis_22 = distancia_cromosoma(padre2, descendiete2)

if (dis_11 + dis_22) <= (dis_12 + dis_21):
    gan1 = max(padre1, descendiete1, key=lambda x: x.aptitud)
    poblacion.append(gan1)
    gan2 = max(padre2, descendiete2, key=lambda x: x.aptitud)
    poblacion.append(gan2)
else:
    gan1 = max(padre1, descendiete2, key=lambda x: x.aptitud)
    poblacion.append(gan1)
    gan2 = max(padre2, descendiete1, key=lambda x: x.aptitud)
    poblacion.append(gan2)


print("\n\nPoblacion Final:")
for i in poblacion:
    print(i)
```

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.