Inteligencia Artificial
Práctica 8: Algoritmos Genéticos

Verónica Esther Arriola Ríos
Pedro Rodríguez Zarazúa
Luis Alfredo Lizárraga Santos

Fecha de entrega: Miércoles 20 de Abril de 2016

1. Objetivo

Conocer el funcionamiento de los algoritmos genéticos, su aplicación, pros y contras.

Entender los mecanismos de herencia, mutación, selección y cruza, así como su relación con la selección natural. Implementar un algoritmo genético que resuelva y optimice un problema dado.

2. Introducción

Un algoritmo genético es una heurística de búsqueda que está basada en el proceso de **selección natural**. La selección natural es un proceso gradual mediante el cual las características biológicas se vuelven más o menos comunes en una población como una función del efecto de las características heredadas de la diferencia de éxito reproductivo de los organismos que interactúan con su entorno. Es el mecanismo clave de la **evolución**. El término de "selección natural" fue popularizado por Charles Darwin desde el año de 1859.

3. Metodología

En un algoritmo genético, una **población** de candidatos a solución (también llamados **individuos** o fenotipos) es evolucionada para obtener soluciones óptimas del problema. Cada individuo tiene una representación (sus cromosomas o genotipo) que puede ser alterada o mutada; tradicionalmente, las representaciones son como cadenas binarias de 0's y 1's, pero otras representaciones son posibles.

La evolución es un proceso iterativo mediante el cual se generan individuos aleatoriamente para crear otra población en cada iteración, llamada **generación**. En cada generación, la **aptitud** (fitness) de cada individuo es evaluada. Usualmente la aptitud es el valor de la función objetivo del problema de optimización que se quiere resolver. Se seleccionan de manera aleatoria individuos de la población para que modifiquen su genoma (**recombinar**) y posiblemente **mutar** aleatoriamente para formar una nueva generación. La nueva generación de posibles soluciones es usada en la siguiente iteración del algoritmo. Comúnmente, el algoritmo termina cuando se alcanza el número máximo de iteraciones o el valor de aptitud de un individuo ha alcanzado el óptimo.

4. Requisitos del algoritmo genético

1. Una **representación** genética del dominio de la solución.
2. Una **función fitness** (aptitud) a evaluar el dominio de la solución.

La representación estándar de cada individuo es una arreglo de bits, sin embargo, arreglos de otro tipo y estructuras funcionan esencialmente de la misma manera. El motivo por el cual se prefiere la representación estándar es porque facilita las operaciones de recombinación en parte a la longitud fija del arreglo, también la operación de mutación se vuelve trivial como se notará más adelante.

5. Componentes del algoritmo genético

5.1. Inicialización

Usualmente se genera una cantidad de posibles soluciones de manera aleatoria para formar la población inicial. El tamaño de la población depende del problema, y pueden llegar a ser cientos de miles de posibles soluciones en la población.

5.2. Selección

La selección es una etapa del algoritmo genético en el cuál se selecciona un individuo de la población que después será recombinado con otro individuo.

Existen diferentes maneras de realizar el proceso de selección, el más común es la **selección proporcional de fitness** (selección de ruleta). En este tipo de selección, el fitness (o aptitud) del individuo se asocia con su probabilidad de selección.

Entonces si f_i es el fitness del individuo i en la población, la probabilidad de ser seleccionado es

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

donde N es el número de individuos en la población.

Esto es similar a imaginar una ruleta de casino. Usualmente una proporción de la rueda de la ruleta es asignada a cada posible individuo basado en su valor fitness. Esto se logra al dividir el fitness de cada individuo entre el total de fitness de todos los individuos, que es equivalente a normalizarlos a 1. Entonces un individuo es aleatoriamente seleccionado de la manera en que lo haría una ruleta que está girando.

5.3. Operadores genéticos

5.3.1. Recombinación

La recombinación es el operador genético que permite la variación de cromosomas de los individuos de la población de una generación a otra. Es análoga a la reproducción y la recombinación biológica. El proceso de recombinación consiste en tomar más de un individuo y producir un nuevo hijo o solución.

Hay varias técnicas de recombinación, la más habitual es la recombinación de un punto y consiste en seleccionar aleatoriamente un mismo punto de corte dentro de la cadena de cromosomas de los padres e intercambiar sus contenido para generar nuevos hijos:

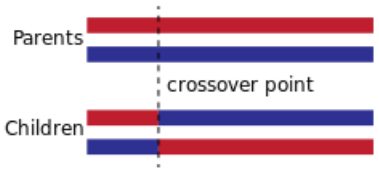

Figura 1: Operación genética recombinación de un punto.

Padre 1	1 1 0 1 0 0 1 0 0 1 1 0 1 1 0
Padre 2	1 0 0 1 1 1 0 1 1 0 0 1 0 1 1
Hijo 1	1 1 0 1 1 1 0 1 1 0 0 1 0 1 1
Hijo 2	1 0 0 1 0 0 1 0 0 1 1 0 1 1 0

Cuadro 1: Ejemplo numérico de recombinación.

5.3.2. Mutación

La mutación es el operador genético que mantiene la diversidad genética de una generación a otra, de manera análoga a la mutación biológica. La mutación altera uno o más de los genes (valores) en el cromosoma. Esta alteración puede cambiar por completo la solución antes de aplicar el operador de mutación y puede llegar a obtener mejores soluciones dentro del algoritmo genético.

Las mutaciones ocurren de acuerdo a una probabilidad de mutación que es definida por el usuario. Esta probabilidad debería ser baja porque si se establece un probabilidad de mutación muy alta el algoritmo genético se convierte en una búsqueda de soluciones de manera aleatoria.

Cromosoma original	1 1 0 1 0 0 1 0 0 1 1 0 1 1 0
Cromosoma mutado	1 1 0 1 1 0 1 0 0 1 1 0 1 0 0

Cuadro 2: Ejemplo del operador de mutación.

5.4. Terminación del algoritmo

Las iteraciones del algoritmo genético se termina hasta que se alcanza alguna de las condiciones de terminaciones, que pueden ser:

- Una solución es encontrada tal que satisface un criterio mínimo.
- Un número fijo de generaciones ha sido alcanzado.
- Se alcanza el máximo de los recursos posibles (por ejemplo: tiempo de procesamiento).
- Se alcanza el valor fitness más alto posible o se llega a un estado en el cual las iteraciones sucesivas no producen mejores resultados (puede deberse a la falta de diversidad por ejemplo).
- Al hacer una inspección manual.
- O combinaciones de las anteriores.

6. Variaciones

6.1. Elitismo

Hay muchas variaciones que pueden hacerse a un algoritmo genético, una de ellas es el proceso de elitismo, el cual es una variante del proceso que permite que algunas de las mejores soluciones pasen a la siguiente generación sin alteraciones por recombinación ni mutación.

7. Desarrollo e implementación

Se desea resolver el problema de las ocho reinas mediante algoritmos genéticos.

Este problema consiste en colocar ocho reinas del juego de ajedrez en un tablero de 8×8 de tal manera que no se ataquen mutuamente. Entonces, encontrar una solución requiere que entre cualesquiera dos reinas no compartan columna, fila ni diagonal entre ellas.
El problema de las ocho reinas es un caso particular más general de las n-reinas, que consiste en colocar n reinas en un tablero de dimensiones $n \times n$.

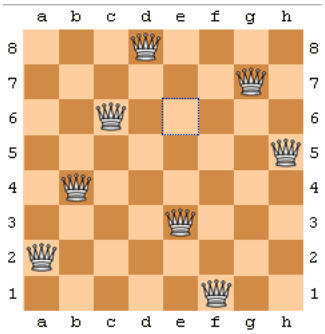

Figura 2: Ejemplo de solución al problema de las ocho reinas. [1]

7.1. Consideraciones de la implementación

7.1.1. Representación genética

Es recomendable emplear una representación de un tablero mediante una arreglo de números que identifica la posición por filas de cada reina. Esta representación es muy conveniente porque evita que las reinas se ataquen por columnas y facilita el proceso de encontrar una solución.

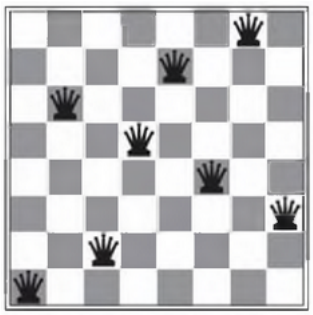

Figura 3: La representación del tablero sería [8, 3, 7, 4, 2, 5, 1, 6].

[1] https://matteoredaelli.wordpress.com/2009/01/05/n-queens-solution-with-erlang/

7.1.2. Función fitness

La optimización que se desea es encontrar un tablero en donde las reinas no se ataquen, de manera que la función fitness es inversamente proporcional a la cantidad de ataques entre reinas en un tablero. La figura 3 muestra un tablero donde sólo existe un ataque entre reinas, casi es un tablero óptimo por lo el resultado de la función fitness debe ser un valor alto.

7.1.3. Operador de recombinación

Se utilizará el operador de corte de un punto eligiendo aleatoriamente un punto de corte, ilustrado con el siguiente ejemplo:

Figura 4: Recombinación de dos tableros para producir un nuevo tablero hijo.

7.1.4. Operador de mutación

El operador de mutación será definido con una probabilidad de mutación de 0.2. Es decir, se recorrerá cada gen del cromosoma (cada número del arreglo) y se modificará su valor con probabilidad 0.2.

Tablero original	[8, 3, 7, 4, 2, 5, 1, 6]
Tablero mutado	[8, 3, 7, 4, 2, 3, 1, 6]

Cuadro 3: Ejemplo de mutación para un tablero de ajedrez.

7.1.5. Selección

Se utilizará el método proporcional de selección por ruleta descrito anteriormente.

7.1.6. Terminación del algoritmo

El algoritmo genético debe terminar cuando encuentra una solución óptima (sin ataques entre reinas) o cuando se hayan alcanzado 1000 generaciones.

7.1.7. Cantidad de población

La población de cada generación estará constituida de 50 individuos.

7.1.8. Elitismo

Para asegurarnos de mantener al menos una solución lo suficientemente óptima, se utilizará elitismo de 1 individuo en cada generación.

7.2. Pseudocódigo

El comportamiento general del algoritmo genético puede representarse a través del siguiente pseudocódigo:

```
poblacion ← newPoblacion(50)
while not limiteDeGeneraciones or optimoEncontrado do
    poblacion.asignarFitness()
    nuevaPoblacion.add(poblacion.elitismo(1))
    while not nuevaPoblacion.llena do
        individuo1 ← poblacion.seleccionRuleta()
        individuo2 ← poblacion.seleccionRuleta()
        hijo ← recombinacion(individuo1, individuo2)
        hijo.mutacion()
        nuevaPoblacion.add(hijo)
    end while
    poblacion ← nuevaPoblacion
end while
print(poblacion.mejorIndividuo())
```

8. Requisitos y resultados

La implementación debe mostrar cada 50 generaciones el mejor individuo encontrado y mostrar la solución óptima una vez terminado el algoritmo.

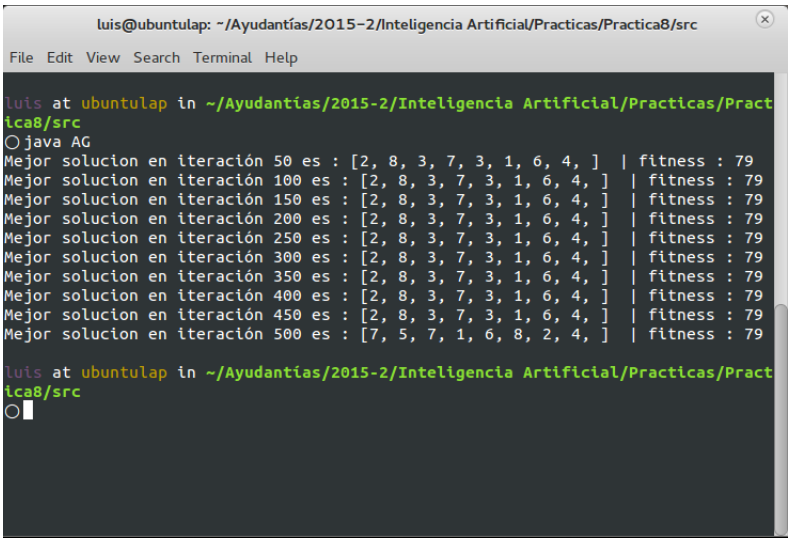

Figura 5: Ejemplo de resultado del algoritmo genético, donde se llegó al límite de generaciones.

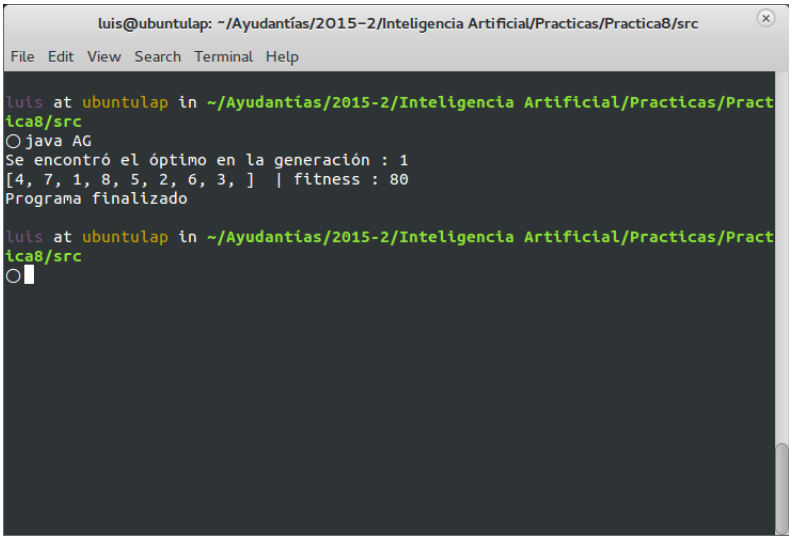

Figura 6: Ejemplo de resultado del algoritmo genético, donde se encontró una solución antes de llegar al límite de generaciones.

Lo podrán programar en `Java` o `Python`. No olviden comentar su código.

Punto Extra: Si extienden su implementación para que pueda resolver el problema de n reinas (tablero de $n \times n$) obtendrán un punto extra.

9. Notas adicionales.

La práctica es individual, anexen a su código un archivo readme.txt con su nombre completo, número de cuenta, número de la práctica y cualquier observación o notas adicionales (posibles errores, complicaciones, opiniones, críticas de la práctica o del laboratorio, cualquier comentario relativo a la práctica).
Pueden agregar cualquier biblioteca extra, sólo asegurense de que se encuentre bien comentada.

Compriman la práctica en un solo archivo (.zip, .rar, .tar.gz) con la siguiente estructura:

- ApellidopaternoNombreNúmerodepráctica.zip (por ejemplo: LizarragaLuis08.zip)
  - ApellidopaternoNombreNúmerodepráctica (por ejemplo: LizarragaLuis08)
    - src
      - aGeneticos(.java / .py)
  - readme.txt

La práctica se entregará en la página del curso en la plataforma AVE Ciencias.
O por medio de correo electrónico a `luislizarraga@ciencias.unam.mx` con asunto `Práctica08[IA 2016-2]`

La fecha de entrega es hasta el día Miércoles 20 de Abril a las 23:59:59 hrs.