

# 10 | Algoritmos Genéticos

Rodrigo Eduardo Colín Rivera

## Objetivo

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

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

## Introducción

Un algoritmo genético es una heurística de búsqueda que está basada en el proceso de **selección natural**. Su objetivo es encontrar soluciones óptimas a problemas combinatorios. La selección natural es un proceso gradual mediante el cual las características biológicas se vuelven más o menos comunes en una población. Esto depende de las características heredadas y de la diferencia de éxito reproductivo de los organismos que interactúan con su entorno. La selección natural es el mecanismo clave de la **evolución**. El término de “selección natural” fue popularizado por Charles Darwin desde el año de 1859.

## 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 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, **recombinando** y posiblemente **mutando** aleatoriamente sus componentes 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 se ha aproximado lo suficiente a un valor óptimo.

## Requisitos del algoritmo genético

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

La representación estándar de cada individuo es un 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 debido a la longitud fija del arreglo, también la operación de mutación se vuelve trivial como se notará más adelante.

## Componentes del algoritmo genético

### Inicialización

Usualmente se genera una cantidad de soluciones posibles de manera aleatoria para formar la población inicial. El tamaño de la población depende del problema y puede llegar a contener cientos de miles de individuos.

### Selección

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

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

## 10. Algoritmos Genéticos

Entonces si  $f_i$  es la aptitud 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.

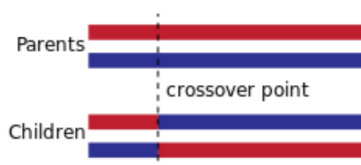
Esto es similar a imaginar una ruleta de casino. Usualmente una proporción de la rueda de la ruleta es asignada a cada posible individuo basado en su aptitud. Esto se logra al dividir la aptitud de cada individuo entre el total de aptitud de todos los individuos, que es equivalente a normalizarlos a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

## Operadores genéticos

### 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 10.1** 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

**Tabla 10.1** Ejemplo numérico de recombinación.

## 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 (o peores) dentro del algoritmo genético.

Las mutaciones ocurren valor por valor en un individuo de acuerdo a una probabilidad de mutación que es definida por el usuario. Esta probabilidad debería ser baja porque si se establece una 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

**Tabla 10.2** Ejemplo del operador de mutación.

## Terminación del algoritmo

Las iteraciones del algoritmo genético se terminan hasta que se alcanza alguna de las condiciones de terminación, 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 de aptitud 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.

## Variaciones

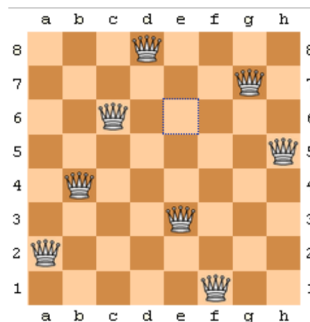
### Elitismo

Hay muchas variaciones que pueden hacerse a un algoritmo genético, una de ellas es el proceso de elitismo. Esta variante permite que algunas de las mejores soluciones pasen

a la siguiente generación sin alteraciones por recombinación ni mutación.

## 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  $8 \times 8$  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 de otro más general, el de las  $n$ -reinas, que consiste en colocar  $n$  reinas en un tablero de dimensiones  $n \times n$ .



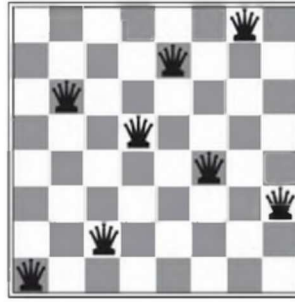
**Figura 10.2** Ejemplo de solución al problema de las ocho reinas. <sup>1</sup>

## Consideraciones de la implementación

### 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 filas y facilita el proceso de encontrar una solución.

<sup>1</sup><https://matteoredaelli.wordpress.com/2009/01/05/n-queens-solution-with-erlang/>



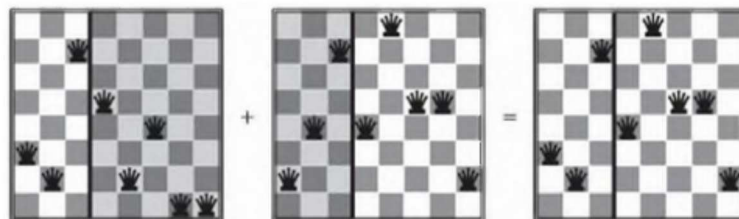
**Figura 10.3** La representación del tablero sería [8, 3, 7, 4, 2, 5, 1, 6].

### Función de aptitud

La optimización que se desea es encontrar un tablero en donde las reinas no se ataquen, de manera que la función de aptitud es inversamente proporcional a la cantidad de ataques entre reinas en un tablero. La figura 10.3 muestra un tablero donde sólo existe un ataque entre reinas, casi es un tablero óptimo por lo que el resultado de la función de aptitud debe ser un valor alto.

### Operador de recombinación

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



**Figura 10.4** Recombinación de dos tableros para producir un nuevo tablero hijo.

### 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 ]

**Tabla 10.3** Ejemplo de mutación para un tablero de ajedrez.

### Selección

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

### Terminación del algoritmo

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

### Cantidad de población

La población de cada generación estará constituida por 50 individuos.

### Elitismo

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

## Pseudocódigo

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

```

población ← newPoblación(50)
población.asignarAptitud()
while not limiteDeGeneraciones or óptimoEncontrado do
    nuevaPoblación.add(población.elitismo(1))
    while not nuevaPoblación.llena do
        individuo1 ← población.seleccionRuleta()
        individuo2 ← población.seleccionRuleta()
        hijo ← recombinación(individuo1, individuo2)
        hijo.mutación()
        nuevaPoblación.add(hijo)
    end while
    población ← nuevaPoblación

```

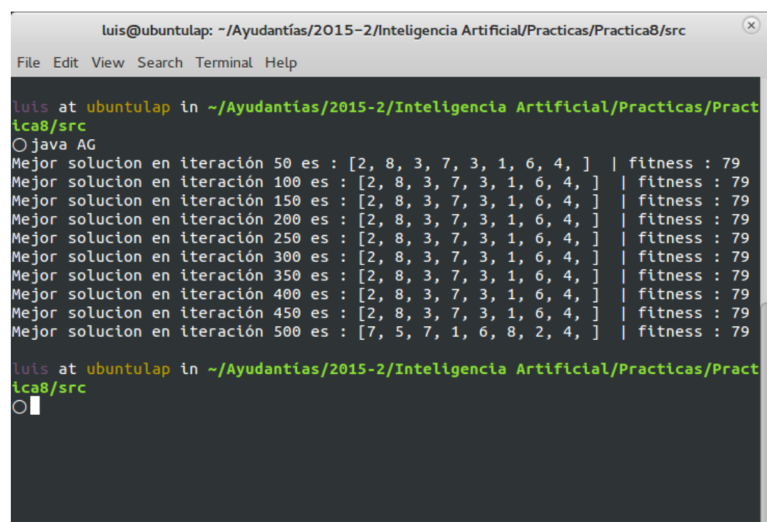
```

    población.asignarAptitud()
end while
print(población.mejorIndividuo())

```

## Requisitos y resultados

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



```

luis@ubuntulap: ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
File Edit View Search Terminal Help

luis at ubuntulap in ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
○ java AG
Mejor solucion en iteración 50 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 100 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 150 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 200 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 250 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 300 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 350 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 400 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 450 es : [2, 8, 3, 7, 3, 1, 6, 4, ] | fitness : 79
Mejor solucion en iteración 500 es : [7, 5, 7, 1, 6, 8, 2, 4, ] | fitness : 79

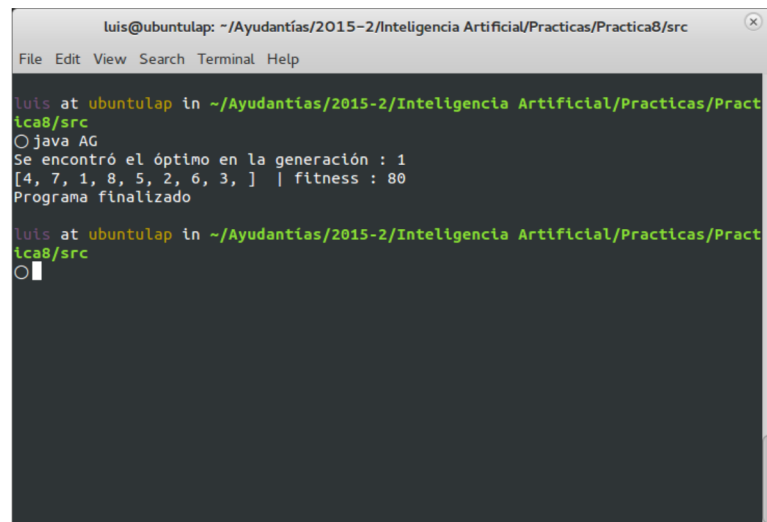
luis at ubuntulap in ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
○

```

**Figura 10.5** Ejemplo de resultado del algoritmo genético, donde se llegó al límite de generaciones.



## 10. Algoritmos Genéticos



```
luis@ubuntulap: ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
File Edit View Search Terminal Help

luis at ubuntulap in ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
○ java AG
Se encontró el óptimo en la generación : 1
[4, 7, 1, 8, 5, 2, 6, 3, ] | fitness : 80
Programa finalizado

luis at ubuntulap in ~/Ayudantías/2015-2/Inteligencia Artificial/Practicas/Practica8/src
○
```

**Figura 10.6** 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.