

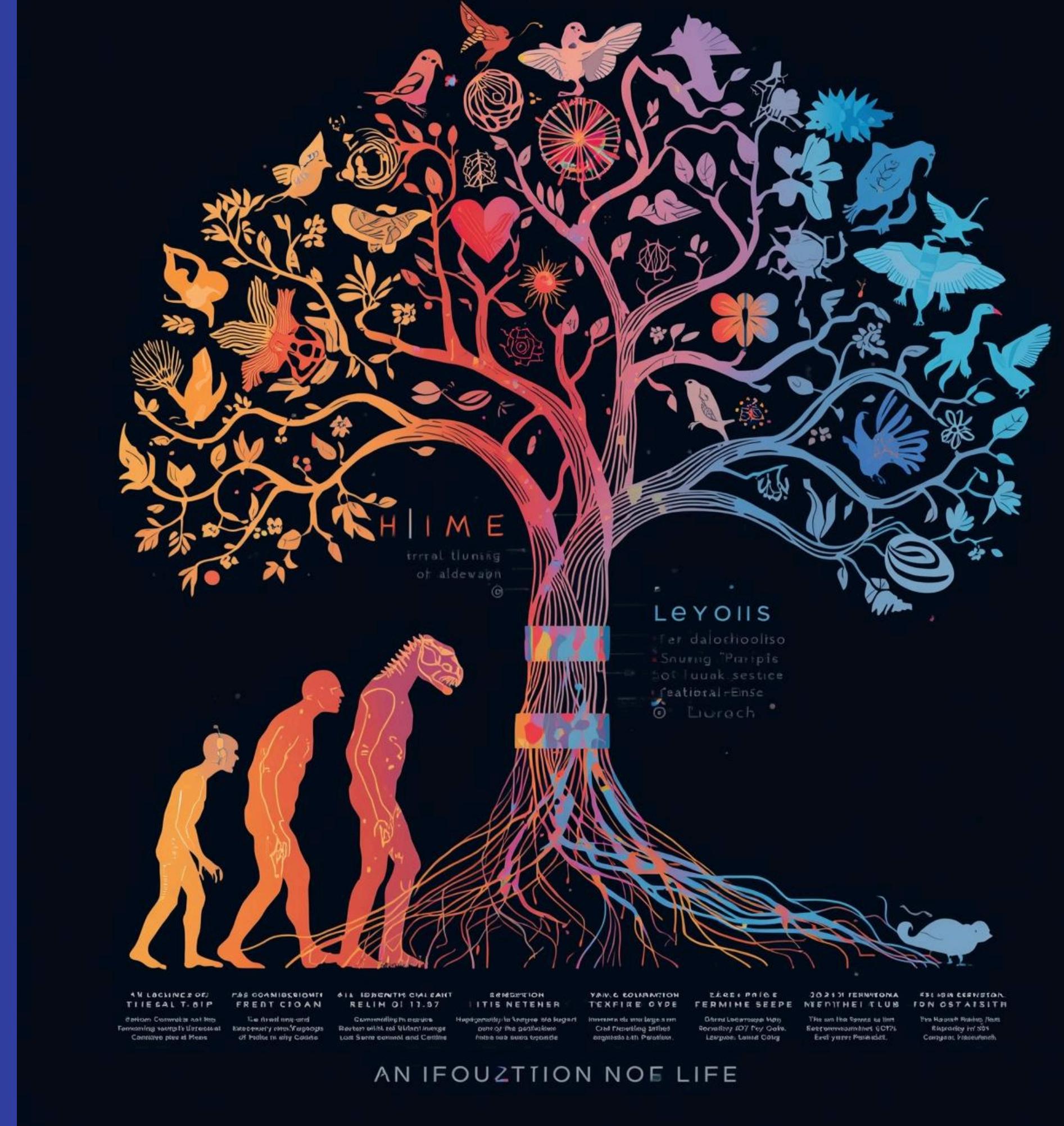
Algoritmos Genéticos

Miguel Trujillo Rojas, 1º DAM



¿Qué es un algoritmo genético?

Los algoritmos genéticos son técnicas de optimización inspiradas en la evolución natural. Su objetivo es encontrar soluciones óptimas a problemas complejos mediante diferentes procesos.



¿Qué partes tiene un Algoritmo Genético?

Para que un programa sea un algoritmo genético debe contar con:

- Población
- Fitness o Evaluación
- Selección
- Cruce
- Mutación
- Reemplazo
- Criterio de Parada



Población Inicial

```
int[] poblacion = Arrays.CreaArrayRandom(INDIVIDUOS, -50, max: 50); // Crea población
```

La población inicial es el primer paso de los algoritmos genéticos. Aquí, cada individuo representa una solución potencial que evolucionará a través de generaciones para encontrar la mejor respuesta posible al problema planteado.

En mi caso, los individuos de la población se eligen de manera aleatoria.



Cantidad de Poblaciones

El **número de poblaciones** es muy importante en la evolución del algoritmo genético. Una mayor cantidad de poblaciones permite una mayor diversidad y disminuye el número de generaciones necesarias para llegar a una solución.

En mi caso sólo he usado una población para hacerlo más simplificado, pero lo óptimo es usar varias.

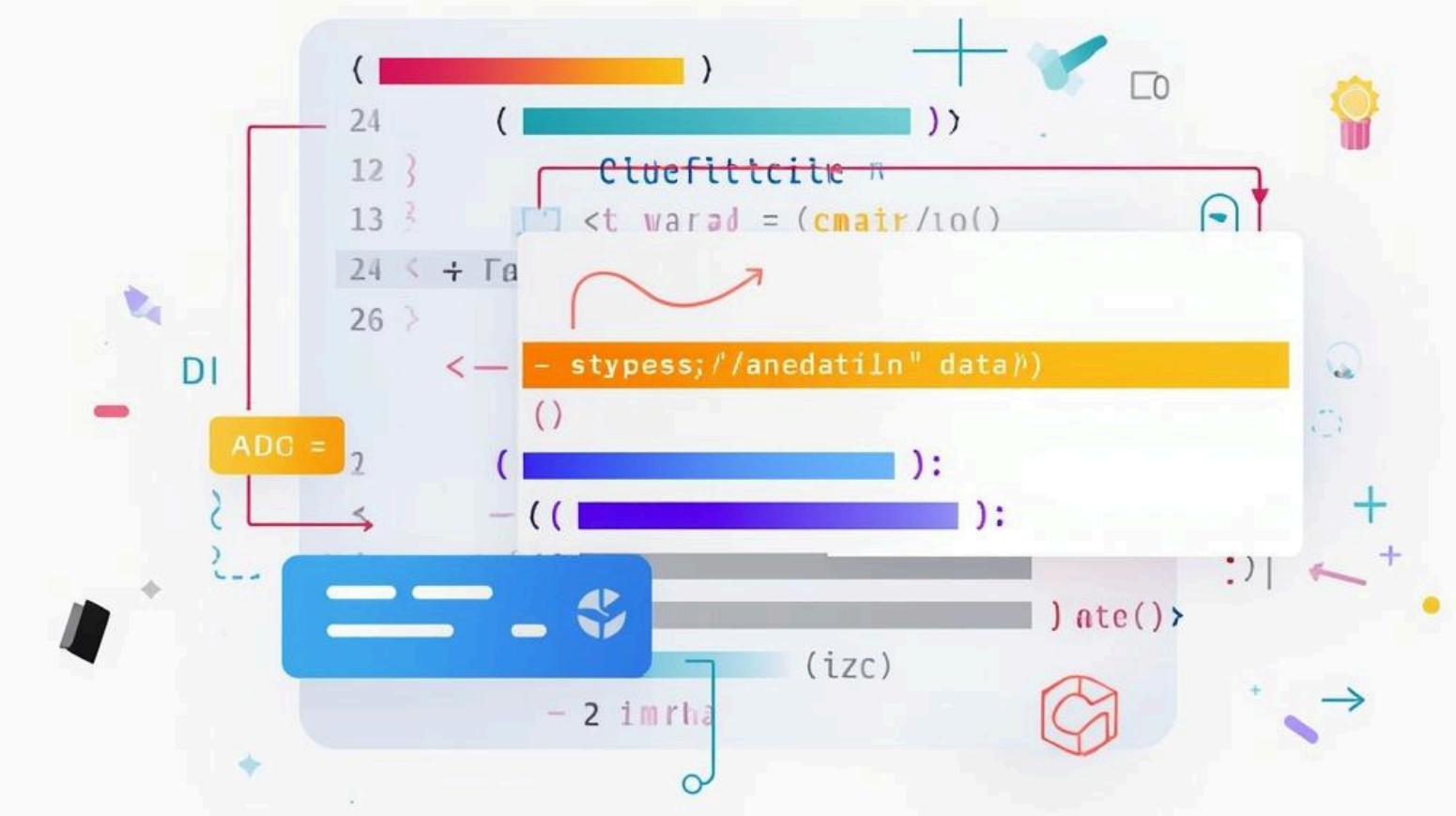


Función de Fitness

```
public static int peorIndividuo(int[] poblacion, int OBJETIVO) {  
    int peor = 0;  
    int indicepeor = 0;  
    for (int i = 0; i < poblacion.length; i++) {  
        int[] copia = copiarArray(poblacion);  
        copia[i] = 0;  
        int diferencia = aptitud(copia, OBJETIVO);  
        if (diferencia > peor) {  
            peor = diferencia;  
            indicepeor = i;  
        }  
    }  
    return indicepeor;  
}
```

La función de fitness es **fundamental en el algoritmo** genético, ya que es la que puntúa la calidad de las soluciones. Esta función evalúa cada individuo y guía la selección de los mejores para la reproducción.

En mi caso, busca el peor, para mejorarla en el cruce.



Proceso de Selección

La **selección de individuos** sirve para determinar que individuos seguirán y cuáles no en la próxima generación. Los métodos buscan mejorar la calidad de las soluciones mediante competencia natural.

En mi caso, siempre elijo al peor candidato y lo cruzo con otro aleatorio



Generación de descendencia

```
poblacion[peor] = poblacion[peor] + (int) [(poblacion[otroPadre] - poblacion[peor]) * Math.random()]; // Cruza al peor con otro individuo
```

La **generación de descendencia** en un algoritmo genético se logra a través de operadores como el cruzamiento y la mutación, permitiendo que las nuevas soluciones hereden características de sus padres.



Mutación

```
if (Math.random() <= MUTABILIDAD) { // Mutación 1  
    poblacion[(int) (Math.random() * INDIVIDUOS)] *= -1;  
}  
if (Math.random() <= MUTABILIDAD) { // Mutación 2  
    poblacion[(int) (Math.random() * INDIVIDUOS)] += 5;  
}  
if (Math.random() <= MUTABILIDAD) { // Mutación 3  
    poblacion[(int) (Math.random() * INDIVIDUOS)] -= 5;  
}
```

La **mutación** introduce **variaciones aleatorias** en los individuos para mantener la diversidad genética. Esto mayormente previene que la población se estanque.



Reemplazo

```
poblacion[peor] = poblacion[peor] + (int) [(poblacion[otroPadre] - poblacion[peor])  
* Math.random()]; // Cruza al peor con otro individuo
```

Se asegura de que los individuos más aptos sean seleccionados para la próxima generación, manteniendo así la diversidad genética.

En mi caso se reemplaza el peor candidato por uno nuevo, que es el cruce entre él mismo y otro aleatorio.



Criterios de Parada

```
while (Arrays.aplitud(poblacion, OBJETIVO) != 0);
```

Indican cuándo finaliza un algoritmo genético.

Los criterios comunes incluyen alcanzar un número máximo de generaciones, un nivel mínimo de error o la falta de mejoras significativas en iteraciones recientes.

En mi caso el algoritmo se detiene cuando alcanza una solución, esto se conoce como criterio de parada basado en aptitud.





FIN