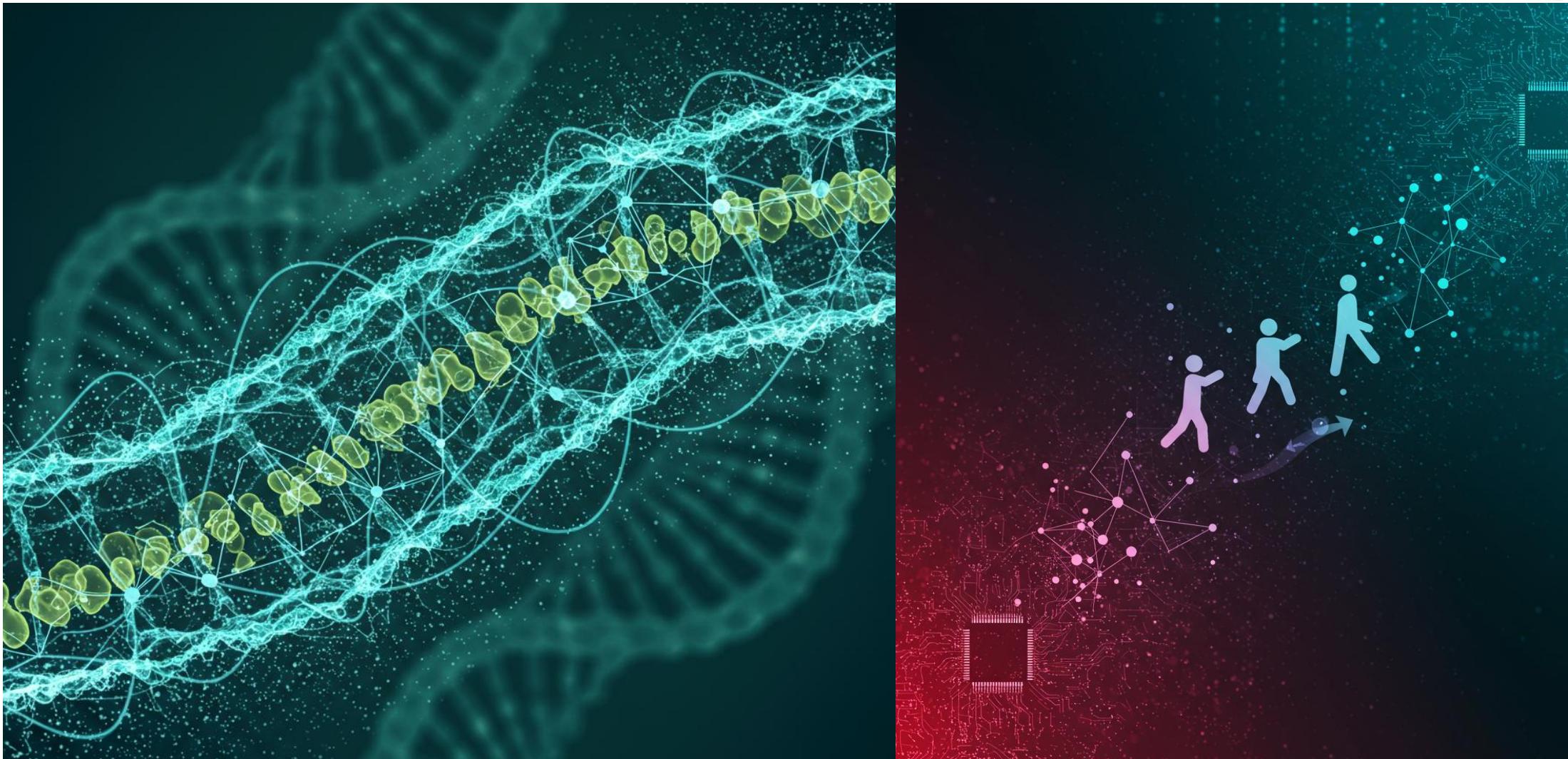


Computación Bioinspirada



Algoritmos genéticos



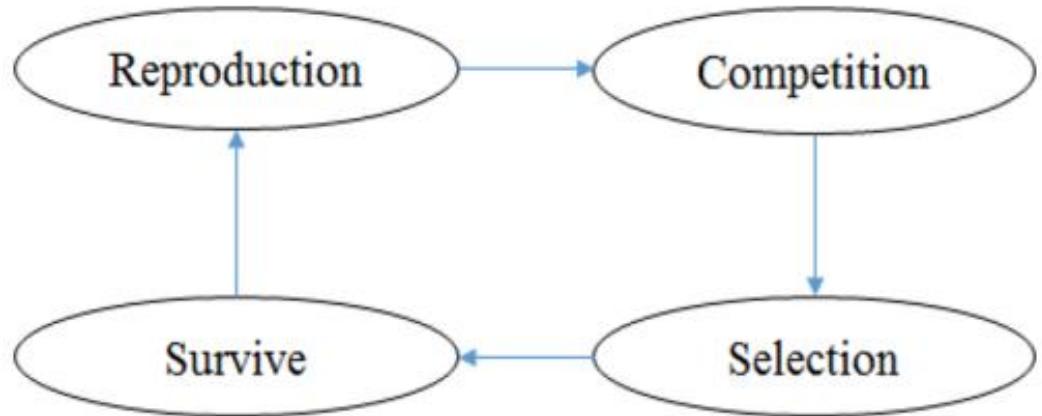
Algoritmos Genéticos y Computación Evolutiva: Resolviendo Problemas con la Lógica de la Naturaleza

- La realidad natural está compuesta por materia y forma. La materia será la condición necesaria, mientras que la forma aduce a la condición suficiente para que una realidad sea lo que es y no otra distinta.
- Lo natural se genera lo artificial se fabrica.
- En el contexto de la computación Bioinspirada, en Algoritmos Genéticos, se usa terminología directamente derivada de la biología evolutiva para describir las soluciones y su representación.

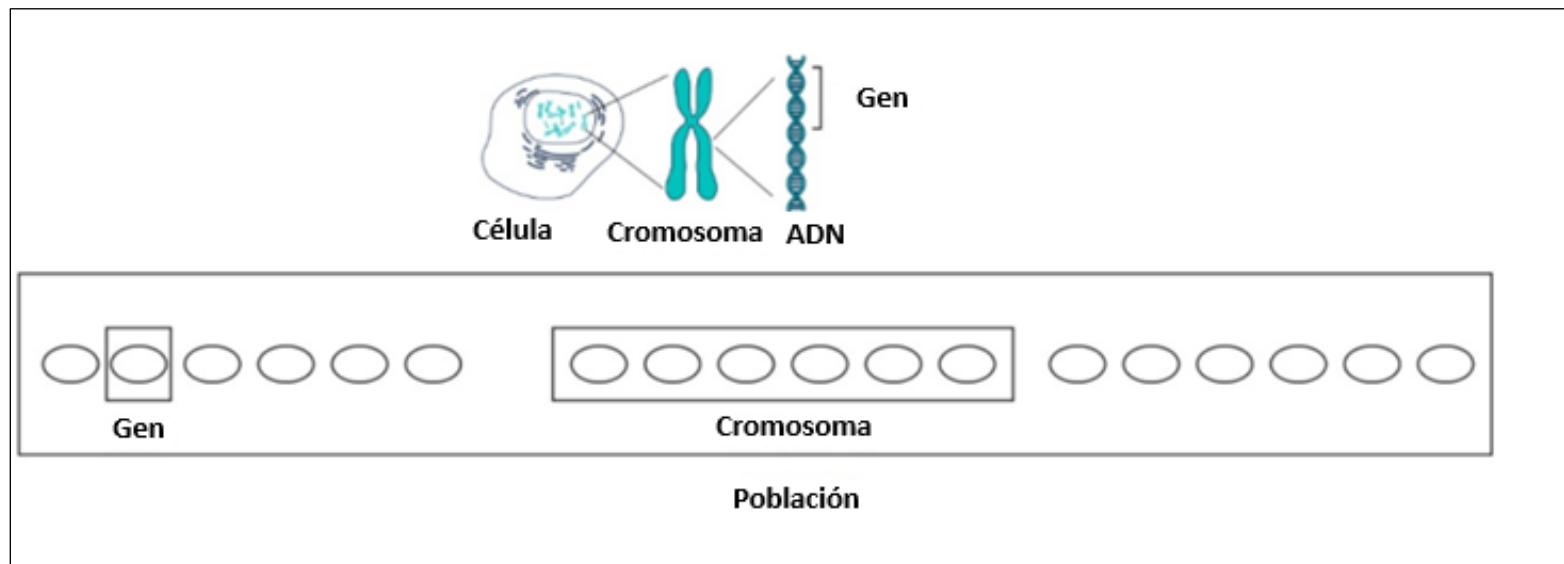
Conceptos base

- Un algoritmo genético es un proceso iterativo que opera sobre una población P de N individuos (cromosomas), donde cada individuo representa una posible solución al problema a resolver.
- La computación evolutiva es un campo de la inteligencia artificial que se inspira en los procesos de evolución biológica para resolver problemas de optimización y aprendizaje. Además de los algoritmos genéticos, existen otros modelos de computación evolutiva que se basan en diferentes aspectos de la evolución.

Algoritmos genéticos



Del principio de Darwin a la codificación.



Algoritmos genéticos

La codificación

Tipo de cromosoma	Descripción
1 0 0 1 1 0 1 1 1 0 0 1 0 1	Codificación binaria: este es el método de codificación más común, donde se representa un cromosoma con una cadena de bits (0 y 1). Útil en problemas optimización de funciones matemáticas.
0.2 0.5 0.03 0.8 0.23 0.76 0.85 0.33	Codificación de valores: representa un cromosoma como un conjunto de valores enteros o de punto flotante, un ejemplo de uso es usar esta codificación para optimizar una red neuronal, para encontrar los mejores pesos y sesgos.
2 4 1 8 3 5 7 6	Codificación con un orden específico: cada cromosoma representa una secuencia de elementos, que se utiliza en problemas como el problema del viajante.

La elección de la codificación es clave para el éxito del algoritmo.

- Codificación Binaria:

- ❖ Qué es: Cadenas de 0s y 1s. Clásica y simple.
- ❖ Ejemplo: problemas de selección de rutas, selección de características en Machine Learning (ejemplo: mejores características para perfiles de riesgo), selección de carteras de proyectos, optimizar costos, configuraciones de equipos.

- Codificación de Valor Real:

- ❖ Qué es: Vectores de números decimales.
- ❖ Ideal para: Ajuste de hiperparámetros en modelos de ML , diseño de ingeniería (ej. optimización de formas aerodinámicas), construir carteras de inversión que maximicen el rendimiento, refinerías, diseño de materiales.

- Codificación por Permutación:

- ❖ Qué es: Una secuencia ordenada de elementos.
- ❖ Ideal para: Problemas de ordenación, planificar rutas, flujos de producción, optimizar flujos en procesos, planificar horarios.

Algoritmos genéticos

💡 ¿Qué son los genotipos?

En un Algoritmo Genético, un genotipo es la representación interna (codificación) de una solución potencial al problema. Es el "ADN" del individuo en el algoritmo.

- Es el código genético de un organismo, en AGs, se suele codificar como una cadena de bits (binario), una lista de números enteros o reales, un árbol, una permutación, etc. Esta cadena o estructura es lo que los operadores genéticos (cruce, mutación) manipulan.
- Ejemplo: si estamos buscando la mejor combinación de switches on/off en un circuito, un genotipo podría ser la cadena 101101 (donde 1 es on y 0 es off).

Algoritmos genéticos

💡 ¿Qué son los fenotipos?

Un fenotipo es la representación externa, observable e interpretable de una solución. Es la forma en que el genotipo se "expresa" o se traduce en una solución real que puede ser evaluada.

- Son las características observables de un organismo (color de ojos, altura, comportamiento, etc.), que resultan de la interacción entre su genotipo y el ambiente. En AGs: es la solución real al problema que el genotipo describe.
- Para obtener el fenotipo a partir del genotipo, a menudo se necesita un proceso de "decodificación" o "interpretación". La función de aptitud (fitness function) se aplica sobre el fenotipo para evaluar su calidad.
- Ejemplo: el genotipo 101101 podría decodificarse en: "Switch 1 ON, Switch 2 OFF, Switch 3 ON, Switch 4 ON, Switch 5 OFF, Switch 6 ON". Esta configuración de switches es el fenotipo que se evalúa para ver qué tan bien funciona el circuito. La relación clave es que los operadores genéticos (cruce y mutación) actúan sobre los genotipos, mientras que la función de aptitud evalúa los fenotipos.

Algoritmos genéticos

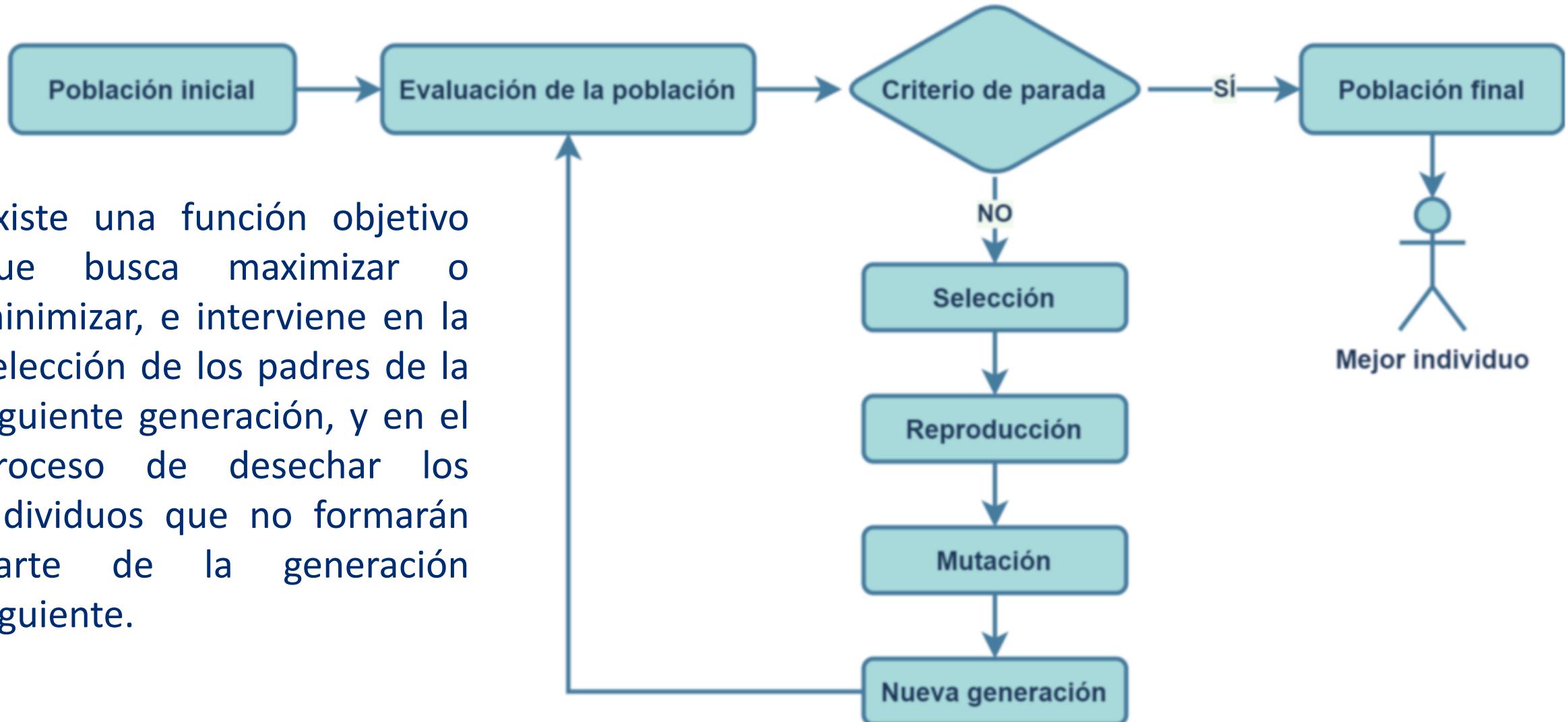
El espacio de búsqueda

El espacio de genotipos es el conjunto de todas las posibles configuraciones válidas de los genotipos que pueden ser representadas. Está definido por la forma en que codificamos las soluciones, por ejemplo, si usamos cadenas binarias de longitud N, el espacio de genotipos tiene 2^N elementos.

- Es el espacio de búsqueda para los algoritmos genéticos a nivel de la representación interna.
- Puede ser muy grande, pero a menudo es discreto y finito, la cantidad de combinaciones puede ser enorme.
- Las distancias y relaciones en este espacio están definidas por los operadores genéticos (por ejemplo, la distancia de Hamming para cadenas binarias).

El espacio de fenotipos es el conjunto de todas las posibles soluciones interpretables y únicas que pueden resultar de la decodificación de los genotipos. Es el espacio de búsqueda del problema en sí mismo, no de su codificación.

La Anatomía de un Algoritmo Genético



Generación de la población inicial

- En computación bio-inspirada, la generación de la población inicial es un paso fundamental. Es el punto de partida a partir del cual el algoritmo evolutivo (como un Algoritmo Genético o Programación Genética) comenzará su búsqueda de soluciones.
- Una población inicial diversa es importante para asegurar una buena exploración del "espacio de búsqueda" y evitar caer en óptimos locales prematuramente.
- Los individuos en esta población se crean generalmente de forma aleatoria, respetando la estructura y el rango de valores definidos para el "cromosoma" y sus "genes".

Generación de la población inicial

- Ejemplo 1: Cromosoma de 16 Bits

```
[ G1 ] [ G2 ] [ G3 ] ... [ G15 ] [ G16 ]    <-- Cromosoma (16 Genes)
|       |       |
v       v       v
(0/1)   (0/1)   (0/1)           (0/1)   (0/1)    <-- Cada Gen es un Bit
```

Individuo	Cromosoma (16 Bits)
#1	0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1
#2	1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 0
#3	0 0 1 1 1 0 1 0 1 1 0 0 0 1 1 1
#4	1 1 0 0 0 1 0 1 0 0 1 1 1 0 1 0
...	... (96 individuos más con patrones aleatorios)
#100	0 1 1 1 0 0 1 0 1 0 0 1 1 1 0 0

Proceso de Génesis Aleatoria

```
+-----+
| Individuo #N (Cromosoma de 16 bits)
|
|   Posición (Gen) 1: Lanzar Moneda -> [0]
|   Posición (Gen) 2: Lanzar Moneda -> [1]
|   Posición (Gen) 3: Lanzar Moneda -> [0]
|   ...
|   Posición (Gen) 16: Lanzar Moneda -> [1]
|
|   Resultado: [0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1]    <-- Un Cromosoma Generado
+-----+
```

Generación de la población inicial

- Ejemplo 2: Cromosoma de 10 Caracteres (A-Z)

```
[ G1 ] [ G2 ] [ G3 ] ... [ G9 ] [ G10 ]    <-- Cromosoma (10 Genes)
|       |       |
v       v       v       v       v
(A-Z)   (A-Z)   (A-Z)   (A-Z)   (A-Z)    <-- Cada Gen es un Carácter
```

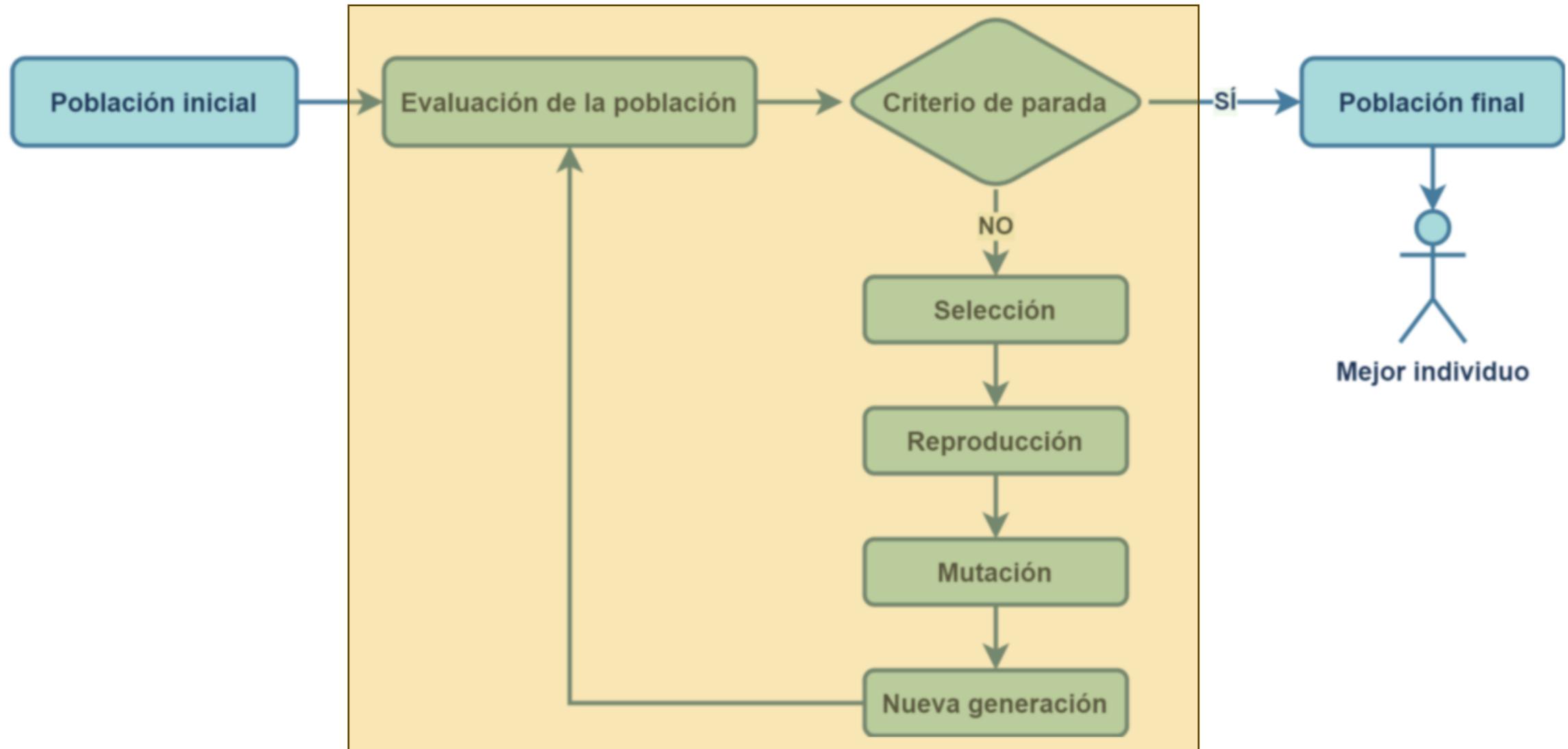
Individuo	Cromosoma (10 Caracteres)
#1	ZXFGHPQWLA
#2	MCVKIRJYZN
#3	QOWEUIRTYA
#4	ALSKDJFHGT
...	... (96 individuos más con cadenas aleatorias)
#100	NVBOPLIUYX

```
Proceso de Génesis Aleatoria
+-----+
| Individuo #N (Cromosoma de 10 Caracteres)
|
|   Posición (Gen) 1: Seleccionar A-Z -> [R]
|   Posición (Gen) 2: Seleccionar A-Z -> [A]
|   Posición (Gen) 3: Seleccionar A-Z -> [N]
|
|   ...
|   Posición (Gen) 10: Seleccionar A-Z -> [T]
|
|   Resultado: [ R A N D O M X Y Z T ]
+-----+                                         <-- Un Cromosoma Generado (una "palabra" aleatoria)
```

Generación de la población inicial

- **Aleatoriedad:** La generación inicial es siempre aleatoria. Esto asegura que no haya sesgos y que el algoritmo explore un amplio "espacio de soluciones" desde el principio.
- **Diversidad:** Una población inicial diversa es crucial. Si todos los individuos fueran iguales o muy parecidos, el algoritmo evolutivo tendría poco "material genético" con el que trabajar, limitando su capacidad para encontrar buenas soluciones.
- **Estructura del Cromosoma:** Aunque el contenido es aleatorio, la forma o estructura del cromosoma (ejemplo: 16 bits, 4 árboles, 10 caracteres) está predefinida por el problema que se intenta resolver.
- **Punto de Partida:** La población inicial es el "ADN" inicial de nuestro algoritmo bio-inspirado. A partir de aquí, la evolución (selección, cruce, mutación) tomará el control para mejorar la "aptitud" de la población.

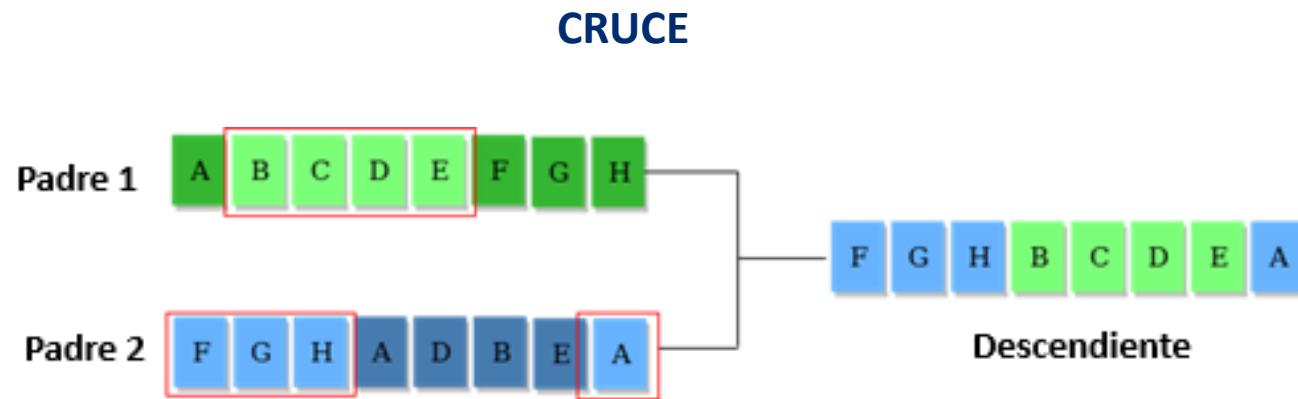
La Anatomía de un Algoritmo Genético, exploración y explotación



La Anatomía de un Algoritmo Genético, exploración y explotación

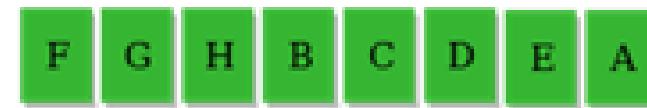
Cruzamiento	Descripción
<p>Padre:</p> <p>Hijo:</p>	<p>Cruzamiento de un punto. Se establece un punto de intercambio en un lugar aleatorio del genoma de los dos individuos, y uno de los individuos contribuye todo su código anterior a ese punto y el otro individuo contribuye todo su código a partir de ese punto para producir una descendencia.</p>
<p>Padre:</p> <p>Hijo:</p>	<p>Cruzamiento en dos puntos. En este caso se intercambian los genes que aparecen en el intervalo de genes delimitados por dos puntos.</p>
<p>Padre:</p> <p>Hijo:</p> <p>Cruce uniforme</p>	<p>Cruzamiento uniforme. Aquí el valor de una posición dada en el genoma de la descendencia corresponde al valor en esa posición del genoma de uno de los padres o al valor en esa posición del genoma del otro parent, elegido con un 50% de probabilidad.</p>

La Anatomía de un Algoritmo Genético, exploración y explotación

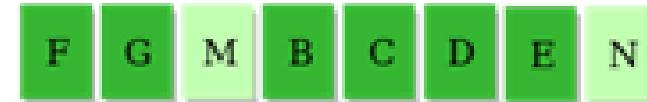


MUTACIÓN

Antes de la mutación



Después de la mutación



La Anatomía de un Algoritmo Genético, exploración y explotación

Estrategias de **exploración** se enfocan en **mantener la diversidad de la población** y en introducir nuevas soluciones en el espacio de búsqueda. Técnicas comunes:

- Diversidad de población inicial
- Alta tasa de mutación
- Reinicio de la población
- Operadores de cruce diversificadores
- Selección menos presionante.

La Anatomía de un Algoritmo Genético, exploración y explotación

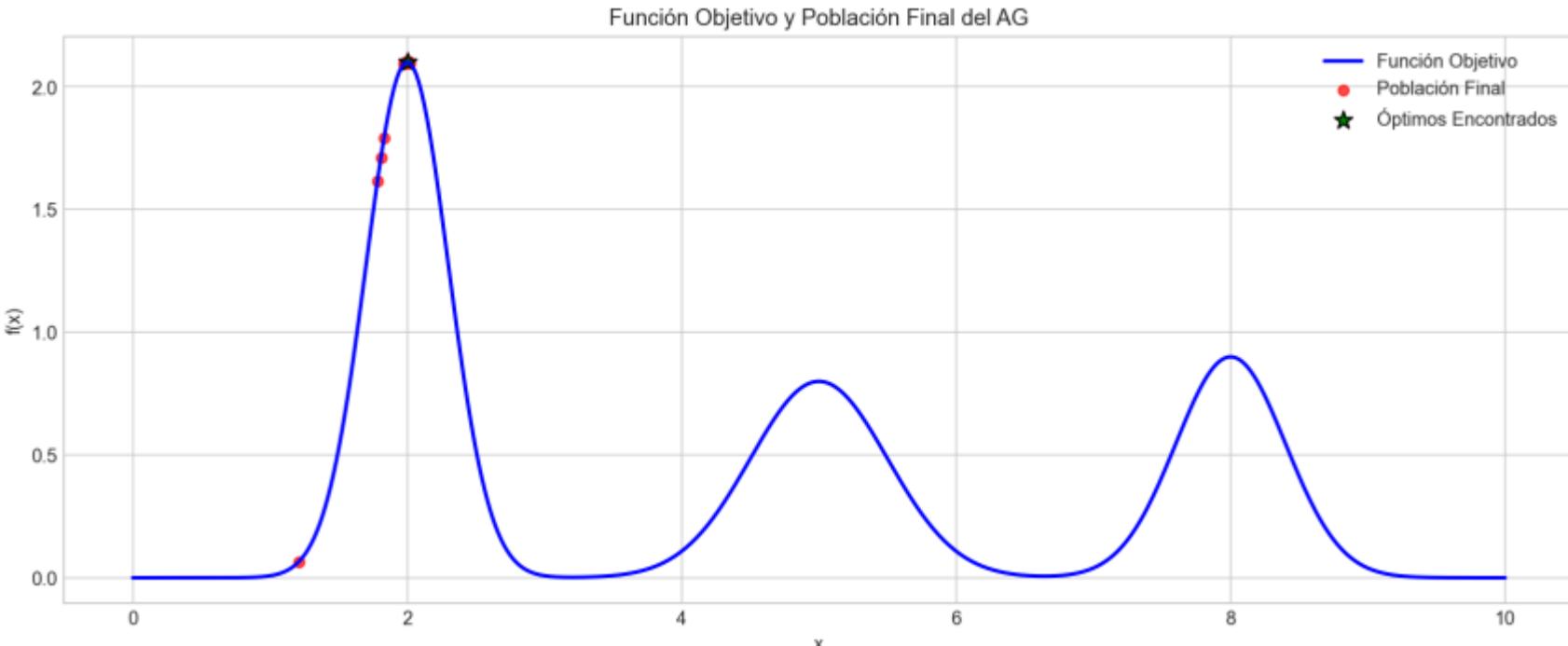
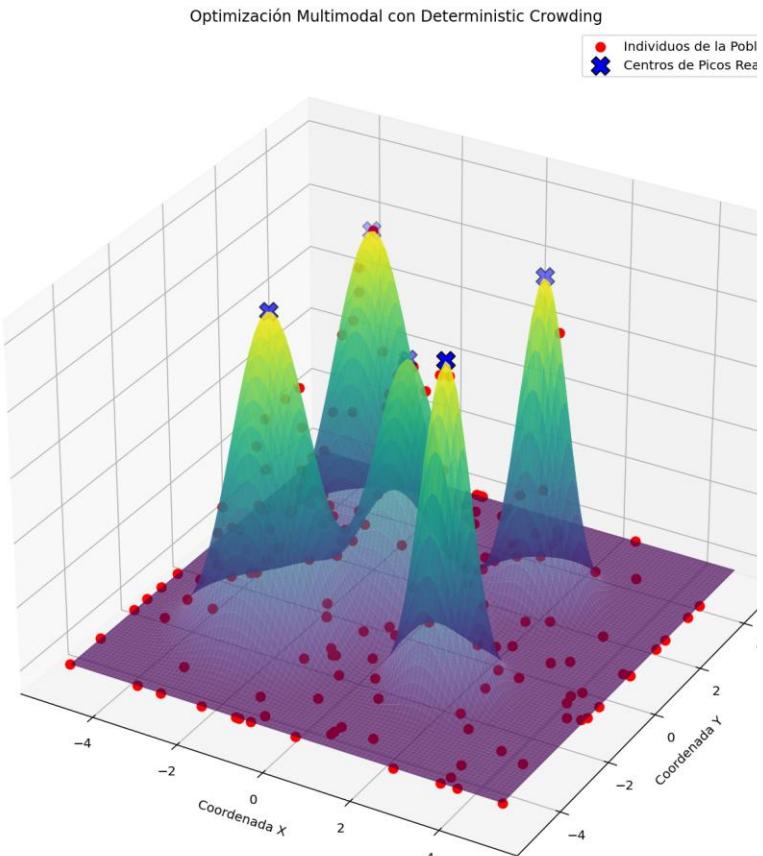
La **exploración** es la búsqueda de una **convergencia eficiente**. Si se explora demasiado, puede perder tiempo buscando en áreas poco prometedoras y no converger a una buena solución de manera eficiente. La **explotación** permite **refinar las soluciones** existentes y acercarse al **óptimo global**. Las estrategias de explotación en AG se centran en refinar las soluciones existentes y en converger hacia un óptimo global. Técnicas comunes:

- Baja Tasa de Mutación en etapas avanzadas
- Selección presionante
- Operadores de cruce conservadores

El dilema fundamental: Exploración vs. Explotación

- El Equilibrio que define el éxito
- Exploración (gracias a la mutación y cruce diverso): buscar en zonas completamente nuevas del mapa.
- Explotación (gracias a la selección y elitismo): excavar más profundo en las zonas prometedoras que ya encontramos.
- Analogía: ¿Deberíamos seguir buscando nuevos restaurantes (exploración) o volver a nuestro favorito para probar otro plato (explotación)?

Más Allá de una Única "Mejor" Solución – Problemas multimodales



- Definición: Problemas que tienen múltiples soluciones óptimas o casi-óptimas.
- El Peligro: Un AG simple convergerá rápidamente a uno de los picos, ignorando por completo los demás.
- ¿Por qué nos importa? En el mundo real, tener varias opciones de alta calidad es extremadamente valioso

Más Allá de una Única "Mejor" Solución – Problemas multimodales

Un AG para Evitar el "Incesto" y Fomentar la Exploración, el algoritmo CHC

- Diseñado para mantener la diversidad y abordar la convergencia prematura.
- Características Clave:
 - ❖ Selección Siempre Elitista: Padres e hijos compiten juntos; solo los mejores sobreviven. La reproducción solo se produce si la distancia de Hamming es mayor al umbral de incesto ($1/4$ del tamaño del cromosoma L).
 - ❖ Cruce HUX con "Control de Incesto": Dos individuos solo se pueden cruzar si son suficientemente "diferentes" (su distancia de Hamming supera un umbral). Esto evita la endogamia.
 - ❖ Sin mutación tradicional. En su lugar mutación catastrófica cuando la población se estanca (no hay cruces posibles), el algoritmo guarda al mejor individuo y reinicia al resto de la población con mutaciones masivas, esto ayuda a escapar de óptimos locales y a explorar nuevas regiones del espacio de búsqueda

Estrategias Evolutivas

Las estrategias evolutivas (EE) son un tipo de algoritmo de optimización evolutiva que se enfoca en la adaptación de los parámetros de la búsqueda a lo largo del tiempo. A diferencia de los AG, las EE suelen representar las soluciones como vectores de números reales y utilizan mutaciones gaussianas para generar nuevas soluciones. Las principales características de las EE son:

- Representación Real
- Mutación Gaussiana
- Adaptación de la desviación estándar.
- Selección determinista o probabilística.

En las más recientes variantes de EE los μ padres generan λ descendencia, cada parente produce λ/μ descendencia en promedio, estas variantes son:

- $(\mu + \lambda)$ -EE: solo los μ mejores individuos obtenidos de la unión de padres e hijos sobreviven.
- (μ, λ) -EE: sólo los μ mejores hijos de la siguiente generación sobreviven.

Programación Evolutiva

La programación evolutiva (PE) es un paradigma de computación evolutiva que se centra en la evolución de estructuras de comportamiento, como máquinas de estados finitos o programas. A diferencia de los AG, la PE no utiliza cruce y se basa únicamente en la mutación para generar nuevas soluciones. Características:

- Representación flexible
- Mutación como operador principal
- Selección estocástica

Los avances recientes de la PE siguen enfatizando el uso de la mutación como único operador, diseñan de forma independiente las desviaciones gaussianas auto adaptativas para variables de valor real y ahora utilizan un esquema de reemplazo de torneo estocástico: cada individuo (entre los $2P$ padres y sus descendientes) se encuentra con T oponentes aleatorios, aumentando su puntuación en un punto si presenta una mejor aptitud. Los P individuos con las puntuaciones más altas pasan a la siguiente generación. El esquema de reemplazo en PE siempre es elitista.

Algoritmos Basados en Evolución Diferencial (ED)

La evolución diferencial (ED) es un algoritmo de optimización evolutiva que utiliza la diferencia entre vectores de soluciones para generar nuevas soluciones. Es especialmente eficaz para problemas de optimización continua. Los algoritmos de Evolución Diferencial es un tipo de algoritmo evolutivo también basados en poblaciones que abarca los procedimientos: cruce, mutación y selección.

La población ED consta de N vectores, y D es la longitud del cromosoma. Inicialmente, se genera una matriz $N * D$ con valores aleatorios de distribución de probabilidad uniforme.

Las características Principales de los algoritmos ED son que usan una representación real, similar a las EE, y las soluciones se representan como vectores de números reales. Tienen una mutación diferencial que se basa en la diferencia entre dos o más vectores seleccionados aleatoriamente de la población. Usan el cruce binomial para combinar la información del vector mutado con el vector original. Cada nuevo vector generado compite directamente con su padre. Si el nuevo vector tiene una mejor aptitud, reemplaza al padre.

Modelos de Evolución Basados en Estimación de Distribuciones (EDA)

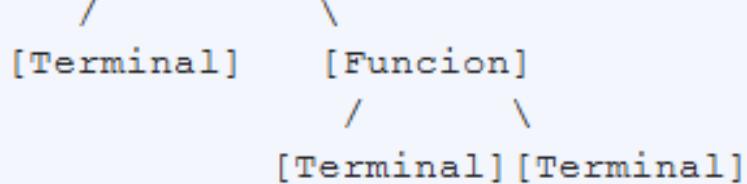
Los algoritmos de estimación de la distribución (EDA) no emplean ni cruce ni mutación, son una clase de algoritmos de optimización que construyen un modelo probabilístico de las soluciones prometedoras y luego muestran nuevas soluciones a partir de este modelo.

Entonces, en lugar de utilizar operadores genéticos como el cruce y la mutación, los EDA aprenden la distribución de probabilidad de las buenas soluciones y generan nuevas soluciones muestreando esta distribución. Los EDA mas conocidos son:

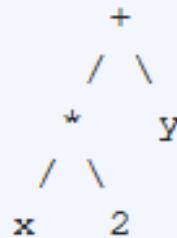
- EDA con independencia entre variables.
- EDA con dependencia entre pares de variables.
- EDA con múltiples dependencias.

Programación Genética (GP)

General: Árbol de Expresión
[Funcion]
/ \



Ejemplo de un Gen (Árbol):



(Representa la expresión: $(x * 2) + y$)

Individuo Cromosoma (4 Árboles - Representación Simplificada)

#1 [Árbol1: ($\sin x$), Árbol2: ($((x^2)-5$), Árbol3: ($\text{if } (> x 1) y x$), Árbol4: ($((y+x)/-1$)]

#2 [Árbol1: ($\cos 2$), Árbol2: (x), Árbol3: ($((y+5)*x$), Árbol4: ($\text{if } (> x y) (x*y) 2$)]

#3 [Árbol1: ($x*y$), Árbol2: ($\sin (x+y)$), Árbol3: (2), Árbol4: ($(- (x / 5))$)]

... ... (47 individuos más con combinaciones de árboles diferentes)

#50 [Árbol1: ($/ x y$), Árbol2: ($+ x (- y 2)$), Árbol3: ($\cos (x*2)$), Árbol4: ($\text{if } (\sin x) y x$)]

Proceso de Génesis Aleatoria

| Individuo #N (Cromosoma de 4 Árboles)

| Gen 1 (Árbol 1):

| Construcción recursiva aleatoria:
| sin (x)

| Gen 2 (Árbol 2):

Construcción recursiva aleatoria:
/ \\
/ \\
* 5
/ \
x 2

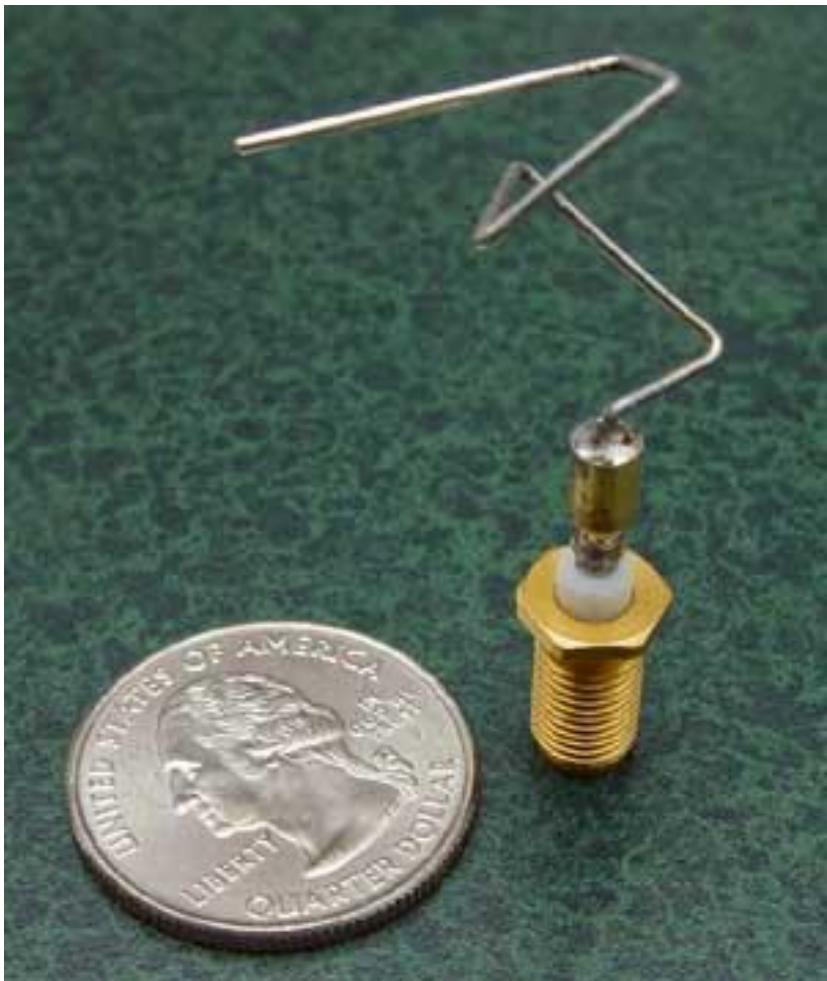
| Gen 3 (Árbol 3):

| Construcción recursiva aleatoria:
| if
| / | \
| > x y
| / \
| x 1

| Gen 4 (Árbol 4):

| Construcción recursiva aleatoria:
| /
| / \
| + -1
| / \
| y x

Cuando la Solución Óptima es Anti-Intuitiva



- Imagen de la antena ST5 de la NASA, con su forma extraña de clip.
- Problema: Diseñar una antena pequeña con un patrón de radiación muy específico.
- Solución con GP: Se evolucionaron diseños de antenas. La solución encontrada por el algoritmo era extraña, no se parecía a nada diseñado por un humano, pero superaba todos los requisitos.
- Lección: La computación evolutiva puede encontrar soluciones innovadoras en espacios que los humanos ni siquiera considerarían.



Powered by
Arizona State University®

Fin de la presentación