# Optimización continua

Algoritmos Genéticos (GA's)

a. Sphere

$$f(x) = \sum_{i=1}^{n} x_i^2$$

$$x_i \in [-5.12, +5.12]$$

b. Ackley

$$f(x) = 20 + e - 20\,exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}(cos\,2\pi\,x_i)\right)$$

$$x_i \in [-30, +30]$$

c. Griewank

$$f(x) = 1 + \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} cos\left(x_i/\sqrt{i}\right)$$

$$x_i \in [-600, +600]$$

d. Rastrigin

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\,cos\left(2\pi x_i\right)]$$

$$x_i \in [-5.12, +5.12]$$

e. Rosenbrock

$$f(x) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2\right]$$

$$x_i \in [-2.048, +2.048]$$

i) Representación binaria para las soluciones

```python
# Representación binaria para las soluciones
def generar_poblacion_binaria(tam_poblacion, tam_solucion):
    return np.random.randint(2, size=(tam_poblacion, tam_solucion))
```


antiguas nuevas
↳ remplazo

- Generar una población inicial de individuos (soluciones) representados como vectores binarios.
- El tamaño de la población (tam_poblacion) determina cuántos individuos habrá en la población (cuantas soluciones "iniciales" se generan)
- El tamaño de la solución (tam_solucion) especifica cuántos bits tendrá cada individuo.

- generar muchas sols (10k, 1m)
  - convergencia rápida
  - ↳ resultados

ii) Selección de padres por el método de la ruleta

```python
# Selección de padres por el método de la ruleta
def seleccion_ruleta(aptitudes):
    total_aptitudes = np.sum(aptitudes)
    probabilidades = aptitudes / total_aptitudes
    padres_indices = np.random.choice(np.arange(len(aptitudes)), size=2, p=probabilidades)
    return padres_indices
```

población n sols
→ evaluar cu sols.
↳ obtener aptitud

- La probabilidad de selección de un individuo (solución) es proporcional a su aptitud.

- Entrada: arreglo de aptitudes de la población
- Salida: Índices de los padres seleccionados para la cruza.

[ 8, 9, 5, 6, 7 ]

- La aptitud se refiere a la medida "qué tan buena es una solución" en términos de la función objetivo que se está optimizando.

↳ opt con ackley isphera

↳ reducir colis
< numero

iii) Operador de cruza de n puntos

**Cruce de un punto**

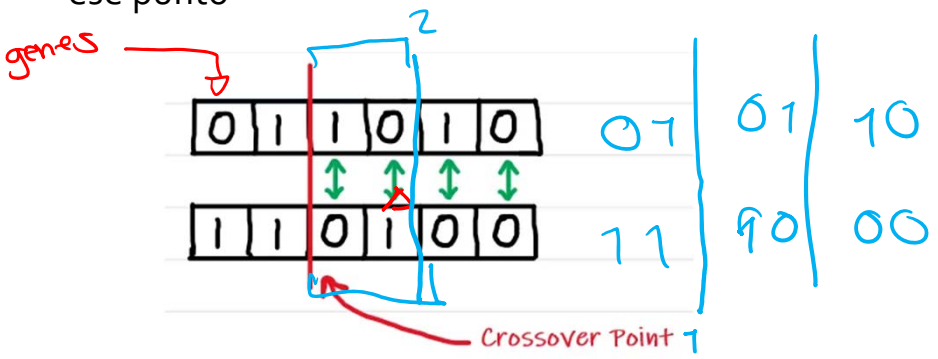
1. Elegir a dos padres para la cruza/cruce/crossover

Parents

P1: 0 1 1 0 1 0

P2: 1 1 0 1 0 0

2. Elegir un punto de cruce aleatorio e intercambiar todo lo que este a la derecha de ese punto




3. Resultados:

Offsprings

O1: 0 1 0 1 0 0

O2: 1 1 1 0 1 0

Proba de cruza: 0.6-0.9

## Operadores de Recombinación


n puntos →

https://medium.com/geekculture/crossover-operators-in-ga-cffa77cddoc8

## iv) Mutación flip

- Realizar pequeñas modificaciones aleatorias en los cromosomas (soluciones) de la población. Estas modificaciones pueden ser cambios en uno o más genes, intercambio de genes, inserción o eliminación.
- Mantener la diversidad genética en la población (evitar estancamiento en mínimos locales y mejorar la capacidad del algoritmo genético para encontrar soluciones óptimas)

- Función que realiza la mutación de un individuo
- Cada bit tiene una probabilidad prob_mutacion de ser invertido (1->0, 0->1).
- Entrada: Toma un individuo (vector binario) y la probabilidad de mutación
- Salida: Devuelve el individuo mutado.

```python
# Mutación flip
def mutacion_flip(individuo, prob_mutacion):
    for i in range(len(individuo)):
        if np.random.rand() < prob_mutacion:
            individuo[i] = 1 - individuo[i]
    return individuo
```

Proba de mutación:
0.01-0.1

v) Reemplazo generacional con elitismo

[ Se debe garantizar que la mejor solución siempre permanece en la población ]

Los mejores individuos de la población actual/inicial se mantienen en la siguiente generación (iteración).
Entrada: población actual, aptitudes de la población, nueva población generada y nuevas aptitudes
Salida: devuelve la población actualizada con los mejores individuos.

```python
# Reemplazo generacional con elitismo
def reemplazo_elitismo(poblacion, aptitudes, nueva_poblacion, nueva_aptitudes):
    mejores_indices = np.argsort(aptitudes)[:len(nueva_poblacion)]
    poblacion[mejores_indices] = nueva_poblacion
    aptitudes[mejores_indices] = nueva_aptitudes
    return poblacion, aptitudes
```

- ite 1600
- tiempo 5 min


aptitud = n

# Graficas evolución de aptitud

Algoritmos Genéticos (GA's)

Representaciones visuales que muestran cómo cambia la aptitud (o valor objetivo) de las soluciones a lo largo del tiempo (de las ejecuciones, de las iteraciones, etc) en un algoritmo de optimización.

Herramienta útil para comprender el comportamiento y la eficacia de un algoritmo en la búsqueda de soluciones óptimas.

Para los algoritmos genéticos, generalmente:
No de generación en el eje X → criterio término
Aptitud promedio/mejor/peor aptitud en el eje Y.

Aptitud Promedio: Esta línea muestra cómo cambia la aptitud promedio de la población en cada generación.
Proporciona info sobre la tendencia general de mejora o empeoramiento de las soluciones a lo largo del tiempo.

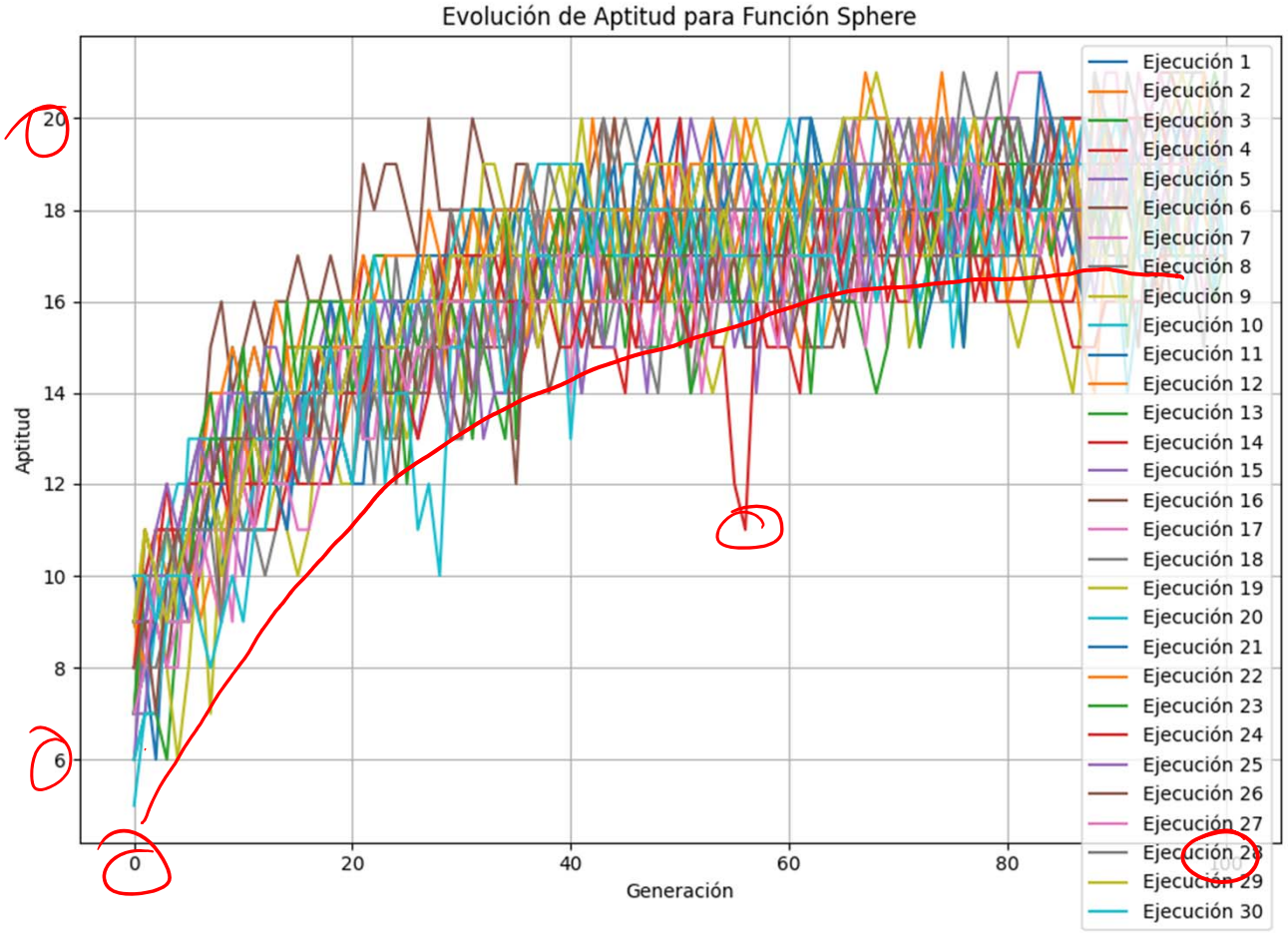
Mejor Aptitud: Representa la mejor aptitud encontrada en cada generación.
Identificar que tan rápido converge el algoritmo hacia soluciones de alta calidad y si alcanza un óptimo global o se estanca en un óptimo local.

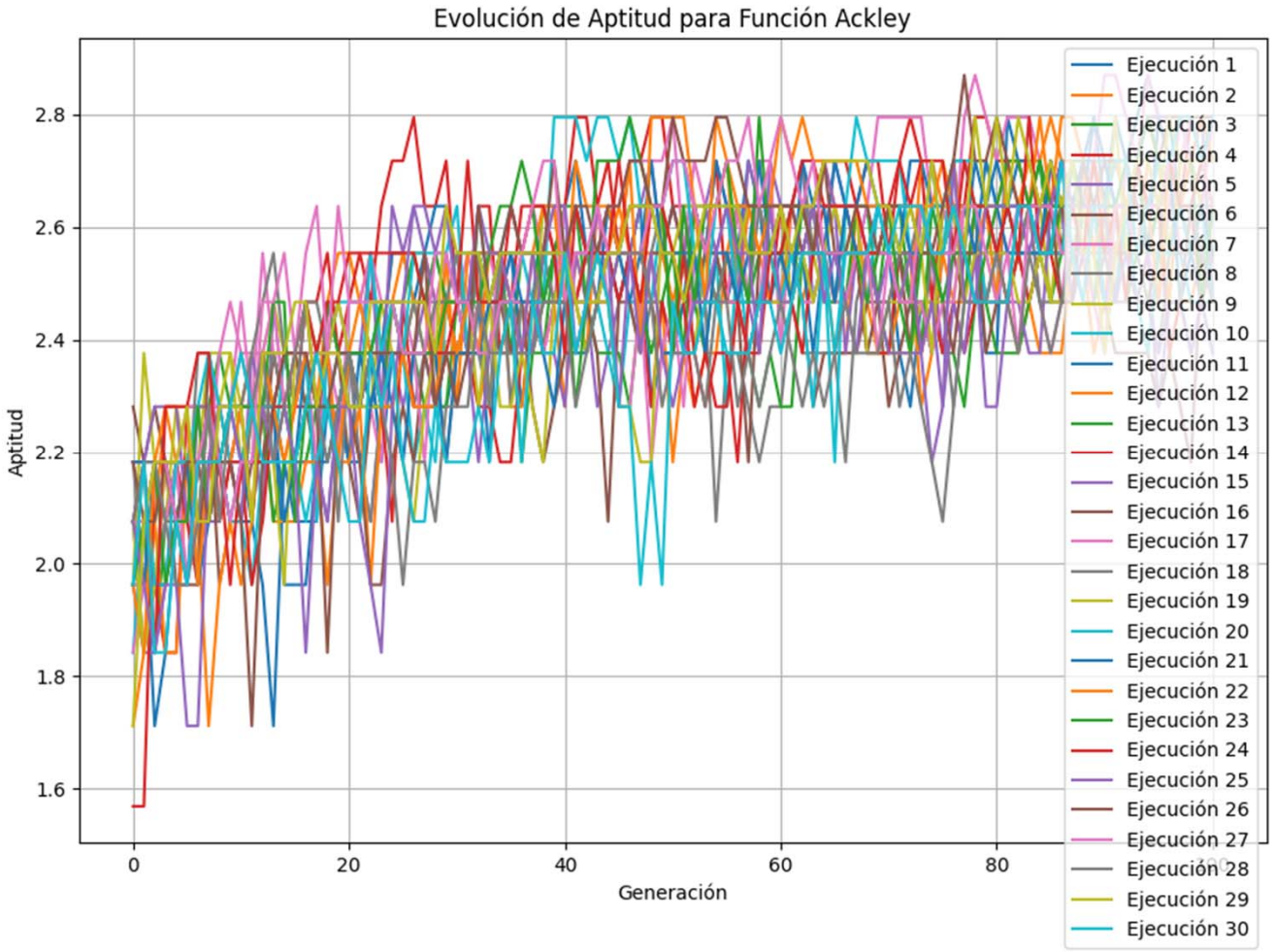
Peor Aptitud: Esta línea muestra la peor aptitud encontrada en cada generación.
Útil para evaluar la diversidad de la población y si el algoritmo es propenso o no a estancarse en mínimos locales.

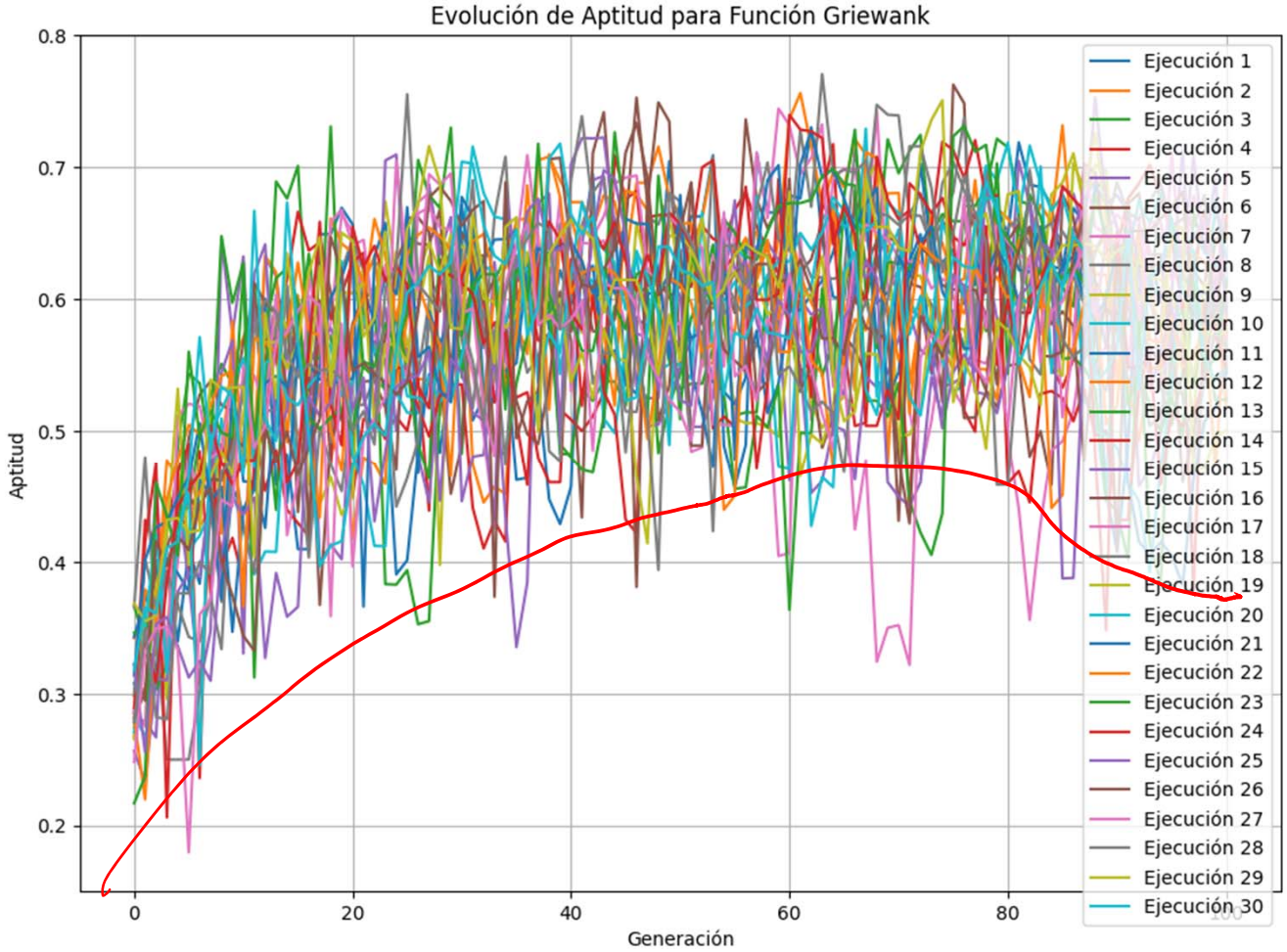
Implementación (Python)

1. *matplotlib* para crear las gráficas
2. Donde se ejecuta el algoritmo genético, registrar la aptitud promedio, la mejor y peor de cada generación.
3. Cuando la ejecución del algoritmo genético termine, usar los datos registrados para crear las gráficas de evolución de aptitud.

```python
# Función para ejecutar el algoritmo genético y graficar la evolución de la aptitud
def ejecutar_y_graficar(funcion_objetivo, nombre_funcion):
    num_ejecuciones = 30
    tam_poblacion = 100
    tam_solucion = 30
    num_generaciones = 100
    prob_mutacion = 0.01

    mejores_aptitudes = []
    for _ in range(num_ejecuciones):
        _, evolucion_aptitudes = algoritmo_genetico(funcion_objetivo, tam_poblacion, tam_solucion, num_generaciones, prob_mutacion)
        mejores_aptitudes.append(evolucion_aptitudes)

    # Graficar la evolución de la aptitud
    plt.figure(figsize=(10, 6))
    for i, evolucion in enumerate(mejores_aptitudes):
        plt.plot(range(num_generaciones + 1), evolucion, label=f'Ejecución {i+1}')
    plt.title(f'Evolución de Aptitud para Función {nombre_funcion}')
    plt.xlabel('Generación')
    plt.ylabel('Aptitud')
    plt.legend()
    plt.grid(True)
    plt.show()

# Ejecutar y graficar para cada función de optimización
funciones = [sphere, ackley, griewank, rastrigin, rosenbrock]
nombres_funciones = ['Sphere', 'Ackley', 'Griewank', 'Rastrigin', 'Rosenbrock']

for funcion, nombre in zip(funciones, nombres_funciones):
    ejecutar_y_graficar(funcion, nombre)
```


Evolución de Aptitud para Función Sphere
Aptitud
Generación
0
20
40
60
80
100
6
8
10
12
14
16
18
20
Ejecución 1
Ejecución 2
Ejecución 3
Ejecución 4
Ejecución 5
Ejecución 6
Ejecución 7
Ejecución 8
Ejecución 9
Ejecución 10
Ejecución 11
Ejecución 12
Ejecución 13
Ejecución 14
Ejecución 15
Ejecución 16
Ejecución 17
Ejecución 18
Ejecución 19
Ejecución 20
Ejecución 21
Ejecución 22
Ejecución 23
Ejecución 24
Ejecución 25
Ejecución 26
Ejecución 27
Ejecución 28
Ejecución 29
Ejecución 30


Evolución de Aptitud para Función Ackley
Aptitud
Generación
2.8
2.6
2.4
2.2
2.0
1.8
1.6
0
20
40
60
80
Ejecución 1
Ejecución 2
Ejecución 3
Ejecución 4
Ejecución 5
Ejecución 6
Ejecución 7
Ejecución 8
Ejecución 9
Ejecución 10
Ejecución 11
Ejecución 12
Ejecución 13
Ejecución 14
Ejecución 15
Ejecución 16
Ejecución 17
Ejecución 18
Ejecución 19
Ejecución 20
Ejecución 21
Ejecución 22
Ejecución 23
Ejecución 24
Ejecución 25
Ejecución 26
Ejecución 27
Ejecución 28
Ejecución 29
Ejecución 30


Evolución de Aptitud para Función Griewank
Aptitud
Generación
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
Ejecución 1
Ejecución 2
Ejecución 3
Ejecución 4
Ejecución 5
Ejecución 6
Ejecución 7
Ejecución 8
Ejecución 9
Ejecución 10
Ejecución 11
Ejecución 12
Ejecución 13
Ejecución 14
Ejecución 15
Ejecución 16
Ejecución 17
Ejecución 18
Ejecución 19
Ejecución 20
Ejecución 21
Ejecución 22
Ejecución 23
Ejecución 24
Ejecución 25
Ejecución 26
Ejecución 27
Ejecución 28
Ejecución 29
Ejecución 30

# Tipos de componentes

Algoritmos Genéticos (GA's)

## Operadores de Recombinación


https://medium.com/geekculture/crossover-operators-in-ga-cffa77cddoc8

**Operadores de Mutación**

**Mutación de cambio de bit (Bit Flip Mutation):**

Seleccionar aleatoriamente uno o más bits en el cromosoma e invierlos (de 0 a 1 o de 1 a 0).

**Mutación de inserción (Insertion Mutation):**

Elegir una posición aleatoria en el cromosoma y se inserta un nuevo gen (o una secuencia de genes) en esa posición.
Por ejemplo:

123456 aplicar la mutación de inserción con el gen 7 en la posición 3

$\Rightarrow$ 1247356

**Mutación de intercambio (Swap Mutation):**

Seleccionar dos posiciones aleatorias en el cromosoma e intercambiar los genes en esas posiciones.
Por ejemplo:

123456 aplicar la mutación swap en las posiciones 2 y 5

$\Rightarrow$ 153426

**Mutación de inversión (Inversion Mutation):**

Seleccionar una subsecuencia aleatoria de genes en el cromosoma y se invierte el orden de los genes en esa subsecuencia.
Por ejemplo:

123456 aplicar mutación de inversión en las posiciones 2 a 5

$\Rightarrow$ 154326

## Operadores de Remplazo

**Reemplazo generacional completo (Full Generational Replacement)**:

Toda la población actual se reemplaza por la nueva descendencia generada en cada generación.
Simple y completa, todos los individuos de la población tienen la misma oportunidad de contribuir a la siguiente generación.
Pérdida de diversidad, si la nueva descendencia no se combina adecuadamente con la población actual.

**Reemplazo por elitismo (Elitism Replacement)**:

Una parte de la población actual (los mejores individuos) se mantiene sin cambios en la siguiente generación, mientras que el resto de la población se reemplaza por la nueva descendencia.
Preservar la información genética de los individuos más aptos y evitar su pérdida.
Puede ayudar a garantizar una convergencia más rápida hacia soluciones de alta calidad
También puede reducir la diversidad genética.

**Reemplazo por torneo (Tournament Replacement)**:

Seleccionar aleatoriamente varios individuos de la población actual y comparar su aptitud.
Los individuos con la mejor aptitud dentro del torneo son seleccionados para formar parte de la siguiente generación, los demás, son reemplazados por la nueva descendencia.
Más robusto que el reemplazo por selección/elitismo de los mejores individuos, ya que no depende de la elección de un número fijo de élites.

**Reemplazo por edad (Age-Based Replacement)**:

Se asigna una edad a cada individuo en la población (aumenta en cada generación).
Los individuos más antiguos son reemplazados por la nueva descendencia, mientras que los individuos más jóvenes tienen la oportunidad de sobrevivir a varias generaciones.
Puede ayudar a mantener la diversidad genética y evitar la convergencia prematura.