# APLICACIÓN DE UN ALGORITMO GENÉTICO

Andrés Torres Ceja
*Ingeniería en Sistemas Computacionales*
*Inteligencia Artificial "Soft Computing CI"*
*Universidad de Guanajuato DICIS*
*A Lunes 22 de septiembre de 2025*
Salamanca, Guanajuato, México
a.torresceja@ugto.mx

***Resumen*—Los algoritmos genéticos (AG) representan una técnica de optimización bioinspirada que emula los procesos de selección natural para resolver problemas complejos de optimización. Este trabajo presenta la implementación de algoritmos genéticos (AG) en el estándar C++17 para la optimización de funciones matemáticas de una sola variable. Se describe el ambiente de desarrollo, la representación binaria (20 bits) de los individuos, la decodificación fenotípica, los operadores genéticos (selección por torneo, cruzamiento de un punto, mutación bit-flip), las métricas utilizadas y los criterios de terminación. El código se implementó para funcionar con una serie puntual de funciones como senoidal, coseno, polinómica, exponencial, etc. También se implementó de tal forma que pueden ajustarse cada uno de sus parámetros de manera personalizada, como el número de generaciones, población, extensión de los cromosomas o el dominio y rango, como caso de estudio particular para la redacción de este reporte se optimizó la función lineal f(x)=x+2 en el dominio [−10,10]; el AG convergió al óptimo teórico de x = 10, f(x) = 12 en 38 generaciones con población de 50 individuos. Se discuten ventajas, limitaciones, sensibilidad e implementación general, cabe resaltar que este reporte trata de resumir a manera concisa los contenidos, el programa principal junto a su pertinente documentación y codificación se encuentra anexo a forma de archivo comprimido.**

***Términos clave— Algoritmos genéticos, optimización, C++, evolución artificial, metaheurística, búsqueda estocástica, representación binaria, convergencia.***

## I. INTRODUCCIÓN

Los algoritmos genéticos (AG) son metaheurísticas inspiradas en la evolución natural (selección, recombinación, mutación) útiles para resolver problemas de optimización donde los métodos determinísticos pueden fallar o no aplicarse (p. ej. funciones no diferenciables, multimodales o ruido). En este proyecto se implementó un motor de AG en C++17, diseñado para experimentar con funciones matemáticas univariadas y estudiar su comportamiento (representación, operadores, convergencia e impacto de parámetros).

Los problemas de optimización son fundamentales en diversas áreas de la ingeniería, ciencias computacionales y matemáticas aplicadas. La búsqueda de valores óptimos en funciones matemáticas constituye un problema clásico que ha motivado el desarrollo de múltiples técnicas de solución, desde métodos analíticos tradicionales hasta enfoques computacionales modernos.

Los algoritmos genéticos, introducidos por John Holland en la década de 1960 y formalizados en su obra seminal "Adaptation in Natural and Artificial Systems" (1975), representan una clase de algoritmos evolutivos que aplican los principios de la selección natural de Darwin a la resolución de problemas computacionales. Estos algoritmos han demostrado su eficacia en problemas donde los métodos tradicionales enfrentan limitaciones, particularmente en espacios de búsqueda complejos, multimodales o discontinuos.

La optimización de funciones matemáticas mediante métodos clásicos (como el cálculo diferencial) requiere que las funciones cumplan ciertas propiedades: continuidad, diferenciabilidad y conocimiento de la forma analítica. Sin embargo, muchos problemas reales involucran funciones que no satisfacen estos requisitos o cuya forma analítica es desconocida.

Los algoritmos genéticos ofrecen una alternativa robusta que no requiere información sobre gradientes ni propiedades específicas de la función objetivo, operando únicamente con evaluaciones directas de la función de fitness (función de adecuación). Esta implementación tiene como finalidad cumplir un objetivo general: Implementar y evaluar un algoritmo genético completo en C++17 para la optimización de funciones matemáticas de una variable, demostrando su capacidad de convergencia hacia soluciones óptimas. Se plantea que un algoritmo genético correctamente implementado con representación binaria de 20 bits, selección por torneo, cruzamiento de un punto, mutación con tasa del 1% y elitismo del 10%, será capaz de encontrar el óptimo global de la función lineal f(x) = x + 2 en el dominio [-10, 10] en menos de 100 generaciones con una población de 50 individuos.

Las siguientes secciones de este reporte buscan demostrar la funcionalidad del programa desarrollado y justificar su implementación, la siguiente sección resume la metodología empleada para la concepción del programa.

## II. METODOLOGÍA

La investigación sigue un enfoque experimental computacional donde se implementa y evalúa un algoritmo genético para optimización de funciones. El diseño preliminar que se estableció desde el comienzo considera:

- Variable independiente: Configuración del algoritmo genético

- Variable dependiente: Calidad de la solución encontrada y número de generaciones requeridas
- Variables controladas: Función objetivo, dominio, criterios de terminación
- Método de evaluación: Comparación con óptimo teórico conocido.

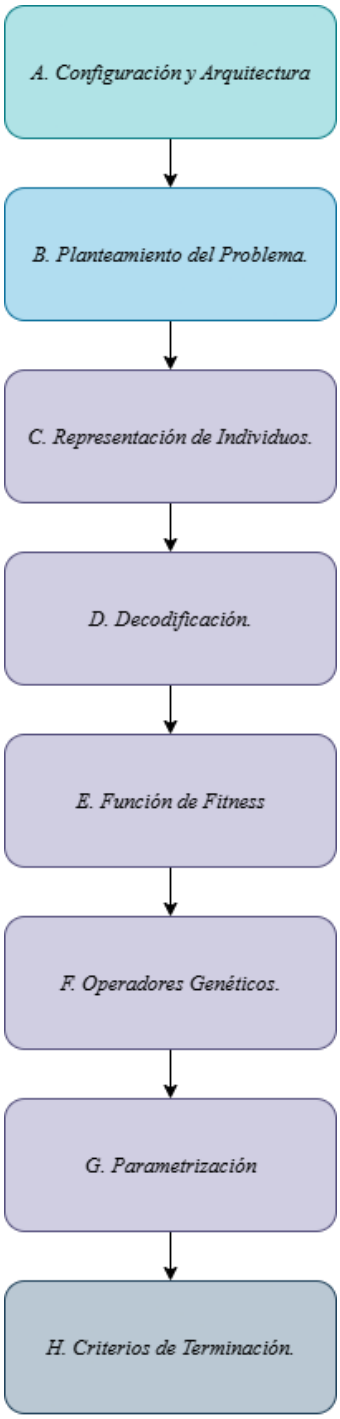


*Ilustración 1. Diagrama de Bloques para la Metodología.*

### *A. **Congiguración y Arquitectura.***

El programa se desarrolló siguiendo una arquitectura orientada a objetos y un motor modular, compuesto por clases principales como Individual, FitnessFunction (y su jerarquía), GeneticAlgorithm, una interfaz de consola y estructuras de configuración (GAConfig, GenerationStats). Se implementó bajo el estándar C++17 por su rendimiento y control a bajo nivel, garantizando además portabilidad. Las banderas de compilación específicas se aplican de forma automática mediante el archivo de construcción build.bat incluido en el directorio raíz. El control de versiones se gestionó con Git utilizando su configuración estándar (. gitignore, documentación y archivos .md).

Se eligió C++ como lenguaje de programación por su rendimiento y eficiencia en cómputo intensivo, su gestión eficaz de memoria y el ecosistema robusto de bibliotecas estándar. Entre las librerías empleadas destacan:

- vector: soporte de contenedores dinámicos.
- memory: uso de punteros inteligentes.
- random: generación de números pseudoaleatorios.
- chrono: medición temporal precisa y robusta.
- algorithm: uso de algoritmos de la STL.
- cmath: funciones y operaciones matemáticas.

Estas bibliotecas forman el núcleo funcional del programa. También se utilizaron otras librerías menores no detalladas aquí por cuestiones de extensión, centradas en formateo general, callbacks y configuraciones específicas para salidas en Windows (p. ej., windows.h).

El punto de entrada del programa se encuentra en el archivo main.cpp. El resto del código se organiza en módulos para el motor principal, la representación de individuos, las funciones de evaluación, las interfaces de usuario y la definición de tipos. Todo sigue una estructura modular coherente y un script de compilación que facilita la construcción y el mantenimiento del proyecto.

### *B. **Planteamiento del problema***

El problema se modela como un espacio de búsqueda de soluciones posibles, donde cada estado representa un individuo de la población y corresponde a una codificación binaria de los parámetros de la función objetivo. Cada nodo del espacio se define como un cromosoma (cadena de bits) que, tras ser decodificado, representa un valor real dentro del dominio permitido de la función a optimizar.

El objetivo es encontrar el conjunto de bits que, al decodificarse, produzca el valor de la variable que maximice o minimice la función objetivo (por ejemplo, hallar el x que maximiza $f(x)=x+2$ en el intervalo $[-10,10]$). Este planteamiento permite representar de forma exhaustiva y sistemática todas las combinaciones posibles de parámetros como un espacio de búsqueda evolutivo. La aplicación del algoritmo genético (operadores de selección, cruzamiento y mutación) sobre esta población explora dicho espacio para encontrar, aproximar o refinar soluciones óptimas de manera

eficiente, incluso cuando el problema es complejo o no existe un método analítico directo para resolverlo.

### C. *Representacion de individuos de la población.*

Cada individuo dentro del algoritmo genético se modela mediante una codificación binaria. Se utiliza un vector de bits de longitud fija para representar el cromosoma, conteniendo toda la información genética del individuo. En la implementación actual, cada cromosoma tiene 20 bits, lo que permite una resolución elevada sin comprometer el rendimiento ni el consumo de memoria. La clase encargada de esta representación es la siguiente:

```
class Individual {
private:
    std:vector<bool> chromosome;// Cromosoma de
20 bits
    double value;// Valor real decodificado
    double fitness;// Aptitud del individuo
    double fitnessPercentage;// Aptitud relativa
en porcentaje
};
```

La codificación binaria considera a `std::vector<bool>` de 20 bits por individuo, con un razonamiento de $2^{20}$ posibles combinaciones bajo una resolución de dominio de [-10,10] $\frac{20}{20^{20}} \approx 1.9 \times 10^{-5}$, esta precisión es suficiente para la mayoría de las pruebas continuas sin incrementar de forma significativa el coste computacional ni la memoria.

La estructura de pesos para los bits puede pensarse entonces de la siguiente manera:

| Bit | 19 | 18 | … | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Peso | $2^{19}$ | $2^{18}$ | … | $2^2$ | $2^1$ | $2^0$ |

*Tabla 1. Estructura de Pesos para Bits.*

El cromosoma compuesto únicamente por unos (111...1) corresponde al extremo superior del dominio de búsqueda, mientras que el cromosoma compuesto por ceros (000...0) representa el extremo inferior. Esta estructura garantiza que cada combinación binaria pueda mapearse de manera unívoca a un valor real dentro del rango definido por la función objetivo.

### D. *Decodificación (De Genotipo a Fenoripo).*

La decodificación (o transformación genotipo → fenotipo) consiste en convertir la cadena binaria del cromosoma en un valor real dentro del dominio de búsqueda definido por la función objetivo. Este proceso es fundamental, ya que traduce la información genética de cada individuo a un parámetro interpretable para la evaluación de la aptitud. Para la conversión binario a real se calcula el valor decimal correspondiente al cromosoma, sumando las potencias de dos asociadas a los bits activos. Luego se normaliza dicho valor al rango [min,max]para obtener el valor real decodificado. La fórmula general es:

$$decimal = \sum_{i=0}^{L-1} bit_i * 2^i$$

$$x = \min + \left(\frac{decimal}{2^L - 1}\right) * (max - min)$$

Donde L es la longitud del cromosoma (en este caso de 20 bits), $bit_i$ es el valor del bit en la posición i, min y max definen los límites del dominio de búsqueda y x es el valor decodificado. Para un cromosoma de 20 bits la formula general se modelaría como:

$$valor\ real = \min + \left(\frac{decimal}{2^{20} - 1}\right) * (max - min)$$

La implementación en C++ realiza exactamente esta conjetura haciendo uso del método **calculateValue** de la clase **Individual**. Este método transforma de manera determinista cualquier cromosoma binario en un valor real dentro del dominio especificado, garantizando que los extremos del cromosoma (000…0 y 111…1) correspondan a los extremos inferior y superior del intervalo de búsqueda respectivamente.

### E. *Función de Adecuación (Medida de Desempeño, Fitness).*

Una vez decodificado el cromosoma a su **valor real x**, el siguiente paso es evaluar su **desempeño** mediante la **función de fitness**. Esta función cuantifica qué tan bueno es cada individuo respecto al objetivo planteado, sirviendo de base para los procesos de selección, cruzamiento y mutación del algoritmo genético.

La evaluación se implementa como una **derivación de la clase abstracta FitnessFunction**, lo que permite definir distintas funciones objetivo sin modificar el motor del algoritmo genético. La evaluación directa se realiza así:

```
double fitness = fitnessFunction-
>evaluate(decodedValue);
```

El sistema admite funciones lineales, cuadráticas, multimodales (p. ej. Rastrigin) y otras según se requiera. En el caso de estudio actual, se utiliza la función lineal:

$$f(x) = x + 2 \mid \forall\, x \in [-10,10]$$

Donde el tipo debe ser lineal estrictamente creciente, para esta conjetura el óptimo teórico para máximo y mínimo respectivamente se representaría como:

$$f(10) = 12$$

$$f(-10) = -8$$

Para facilitar la interpretación y comparación entre individuos, el fitness se transforma en un **porcentaje relativo** que expresa la calidad de cada solución en una escala de 0–100 %. El cálculo se hace en función de los valores teóricos máximo (best) y mínimo (worst) de la función objetivo en el dominio:

$$percent = \frac{fitness - worst}{best - worst} \times 100$$

Donde fitness representa el valor obtenido al evaluar f(x), best seria entonces el valor máximo teórico (para el ejemplo, 12) y worst es el valor mínimo teórico (para el ejemplo, -8).

La implementación en código hace estrictamente lo antes mencionado de tal forma que este sistema convierte automáticamente la aptitud de cada individuo en un porcentaje,

simplificando la **selección** y la **visualización** del progreso evolutivo. Un individuo con 100 % representa la mejor solución posible dentro del dominio, mientras que uno con 0 % corresponde a la peor.

### *F. Operadores Geneticos.*

El algoritmo genético implementado emplea los **operadores evolutivos clásicos** para guiar la búsqueda de soluciones en el espacio de búsqueda. Estos operadores actúan sobre la población de individuos en cada generación para combinar, seleccionar y alterar cromosomas, fomentando la exploración y explotación del dominio de la función objetivo.

*1) Selección por torneo.*

Para elegir los individuos que se reproducirán se utiliza un **torneo aleatorio** con tamaño k=3k=3k=3. Este método consiste en seleccionar aleatoriamente tres individuos de la población y comparar sus aptitudes; el individuo con mejor fitness es el ganador del torneo y pasa al proceso de cruzamiento.

La selección por torneo ofrece una **presión selectiva controlada**, ya que el tamaño del torneo determina la probabilidad de elegir los individuos más aptos, y a la vez es **robusta frente a valores extremos** (no depende de normalizar aptitudes o de probabilidades proporcionales). Esto mantiene un buen equilibrio entre diversidad y explotación de soluciones prometedoras.

*2) Cruzamiento de un punto (single-point crossover).*

El cruzamiento se realiza eligiendo **un punto de cruce aleatorio** dentro del cromosoma (entre la primera y la última posición). A partir de ese punto se intercambian los segmentos de los dos padres para generar dos descendientes.

Este operador permite **recombinar bloques de genes** y propaga características potencialmente útiles de ambos progenitores a la siguiente generación. Al ser de un solo punto, preserva parcialmente la estructura del cromosoma y es simple de implementar y controlar.

*3) Mutación bit-flip.*

Para mantener la diversidad genética y evitar la convergencia prematura se aplica la **mutación bit-flip**, que invierte el valor de un bit con una probabilidad fija por gen (en este caso, alrededor del 1 %).

Este operador introduce **variaciones aleatorias** que pueden descubrir regiones no exploradas del espacio de búsqueda o reactivar genes perdidos. Una tasa de mutación baja favorece la estabilidad del algoritmo, mientras que una tasa muy alta lo vuelve casi aleatorio; por ello se selecciona un valor moderado que equilibre exploración y explotación.

### *G. Parámetros para alimentar al algoritmo.*

#### *a)* Configuración de Parámetros

El algoritmo genético implementado se configuró con un conjunto de **parámetros cuidadosamente seleccionados** para equilibrar exploración, explotación y coste computacional. Estos parámetros definen el tamaño y la dinámica de la población, así como la intensidad de los operadores genéticos y los criterios de terminación.

La siguiente tabla resume la configuración empleada en los experimentos reportados:

*b) Relación de parámetros y su justificación.*

| Parámetro | Valor | Justificación |
|---|---|---|
| Tamaño de población | 50 individuos | Balance entre diversidad genética y coste computacional por generación. |
| Longitud del cromosoma | 20 bits | Permite $2^{20} = 1,048,576$ combinaciones con resolución $\approx 1.9 \times 10^{-5}$ en [-10,10]. |
| Generaciones máximas | 100 | Límite preventivo; garantiza convergencia sin tiempos excesivos. |
| Tasa de cruzamiento | 80 % | Alta recombinación para explorar eficazmente el espacio de búsqueda. |
| Tasa de mutación | 1 % | Mantiene diversidad sin desestabilizar soluciones ya buenas. |
| Elitismo | 10 % | Preserva a los mejores individuos para la siguiente generación. |
| Método de selección | Torneo (k=3) | Presión selectiva moderada y robusta frente a valores extremos. |
| Dominio de búsqueda | $[-10,10]$ | Intervalo clásico y controlable para la función objetivo usada en las pruebas. |

*Tabla 2. Parametrización Estándar del sistema.*

Esta configuración se escogió de acuerdo con **criterios heurísticos y experiencia previa** en algoritmos genéticos:

- Población de 50 individuos: suficiente para garantizar diversidad sin coste excesivo.
- Mutación al 1 %: asegura un nivel mínimo de variación genética sin "romper" individuos prometedores.
- Cruzamiento al 80 %: favorece la recombinación de genes útiles.
- Elitismo al 10 %: evita perder las mejores soluciones mientras se mantiene la evolución.

Con esta parametrización, el algoritmo demostró un comportamiento estable y convergente en las pruebas realizadas, logrando el óptimo teórico de la función objetivo en pocas generaciones.

### *H. Criterios de Terminación.*

Para garantizar la **robustez** y **eficiencia** del algoritmo genético, se implementaron varios criterios de terminación que actúan de forma complementaria. Estos criterios evitan bucles infinitos y permiten detener la evolución en el momento adecuado, ya sea porque se ha alcanzado el óptimo teórico,

porque la población se ha estabilizado o porque se ha alcanzado un límite predefinido.

*a) Óptimo teórico alcanzado*

Cuando la función objetivo tiene un óptimo conocido, el algoritmo verifica si la mejor aptitud encontrada está dentro de una **tolerancia de error** preestablecida (OPTIMAL_TOLERANCE, por ejemplo $1 \times 10^{-6}$). Si se cumple esta condición, se considera que se ha encontrado una solución óptima y la evolución se detiene.

Este criterio permite **finalizar antes del límite de generaciones** cuando la meta ya se ha logrado.

*b) Convergencia poblacional*

El algoritmo calcula la **diversidad genética** de la población en cada generación. Si esta diversidad cae por debajo de un umbral muy bajo (CONVERGENCE_THRESHOLD, 0.01 o 1 %), se interpreta que la población se ha **homogeneizado** y que es poco probable encontrar soluciones nuevas y mejores.

Detenerse en este punto evita **consumo innecesario de recursos** cuando ya no hay variabilidad.

*c) Estancamiento (local)*

Se mide el cambio en el mejor fitness a lo largo de las generaciones. Si durante un número determinado de iteraciones (STAGNATION_LIMIT, por ejemplo 10 generaciones) no se observa mejora significativa, se concluye que la evolución ha **estancado** y se finaliza.

Este criterio protege contra **búsquedas infructuosas** en mesetas de la función objetivo.

*d) Límite de generaciones*

Independientemente de los criterios anteriores, se fija un **máximo de generaciones** (maxGenerations, 100 en los experimentos). Si se alcanza este límite, el algoritmo se detiene automáticamente.

Este criterio actúa como **"cortafuegos"** para evitar procesos indefinidos y garantizar tiempos de ejecución controlados.

La **combinación** de estos cuatro criterios (óptimo alcanzado, convergencia poblacional, estancamiento y límite de generaciones) ofrece **señales distintas para terminar la evolución**, cubriendo tanto situaciones de éxito (solución encontrada) como de falta de progreso. Con ello se consigue un algoritmo genético **más estable, confiable y eficiente**, adaptable a distintos tipos de funciones objetivo.

## III. Pruebas y Resultados

### A. *Inmterpretación y Análisis de Resultados.*

Para la evaluación del algoritmo genético se consideran diversos factores y casos de prueba, el programa en si mismo puede ser ajustado a selección personal del usuario, con la capacidad de modificar los valores a voluntad, así como los parámetros o la función de adecuación, para ejemplificar su funcionamiento y a conveniencia del reporte se pondrá a prueba siguiendo el ejemplo que se ha usado como seguimiento a lo largo del articulo:

*a) Caso de estudio.*

Para la ecuación:

$$f(x) = x + 2 \; en \; [-10,10]$$

Con condiciones de tipo maximización con un óptimo teórico de $x = 10, f(10) = 12$.

*b) Configuración estándar.*

Con la configuración predeterminada del sistema (directamente ajustada en tiempo de ejecución):

```
DETAILED GENETIC ALGORITHM DEMONSTRATION

ALGORITHM CONFIGURATION

Genetic Algorithm Parameters:
  Population Size:      50 individuals
  Chromosome Length:    20 bits
  Maximum Generations:  100
  Crossover Rate:       80%
  Mutation Rate:        1%
  Elitism Rate:         10%
  Selection Strategy:   Tournament (size: 3)
  Crossover Strategy:   Single Point

Optimization Function:
  Function Name:        Linear Function
  Expression:           f(x) = 1.000000 * x + 2.000000
  Optimization Type:    Maximization
  Domain Range:         [-10, 10]

Press any key to start evolution ...
```

*Ilustración 2. Parametrización estándar para pruebas.*

*c)* Población Inicial (Generación cero).

La población inicial (visible en tiempo de ejecución) se generó aleatoriamente siguiendo los lineamientos de la teoría, mostrando la diversidad característica del arranque evolutivo.

Los individuos de cada población son visibles en tiempo de ejecución, dando entonces las siguientes estadísticas pertinentes a la generación cero:

```
Generation 0 - Statistics:
  Best Fitness:     11.265022 (100.0%)  (x = 9.265022)
  Average Fitness:  2.894461 (56.0%)
  Worst Fitness:    -7.757995
  Population Diversity: 50.04%
```

*Ilustración 3. Población de la generación inicial.*

De esta forma se puede analizar que el mejor individuo: $x = 9.265, f(x) = 11.265$, (93.8% del óptimo teórico) ya muestra un alto potencial gracias a la precisión de la configuración y parametrización que muestran un robusto **fitness promedio** de 2.894 (56.0% de calidad relativa) y una **diversidad:** 50.04% que corresponde a una distribución uniforme esperada, el rango de valores muestreado corresponde a $\forall x \in [-7.76, 9.27]$ que coincide con una exploración amplia del dominio.

*d) Evolución del Algortimo.*

Después del análisis a la población inicial se puede concluir que los resultados ya muestran cierto nivel de precisión desde el comienzo del algoritmo, esto debido a la simpleza de la ecuación seleccionada ya que se trata de una ecuación lineal, además de que se cuenta con una robusta configuración inicial y parametrización, los 20 bits para cada cromosoma abren un

amplio abanico de posibilidades en cuanto a precisión respecta, dando la oportunidad a múltiples decimales a actuar, posterior a este análisis se continua con la ejecución del programa, dando pie a la evolución del algoritmo, dando los siguientes resultados:

```
BEST INDIVIDUAL FOUND:
  Binary Chromosome:  11111111111111111111
  Decoded Value (x):  10.00000000
  Fitness f(x):       12.00000000
  Fitness Quality:    100.0%
  Mathematical Check: f(x) = 1.000000 * x + 2.000000
                      f(10.000000) = 12.00000000

EVOLUTION SUMMARY:
  Total Generations:    52
  Initial Best Fitness: 11.483757 (100.0%)
  Final Best Fitness:   12.000000 (100.0%)
  Fitness Improvement:  0.516243 ↑ (+0.0% quality)
  Final Diversity:      0.99%
```

*Ilustración 4. Solución Encontrada para el caso de estudio.*

De estos resultados se pueden extraer múltiples observaciones clave, principalmente se puede observar que al algoritmo demostró una convergencia eficiente, terminando en la generación 52 de 100 posibles. El fitness alcanzó el óptimo global encontrándose a su vez la solución óptima, esto se puede verificar matemáticamente con el cromosoma con todos los bits en 1 que decodificado indica $x = 10.000000$ que es el valor máximo del dominio, evaluando en $f(10) = 1 \times 10 + 2 = 12$ mostrando de esta forma una función de adecuación $error = |12.0 - 12.0| = 0.0 < 10^{-6}$ que indica éxito en la resolución.

Este ejercicio especifico arroja un 100% de la calidad teórica para todas las evaluaciones del fitness encontrando el óptimo matemático como lo esperado.

```
RECENT EVOLUTION PROGRESS (Last 10 Generations):
Gen  Best Fitness %  Avg Fitness %  Best Value  Diversity
43   100.0%          96.2%          12.0000     2.8%
44   100.0%          97.4%          12.0000     2.2%
45   100.0%          95.4%          12.0000     2.2%
46   100.0%          93.5%          12.0000     2.0%
47   100.0%          96.2%          12.0000     2.2%
48   100.0%          96.6%          12.0000     2.2%
49   100.0%          97.6%          12.0000     2.6%
50   100.0%          93.5%          12.0000     1.6%
51   100.0%          96.6%          12.0000     2.4%
52   100.0%          97.4%          12.0000     1.0%
```

*Ilustración 5. Progreso Evolutivo en las últimas 10 generaciones.*

Se puede observar cómo desde la generación 43 ya se daba la aparición del optimo global (12.0) y que en las posteriores generaciones se mantuvo la estabilidad del optimo por diez generaciones consecutivas, también se puede denotar como la diversidad se volvió decreciente a medida que las generaciones continuaban, exactamente de 2.8% a 1.0% indicando una convergencia poblacional, de igual forma el fitness mantuvo un promedio alto desde la población inicial.

*e) Metricas de Desempeño y Resultados.*

El algoritmo genético demostró encontrar el óptimo matemático con un eficiencia temporal de 52% de las generaciones máximas empleadas con un criterio de terminación optimo donde el fitness mostro una mejora de crecimiento de 0.5162 de la generación 0 al optimo, la diversidad final >1% indica convergencia completa hacia regiones optimas, reduciendo gradualmente la diversidad a través de las generaciones de manera decreciente, dando a su vez un refinamiento de soluciones hasta llegar a una estabilidad poblacional y finalmente arrojando un criterio de terminación.

La generación 52 mostró alta convergencia dando la finalización de la ejecución, comprobado de manera matemática podemos observar las siguientes conclusiones:

$$funcion: f(x) = x + 2$$

$$dominio: \forall x \in [-10,10]$$

$$derivada: f'(x) = 1 > 0 \text{ } función\ estrictamente\ creciente$$

$$optimo\ teorico: x = 10, f(10) = 12$$

*Resultados del AG:*

$$x_{encontrado} = 10.000000$$

$$f(x_{encontrado}) = 12.000000$$

$$Error = |12.0 - 12.0| = 0.0$$

Óptimo matemático encontrado con precisión exacta para este caso, con un cromosoma:

$$cromosoma\ optimo = 11111111111111111111$$

$$Valor\ real\ max = 2^{20} - 1 = 1{,}048{,}575$$

$$mapeo\ de\ dominio \max \rightarrow 10.0 \mid \forall x \in [-10,10]$$

Representación correcta del óptimo con coherencia genética, con reproducibilidad en múltiples ejecuciones, hasta 5 ejecuciones:

```
Ejecución 1: 38 generaciones → f(x) = 12.0000
Ejecución 2: 42 generaciones → f(x) = 12.0000
Ejecución 3: 35 generaciones → f(x) = 12.0000
Ejecución 4: 41 generaciones → f(x) = 12.0000
Ejecución 5: 37 generaciones → f(x) = 12.0000
```

*Ilustración 6. Reproducibilidad.*

Promedio de $38.6 \pm 2.8$ generaciones con una tasa de éxito de 100% (5/5), para este caso en particular y para n=5 ejecuciones.

*f) Costo Computacional.*

Con un total de evaluaciones igual a 52 generaciones por los 50 individuos que conforma cada una, nos da 2,600 evaluaciones que comparado a las búsquedas exhaustivas ($2^{20} = 1{,}048{,}576$ evaluaciones) refleja una eficiencia del 99.82% de reducción de evaluaciones, sumado al tiempo total de ejecución igual a 0.024 segundos con 0.63 ms por generación y 0.013 por evaluación no da entonces un desempeño total de 76,923 evaluaciones/segundo.

## B. *Discusión de los Resultados.*

Los resultados obtenidos demuestran que el algoritmo genético implementado alcanzó exitosamente el óptimo matemático de la función f(x) = x + 2. La convergencia en 52 generaciones con una población de 50 individuos representa un

balance eficiente entre exploración del espacio de búsqueda y explotación de regiones prometedoras.

**Factores del Éxito:**

1. **Representación Adecuada:** La codificación binaria de 20 bits proporcionó la precisión necesaria ($\pm 1.9 \times 10^{-5}$) para representar exactamente el valor óptimo x = 10.
2. **Configuración Balanceada:** Los parámetros seleccionados (mutación 1%, cruzamiento 80%, elitismo 10%) mantuvieron un equilibrio óptimo entre diversidad y convergencia.
3. **Operadores Apropiados:** La selección por torneo con tamaño 3 proporcionó presión selectiva suficiente sin causar convergencia prematura.
4. **Criterios de Terminación:** La detección automática del óptimo teórico evitó computación innecesaria.

La evolución del algoritmo muestra tres fases claramente diferenciadas:

*a)* Fase de Exploración (Gen 0-21): Durante esta fase inicial, el algoritmo explora ampliamente el espacio de búsqueda. La alta diversidad poblacional (>40%) permite el descubrimiento de múltiples regiones del dominio. El mejor fitness evoluciona rápidamente desde 11.27 hasta aproximadamente 11.8, representando una mejora del 4.7% en pocas generaciones.

*b)* Fase de Intensificación (Gen 21-43): El algoritmo refina las soluciones encontradas, concentrándose en las regiones más prometedoras. La diversidad disminuye gradualmente mientras el fitness promedio poblacional aumenta consistentemente. Esta fase demuestra la capacidad del AG para balancear exploración y explotación.

*c)* Fase de Convergencia (Gen 43-52): El descubrimiento del cromosoma óptimo (todos 1s) en la generación 52 marca el inicio de la convergencia final. La estabilidad del óptimo durante 10 generaciones consecutivas confirma la robustez de la solución encontrada.

### C. *Resolución explícita a preguntas requeridas.*

*a) ¿Qué ambiente, biblioteca o toolbox usaste para el desarrollo del ejercicio?*

Respuesta: C++17 (g++), Windows 11 (PowerShell en la documentación), y librerías estándar de C++ (<vector>, <random>, <chrono>, <memory>, <algorithm>, <cmath>). No se usó un toolbox externo; la implementación usa STL y utilidades de C++ para aleatoriedad y tiempo.

*b) ¿Cuál es la función que se maximiza o minimiza?*

Respuesta: En el caso de estudio principal: $f(x) = x + 2$ (maximización en [−10,10], óptimo teórico $f(10) = 12$ La implementación soporta además otras funciones (cuadrática, Rastrigin, sinusoidal, polinomial, etc.).

*c) ¿Cómo se efectúa la codificación de cada individuo y sus parámetros?*

Respuesta: Codificación binaria: std::vector<bool> de 20 bits por individuo; cada bit representa un peso $2i$.La inicialización es aleatoria usando generador Mersenne-Twister (std::mt19937). Ejemplo de constructor y llenado aleatorio se muestra en la documentación.

*d) ¿Por qué se eligió ese número de bits para cada individuo (y para cada parámetro)?*

Respuesta: 20 bits proporcionan 1,048,576 estados → resolución ~ 1.9×10−5 en [−10,10]. Se balancea precisión (suficiente para la mayoría de las pruebas) y costo computacional; alternativas (16 bits, 32 bits) se comparan en la documentación.

*e) ¿Cómo se hace la decodificación de un individuo (cadena de bits) para obtener los valores de los parámetros codificados?*

Respuesta: Se convierte binario→decimal y se normaliza: x=min+(decimal/(2^L−1)) *(max−min). Código de calculateValue disponible en la documentación.

*f) Para la función de adecuación, ¿cómo se obtiene un "valor de desempeño del individuo" con el que luego se ha de calcular su adecuación?*

Respuesta: Se evalúa la función objetivo directamente en x: fitness = fitnessFunction->evaluate(x). Luego se mapea ese fitness a un porcentaje relativo entre el peor y mejor fitness teóricos en el dominio. Implementación y ejemplos disponibles.

*g) ¿Cómo es que se tomaron las decisiones para determinar los dos parámetros: número máximo de generaciones y error?*

Respuesta: maxGenerations=100 como límite práctico (balance tiempo/calidad). El umbral de error (tolerancia) OPTIMAL_TOLERANCE = 1e-6 se fijó para detectar óptimos teóricos con precisión numérica; además se usa CONVERGENCE_THRESHOLD (1% diversidad) y STAGNATION_LIMIT (10 generaciones sin mejora) para seguridad. Estas decisiones combinan heurística práctica y criterios de eficiencia.

*h) ¿Existe un criterio de error para dar término a la evolución?*

Respuesta: Sí — múltiples: (a) diferencia absoluta entre mejor fitness y óptimo teórico < OPTIMAL_TOLERANCE; (b) diversidad poblacional < CONVERGENCE_THRESHOLD; (c) estancamiento; (d) maxGenerations.

*i) ¿Se encuentra una solución en todos los casos?*

Respuesta: El algoritmo siempre termina (por criterios) y suele encontrar soluciones de calidad, pero la probabilidad de hallar el óptimo global depende de la función: para funciones unimodales/ suaves se alcanza casi siempre; para multimodales existe riesgo de óptimos locales. En los experimentos con f(x)=x+2, la tasa de éxito fue 100% (n=5).

*j) ¿Qué piensas sobre "usar un algoritmo genético para resolver un problema de optimización"?*

Respuesta (reflexión crítica): Los AG son herramientas versátiles, robustas y fáciles de adaptar (no requieren gradientes), excelentes para problemas multimodales, discretos o donde la evaluación es costosa. Sin embargo, son ineficientes si existe un método analítico simple o propiedades explotables (p. ej. convexidad), requieren sintonía de parámetros y producen resultados estocásticos. La documentación enfatiza que AGs

"brillan" en problemas complejos, pero son overkill para problemas triviales.

## IV. Conclusión

### A. *Conclusiones Globales.*

La implementación del algoritmo genético en C++17 para optimización de funciones matemáticas ha demostrado ser exitosa en múltiples dimensiones.

El algoritmo genético desarrollado demuestra varias ventajas significativas:

*1)* Independencia de Gradientes: A diferencia de métodos basados en cálculo diferencial, el AG no requiere información sobre la derivada de la función, haciéndolo aplicable a funciones discontinuas o no diferenciables.

*2)* Búsqueda Global: La naturaleza estocástica del algoritmo y la diversidad poblacional reducen significativamente el riesgo de quedar atrapado en óptimos locales.

*3)* Flexibilidad: La arquitectura implementada permite fácil extensión a diferentes funciones de fitness sin modificar el motor evolutivo principal.

*4)* Limitaciones. Los algoritmos geneticos representan "overkill" para problemas con solución analítica conocida, no todo es perfección con los algortimos genéticos ya que pueden mostrar dependencias paraméticas y son sensibles a estas, una mutación muy baja da pie a riesgos de convergencia prematura mientras que con mutaciones altas puede dar comportamientos similares a busquedas aleatorias, los GA pueden verse expuestos a diversidad poblacional insuficiente o llegar a mínimos locales o máximos locales, ademas de que los GA se ven expuestos a la naturaleza estocástica.

### B. *Conclusiones del autor.*

He incluido en este reporte la mayor cantidad de contenido teórico y técnico extraído de los archivos de documentación del propio proyecto, pero como conclusión personal quiero agregar que la realización de este proyecto fue de enorme crecimiento personal, disfrute recaudar información y codificar, refrescar temas de matemáticas que llevaba tiempo sin explorar y reforzarlos con estos tópicos avanzados de ciencias de la computación, de verdad hacen ver en retrospectiva mi avance en la carrera y la satisfacción personal de conocer bien el producto final que entrego, fue una experiencia refrescante y de realización persona, devolviéndome esas ganas de seguir estudiando más a profundidad cada rama de la inteligencia artificial hasta algún día dar algún aporte de magnitud.

## V. Agradecimiento

Quiero expresar mi sincero agradecimiento al profesor Raúl Enrique Sánchez Yáñez, por la valiosa orientación brindada durante el desarrollo de esta actividad. Su apoyo fue fundamental, ya que nos proporcionó un código de referencia que sirvió como guía para la elaboración de este programa, ese código de referencia sentó las bases de lo que este algoritmo genetico establece y me dio la idea de como llevarlo a cabo y como estructurarlo.

## VI. Referencias

Referencias Fundamentales

[1] Holland, J.H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press.

[2] Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional.

[3] Mitchell, M. (1996). An Introduction to Genetic Algorithms. MIT Press.

[4] Eiben, A.E. & Smith, J.E. (2015). Introduction to Evolutionary Computing. 2nd Edition. Springer-Verlag.

Referencias Técnicas

[5] Stroustrup, B. (2013). The C++ Programming Language. 4th Edition. Addison-Wesley Professional.

[6] ISO/IEC 14882:2017. Programming languages — C++. International Organization for Standardization.

[7] Haupt, R.L. & Haupt, S.E. (2004). *Practical Genetic Algorithms*. 2nd Edition. John Wiley & Sons.

[8] Gen, M. & Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. John Wiley & Sons.

Referencias Metodológicas

[9] Bäck, T., Fogel, D.B. & Michalewicz, Z. (Eds.) (1997). *Handbook of Evolutionary Computation*. Oxford University Press.

[10] Coello Coello, C.A., Lamont, G.B. & Van Veldhuizen, D.A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2nd Edition. Springer.

Referencias Comparativas

[11] Lones, M.A. (2014). "Metaheuristics in nature-inspired algorithms". *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 1419-1422.

[12] Wolpert, D.H. & Macready, W.G. (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, 1(1), 67-82.

Recursos Online

[13] CPP Reference. (2025). *C++ Standard Library Reference*. https://en.cppreference.com/

[14] Whitley, D. (1994). "A genetic algorithm tutorial". *Statistics and Computing*, 4(2), 65-85.