Cómputo Evolutivo

Tarea 3. Comparación de Métodos de Codificación.

Daniel Rojo Mata

danielrojomata@gmail.com

Fecha de Entrega: 21 de Septiembre de 2025

Cuadro 1: Parámetros de configuración del Algoritmo Genético para cada problema.

Problema	Parámetros Generales	Operadores (Cruza)	Operadores (Mutación)
N-Reinas	$N = 8$: 100 individuos, 1000 gen. $N = 12$: 200 individuos, 2000 gen.	Entera: Orden (OX) Binaria: Un punto Real: BLX-α	Entera: Intercambio (Swap) Binaria: Bit-flip Real: No uniforme
TSP (17 Ciudades)	100 individuos 500 generaciones P(muta) = 0.2	Entera: Orden (OX) Binaria: Un punto Real: Un punto	Entera: Intercambio (Swap) Binaria: Intercambio de gen Real: No uniforme
Función Esfera	50 individuos 100 generaciones P(cruza) = 0.9 Precisión: 4 decimales	Binaria: Un punto Real: BLX-α	Binaria: Bit-flip (P=1/L) Real: No uniforme
Función Eggholder	250 individuos 1000 generaciones P(cruza) = 0.9 Precisión: 4 decimales	Binaria: Un punto Real: BLX-α	Binaria: Bit-flip (P=1/L) Real: No uniforme

1. N-REINAS

1.0.1. Codificación Entera

La primera aproximación utilizó una codificación entera, donde cada individuo del cromosoma se representa como una permutación del conjunto $\{0, 1, \ldots, N-1\}$. Un individuo $\pi = (\pi_0, \pi_1, \ldots, \pi_{N-1})$ se interpreta como un tablero donde la reina de la columna i se encuentra en la fila π_i. Esta representación tiene la ventaja de que no pueden existir dos reinas en la misma fila o columna, restringiendo el espacio de búsqueda únicamente a soluciones parcialmente válidas. Por lo tanto, la función de aptitud solo necesita minimizar el número de conflictos en las diagonales.

Ejemplo para N=8: *Un individuo podría ser el vector:*

$$\pi = [7, 3, 0, 2, 5, 1, 6, 4]$$

Esto significa que la reina de la columna 0 está en la fila 7, la de la columna 1 en la fila 3, y así sucesivamente.

1.0.2. Codificación Binaria

En la segunda estrategia, cada individuo se representó como una cadena de bits de longitud $L = N \times m$, donde $m = \lceil \log_2(N) \rceil$ es el número de bits necesarios para representar la posición de una reina en una fila. Para evaluar un individuo, se requiere un paso de decodificación: la cadena de bits se segmenta en N bloques de m bits cada uno. Cada bloque se convierte a su valor entero, generando un vector de posiciones de la misma naturaleza que la codificación entera. Por ejemplo, para $N = 8$, un cromosoma se representa con $8 \times 3 = 24$ bits.

Ejemplo para N=8: *Se necesitan* $m = \lceil \log_2(8) \rceil = 3$ *bits por reina, para un total de 24 bits. El cromosoma que representa la misma solución anterior sería la concatenación de los valores binarios de cada posición:*

$$C = (\underbrace{111}_{7}\ \underbrace{011}_{3}\ \underbrace{000}_{0}\ \underbrace{010}_{2}\ \underbrace{101}_{5}\ \underbrace{001}_{1}\ \underbrace{110}_{6}\ \underbrace{100}_{4})$$

Al decodificar, el segmento `111` se convierte en 7, `011` en 3, etc., resultando en el vector $[7, 3, 0, 2, 5, 1, 6, 4]$.

1.0.3. Codificación Real

La tercera representación utilizó un vector de N números reales, donde cada gen $g_i \in [0, N)$ representa la posición de la reina en la columna i. Similar a la codificación binaria, se necesita un paso previo a la evaluación de la aptitud, llamado **discretización**. Este proceso mapea cada valor real a su correspondiente entero mediante la función piso, $p_i = \lfloor g_i \rfloor$. El vector de enteros resultante, $P = (p_0, p_1, \ldots, p_{N-1})$, se utiliza para calcular los conflictos.

Ejemplo para N=8: *Un individuo podría ser el siguiente vector de números reales:*

$$G = (7.15, 3.98, 0.42, 2.76, 5.33, 1.09, 6.81, 4.50)$$

Aplicando la función piso a cada elemento ($\lfloor 7.15 \rfloor = 7$, $\lfloor 3.98 \rfloor = 3$, etc.), se obtiene el mismo vector de posiciones enteras: $(7, 3, 0, 2, 5, 1, 6, 4)$.

1.0.4. Manejo de Restricciones y Función de Aptitud

Un problema en las codificaciones binaria y real es que los procesos de decodificación y discretización pueden generar soluciones inválidas, donde $p_i = p_j$ para $i \neq j$ (múltiples reinas en

la misma fila). Para manejar esta restricción, se modificó la función de aptitud para incluir un término de penalización. La aptitud $f(P)$ de un individuo se calculó como:

$$f(P) = \alpha \cdot C_{filas} + C_{diagonales}$$

donde $C_{diagonales}$ es el número de ataques diagonales, C_{filas} es el número de colisiones en filas, y α es un coeficiente de penalización alto (en nuestro caso, $\alpha = N$) que castiga severamente las soluciones inválidas, guiando al algoritmo hacia permutaciones válidas.

1.1. Resultados y Discusión para N-Reinas

1.2. Configuración Experimental

Para cada instancia del problema, se ejecutó el algoritmo genético 30 veces con semillas aleatorias distintas para garantizar la importancia estadística de los resultados. Las gráficas de convergencia mostradas en las Figuras 1 y 2 representan el promedio del mejor fitness (número de ataques) encontrado en cada generación a lo largo de estas 30 ejecuciones. La sombra alrededor de cada curva indica la desviación estándar.

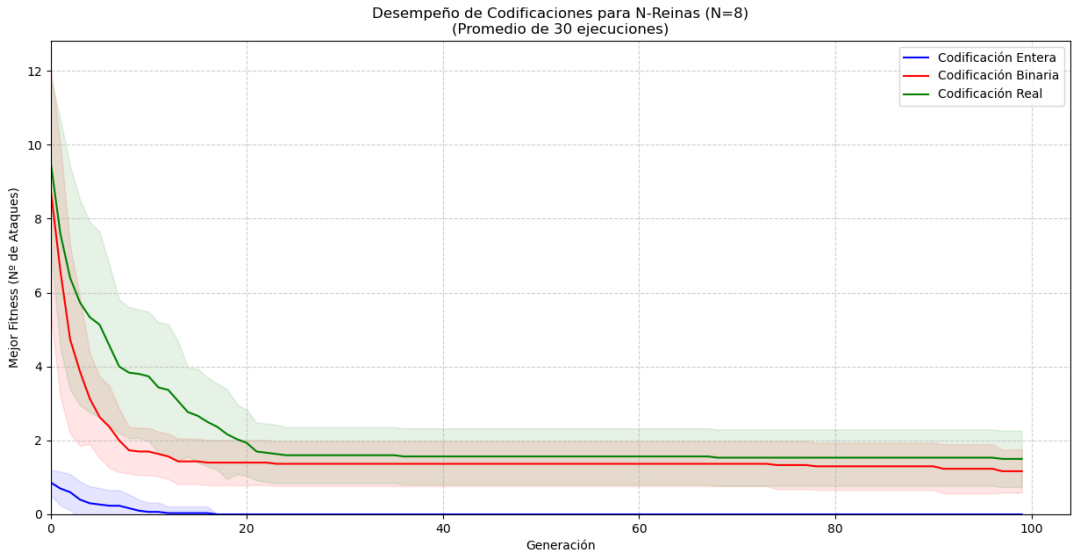

Figura 1: Convergencia de las codificaciones para N-Reinas con N=8.

1.3. Análisis de Desempeño por Codificación

1.3.1. Codificación Entera (Permutación)

La codificación entera exhibe un **desempeño notablemente superior** en ambas instancias del problema. La curva de convergencia (en azul) desciende de manera abrupta hasta alcanzar el fitness óptimo de 0 en muy pocas generaciones. Además, la ausencia casi total de desviación estándar indica que este método no solo es rápido, sino también **extremadamente fiable**,

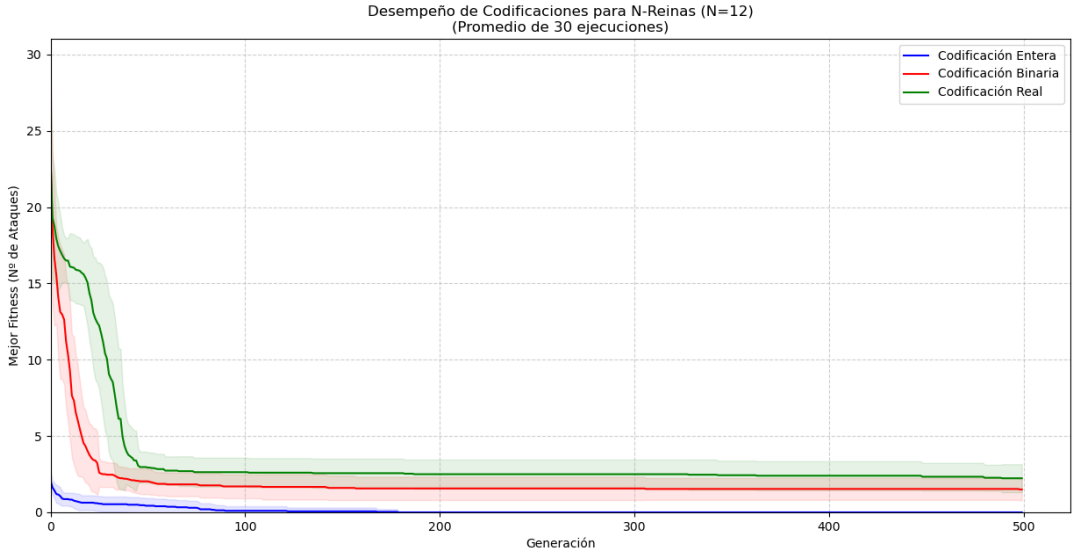

Figura 2: Convergencia de las codificaciones para N-Reinas con N=12.

encontrando una solución perfecta en prácticamente todas las ejecuciones. Este éxito se atribuye a que la representación por permutación restringe el espacio de búsqueda exclusivamente a soluciones donde no existen conflictos de filas o columnas, permitiendo al algoritmo concentrar su esfuerzo únicamente en resolver los ataques diagonales.

1.3.2. Codificación Binaria y Real

Ambas codificaciones, binaria (rojo) y real (verde), muestran un comportamiento similar entre sí, pero marcadamente inferior a la codificación entera. Ambas logran una reducción inicial del fitness, pero sufren de un **estancamiento prematuro**, convergiendo a un mínimo local del cual no logran escapar.

Este comportamiento se explica por la naturaleza de su espacio de búsqueda. A diferencia de la codificación por permutación, estas representaciones pueden generar soluciones inválidas (con reinas en la misma fila). Aunque se implementó una **función de penalización** en la aptitud para castigar dichas soluciones, el paisaje de búsqueda resultante es más complejo. El algoritmo aprende primero a evitar las soluciones penalizadas (generando permutaciones), pero luego le resulta difícil navegar el espacio de soluciones válidas para eliminar los conflictos diagonales restantes.

Entre estas dos, la codificación binaria muestra un desempeño ligeramente mejor y más consistente (menor desviación estándar) que la codificación real.

1.4. Impacto de la Escalabilidad (N=8 vs. N=12)

Al comparar la Figura 1 con la 2, se observa que la codificación entera escala de manera muy eficiente, encontrando la solución para $N = 12$ casi con la misma facilidad que para $N = 8$, las codificaciones binaria y real se estancan en valores de fitness progresivamente peores. Esto demuestra que la ventaja de una representación especializada se vuelve aún más crítica a medida

que la dimensionalidad del problema aumenta.

1.5. Conclusiones para N-Reinas

Los resultados experimentales sugieren de manera decisiva que para problemas combinatorios con restricciones de permutación, como el de las N-Reinas, una **codificación entera especializada es la mejor estrategia**. Las codificaciones binaria y real, aunque son adaptables mediante funciones de penalización, introducen una complejidad adicional en el espacio de búsqueda que dificulta la convergencia hacia el óptimo global, especialmente en problemas de mayor escala.

2. PARAMÉTRICOS

Para la optimización de funciones paramétricas continuas, como Esfera y Eggholder, se evaluaron dos enfoques de codificación: una representación directa con valores reales y una representación indirecta mediante cadenas de bits.

2.0.1. Codificación Real

Esta es la representación más natural para problemas de dominio continuo. Cada individuo es un vector $\vec{x} = (x_1, x_2, \ldots, x_n)$, donde cada gen x_i es un número de punto flotante que representa directamente un valor dentro del dominio de la variable correspondiente, $[a_i, b_i]$. La aptitud del individuo se calcula directamente aplicando la función objetivo $f(\vec{x})$. Para la recombinación se utilizó **Blend Crossover (BLX-α)** y para la variación se aplicó una **mutación no uniforme**.

***Ejemplo para la función Esfera 2D**, $f(x, y) = x^2 + y^2$, con dominio $[-10, 10]$ para ambas variables:*

$$\textit{Un individuo podría ser el vector: } \vec{x} = (5.25, -3.10)$$

El cálculo de su aptitud es directo: $f(5.25, -3.10) = (5.25)^2 + (-3.10)^2 = 27.5625 + 9.61 = 37.1725$.

2.0.2. Codificación Binaria

En este enfoque, cada variable continua x_i del vector $\vec{x}$ se representa mediante una subcadena de m_i bits. El cromosoma completo es la concatenación de estas subcadenas. Para evaluar un individuo, es requerido un proceso de **decodificación** que mapea el valor entero de cada subcadena binaria a un número real dentro del dominio $[a_i, b_i]$ de la variable. La fórmula de mapeo utilizada es:

$$x_i = a_i + \text{decimal}(\text{bits}_i) \times \frac{b_i - a_i}{2^{m_i} - 1}$$

donde $\text{decimal}(\text{bits}_i)$ es el valor entero de la subcadena de m_i bits. Los operadores genéticos utilizados fueron la **cruza de un punto** y la **mutación por bit-flip**.

***Ejemplo para la misma función Esfera 2D**, usando 10 bits para cada variable (total 20 bits):*

$$\textit{Un individuo podría ser: } C = (\underbrace{1100000000}_{x}\underbrace{0100000000}_{y})$$

- ***Decodificación de x:** El valor decimal de `1100000000` es* 768*. Aplicando la fórmula:*

$$x = -10 + 768 \times \frac{10 - (-10)}{2^{10} - 1} = -10 + 768 \times \frac{20}{1023} \approx 5.01$$

- ***Decodificación de y:** El valor decimal de `0100000000` es* 128*. Aplicando la fórmula:*

$$y = -10 + 128 \times \frac{20}{1023} = -10 + \frac{2560}{1023} \approx -7.5$$

El individuo decodificado es $(5.01, -7.5)$*, y su aptitud se calcula sobre estos valores.*

2.1. Resultados y Discusión

Finalmente, se evaluó el desempeño de las codificaciones real y binaria en dos funciones de prueba paramétricas con características muy diferentes: la función **Esfera** y **Eggholder**.

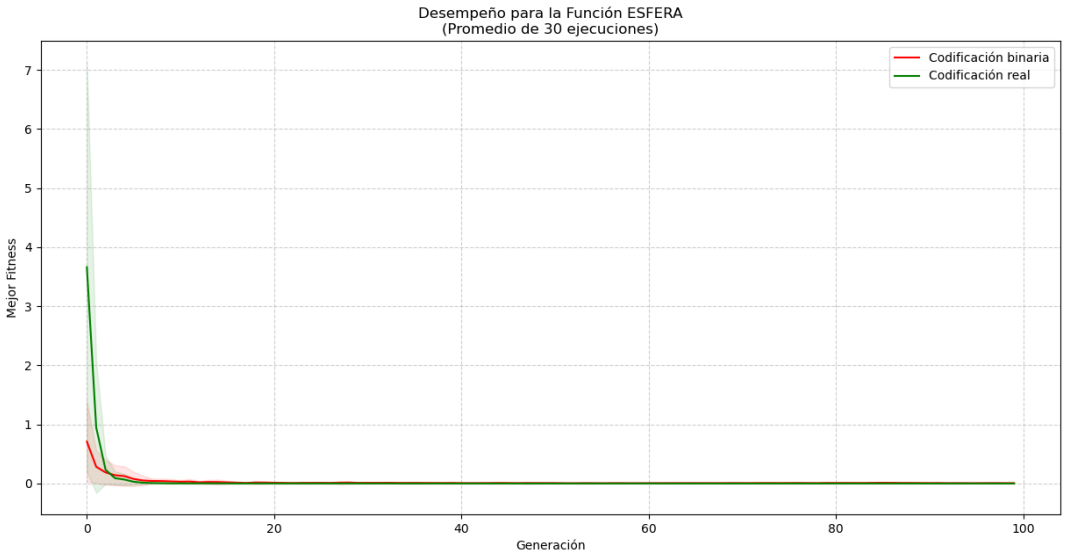

Figura 3: Convergencia de las codificaciones para la función Esfera.

2.2. Análisis para la Función Esfera

Como se observa en la Figura 3, ambas codificaciones demostraron ser **altamente efectivas** para minimizar la función Esfera, convergiendo al óptimo global de 0 en muy pocas generaciones.

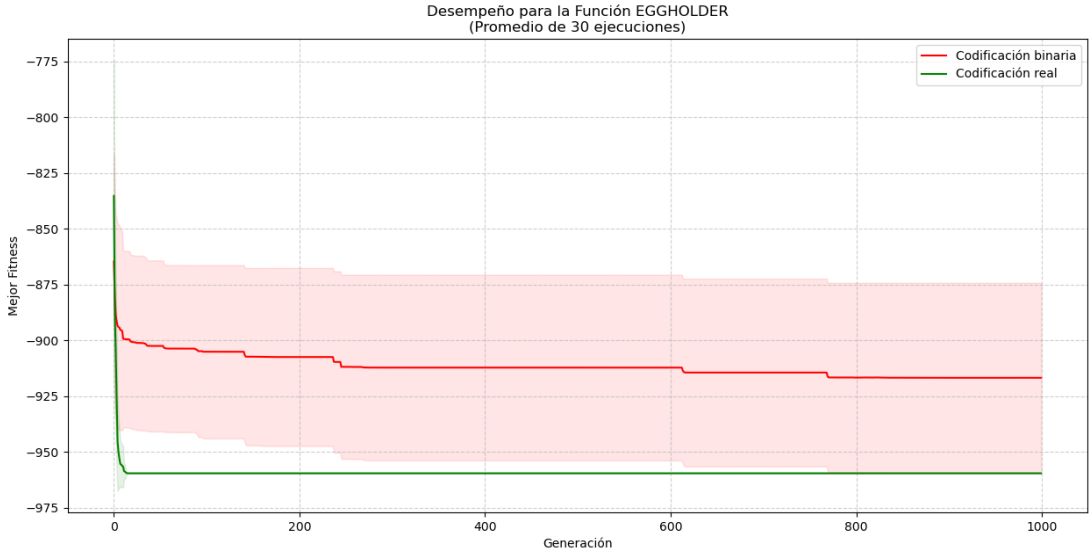

Figura 4: Convergencia de las codificaciones para la función Eggholder.

- **La codificación real (verde)** muestra una convergencia más rápida y precisa, alcanzando el valor de 0 de manera estable. Esto es esperable, ya que su representación directa del espacio de soluciones permite a los operadores, como la mutación no uniforme, realizar ajustes finos y precisos para aproximarse al mínimo.
- **La codificación binaria (rojo)** también converge de manera excelente, aunque se estabiliza en un valor infinitesimalmente superior a cero. Esto se debe a la **resolución finita** de la representación binaria; con un número finito de bits, solo se puede representar un número discreto de puntos en el espacio de búsqueda, y es posible que el punto exacto $(0, 0)$ no sea uno de ellos.

Para un problema simple y unimodal, ambas codificaciones son adecuadas, pero la real ofrece una mayor precisión.

2.3. Análisis para la Función Eggholder

La Figura 4 revela una **diferencia de desempeño notable** entre las dos codificaciones al enfrentar un paisaje de búsqueda complejo y multimodal.

- **La codificación real (verde)** demuestra una superioridad notable. Converge rápidamente a un valor de fitness cercano a -960, que es el óptimo global conocido para esta función (aprox. -959.64). La desviación estándar es mínima, lo que indica que encuentra consistentemente soluciones de alta calidad. Su éxito radica en la eficacia de sus operadores (Blend Crossover y mutación no uniforme) para explorar un paisaje continuo y complejo, combinando soluciones y realizando ajustes precisos que le permiten navegar entre múltiples óptimos locales.
- **La codificación binaria (rojo)**, en contraste, tiene un desempeño pobre. Se estanca prematuramente en un valor de fitness muy superior (peor), alrededor de -915. Más importante

aún, su **desviación estándar es grande**, lo que significa que sus resultados son erráticos e impredecibles; en algunas ejecuciones encuentra un mínimo local decente, y en otras queda atrapada en regiones muy subóptimas. La naturaleza discreta de la codificación binaria y sus operadores (cruce de un punto) son menos eficientes para explorar paisajes continuos y rugosos", donde pequeños cambios en los valores reales pueden requerir que muchos bits cambien simultáneamente.

2.4. Conclusiones para Problemas Paramétricos

Para la optimización de funciones paramétricas, la **codificación real es la estrategia dominante y preferida**. Mientras que en problemas simples como la función Esfera la codificación binaria es una alternativa viable, su eficacia se degrada severamente en paisajes de búsqueda complejos y multimodales. La representación directa y los operadores especializados de la codificación real le otorgan una ventaja fundamental en precisión, poder exploratorio y fiabilidad, haciéndola la herramienta superior para problemas de este dominio.

3. Problema del Agente Viajero (TSP)

Para el TSP, un problema de optimización combinatoria basado en permutaciones, se implementaron y compararon tres representaciones cromosómicas. El objetivo fue determinar cómo cada enfoque maneja la restricción fundamental de que cada ciudad debe ser visitada exactamente una vez.

3.0.1. Codificación Entera (Permutación)

Esta es la representación más directa y natural para el TSP. Un individuo se codifica como un vector que contiene una permutación de los índices de las ciudades $\{0, 1, \ldots, N-1\}$. El orden de los elementos en el vector define directamente el recorrido del tour. Para la recombinación de soluciones, se utilizó la **cruza de orden (Order Crossover, OX)**, un operador diseñado específicamente para preservar sub-rutas o "bloques constructivos"de los padres.

Ejemplo para 4 ciudades (0, 1, 2, 3): *Un individuo podría ser el vector:*

$$P = (2, 0, 3, 1)$$

Esto representa el tour Hamiltoniano $2 \to 0 \to 3 \to 1 \to 2$.

3.0.2. Codificación Real

En este enfoque indirecto, un individuo es un vector de N números reales, generalmente en el rango $[0, 1)$. La permutación que define el tour no se almacena directamente, sino que se obtiene a través de un proceso de decodificación. Se utiliza la función `argsort`, que devuelve los índices de los

elementos del vector ordenados de menor a mayor. Esta lista de índices constituye la permutación del recorrido.

Ejemplo para 4 ciudades: *Un individuo podría ser el vector de reales:*

$$R = (0.78, 0.21, 0.95, 0.44)$$

Al ordenar los valores de menor a mayor se obtiene $(0.21, 0.44, 0.78, 0.95)$*. Los índices originales de estos valores son* $(1, 3, 0, 2)$*. Por lo tanto, el tour decodificado es* $1 \to 3 \to 0 \to 2 \to 1$*.*

3.0.3. Codificación Binaria

Esta es la representación más abstracta. El tour se codifica como una cadena de bits de longitud $L = N \times m$, donde $m = \lceil \log_2(N) \rceil$ es el número de bits por ciudad. El proceso de decodificación es más complejo, ya que la conversión directa de los segmentos de bits a enteros puede generar ciudades duplicadas. Para resolver esto, se implementó un mecanismo de **reparación**:

1. La cadena de bits se segmenta y se convierte en un vector de enteros, que puede contener duplicados.
2. Se eliminan los duplicados, conservando solo la primera aparición de cada ciudad.
3. Se identifican las ciudades que no aparecen en el vector resultante.
4. Las ciudades faltantes se añaden al final del vector para completar la permutación.

Ejemplo para 4 ciudades: *Se necesitan* $m = \lceil \log_2(4) \rceil = 2$ *bits por ciudad, para un total de 8 bits. Consideremos el cromosoma:*

$$C = (\underbrace{10}_{2}\ \underbrace{01}_{1}\ \underbrace{10}_{2}\ \underbrace{00}_{0})$$

- ***Decodificación inicial (con duplicados):*** $(2, 1, 2, 0)$
- ***Reparación:*** *Se eliminan los duplicados, resultando en* $(2, 1, 0)$*. La ciudad faltante es la* $\{3\}$*. Se añade al final.*
- ***Tour final válido:*** $(2, 1, 0, 3)$*, que representa el recorrido* $2 \to 1 \to 0 \to 3 \to 2$*.*

Este proceso de reparación, aunque garantiza una solución válida, puede ser disruptivo para la herencia de buenas características.

3.1. Resultados y Discusión para TSP

Se evaluó el desempeño de las tres codificaciones (entera, real y binaria) en la resolución del Problema del Agente Viajero (TSP) para una instancia de 17 ciudades. Los resultados, promediados a lo largo de 30 ejecuciones independientes por cada codificación, se presentan en la gráfica de convergencia de la Figura 5.

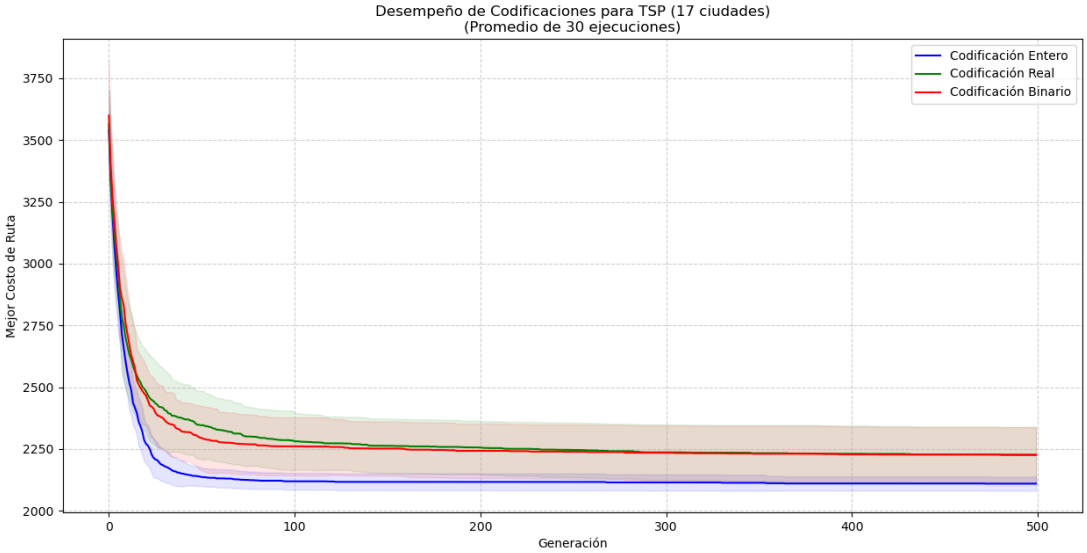

Figura 5: Convergencia de las codificaciones para TSP (17 ciudades).

3.2. Análisis de las Curvas de Convergencia

La gráfica muestra que, si bien todas las codificaciones logran una mejora significativa en las primeras generaciones, existen diferencias claras en su velocidad de convergencia y en la calidad de la solución final a la que llegan.

3.2.1. Codificación Entera (Permutación)

La codificación entera (curva azul) demuestra ser, con una gran diferencia, la **estrategia más efectiva**. Converge más rápidamente y alcanza un costo de ruta final significativamente más bajo que las otras dos. La desviación estándar (sombra azul) es la más estrecha, lo que indica una alta **fiabilidad y consistencia** en sus resultados. Este éxito se debe a que la representación por permutación es un análogo directo del problema. Los operadores genéticos seleccionados, como la **cruza de orden (OX)** y la **mutación por intercambio (swap)**, están diseñados específicamente para manipular permutaciones, permitiendo preservar y combinar sub-rutas eficientes (bloques constructivos) de manera efectiva.

3.2.2. Codificación Real

La codificación real (curva verde), que utiliza el método `argsort` para decodificar un vector de números reales en una permutación, muestra el **desempeño más pobre**. Aunque la mutación no uniforme que implementamos permite una exploración fina, la representación es indirecta y menos intuitiva. Un pequeño cambio en un valor real puede provocar una reorganización drástica en la permutación resultante, dificultando la preservación de sub-rutas valiosas. Esto se traduce en una convergencia más lenta y un estancamiento en soluciones de mayor costo. La amplia desviación estándar sugiere que es el método menos consistente.

3.2.3. Codificación Binaria

El desempeño de la codificación binaria (curva roja) se sitúa en un punto intermedio. Es superior a la real pero claramente inferior a la entera. El principal desafío de esta representación es la decodificación. El mecanismo de **reparación** que implementamos, el cual elimina ciudades duplicadas y añade las faltantes, asegura la validez de cada individuo, pero a un costo. Este proceso de reparación puede ser **disruptivo**, rompiendo patrones genéticos que el algoritmo podría estar formando. Aunque logra encontrar rutas decentes, le cuesta refinar la solución de manera tan eficiente como la codificación entera.

3.3. Conclusiones para TSP

Al igual que en el problema de N-Reinas, los resultados para el TSP refuerzan la idea de que la elección de la representación es más que importante. Para problemas cuya estructura fundamental es una **permutación**, una codificación entera directa junto con operadores especializados es la mejor estrategia. Las codificaciones de propósito general (real y binaria) requieren mecanismos de decodificación o reparación que, si bien hacen que el problema sea tratable, imponen una sobrecarga en la búsqueda y limitan la capacidad del algoritmo para converger a soluciones de alta calidad de manera eficiente y consistente.