UNIVERSIDAD
Ð CÓRDOBA

Metaheurísticas: Práctica 2

Autores: **Javier Gómez Aparicio, Carlos David López Hinojosa, Alejandro Luque Núñez**

(Profesor: José María Luna Ariza)

Universidad de Córdoba

Índice general

Capítulo 1

Objetivos

En la presente práctica se nos da una función polinomial $f(x)$, la cual se nos pide encontrar los coeficientes de la misma con un dataset dado, dicha función es la siguiente:

$$f(x) = e^a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 + hx^7 \tag{1.1}$$

El dataset que se nos proporciona para realizar la práctica es el siguiente:

$f(x)$	x
3.490342957	0
3.649406057	0.1
3.850310157	0.2
4.110680257	0.3
4.444613357	0.4
4.864490457	0.5
8.945762957	1
14.907555257	1.3
3.3508758574	-0.1
-10.443986642	-1.6
-10.134869742	-1.7
-0.0700854481	-0.83
0.0372406176	-0.82
-0.2501897336	-1.98
0.4626335969	-1.99

Cuadro 1.1: Valores del dataset proporcionado.

Habiendo analizado las características de nuestro problema se nos pide construir un algoritmo genético que nos permita encontrar los coeficientes de la función polinomial, que se ajusten mejor a los datos y minimicen el error.

Capítulo 2

Decisiones tomadas

2.1. Aclaraciones

- El algoritmo genético se ha implementado en Python.
- El modelo de algoritmo genético utilizado ha sido el de *"Generational Genetic Algorithm"*.
- La función de fitness utilizada es la media de los errores al cuadrado entre el valor real y el valor calculado por la función polinomial.
- Se ha utilizado la librería NumPy para realizar operaciones matemáticas y optimizaciones.

2.2. Representación de la solución

La representación de la solución se ha realizado mediante una lista de coeficientes, donde cada coeficiente representa un parámetro de la función polinomial. La longitud de la lista es 8, ya que la función polinomial tiene 8 coeficientes (incluyendo el término independiente). Cada coeficiente se representa como un número real, y la lista completa representa una posible solución al problema.

0.393	0.310	0.145	0.287	0.298	0.235	0.916	0.402

Figura 2.1: Representación de la solución.

2.3. Función Objetivo

La función objetivo se ha definido como la media de los errores al cuadrado entre el valor real y el valor calculado por la función polinomial. La función objetivo se calcula como:

$$MSE(x) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - \hat{f}(x_i))^2 \tag{2.1}$$

donde n es el número de puntos en el dataset, $f(x_i)$ es el valor real de la función polinomial en el punto x_i, y $\hat{f}(x_i)$ es el valor calculado por la función polinomial con los coeficientes actuales. La función objetivo se minimiza durante el proceso de optimización del algoritmo genético, buscando los coeficientes que mejor se ajusten a los datos del dataset.

En el caso de la solución anterior, el error cuadrático medio es de 36.362 (truncado a la milesima).

2.4. Inicialización de la población

La población inicial se ha generado aleatoriamente, creando una lista de coeficientes aleatorios para cada individuo en la población. Cada coeficiente se ha generado en un rango $[0, 1]$, hemos escogido este método por su simplicidad y velocidad, además de la facilidad de implementación.

2.5. Modificación del modelo generacional

Como se ha mencionado anteriormente, el modelo de algoritmo genético utilizado ha sido el de *"Generational Genetic Algorithm"*, una de las ventajas de estos modelos geneticos es la facilidad que tienen para paralelizarse, aprovechando esto hemos decidio aplicarlo como un modelo de islas, donde cada isla es un proceso que se ejecuta en paralelo, y cada uno de estos procesos tiene su propia población. Esto nos permite aprovechar al máximo los recursos de la máquina, y acelerar el proceso de optimización.

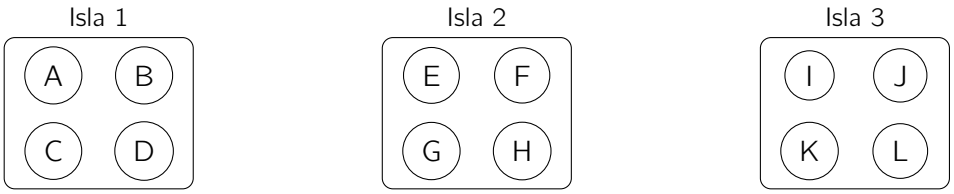

Figura 2.2: Modelo de islas con poblaciones independientes

Migración entre islas (no implementado)

También pensamos en realizar migraciones entre islas, pero al final decidimos no hacerlo, ya que el coste de la migración era mayor que el beneficio que se obtenía, deberíamos crear una entre los procesos lo que hacia que el algoritmo fuera mucho más lento, y al final no se obtenía una mejora significativa en los resultados.

Entrenamiento de los individuos (no implementado)

Pensamos en realizar un entrenamiento de individuos seleccionados aleatoriamente al final de cada generación, como ocurre en la vida real, las personas se dedican a una habilidad o un trabajo en concreto, hay carpinteros, fontaneros, electricistas, etc. Se escogia un gen aleatorio de un individuo aleatorio y se le aplicaba una busqueda local, es decir se incrementaba o decrementaba el gen aleatorio, y se calculaba el error cuadrático medio, si este era menor que el anterior se quedaba con el nuevo gen, si no se quedaba con el antiguo. dicha búsqueda sería una búsqueda Tabú paralelizada, para mantener la mejora de distintos genes cada vez que se entrenase un individuo. Esto se hacia para intentar mejorar el resultado final, pero al final decidimos no implementarlo, ya que el coste de la busqueda local era mayor que el beneficio que se obtenía, y al final no se obtenía una mejora significativa en los resultados.

2.6. Diagrama del Algoritmo

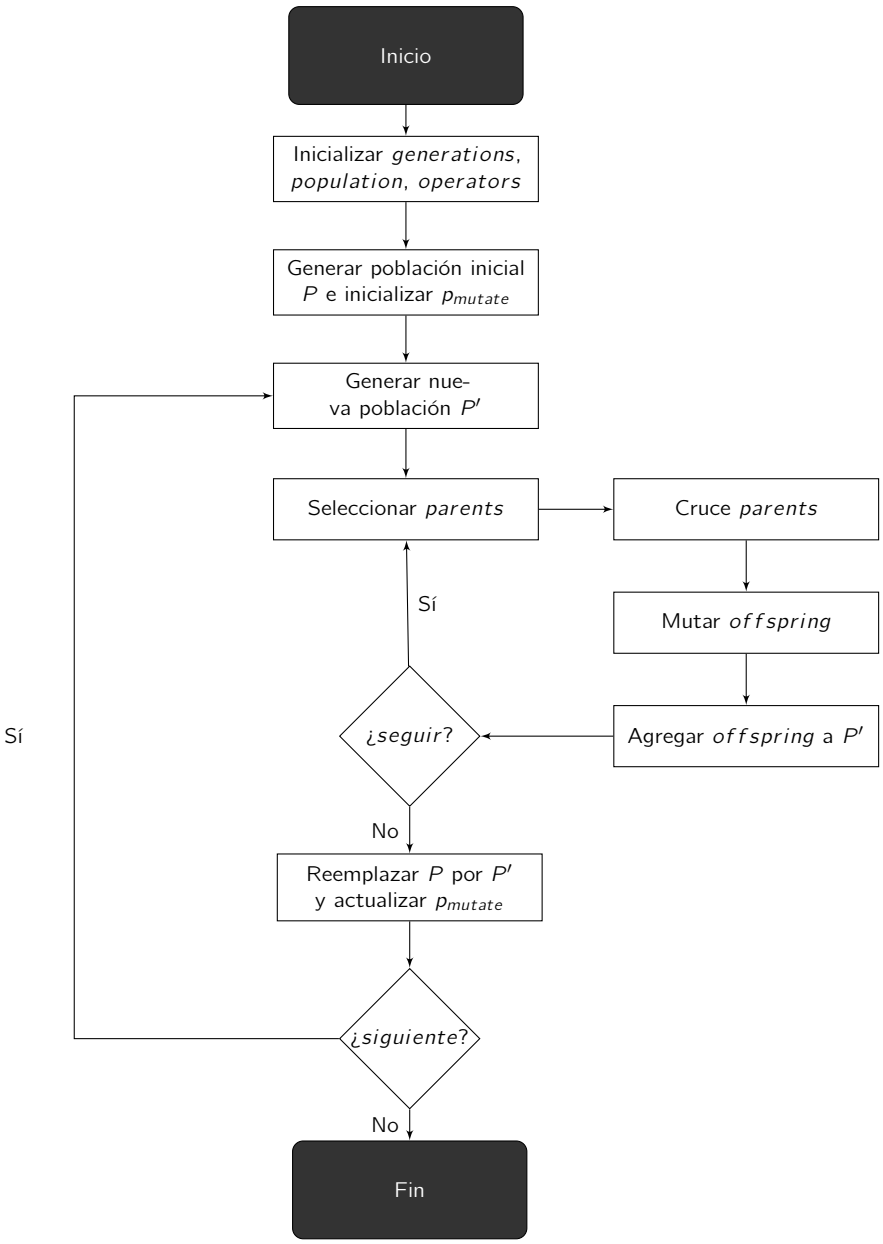

Figura 2.3: Diagrama de flujo del algoritmo genético.

Nota: Este digrama se refiere al algoritmo en una isla independiente.

2.7. Operadores genéticos

En esta sección solamente se van a mencionar los operadores genéticos utilizados, más adelante se hablara en profundidad de cada uno de ellos.

Selección

- **Selección por torneo .**
- **Selección aleatoria.**
- **Selección por ruleta.**
- **Selección por emparejamiento variado inverso.**

Cruce

- **Cruce Aritmético.**
- **Cruce de Un Punto.**
- **Cruce Uniforme.**
- **Cruce BLX.**

Hay que aclarar que en el caso del cruce hemos decidido siempre realizarlo, no vimos sentido a realizar el cruce con cierta probabilidad, ya que estas limitando el algoritmo y manteniendo la diversidad muy baja.

Mutación

- **Mutación Gaussiana.**
- **Mutación Uniforme.**
- **Mutación No Uniforme**
- **Mutación Polinómica.**

En nuestro modelo, la probabilidad de mutación se ajusta dinámicamente en función de la **diversidad actual de la población**. Este enfoque tiene como objetivo mantener una exploración saludable del espacio de búsqueda y evitar la convergencia prematura a óptimos locales.

La lógica detrás de este mecanismo es la siguiente:

- Cuando la población presenta *baja diversidad* (los individuos son muy similares entre sí), la **probabilidad de mutación aumenta**.
- Cuando la población es *muy diversa* (los individuos son muy distintos entre sí), la **probabilidad de mutación disminuye**.

Este ajuste adaptativo favorece la exploración cuando es necesaria, y conserva soluciones prometedoras cuando la población ya cubre suficientemente el espacio de búsqueda.

Para cuantificar la diversidad, utilizamos la **norma L2** (o distancia euclídea) entre los individuos de la población. A partir de esto, definimos un coeficiente de diversidad como:

$$D = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{m} (x_{ij} - x_{kj})^2} \tag{2.2}$$

donde:

- n es el número de individuos en la población,
- m es el número de genes por individuo,
- x_{ij} es el valor del gen j del individuo i,
- x_{kj} es el valor del gen j del individuo k (tomado como referencia).

Además de la media de las distancias euclídeas, incorporamos también la **desviación estándar** de dichas distancias. Esta medida adicional nos permite reflejar la dispersión interna de la población. Empíricamente comprobamos que su uso mejora la sensibilidad del sistema ante posibles estancamientos.

Dado que un mayor valor de D indica mayor diversidad, y buscamos una relación inversa con la tasa de mutación, definimos la probabilidad de mutación (reescalada al rango $[0, 1]$) como:

$$p_{\text{mutate}} = 1 - (D + \sigma) \tag{2.3}$$

Reemplazo

- **Reemplazo Generacional Completo.**
- **Reemplazar al peor de la población.**
- **Torneo restringido.**
- **Peor entre semejantes.**
- **Elitismo.**

2.8. Optimización de parámetros

De nuevo para la optimización de parametros hemos utilizado al igual que la prática 1, el algoritmo de optimización Bayesiano, el cual nos permite optimizar los parámetros del algoritmo genético de forma eficiente y rápida, aunque por como codificamos el algoritmo podriamos haber hecho una meta optimización "recursiva", esto era muy aparatoso y decidimos la opción mencionada.

Parámetros del algoritmo genético

En base a las pruebas realizadas y la optimización de parámetros mediante el algoritmo de optimización Bayesiano, se ha determinado que la mejor configuración para el problema de optimización es la siguiente:

- **Tamaño de la población:** 23 individuos.
- **Número de generaciones:** 143.
- **Operador de selección:** Torneo con $k = 7$.
- **Operador de cruce:** BLX-α.
- **Operador de mutación:** Mutación Gaussiana con $\sigma = 0{,}05$.
- **Operador de reemplazo:** Elitismo.

El porqué de esta selección de parámetros y como afectan al desempeño del algoritmo a la vez que la explicación de porque casan será abordado en el capítulo de conclusiones, tras haber visto todos los operadores.

Capítulo 3

Operadores de Selección

3.1. Selección por torneo.

Este método es útil porque permite un balance entre explotación y exploración. Al seleccionar k individuos al azar y elegir al mejor, se fomenta la supervivencia de los más aptos, pero sin descartar completamente a los menos aptos, lo que ayuda a mantener la diversidad en la población.

Bondades:

- Balance entre exploración y explotación.
- Fácil de implementar y rápido computacionalmente.
- Favorece a los más aptos sin eliminar completamente la diversidad.

Desventajas:

- Si el torneo es muy grande, puede llevar a convergencia prematura.
- Si es muy pequeño, se asemeja a una selección aleatoria.

Parámetros:

- Tamaño del torneo k (en el caso binario, $k = 2$).

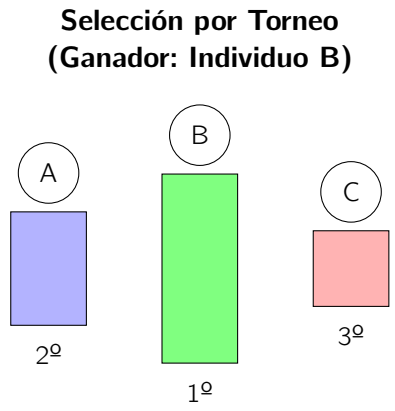

Figura 3.1: Representación visual del torneo con tres participantes ($k = 3$).

3.2. Selección aleatoria.

Este selector es ideal para mantener una alta diversidad en la población, ya que todos los individuos tienen la misma probabilidad de ser seleccionados, independientemente de su calidad. Esto puede ser útil en las primeras etapas del algoritmo, cuando la exploración del espacio de búsqueda es más importante que la explotación.

Bondades:

- Máxima diversidad genética en la selección.
- Útil en las etapas iniciales del algoritmo para evitar estancamientos.

Desventajas:

- No favorece a los individuos más aptos.
- Baja eficiencia en fases avanzadas del algoritmo.

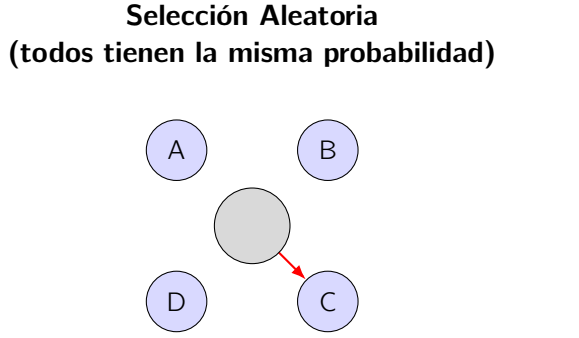

Figura 3.2: Representación visual de selección aleatoria.

3.3. Selección por ruleta.

Este método asigna probabilidades de selección proporcionales a la calidad de los individuos. Es útil para problemas donde queremos que los mejores individuos tengan más oportunidades de reproducirse, pero sin eliminar completamente la posibilidad de que los menos aptos contribuyan, lo que puede evitar la convergencia prematura.

Bondades:

- La probabilidad de selección es proporcional a la calidad del individuo.
- Favorece a los mejores individuos sin excluir completamente a los menos aptos.

Desventajas:

- Sensible a diferencias grandes en los valores de fitness.
- Puede favorecer excesivamente a pocos individuos dominantes.

Parámetros:

- Número de individuos a seleccionar.

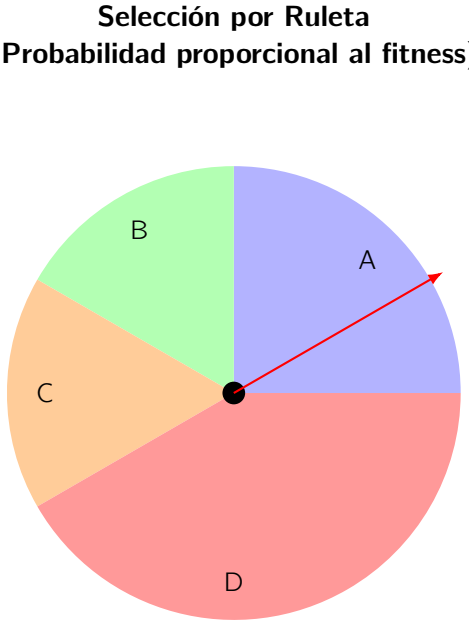

Figura 3.3: Representación visual de la selección por ruleta.

3.4. Selección por emparejamiento variado inverso.

Este enfoque fomenta la diversidad al emparejar individuos con características opuestas o complementarias. Es especialmente útil en problemas donde la diversidad genética es clave para evitar estancamientos en óptimos locales y para explorar regiones del espacio de búsqueda que de otro modo podrían ser ignoradas.

Bondades:

- Favorece la diversidad genética.
- Ayuda a escapar de óptimos locales.
- Explora regiones del espacio de búsqueda poco explotadas.

Desventajas:

- Requiere cálculo de distancias entre individuos, lo cual puede ser costoso.
- Puede emparejar individuos incompatibles y generar descendencia de baja calidad.

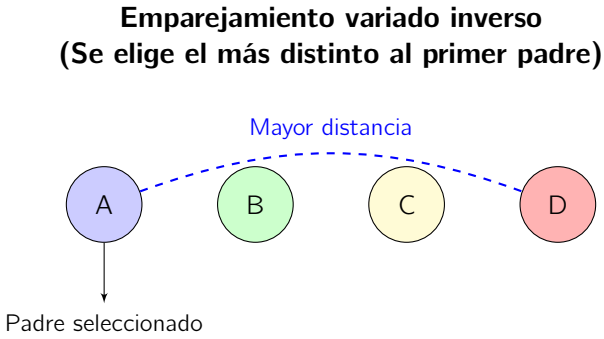

Figura 3.4: Selección por emparejamiento variado inverso.

3.5. Mediciones de los operadores de selección

Tiempo consumido

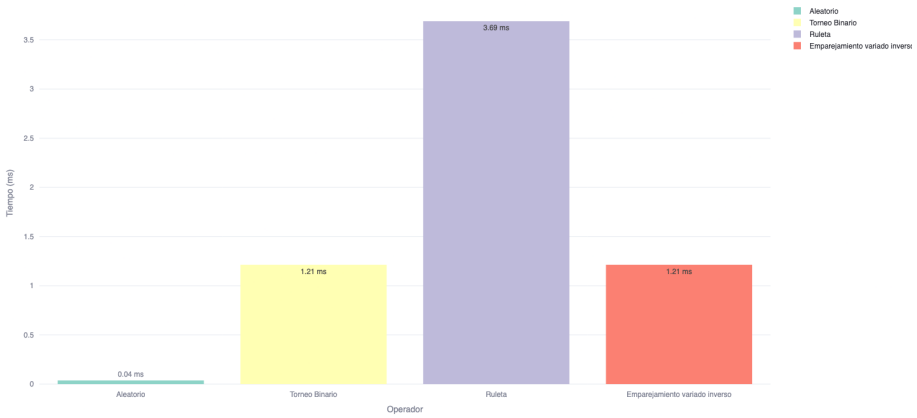

Figura 3.5: Tiempo consumido por cada operador de selección.

En este gráfico se pueden observar los tiempos consumidos medios de cada operador de selección. Se observa que el operador de selección por ruleta es el que más tiempo consume de todos los operadores de selección, le sigue así los operadores de selección por torneo y por emparejamiento variado inverso, que tienen un tiempo de ejecución similar. Por último, el operador de selección aleatoria es el que menos tiempo consume.

Estadísticos de las métricas de tiempo

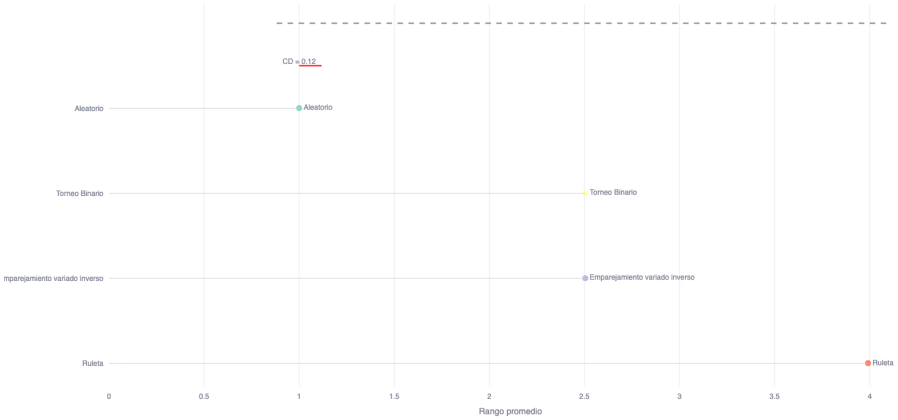

Figura 3.6: Gráfico de Nemenyi para los tiempos de cada operador de selección.

En este gráfico de nemenyi se pueden observar las tendencias anteriores.

Memoria consumida

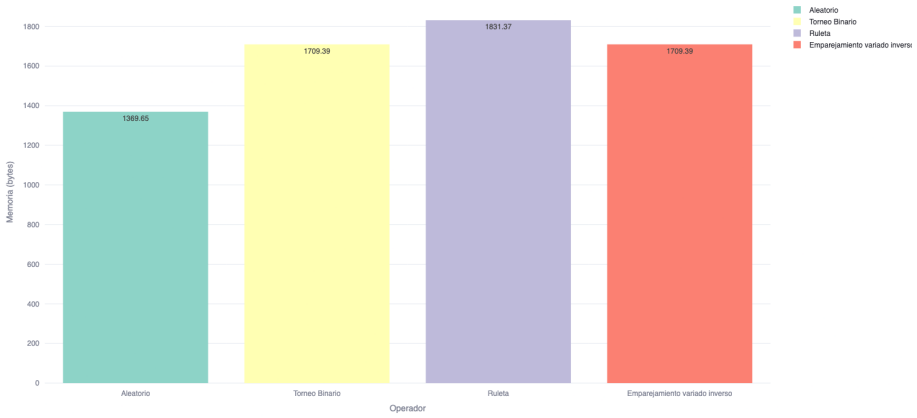

Figura 3.7: Memoria consumida por cada operador de selección.

Aquí podemos observar una tendencia similar a la de los tiempos consumidos, donde el operador de selección por ruleta es el que más memoria consume, seguido por el operador de selección por torneo y por emparejamiento variado inverso, que tienen un consumo de memoria similar. Por último, el operador de selección aleatoria es el que menos memoria consume.

Estadísticos de las métricas de memoria

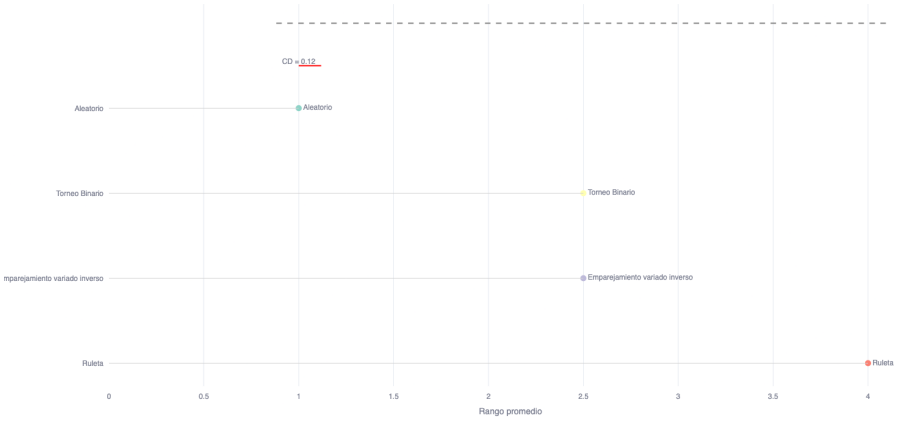

Figura 3.8: Gráfico de Nemenyi para la memoria de cada operador de selección.

En este gráfico de nemenyi se pueden observar las tendencias anteriores.

Convergencia

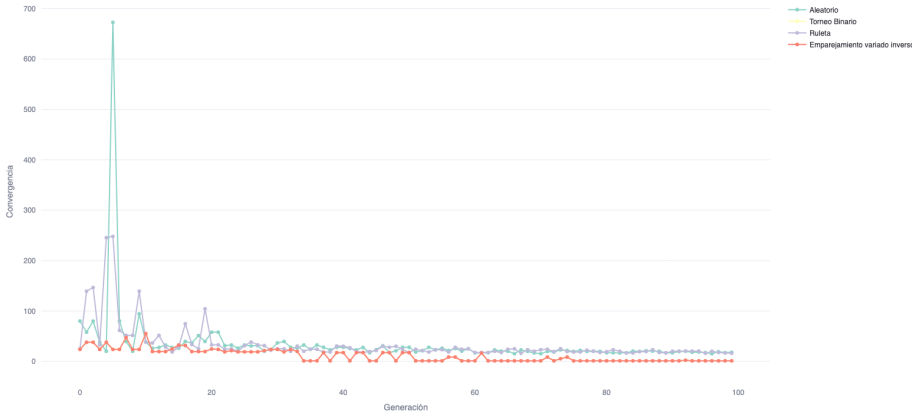

Figura 3.9: Convergencia de cada operador de selección.

En este gráfico se puede observar las convergencias de los distintos operadores de selección. No vamos a sacar conclusiones directas de este gráfico, en este caso nos dará más información el gráfico de Nemenyi que se presenta a continuación. Solo remarcar que el operador de selección aleatorio parece tener una convergencia más lenta al inicio y el operador por torneo parece tener una rápida convergencia.

Estadísticos de las métricas de convergencia

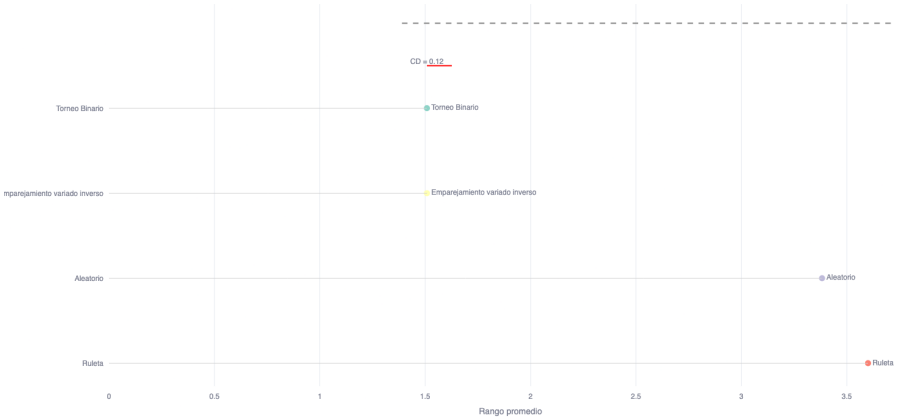

Figura 3.10: Gráfico de Nemenyi para la convergencia de cada operador de selección.

Aquí ya sí podemos observar que el operador de selección por torneo es el que más rápido converge, sorprendentemente le sigue reñido emparejamiento variado inverso (estadísticamente iguales), siendo los peores en convergencia el operador de selección aleatorio y el operador de selección por ruleta, que tienen una convergencia muy similar.

Conclusiones sobre los operadores de selección

En base a las métricas evaluadas, se pueden extraer las siguientes conclusiones sobre los operadores de selección:

1. Tiempo consumido:

- El operador de selección por ruleta es el más costoso en términos de tiempo, debido al cálculo de probabilidades proporcionales al fitness.
- Los operadores de selección por torneo y emparejamiento variado inverso tienen tiempos de ejecución similares, siendo intermedios.
- El operador de selección aleatoria es el más eficiente en tiempo, ya que no requiere cálculos complejos.

2. Memoria consumida:

- El operador de selección por ruleta también es el que más memoria consume.
- Los operadores de torneo y emparejamiento variado inverso tienen un consumo de memoria similar.
- El operador de selección aleatoria es el más eficiente en términos de memoria, al igual que en tiempo.

3. Convergencia:

- El operador de selección por torneo es el que converge más rápido.
- El operador de emparejamiento variado inverso le sigue de cerca, siendo estadísticamente similar.
- Los operadores de selección aleatoria y por ruleta tienen una convergencia más lenta, siendo estadísticamente iguales en este aspecto.

Conclusión general:

- Si se busca rapidez y eficiencia en recursos, el operador de selección aleatoria es el más adecuado, aunque sacrifica convergencia.
- Para una convergencia rápida y balanceada, el operador de selección por torneo es la mejor opción.
- El operador de emparejamiento variado inverso es útil para mantener diversidad y evitar estancamientos, aunque con un costo computacional intermedio.
- El operador de selección por ruleta, aunque favorece a los mejores individuos, es el menos eficiente en tiempo y memoria, y tiene una convergencia más lenta.

Capítulo 4

Operadores de Cruce

4.1. Cruce Aritmético

Este operador combina a los padres mediante una media ponderada de sus genes. Es especialmente útil en problemas de optimización continua donde queremos explorar el espacio entre soluciones parentales, manteniendo la diversidad mediante combinaciones lineales controladas.

Bondades:

- Mantiene diversidad mediante combinaciones suaves.
- Garantiza descendencia dentro del espacio de búsqueda (para problemas convexos).
- Funciona bien en representaciones reales.

Desventajas:

- Puede generar convergencia prematura en espacios no convexos.
- No es adecuado para representaciones discretas.

Parámetros:

- Factor de mezcla α ($\alpha \in [0, 1]$).

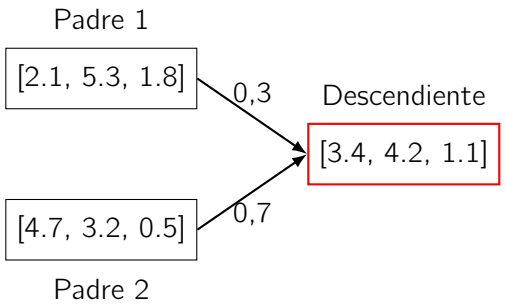

Figura 4.1: Ejemplo de cruce aritmético con $\alpha = 0{,}3$.

4.2. Cruce de Un Punto

Este operador clásico divide a los padres en un punto aleatorio y combina sus segmentos. Es eficaz para problemas donde los genes adyacentes tienen correlaciones significativas, permitiendo preservar bloques de información útil.

Bondades:

- Simple y computacionalmente eficiente.
- Preserva esquemas genéticos adyacentes.
- Funciona bien en representaciones binarias y discretas.

Desventajas:

- Limitada exploración del espacio de búsqueda.
- Sensible a la ordenación de los genes.

Aunque pueda parecer que este operador no vaya bien con la codificación de nuestro problema a la hora de experimentar decidimos incluirlo ya que daba buenos resultados, había ocasiones en la que los coeficientes de un gen de una solución eran mejores en otra posición.

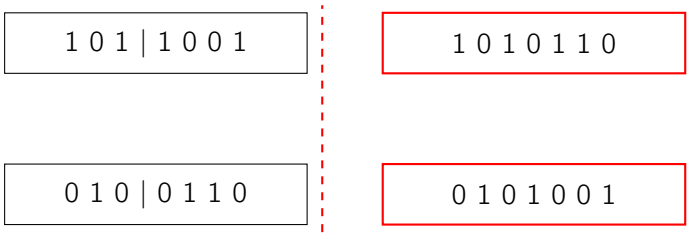

Figura 4.2: Ejemplo de cruce de un punto en representación binaria.

4.3. Cruce Uniforme

Este operador genera descendencia cruzando genes aleatoriamente de cada padre. Es ideal para problemas donde los genes son independientes entre sí, permitiendo máxima exploración del espacio de combinaciones posibles.

Bondades:

- Máxima diversidad en la descendencia.
- Neutral a la ordenación de genes.
- Efectivo en problemas con epistasis baja.

Desventajas:

- Puede romper buenos esquemas genéticos.
- Requiere mayor tiempo de convergencia.

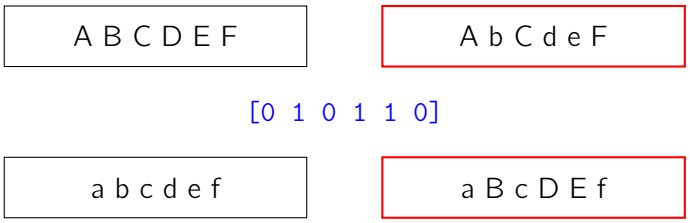

Figura 4.3: Cruce uniforme usando máscara binaria. Los 1 cruzan al padre 2 y los 0 al padre 1.

4.4. Cruce BLX-α

Este operador para representaciones reales genera descendencia dentro de un intervalo extendido alrededor de los valores parentales. Combina explotación local con exploración controlada, siendo efectivo en espacios de búsqueda continuos.

Bondades:

- Balance ajustable entre exploración/explotación.
- Genera soluciones innovadoras dentro de un rango seguro.
- Robustez ante distribuciones parentales diversas.

Desventajas:

- Sensible a la elección del parámetro α.

Parámetros:

- Factor de expansión α (en nuestro caso $\alpha \in [0, 1]$).

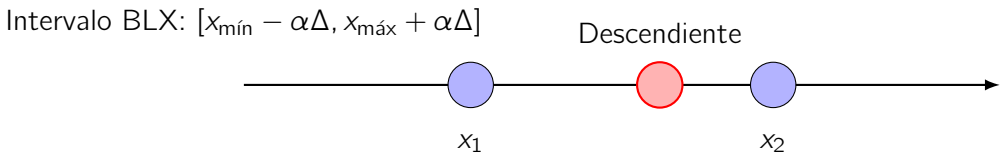

Figura 4.4: Representación del cruce BLX-α en una dimensión.

4.5. Mediciones de los operadores de cruces

Tiempo consumido

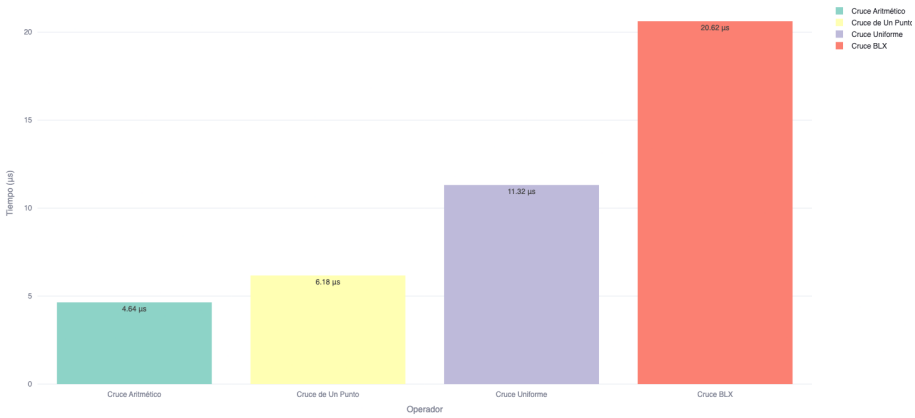

Figura 4.5: Tiempo consumido por cada operador de cruce.

En la figura 4.5 se puede observar que una distribución escalonada ascendente de las medias de los tiempos de cada operador de cruce, siendo el primero y que menos tiempo consume el cruce aritmético, seguido por el cruce de un punto, el cruce uniforme y finalmente el cruce BLX-α que es el que más tiempo consume.

Estadísticos de las métricas de memoria

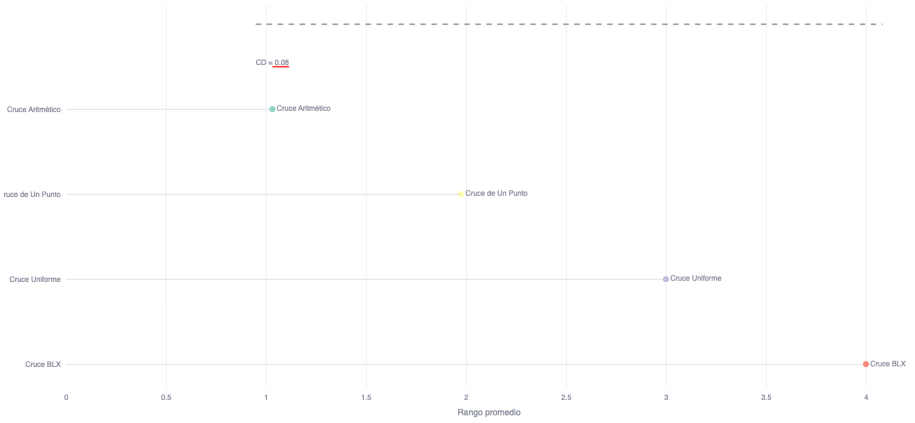

Figura 4.6: Gráfico de Nemenyi para los tiempos de cada operador de cruce.

En el gráfico de Nemenyi 4.6 se puede observar la misma distribución escalonada.

Memoria consumida

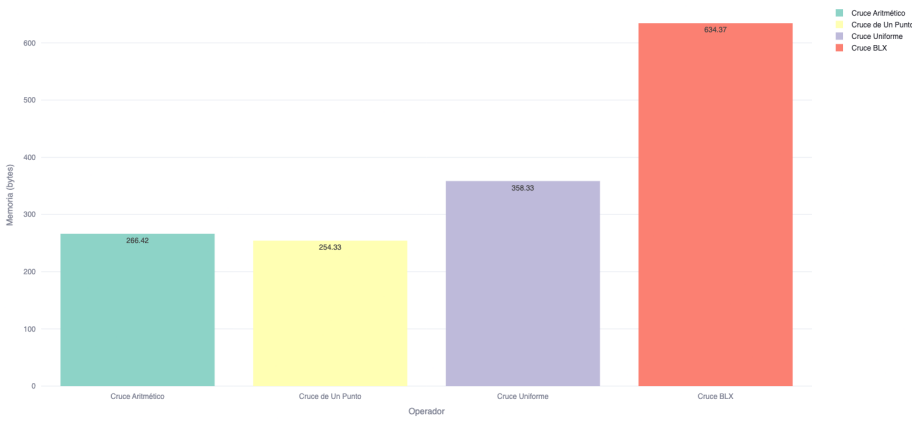

Figura 4.7: Memoria consumida por cada operador de cruce.

En la figura 4.7 se puede obser de nuevo una distribución escalonada ascendente de las medias de uso de memoria de cada operador de cruce, siendo el cruce aritmético un operador que ha usado menos memoria, sin embargo en esta ocasión el cruce de un punto lo mejora por muy poco, seguido por el cruce uniforme y finalmente el cruce BLX-α que es el que más memoria consume.

Estadísticos de las métricas de memoria

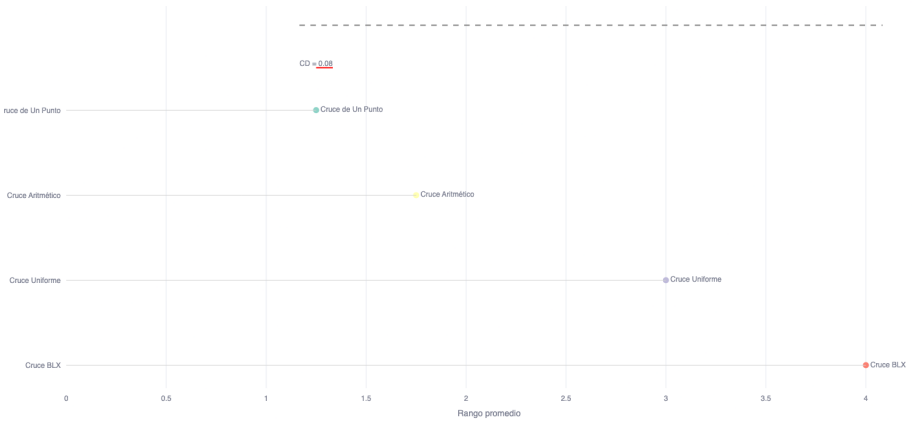

Figura 4.8: Gráfico de Nemenyi para la memoria de cada operador de cruce.

En el gráfico de Nemenyi 4.8 se puede observar como estadísticamente el cruce aritmético resulto ser al final que menos memoria consumió, seguido por el cruce de un punto, el cruce uniforme y finalmente el cruce BLX-α.

Convergencia

Figura 4.9: Convergencia de cada operador de cruce.

De esta gráfica 4.9 aunque a simple vista no deberiamos poder concluir nada, se puede aprecias que el cruce BLX-α tiene al principio una convergencia más lenta teniendo a lo largo de las generaciones repuntes donde se obtenian soluciones peores, por otro lado el cruce aritmético desde el comienzo tiene una convergencia muy rápida.

Estadísticos de la convergencia

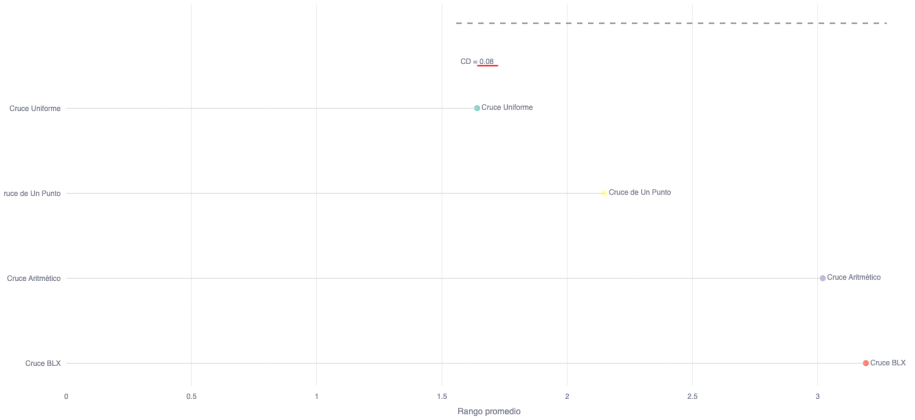

Figura 4.10: Gráfico de Nemenyi para la convergencia de cada operador de cruce.

En este caso podemos observar que el cruce aritmético es el que mejor convergencia tiene, seguido por el cruce de un punto, el cruce uniforme y finalmente el cruce BLX-α que es el que peor convergencia tiene.

Conclusiones sobre los operadores de cruce

En base a las métricas evaluadas, se pueden extraer las siguientes conclusiones sobre los operadores de cruce:

1. Tiempo consumido:

- El cruce aritmético es el más eficiente en términos de tiempo, siendo el que menos tiempo consume.
- El cruce de un punto le sigue, con un tiempo de ejecución ligeramente mayor.
- El cruce uniforme ocupa el tercer lugar en consumo de tiempo.
- El cruce BLX-α es el que más tiempo consume, debido a su complejidad en la generación de descendencia.

2. Memoria consumida:

- El cruce aritmético es el que menos memoria consume, seguido muy de cerca por el cruce de un punto.
- El cruce uniforme tiene un consumo de memoria intermedio.
- El cruce BLX-α es el que más memoria consume, debido a la necesidad de manejar intervalos extendidos.

3. Convergencia:

- El cruce aritmético tiene la mejor convergencia, logrando soluciones óptimas rápidamente.
- El cruce de un punto tiene una buena convergencia, aunque ligeramente inferior al cruce aritmético.
- El cruce uniforme presenta una convergencia más lenta en comparación con los anteriores.
- El cruce BLX-α es el que peor convergencia tiene, mostrando repuntes en las generaciones donde se obtienen soluciones de menor calidad.

Conclusión general:

- Si se busca eficiencia en tiempo y memoria, el cruce aritmético es la mejor opción, además de tener la mejor convergencia.
- El cruce de un punto es una alternativa balanceada, con buen rendimiento en todas las métricas.
- El cruce uniforme es útil para mantener diversidad, pero sacrifica algo de eficiencia y convergencia.
- El cruce BLX-α, aunque permite explorar soluciones innovadoras, es el menos eficiente en tiempo y memoria, y tiene la peor convergencia.

Capítulo 5

Operadores de Mutación

5.1. Mutación Gaussiana

Este operador añade ruido gaussiano a los genes seleccionados. La mutación se aplica con una probabilidad dada, modificando los valores mediante una distribución normal centrada en el valor original. Ideal para ajustes progresivos en espacios continuos.

Bondades:

- Perturbaciones controladas por σ.
- Mantiene diversidad local.
- Eficiente computacionalmente.

Desventajas:

- Sensible a la elección de σ.
- No garantiza saltos grandes en el espacio de búsqueda.

Parámetros:

- Desviación estándar (σ): Controla la magnitud de la mutación.
- Probabilidad de mutación por gen (p_m).

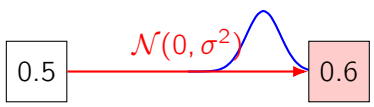

Figura 5.1: Mutación gaussiana aplicada a un gen con $\sigma = 0{,}1$.

5.2. Mutación Uniforme

Reemplaza valores seleccionados con números aleatorios uniformes en [0,1]. Efectivo para reinicializar genes y escapar de óptimos locales, especialmente en representaciones normalizadas.

Bondades:

- Máxima exploración en el espacio [0,1].
- Implementación simple y rápida.
- Efectivo contra convergencia prematura.

Desventajas:

- Pérdida de información valiosa.
- No direccional - puramente aleatorio.

Parámetros:

- Probabilidad de mutación por gen (p_m).

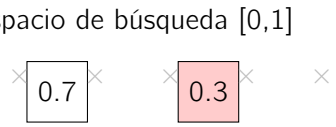

Figura 5.2: mutación uniforme dentro del rango normalizado [0,1].

5.3. Mutación No Uniforme

Este operador adapta dinámicamente la magnitud de la mutación en función del progreso del algoritmo. La intensidad disminuye exponencialmente con las generaciones, permitiendo grandes cambios iniciales y ajustes finos al final. Formalmente: $\Delta(t) = r \cdot (1 - \frac{t}{T})^b$, donde t es la generación actual, T el máximo de generaciones, r el rango máximo, y b controla la tasa de decaimiento.

Bondades:

- Balance automático exploración/explotación
- Adaptación progresiva a la fase del algoritmo
- Efectivo en espacios de búsqueda complejos

Desventajas:

- Requiere conocer el número máximo de generaciones
- Sensible a la configuración del parámetro b

Parámetros:

- Generación actual t y máximo generaciones T
- Factor de forma ($b \in [2, 5]$)
- Probabilidad de mutación p_m

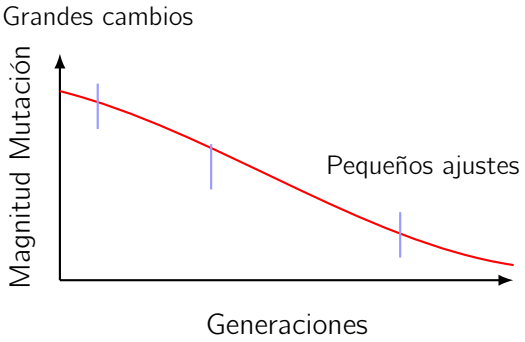

Figura 5.3: Evolución típica de la magnitud de mutación no uniforme ($b = 3$).

5.4. Mutación Pareto (Polinómica)

Utiliza la distribución de Pareto para generar mutaciónes con colas pesadas. Produce principalmente cambios pequeños pero permite ocasionalmente mutaciónes grandes, ideal para exploración multi-escala.

Bondades:

- Balance entre exploración local y saltos grandes.
- Efectivo en problemas multi-modales.
- Distribución de cola pesada para diversidad.

Desventajas:

- Puede generar valores extremos no acotados.
- Sensible al parámetro de forma (α).

Parámetros:

- Probabilidad de mutación por gen (p_m).
- Parámetro de forma fijo ($\alpha = 4{,}7$).

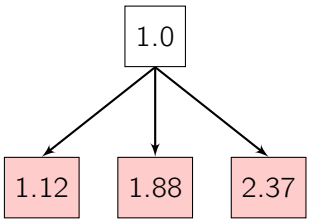

Figura 5.4: Mutación Pareto con $\alpha = 4{,}7$ permitiendo saltos variables.

5.5. Mediciones de los operadores de mutación

Tiempo consumido

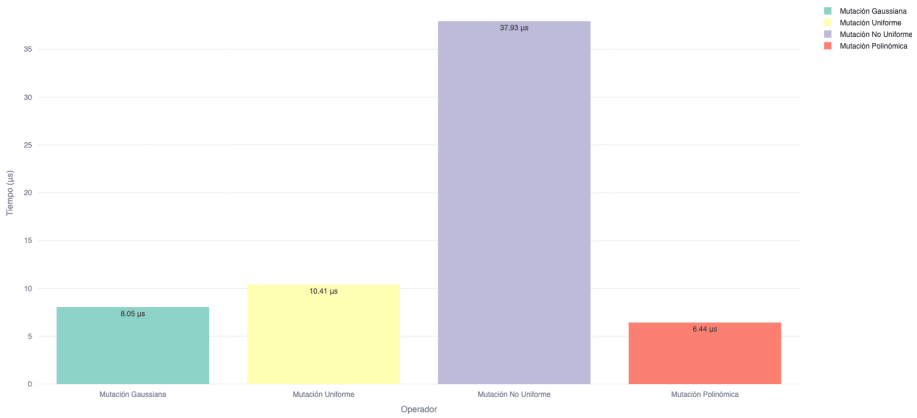

Figura 5.5: Tiempo consumido por cada operador de mutación.

En esta ocasion podemos ver que hay una distribución bastante variada de las medias de los tiempos de ejecución de los operadores de mutación. Resalta que la mutación polinomica es la que menos tiempo consume, seguido de la gaussiana y la uniforme, la mutación no uniforme destaca por su alto consumo de tiempo.

Estadísticos de las métricas de tiempo

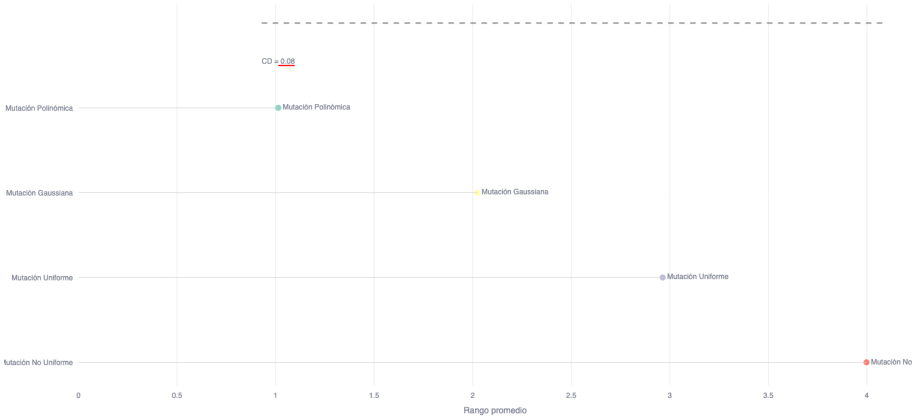

Figura 5.6: Gráfico de Nemenyi para tiempos de mutación.

En el gráfico de Nemenyi podemos ver que se respeta la jerarquía de los tiempos de ejecución anteriormente mencionada.

Memoria consumida

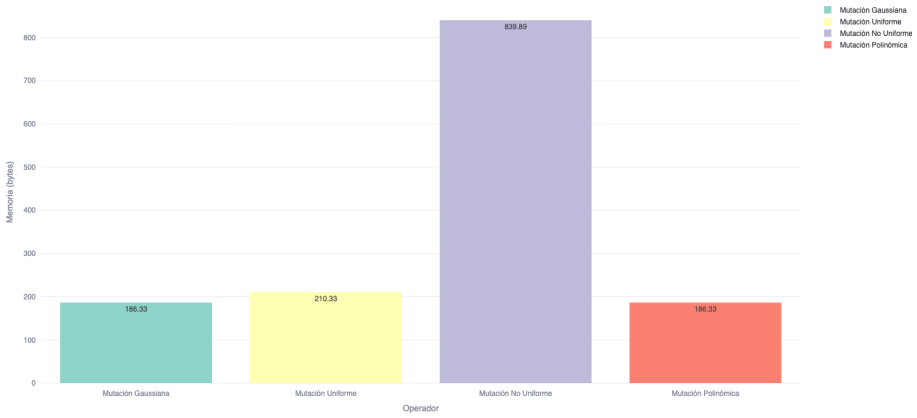

Figura 5.7: Memoria utilizada por cada operador.

En el caso de la memoria consumida se respeta un esquema similar al del tiempo consumido, salvo que ahora la mutación gaussiana y la polinomica están a la par.

Estadísticos de las métricas de memoria

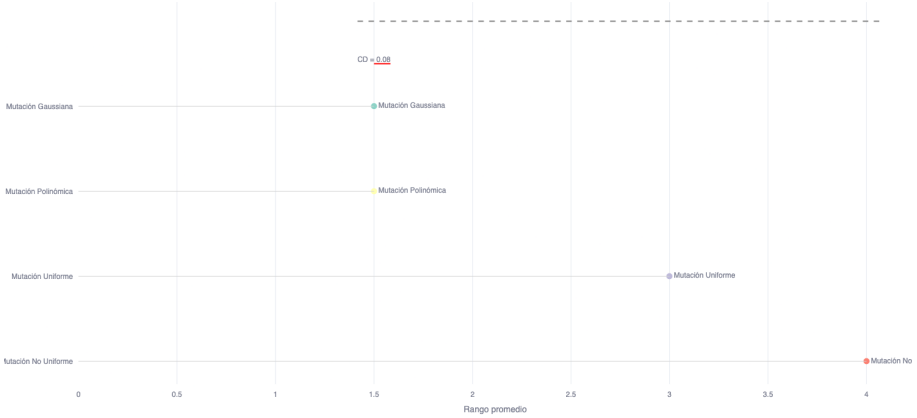

Figura 5.8: Comparación estadística de uso de memoria.

No hay comportamiento anómalo en el gráfico de Nemenyi, se respeta la jerarquía de los tiempos de ejecución y la memoria consumida. La mutación polinómica y la gaussiana son estadísticamente iguales.

Convergencia

Figura 5.9: Tasa de convergencia por operador.

En esta ocasión debido a la naturaleza de salto variable de la mutación polinómica, podemos ver que suele tener una convergencia constante salvo en algunos casos donde tenemos repuntes de soluciones que son peores, en cuanto a la convergencia de la mutación gaussiana y no uniforme, podemos ver que podrían a llegar a ser bastante similares.

Estadísticos de convergencia

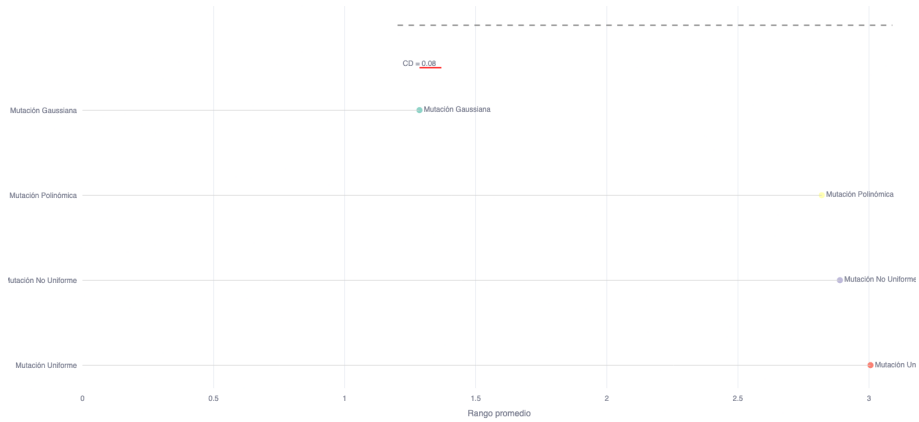

Figura 5.10: Análisis Nemenyi para convergencia.

En el gráfico de Nemenyi 5.10 podemos observar como de manera arrasadora la mutación gaussiana es la que tiene una convergencia más rápida.

Conclusiones sobre los operadores de mutación

En base a las métricas evaluadas, se pueden extraer las siguientes conclusiones sobre los operadores de mutación:

1. Tiempo consumido:

- La mutación polinómica es la más eficiente en términos de tiempo, siendo la que menos tiempo consume.
- La mutación gaussiana ocupa el segundo lugar, seguida de cerca por la mutación uniforme.
- La mutación no uniforme es la que más tiempo consume, debido a su complejidad en la adaptación dinámica.

2. Memoria consumida:

- La mutación polinómica y la gaussiana tienen un consumo de memoria similar, siendo las más eficientes.
- La mutación uniforme tiene un consumo de memoria intermedio.
- La mutación no uniforme es la que más memoria consume, debido a la necesidad de manejar parámetros dinámicos.

3. Convergencia:

- La mutación gaussiana tiene la mejor convergencia, logrando soluciones óptimas rápidamente.
- La mutación polinómica presenta una convergencia constante, aunque con repuntes ocasionales de soluciones de menor calidad debido a su naturaleza de salto variable.
- La mutación no uniforme y la uniforme tienen una convergencia similar, pero más lenta en comparación con la gaussiana.

Conclusión general:

- Si se busca rapidez y eficiencia en memoria, la mutación polinómica es la mejor opción, aunque puede tener repuntes en la calidad de las soluciones.
- Para una convergencia rápida y estable, la mutación gaussiana es la más adecuada.
- La mutación uniforme es útil para mantener diversidad, pero sacrifica algo de eficiencia y convergencia.
- La mutación no uniforme, aunque permite un balance dinámico entre exploración y explotación, es la menos eficiente en tiempo y memoria.

Capítulo 6

Operadores de Reemplazo

6.1. Reemplazo Generacional Completo

Este método reemplaza toda la población anterior con la nueva generación de descendientes. Es útil para mantener máxima diversidad y exploración del espacio de búsqueda, aunque puede perder buenas soluciones existentes.

Bondades:

- Máxima renovación de la población.
- Simple y rápido de implementar.
- Promueve amplia exploración.

Desventajas:

- Pérdida potencial de buenos individuos.
- Puede ralentizar la convergencia.

Figura 6.1: Reemplazo generacional completo.

6.2. Reemplazar al peor de la población

Este operador sustituye solo al peor individuo de la población actual con un nuevos descendiente. Conserva las mejores soluciones encontradas mientras permite cierta renovación.

Bondades:

- Conserva elites naturales.
- Balance entre exploración y explotación.
- Fácil de implementar.

Desventajas:

- Convergencia más lenta.
- Riesgo de estancamiento.

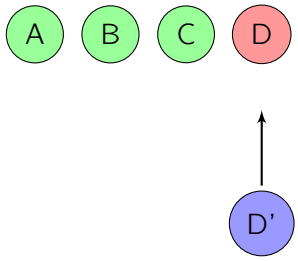

Figura 6.2: Reemplazo del peor individuo

6.3. Torneo restringido

Este método realiza torneos locales entre padres e hijos para mantener diversidad. Selecciona los mejores individuos en vecindarios específicos, previniendo convergencia prematura.

Bondades:

- Mantiene diversidad nichada.
- Previene dominancia temprana.
- Efectivo en problemas multimodales.

Desventajas:

- Coste computacional elevado.
- Sensible al tamaño del vecindario.

Parámetros:

- Tamaño del torneo.

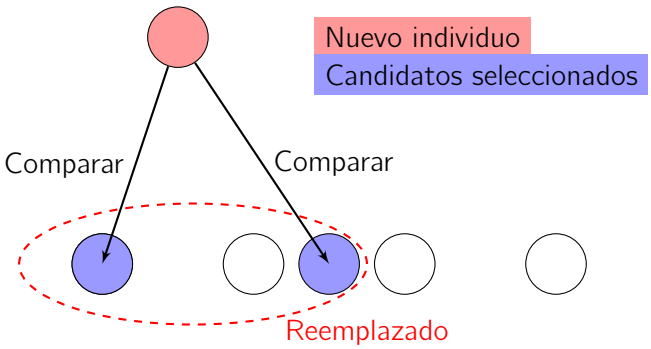

Figura 6.3: Torneo restringido: selección de candidatos.

6.4. Peor entre semejantes

Este operador identifica grupos de individuos similares y reemplaza el peor de cada grupo. Favorece la preservación de soluciones diversas mientras elimina malas copias.

Bondades:

- Mantiene diversidad cualitativa.
- Elimina redundancias.
- Eficiente en espacio de soluciones.

Desventajas:

- Complejidad en cálculo de similitud.
- Sensible a la métrica de similaridad.

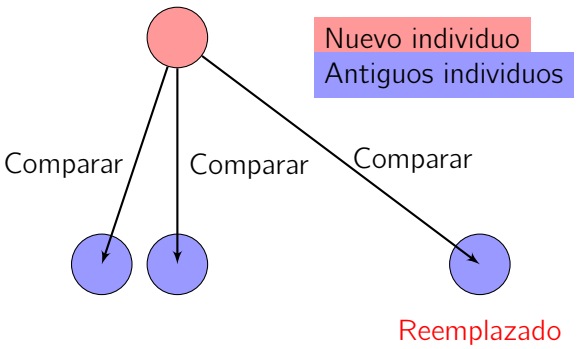

Figura 6.4: Peor entre semejantes

6.5. Elitismo

Esta estrategia preserva automáticamente los mejores individuos entre generaciones. Garantiza que la calidad de la solución no disminuya, aunque puede reducir diversidad.

Bondades:

- Garantiza mejora monótona.
- Simple y efectivo.
- Esencial en problemas complejos.

Desventajas:

- Puede causar convergencia prematura.
- Reduce presión selectiva.

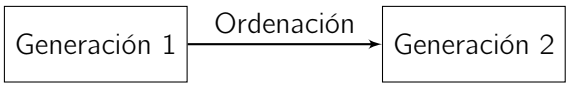

Figura 6.5: Preservación de individuos élite entre generaciones.

6.6. Mediciones de los operadores de reemplazo

Tiempo consumido

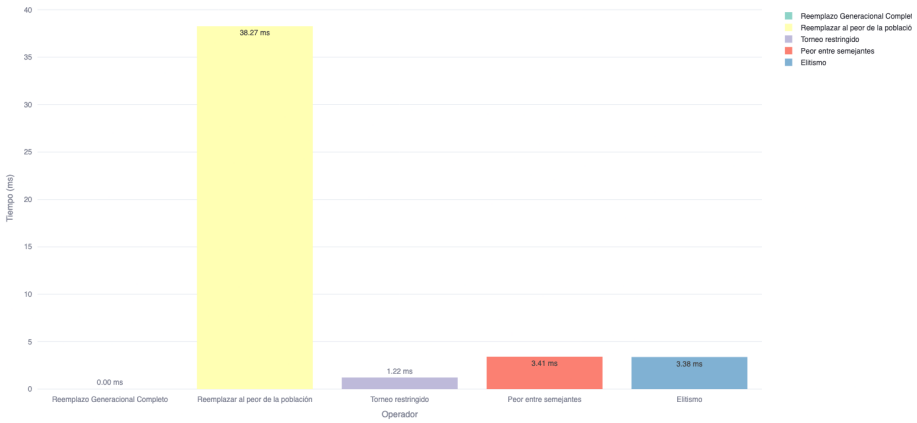

Figura 6.6: Tiempo consumido por cada operador de reemplazo.

Como se puede observar en la figura 6.6, los operadores de reemplazo tienen tiempos muy bajos de ejecución, destacando el reemplazo generacional completo por su rapidez casi instantánea y el reemplazo al peor de la población como el peor con un tiempo considerablemente mayor si lo comparamos con el resto de los operadores.

Estadísticos de las métricas de tiempo

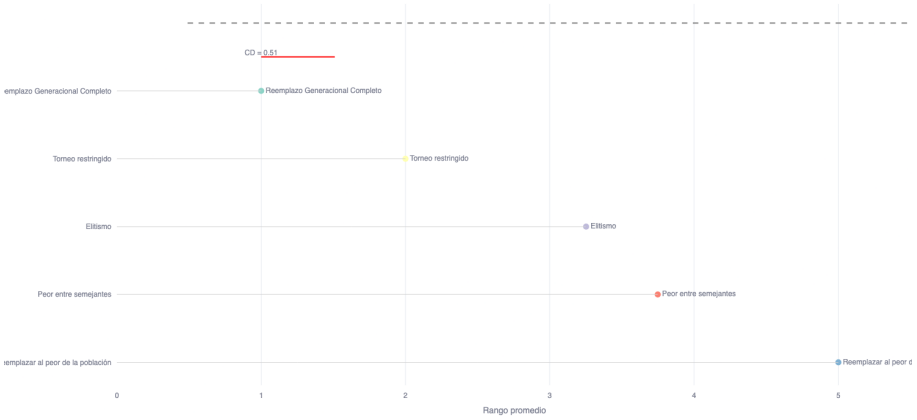

Figura 6.7: Gráfico de Nemenyi para tiempos de reemplazo.

En el gráfico de Nemenyi 6.7 se puede observar lo anteriormente mencionado, destacar que el operador de elitismo y el de peor entre semejantes tienen un tiempo similar y el torneo restringido les sigue por detrás.

Memoria consumida

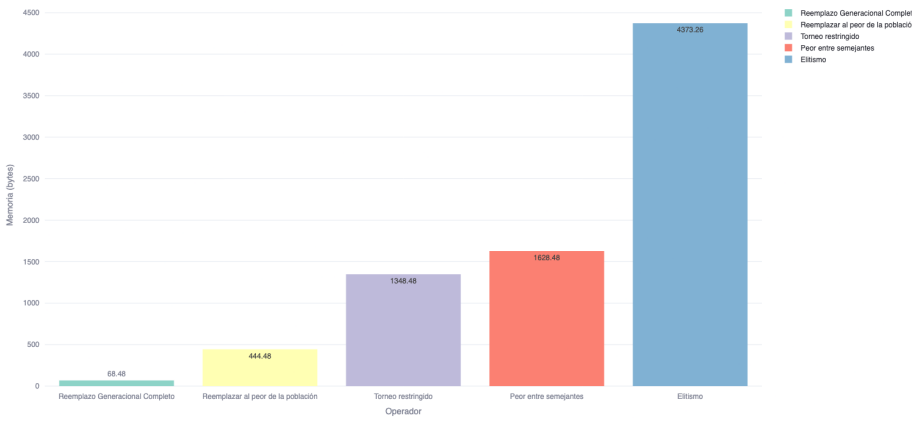

Figura 6.8: Memoria utilizada por cada operador.

En el caso de la memoria consumida por los operadores de reemplazo, podemos ver una distribución escalonada ascendente del uso de memoria, que va desde el reemplazo generacional completo, que es el que menos memoria consume, hasta elitismo que más memoria consume de todos.

Estadísticos de las métricas de memoria

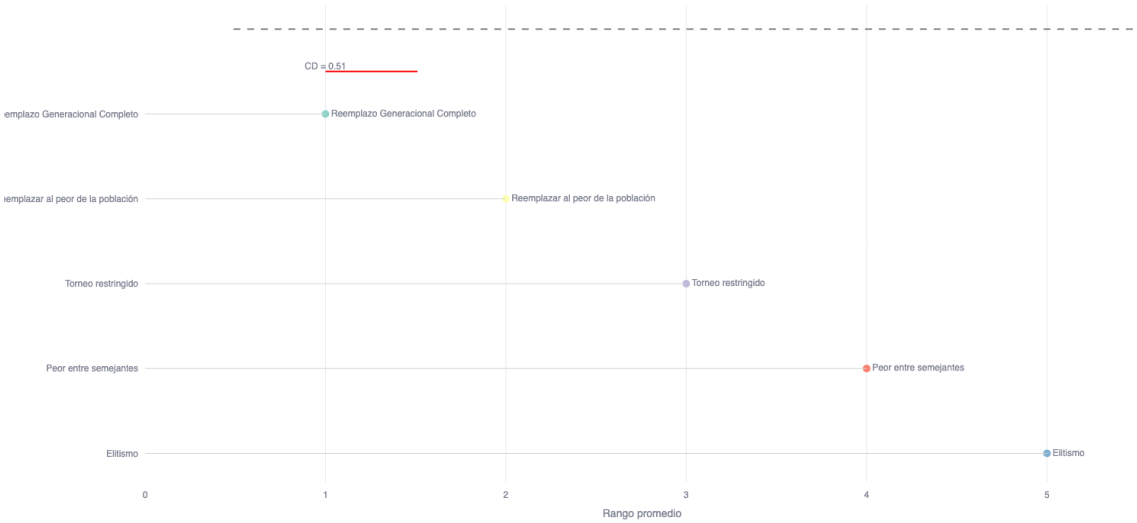

Figura 6.9: Comparación estadística de uso de memoria.

Dicha tendencia mencionada anteriormente se puede observar en el gráfico de Nemenyi 6.9.

Convergencia

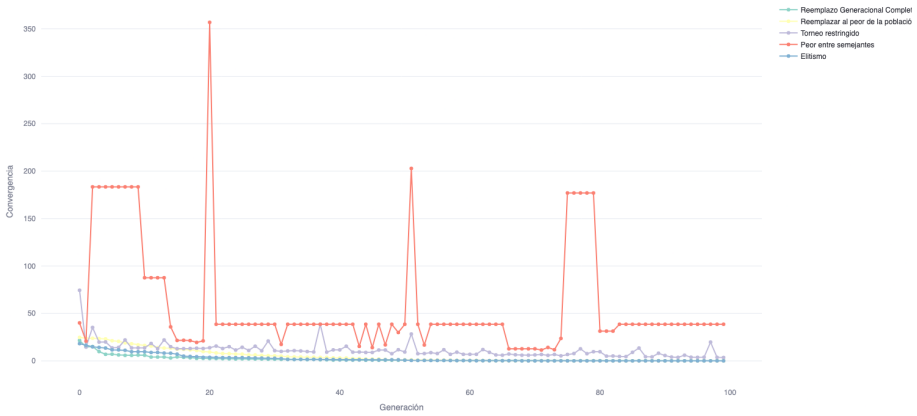

Figura 6.10: Tasa de convergencia por operador.

En concreto las conclusiones que podemos sacar de la figura 6.10 es que en general los operadores de reemplazo tienen una convergencia similar, aunque el operador de peor entre semejantes tiene una convergencia más lenta, alcanzando ocasionalmente soluciones de peor calidad.

Estadísticos de convergencia

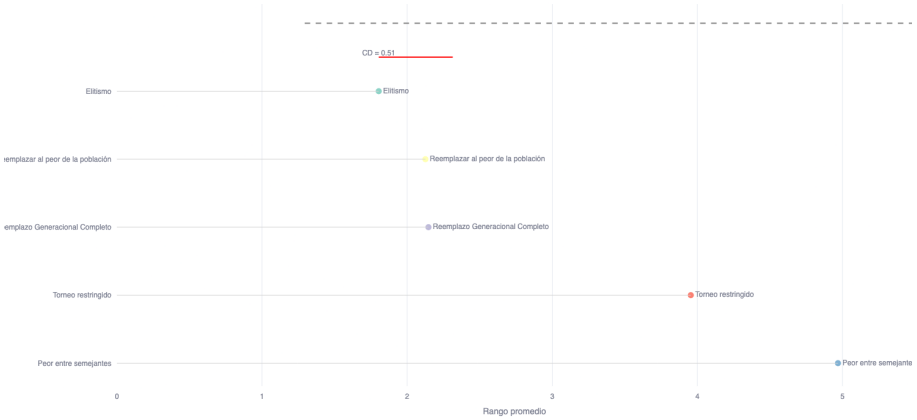

Figura 6.11: Análisis Nemenyi para convergencia.

Como era de esperar en el gráfico de Nemenyi 6.11 el operador de elitismo es el que tiene una convergencia más rápida de todas, siendo estadísticamente iguales el reemplazo generacional completo, el reemplazo al peor de la población, les sigue el torneo restringido y por último el peor entre semejantes que es el que tiene una convergencia más lenta de todos.

Conclusiones sobre los operadores de reemplazo

En base a las métricas evaluadas, se pueden extraer las siguientes conclusiones sobre los operadores de reemplazo:

1. Tiempo consumido:

- El reemplazo generacional completo es el más eficiente en términos de tiempo, siendo el que menos tiempo consume.
- El reemplazo al peor de la población es el que más tiempo consume, debido a la necesidad de identificar al peor individuo.
- El elitismo y el peor entre semejantes tienen tiempos de ejecución similares, seguidos por el torneo restringido.

2. Memoria consumida:

- El reemplazo generacional completo es el que menos memoria consume, debido a la simplicidad de su implementación.
- El elitismo es el que más memoria consume, ya que requiere mantener un registro de los mejores individuos.
- Los demás operadores (reemplazo al peor, torneo restringido y peor entre semejantes) tienen un consumo de memoria intermedio.

3. Convergencia:

- El elitismo tiene la mejor convergencia, garantizando una mejora continua en la calidad de las soluciones.
- El reemplazo generacional completo, el reemplazo al peor y el torneo restringido tienen una convergencia similar, aunque más lenta que el elitismo.
- El peor entre semejantes es el que tiene la convergencia más lenta, llegando ocasionalmente a soluciones de menor calidad.

Conclusión general:

- Si se busca rapidez y eficiencia en memoria, el reemplazo generacional completo es la mejor opción.
- Para garantizar una mejora continua en la calidad de las soluciones, el elitismo es el más adecuado, aunque consume más memoria.
- El torneo restringido y el peor entre semejantes son útiles para mantener diversidad, pero sacrifican algo de eficiencia y convergencia.
- El reemplazo al peor es una opción balanceada, aunque con un mayor costo en tiempo de ejecución.

Capítulo 7

Conclusiones

En esta práctica, se ha implementado y evaluado un algoritmo genético para ajustar los coeficientes de una función polinomial a un conjunto de datos. A continuación, se presentan las conclusiones generales obtenidas:

Trás haber estudiado cada uno de los operadores, sus pros, sus contras y el efecto que tienen en el rendimiento del algoritmo ahora vamos a concluir con los parametros finales de nuestro algoritmo, presentar nuestra solución final y una reflexión del equipo.

7.1. Explicación de los mejores parámetros

- **Tamaño de la población:** 23 individuos.
- **Número de generaciones:** 143.
- **Operador de selección:** Torneo con $k = 7$.
- **Operador de cruce:** BLX-α.
- **Operador de mutación:** Mutación Gaussiana con $\sigma = 0{,}05$.
- **Operador de reemplazo:** Elitismo.

El análisis que le podemos dar a estos parámetros es el siguiente:

El tamaño de la población y el número de generaciones relativamente pequeños ayudaron a la preservación de las poblaciones y tener una mayor velocidad en los operadores, además debido a la rápida convergencia de estos no es necesario tener un número elevado de generaciones para obtener soluciones de buena calidad.

El tamaño $k = 7$ del torneo selecciona lo que sería casi un tercio de población permitia escoger a los mejores padres de un percentil estable de la población, añadiendo una alta presión selectiva que junto también al reemplazo con elitismo se conseguian las mejores soluciones, por otro lado el operador de mutación gaussiana con una $\sigma = 0{,}05$ añadia un ajuste fino a los individuos sin alterarlos demasiado preservando sus mejores características, el cruce BLX-α fue lo que añadio la exploración de soluciones que necesitaba esta configuración con una convergencia tan rápida, ampliando o reduciendo el rango de algunos genes añadiendo la variabilidad genética necesaria.

7.2. Nuestra solución final

Nuestro individuo final fue obtenido con los parámetros anteriores usando cuatro islas en paralelo:

$$
\begin{aligned}
a &= 1.2386246519010953 \\
b &= 2.247436127001707 \\
c &= -0.09333146986430593 \\
d &= 1.923377600883105 \\
e &= -0.20332822348937588 \\
f &= 1.6381424682443946 \\
g &= 0.33636499482180904 \\
h &= -0.40839797663146593
\end{aligned}
$$

$$\text{MSE} = 0.0083$$

$$\text{Memoria utilizada} = 125706 \text{ bytes}$$

$$\text{Tiempo de ejecución} = 3.2990 \text{ segundos}$$

Figura 7.1: Captura de pantalla el individuo final

Con un MSE de 0.0083 es el mejor individuo que obtuvimos, dentro de lo que cabe en un tiempo razonable y con un consumo no demasiado alto de memoria.

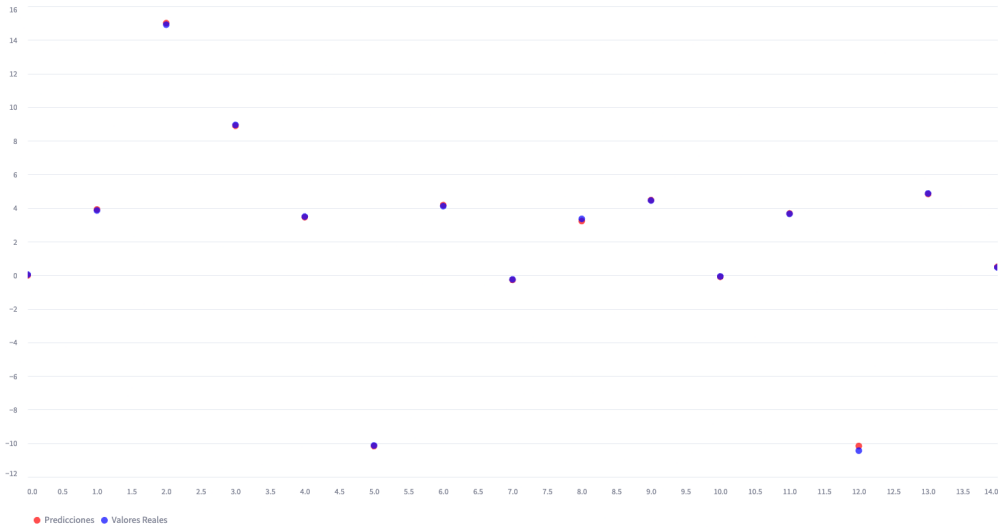

Figura 7.2: Aciertos de nuestro individuo

En la figura 7.2 podemos ver como aunque el individuo tiene un MSE muy bajo aun hay alguno puntos de los datos que no termina de predecir a la perfección, pero con un error asumible.

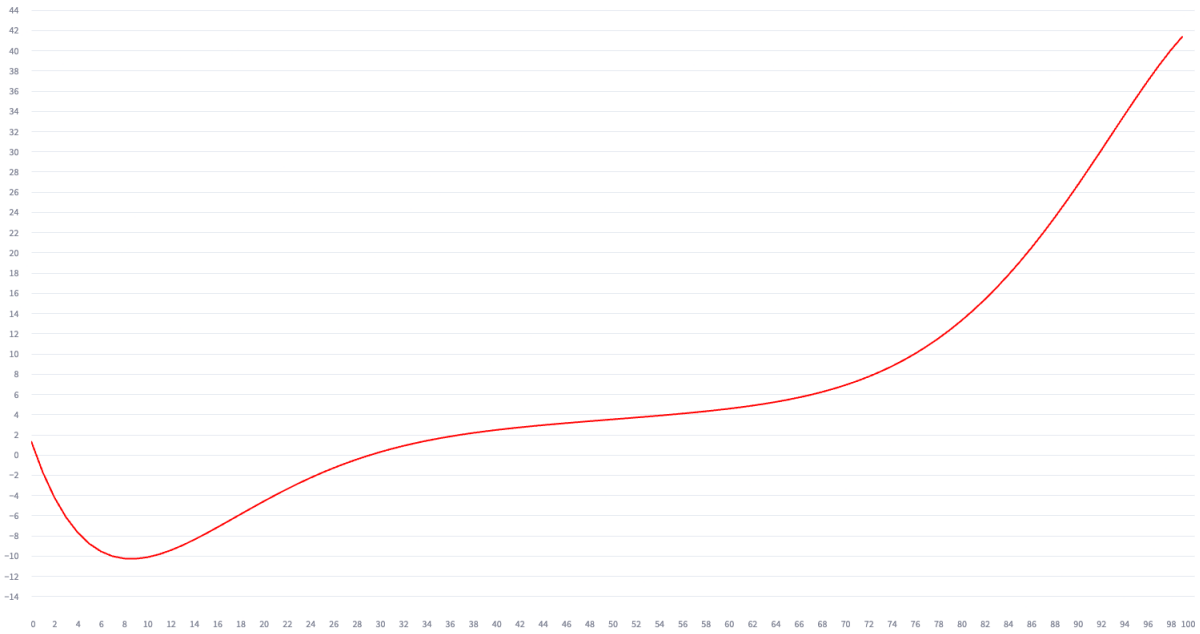

Figura 7.3: Gráfica de nuestra función

7.3. Aprendizaje del equipo

Podemos afirmar que después de realizar la práctica hemos aprendido mucho sobre algoritmos genéticos y sobre todo la configuración de estos y para más adelante una lección muy importante, y es que a veces es mejor una buena configuración de parámetros con un algoritmo simple es mejor (al menos en nuestro caso) que un algoritmo sofisticado, tosco y complejo, nos referimos con esto a que en la primera fase de la práctica nos centramos más en pulir el algoritmo con paralelismo ya que creiamos que era una baza muy importante para poder resolver el problema, al final fue muy aparatoso y con un consumo muy alto de recursos, al optimizar los parámetros nos dimos cuenta de esto y el paralelismo quedo en segundo plano, estaba ahí pero porque lo habiamos implementado y era un arma que debiamos usar, aunque no tuviera a posteriori un efecto tan grande en la calidad de las soluciones finales.

A parte y para finalizar después de analizar también el problema en profundidad no resultó ser un problema tan complejo, podría haber sido resuelto con un algoritmo regresión lineal sin necesidad de implementar tantas funcionalidades, de hecho en nuestro código tenemos implementado un algoritmo de la librería *scikit* que resuelve el problema de manera veloz usando muy pocas lineas de código, por lo que consideramos que el acercamiento con un algoritmo genético no era el mejor para resolver este tipo de problemas y hay mejores opciones como la mencionada.

Índice de cuadros

Índice de figuras

Bibliografía

[1] J. M. Luna Ariza. Metaheurísticas: Notas de clase. Material docente, Universidad de Córdoba, 2025.

[2] Wikipedia contributors. Bayesian optimization. https://en.wikipedia.org/wiki/Bayesian_optimization, 2025. Accessed: 2025-04-29.

[3] Wikipedia contributors. Algoritmo genético. https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico, 2025. Accessed: 2025-04-29.

[4] Wikipedia contributors. Pareto distribution. https://en.wikipedia.org/wiki/Pareto_distribution, 2025. Accessed: 2025-04-29.

[5] Scikit-Optimize Developers. Scikit-optimize: Sequential model-based optimization in python. https://scikit-optimize.github.io/stable/, 2025. Accessed: 2025-04-29.