Análisis Comparativo de EDAs Univariantes y Bivariantes en Problemas de Optimización Binaria

Lucía Martín-Núñez

19 de mayo de 2025

1. Introducción

Los algoritmos evolutivos son técnicas que imitan el proceso de evolución natural para resolver problemas complejos. Una de sus variantes más interesantes son los algoritmos de estimación de distribuciones (Estimation of Distribution Algorithms), conocidos como EDAs [Larrañaga and Lozano, 2002, Mühlenbein and Paaß, 1996]. Estos algoritmos constituyen una familia de metaheurísticas basadas en poblaciones que, a diferencia de los algoritmos genéticos tradicionales, no utilizan operadores de cruce ni mutación. En su lugar, aprenden un modelo probabilístico de las soluciones más prometedoras y generan nuevas soluciones mediante muestreo de dicho modelo. El interés por los EDAs ha crecido debido a su capacidad para aprovechar modelos probabilísticos, lo que permite evitar operadores heurísticos manuales y facilita su aplicación a problemas con estructuras complejas.

Este informe analiza comparativamente dos variantes discretas de EDAs:

- **Univariate Marginal Distribution Algorithm (UMDA)**, que asume que todas las variables del problema son independientes.
- **Mutual Information Maximization for Input Clustering (MIMIC)**, que tiene en cuenta posibles dependencias entre variables.

y las contrasta con un algoritmo genético básico, utilizando como base problemas clásicos de optimización combinatoria (OneMax y Knapsack).

2. Fundamentos Teóricos

2.1. UMDA

Tal como se plantea en el trabajo de [Mühlenbein, 1997] UMDA es un algoritmo de estimación de distribuciones que parte de una hipótesis fuerte: las variables que definen una solución son independientes entre sí. Esto significa que puede modelar la probabilidad de cada bit de manera separada, sin tener en cuenta lo que ocurra con los demás.

Modelo probabilístico:

$$p(x) = \prod_{i=1}^{n} p(x_i)$$

Funcionamiento paso a paso:

1. Se genera una población inicial de M soluciones binarias (por ejemplo, cadenas de 0s y 1s).
2. Cada individuo es evaluado según una función objetivo que mide su calidad.
3. Se seleccionan los N mejores individuos (normalmente, un 50 %).
4. Se calcula, para cada posición x_i, la proporción de individuos seleccionados que tienen un 1 en esa posición. Esto da como resultado un vector de probabilidades $p(x_i = 1)$.
5. Se generan nuevos individuos muestreando de forma independiente cada bit según estas probabilidades.
6. Se repite el proceso hasta alcanzar el número máximo de generaciones o una condición de parada.

Ejemplo ilustrativo: Si tras seleccionar los mejores individuos, la tercera posición del vector tiene un 70 % de unos y un 30 % de ceros, entonces en la próxima generación, cada nuevo individuo tendrá un 70 % de probabilidad de tener un 1 en esa posición.

2.2. MIMIC

MIMIC de [Bonet et al., 1996] es una versión más sofisticada de EDA que intenta capturar relaciones entre variables. En lugar de asumir que cada bit puede analizarse por separado, MIMIC busca aprender una secuencia de generación donde cada variable puede depender de la anterior.

Modelo probabilístico:

$$p_\pi(x) = p(x_{i_1} \mid x_{i_2}) \cdot p(x_{i_2} \mid x_{i_3}) \cdots p(x_{i_{n-1}} \mid x_{i_n}) \cdot p(x_{i_n})$$

Aquí, π es una permutación del conjunto de variables. MIMIC estima la entropía condicional entre pares de variables y ordena aquellas que muestran mayor dependencia informativa.

Funcionamiento paso a paso:

1. Se genera una población inicial de soluciones.
2. Se seleccionan los mejores individuos (según la función objetivo).
3. Se calcula cuánta información aporta cada variable sobre otra, usando medidas de entropía condicional.

4. Se determina un orden π de variables que minimiza la entropía total.

5. Se estiman las probabilidades condicionales entre variables ordenadas.

6. Se generan nuevos individuos siguiendo esta cadena condicional.

Ejemplo ilustrativo: Supongamos que los bits 3 y 4 están altamente correlacionados (si uno vale 1, el otro también). MIMIC intentará poner uno de ellos justo después del otro en su orden π, y estimará $p(x_3 \mid x_4)$.

2.3. Tabla comparativa: GA vs UMDA vs MIMIC

Características	**GA**	**UMDA**	**MIMIC**
Modelo probabilístico	No	Independencia marginal	Dependencias bivariadas ordenadas
Mecanismo de generación	Cruce y mutación	Muestreo Bernoulli	Muestreo condicional secuencial
Representación del conocimiento	Implícita (población)	Vector de probabilidades	Modelo condicional ordenado
Suposición de independencia	No	Total	Parcial
Complejidad computacional	Media	Baja	Media-Alta

Explicación de cada fila:

- **Modelo probabilístico**: Indica si el algoritmo utiliza un modelo explícito para representar la distribución de las soluciones. GA no usa modelo; UMDA usa distribuciones marginales independientes; MIMIC usa distribuciones condicionales ordenadas.

- **Mecanismo de generación**: Describe cómo se crean nuevas soluciones. GA usa operadores como cruce y mutación; UMDA y MIMIC muestrean desde el modelo aprendido.

- **Representación del conocimiento**: Indica si el algoritmo almacena conocimiento aprendido de forma explícita (por ejemplo, un vector de probabilidades en UMDA o una estructura ordenada en MIMIC) o lo deja implícito en la población como hace GA.

- **Suposición de independencia**: Describe qué hipótesis hace cada algoritmo sobre las relaciones entre variables. UMDA asume independencia total; MIMIC permite dependencias; GA no modela estas relaciones.

- **Complejidad computacional**: Mide cuánto cuesta computacionalmente ejecutar cada algoritmo por generación. MIMIC es el más costoso debido al cálculo de entropías condicionales y ordenación de variables.

3. Problemas de Prueba

Con el objetivo de comparar empíricamente el rendimiento de UMDA, MIMIC y un algoritmo genético básico, se utilizarán tres problemas clásicos de optimización combinatoria en dominio binario:

3.1. OneMax

OneMax es uno de los problemas más simples y utilizados en optimización evolutiva. Consiste en maximizar el número de unos en un vector binario:

$$f(x) = \sum_{i=1}^{n} x_i \quad \text{con } x_i \in \{0, 1\}$$

La solución óptima es el vector formado completamente por unos. A pesar de su sencillez, OneMax es muy útil para evaluar la capacidad de convergencia y la estabilidad de los algoritmos evolutivos.

3.2. Knapsack binario

En este clásico problema de optimización combinatoria, se tiene un conjunto de objetos, cada uno con un peso w_i y un beneficio v_i, y una capacidad total W. El objetivo es seleccionar un subconjunto de objetos que maximice el valor total sin superar la capacidad:

$$\text{máx} \sum_{i=1}^{n} v_i x_i \quad \text{sujeto a } \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}$$

Este problema permite evaluar cómo los algoritmos manejan restricciones explícitas y compromisos entre exploración y factibilidad. Es particularmente útil para comparar la capacidad adaptativa y la robustez de cada enfoque.

4. Consideraciones de implementación

Durante la fase de desarrollo experimental se valoró inicialmente el uso de la librería `EDAspy` de [Soloviev et al., 2024], una herramienta especializada en algoritmos de estimación de distribuciones. No obstante, se encontraron diversos problemas de compatibilidad relacionados con versiones de dependencias y restricciones en la definición de funciones de evaluación personalizadas, lo que impidió su integración efectiva en el entorno de ejecución.

Como consecuencia, se optó por realizar una implementación manual completa de los algoritmos UMDA, MIMIC y GA. Esta decisión proporcionó un control total sobre el proceso evolutivo y permitió personalizar el flujo de trabajo, incluyendo la selección de individuos, la estimación de distribuciones y el muestreo, así como la instrumentación para recoger métricas específicas (e.g., orden de variables, probabilidades marginales, diversidad).

El código fue desarrollado íntegramente en `Python`, utilizando bibliotecas estándar como `NumPy`, `Pandas` y `Matplotlib`. Se definió una estructura modular reutilizable, común para todos los problemas abordados, que incluye:

- Representación binaria de soluciones y funciones de evaluación específicas por problema.
- Esquema común de repetición de experimentos con semillas fijas.
- Generación de gráficos comparativos y exportación de resultados en formato `.csv`.

Este enfoque permitió validar los algoritmos en condiciones homogéneas y facilitó la trazabilidad de los resultados obtenidos.

Con el objetivo de garantizar la reproducibilidad de los experimentos presentados, todo el código fuente ha sido desarrollado en Python de forma modular y está disponible públicamente. El repositorio incluye:

- Implementaciones completas de UMDA, MIMIC y un algoritmo genético básico.
- Funciones auxiliares para el análisis y visualización de métricas (fitness, diversidad, orden de variables).
- Archivos `.csv` con los resultados por generación.
- Cuadernos ejecutables en formato `.ipynb` y `.html` para facilitar la inspección y replicación del análisis.

Este material puede consultarse en el siguiente repositorio abierto de GitHub[1], que permite seguir paso a paso la evolución de los experimentos, revisar las configuraciones utilizadas y extender el trabajo con nuevos problemas o variantes algorítmicas.

5. Experimentos con OneMax

5.1. Metodología

Tamaño de población	100
Ratio de selección	0.5 (50 % superior)
Generaciones	100 (máximo)
Repeticiones	1 ejecución por algoritmo
Lenguaje	Python

Se evaluó el rendimiento de tres algoritmos evolutivos sobre el problema OneMax con cadenas binarias de longitud 50. El objetivo era maximizar el número de bits a 1 en cada individuo, una función aditiva y separable ampliamente utilizada como benchmark en algoritmos de optimización.

Configuración común

Todos los algoritmos utilizaron una población de 100 individuos y ejecutaron un máximo de 100 generaciones, seleccionando en cada iteración el 50 % de mejores soluciones según su valor de fitness. La longitud del cromosoma fue de $n = 50$ bits. No se emplearon múltiples repeticiones, ya que el análisis se centró en la comparación directa del comportamiento convergente de cada técnica.

[1] `https://github.com/luciamartinnunez/Analisis-Comparativo-de-EDAs`

Algoritmo Genético (GA)

El algoritmo genético clásico empleó operadores estándar:

- **Selección**: truncamiento de élite (mejor 50 %).
- **Cruce**: de un punto.
- **Mutación**: bit a bit con una tasa de 0.01.

La generación de la nueva población se efectuó mediante recombinación aleatoria entre padres seleccionados, seguidos de una etapa de mutación independiente para cada bit.

UMDA

Se aplicó el algoritmo UMDA tal como se expone en la Sección 2, asumiendo independencia entre variables. En cada generación se calcularon las probabilidades marginales univariantes a partir de los individuos seleccionados, y se generó una nueva población mediante muestreo Bernoulli independiente.

MIMIC

MIMIC se implementó siguiendo el modelo condicional secuencial descrito previamente. Se calcularon órdenes de variables basados en entropía condicional y se generaron nuevos individuos a través de muestreo condicional. Su uso permite capturar dependencias entre variables, a diferencia de UMDA.

Implementación y recolección de métricas

Véase la Sección 4 para detalles sobre la implementación técnica común y recogida de métricas. Para UMDA y MIMIC se instrumentaron funciones auxiliares que permiten capturar: la evolución de las probabilidades marginales por posición (UMDA), y el orden de variables y su estabilidad generacional (MIMIC).

Se recogieron las siguientes métricas por generación:

- Mejor fitness alcanzado.
- Fitness medio de la población.
- Diversidad (distancia hamming media entre individuos).

Estas métricas permitieron analizar tanto la calidad de la solución como la velocidad y estabilidad del proceso de convergencia.

5.2. Resultados

En esta sección se presentan los resultados obtenidos por los tres algoritmos analizados: UMDA, MIMIC y un algoritmo genético (GA) clásico, aplicados al problema OneMax con cadenas binarias de longitud 50. La evaluación se basa en tres métricas principales: el fitness medio alcanzado, la desviación estándar del fitness durante la evolución, y la generación en la que se alcanza la convergencia.

Algoritmo	Fitness medio	Desv. estándar	Generación de convergencia
UMDA	47.2	7.17	9
MIMIC	48.0	7.65	11
GA	47.1	6.33	22

Cuadro 1: Resultados de los algoritmos evolutivos en el problema OneMax.

Análisis de resultados

Fitness medio. El algoritmo MIMIC alcanza el mayor valor de fitness medio final (48.0), lo que sugiere una mayor capacidad para capturar dependencias entre variables. Su modelo bivariante, basado en permutaciones y estimación condicional, permite una exploración más eficaz del espacio de soluciones. UMDA obtiene un fitness medio de 47.2, muy cercano al de MIMIC. Este rendimiento es coherente con la naturaleza del problema OneMax, el cual es separable y, por tanto, adecuado para modelos univariantes. En contraste, el algoritmo genético obtiene un valor ligeramente inferior (47.1), lo cual pone de manifiesto que, en este contexto, sus operadores de cruce y mutación no son tan efectivos como los mecanismos probabilísticos de los EDAs.

Desviación estándar. El algoritmo genético muestra la menor desviación estándar (6.33), lo que indica una evolución más estable de la población y menor variabilidad entre generaciones. Esto podría estar relacionado con una presión selectiva más moderada o una exploración más uniforme del espacio. En cambio, UMDA (7.17) y MIMIC (7.65) presentan una mayor variabilidad, lo cual es esperable en algoritmos basados en modelos probabilísticos, donde pequeñas diferencias en los individuos seleccionados pueden amplificarse en la estimación de distribuciones, sobre todo en las primeras generaciones.

Generación de convergencia. UMDA destaca por su rápida convergencia, alcanzando el óptimo en la generación 9. Este comportamiento está en línea con resultados teóricos previos que establecen cotas inferiores de convergencia ajustadas para UMDA en problemas separables como OneMax. MIMIC requiere dos generaciones más (11), posiblemente debido a la complejidad adicional asociada al aprendizaje del orden de las variables y sus dependencias condicionales. El algoritmo genético, por su parte, necesita 22 generaciones para converger, reflejando una dinámica evolutiva más lenta y menos eficiente.

Síntesis. Los resultados empíricos obtenidos confirman que UMDA y MIMIC son algoritmos especialmente eficaces en contextos donde la estructura del problema puede ser explotada mediante modelado probabilístico. En particular, UMDA se beneficia de la independencia entre variables en OneMax, mientras que MIMIC, aunque más complejo, muestra un rendimiento ligeramente superior. El algoritmo genético, si bien competitivo, se ve superado tanto en calidad de solución como en velocidad de convergencia.

Estos resultados están en consonancia con estudios teóricos que han establecido cotas inferiores ajustadas para la convergencia de UMDA en este tipo de problemas, concretamente

del orden:

$$\lambda + \mu\sqrt{n} + n \log n,$$

cuando $\lambda = \mathcal{O}(\mu)$ o $\mu = \mathcal{O}(\log n)$ [Krejca and Witt, 2020].

Análisis gráfico

La Figura 1 representa la evolución del mejor fitness alcanzado en cada generación para los tres algoritmos. Se observa una rápida mejora en las primeras generaciones para UMDA y MIMIC, alcanzando valores próximos al óptimo en menos de 15 generaciones. El algoritmo genético presenta una pendiente más moderada, reflejando su convergencia más lenta.

La Figura 2 muestra la evolución de las probabilidades univariantes aprendidas por UMDA a lo largo de las generaciones. Se aprecia una clara tendencia hacia la fijación (probabilidad cercana a 0 o 1) de cada bit, indicando convergencia hacia una solución óptima.

La Figura 3 muestra cómo se estabiliza la permutación de variables que define el orden condicional aprendido por MIMIC. A partir de la generación 10, la estructura de dependencias apenas varía, lo que refleja que el modelo aprendido se ha asentado y guía consistentemente la generación de nuevos individuos.

6. Experimentos con Knapsack

Configuración experimental

Para el problema de la mochila binaria (*0-1 Knapsack*), se utilizó una instancia generada aleatoriamente con las siguientes características:

- **Número de objetos:** 50
- **Pesos:** valores enteros generados aleatoriamente en el rango [1, 20]
- **Valores:** enteros entre 10 y 100
- **Capacidad total de la mochila:** 40 % de la suma total de los pesos

Todos los algoritmos se ejecutaron durante un máximo de 100 generaciones, con una población de 100 individuos y una tasa de selección del 50 %. La representación de los individuos es binaria, donde cada bit indica si un objeto es seleccionado o no.

Algoritmo Genético (GA). Se empleó selección por truncamiento, cruce de un punto, y mutación bit a bit con una probabilidad de 0.01.

UMDA. En cada generación se calcularon las probabilidades marginales univariantes a partir de los individuos seleccionados, y se generó una nueva población mediante muestreo independiente para cada bit.

MIMIC. Se modelaron dependencias bivariadas mediante el cálculo de entropía condicional y se utilizó un orden secuencial de las variables para construir un modelo condicional tipo cadena, a partir del cual se generaron nuevos individuos.

Para evaluar la estabilidad de los algoritmos, se repitieron los experimentos 10 veces con semillas distintas, y se midió tanto el valor promedio como la desviación estándar del mejor fitness alcanzado por generación.

Algoritmo	**Fitness medio**	**Desv. estándar**	**Mejor solución**
UMDA	1842.00	458.34	1876
MIMIC	1842.00	385.48	1842
GA	1509.35	291.10	1936

Cuadro 2: Resumen de resultados en Knapsack.

Análisis de resultados

Se presenta a continuación una evaluación cuantitativa de la ejecución final de los algoritmos aplicados al problema Knapsack. Además del fitness medio final y la mejor solución alcanzada, se analiza la desviación estándar de los resultados y el fitness medio global a lo largo de todas las generaciones.

- **UMDA** alcanza un fitness medio final elevado (1842.00), igualando a MIMIC, pero con la desviación estándar más alta (458.34), lo que sugiere una mayor variabilidad en la evolución de la población. Su mejor solución fue 1876, y la peor, 1206. Su fitness medio global fue 1573.06, lo que indica oscilaciones entre generaciones.
- **MIMIC** muestra un comportamiento más robusto, con el mismo fitness final que UMDA, pero con menor dispersión (385.48). Su fitness medio global (1669.97) es el más alto de los tres algoritmos, reflejando una evolución más estable. Su peor solución fue 1229 y la mejor 1842, lo que muestra una trayectoria consistente.
- **GA** obtiene el fitness medio más bajo (1509.35), pero también la menor desviación estándar (291.10), evidenciando una evolución más estable pero menos eficiente. A pesar de ello, logra la mejor solución individual de toda la ejecución (1936), indicando su capacidad exploratoria aleatoria, aunque con bajo rendimiento sostenido. Su fitness global fue 1344.61.

Síntesis. Los resultados confirman que MIMIC es el algoritmo más consistente y eficaz globalmente, mientras que UMDA presenta un buen desempeño con mayor dispersión. GA, aunque menos competitivo en promedio, puede ser útil en contextos donde se prioriza alcanzar puntualmente soluciones excepcionales.

Análisis gráfico

La Figura 4 muestra la evolución del fitness medio por generación. MIMIC presenta una mejora rápida y estabilización precoz, mientras que UMDA evoluciona de forma más irregular pero competitiva. GA progresa más lentamente, reflejando una estrategia más conservadora.

La Figura 5 representa la media del mejor valor alcanzado junto con su desviación estándar. UMDA alcanza el mejor rendimiento final, aunque con más variabilidad. MIMIC destaca por su estabilidad, mientras que GA muestra más dispersión inicial pero resultados competitivos en las últimas generaciones.

7. Líneas futuras

Una extensión natural de este trabajo sería aplicar los algoritmos evaluados al problema *MaxSAT*, una variante de satisfacción booleana ampliamente utilizada como benchmark en optimización combinatoria. MaxSAT consiste en encontrar una asignación de variables booleanas que maximice el número de cláusulas satisfechas en una fórmula en forma normal conjuntiva (CNF). Cada cláusula es una disyunción de variables booleanas o sus negaciones. Por ejemplo, una instancia 3-SAT puede representarse como:

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge \dots$$

Este problema, clasificado como *NP*-completo, representa un desafío importante para los algoritmos evolutivos debido a la presencia de múltiples óptimos locales y a la fuerte dependencia entre variables, lo que da lugar a un espacio de búsqueda altamente no lineal.

El uso de MaxSAT permitiría evaluar más profundamente la capacidad de los algoritmos para capturar dependencias complejas y explorar espacios de búsqueda con restricciones duras y soluciones dispersas. En particular, sería interesante analizar el comportamiento de UMDA y MIMIC en presencia de correlaciones lógicas entre variables, así como evaluar si el modelado bivariante de MIMIC ofrece ventajas sustanciales. Además, el algoritmo genético podría beneficiarse del uso de mecanismos adaptativos de exploración, dada la naturaleza altamente epistática del problema.

Esta línea de trabajo permitiría generalizar las conclusiones del presente estudio y analizar la robustez de cada enfoque en problemas de mayor complejidad estructural.

8. Conclusiones

A lo largo de este trabajo se ha analizado el rendimiento de tres algoritmos evolutivos —UMDA, MIMIC y un algoritmo genético clásico— aplicados a dos problemas representativos de optimización binaria: OneMax y Knapsack. La comparación experimental ha permitido observar diferencias relevantes en su comportamiento, tanto en cuanto a calidad de las soluciones como a estabilidad y capacidad de adaptación a distintas estructuras de problema.

UMDA ha funcionado especialmente bien en OneMax, donde su hipótesis de independencia entre variables encaja con la naturaleza separable del problema. Su principal ventaja ha sido la rapidez con la que converge, aunque en problemas con restricciones, como Knapsack, su rendimiento ha sido más irregular.

MIMIC ha mostrado un comportamiento más sólido y estable en ambos casos. Su capacidad para capturar dependencias entre variables le ha permitido mantener buenos resultados incluso en Knapsack, donde hay interacciones más complejas y restricciones duras. Esto lo convierte en una opción más robusta cuando la estructura del problema no es trivial.

El algoritmo genético, aunque en promedio por debajo de los EDAs, ha sido competitivo en algunos momentos concretos, especialmente en Knapsack, donde ha llegado a encontrar soluciones de alta calidad. Sin embargo, su falta de modelado explícito de la distribución poblacional lo hace menos eficaz cuando se requiere explotar relaciones internas entre variables.

En general, los EDAs han ofrecido mejores resultados globales. UMDA destaca por su sencillez y eficiencia en problemas simples. MIMIC, aunque más costoso computacionalmente, ha resultado más versátil y consistente. El algoritmo genético sigue siendo una alternativa razonable en escenarios donde se busque simplicidad o se desconozca la estructura del problema.

Estos resultados aportan a las bases de los EDAs soporte experimental para futuras aplicaciones en contextos reales donde la naturaleza del problema no sea completamente conocida, y donde la capacidad de modelar dependencias condicionales pueda suponer una ventaja significativa frente a métodos evolutivos tradicionales.

Referencias

[Bonet et al., 1996] Bonet, J. S. D., Jr., C. L. I., and Viola, P. A. (1996). MIMIC: finding optima by estimating probability densities. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 424–430. MIT Press.

[Krejca and Witt, 2020] Krejca, M. S. and Witt, C. (2020). Lower bounds on the run time of the univariate marginal distribution algorithm on onemax. *Theoretical Computer Science*, 832:143–165. Theory of Evolutionary Computation.

[Larrañaga and Lozano, 2002] Larrañaga, P. and Lozano, J. A., editors (2002). *Estimation of Distribution Algorithms*. Genetic Algorithms and Evolutionary Computation. Springer.

[Mühlenbein and Paaß, 1996] Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions i. binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature — PPSN IV*, pages 178–187, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Mühlenbein, 1997] Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346.

[Soloviev et al., 2024] Soloviev, V. P., Larrañaga, P., and Bielza, C. (2024). Edaspy: An extensible python package for estimation of distribution algorithms. *Neurocomputing*, 598:128043.

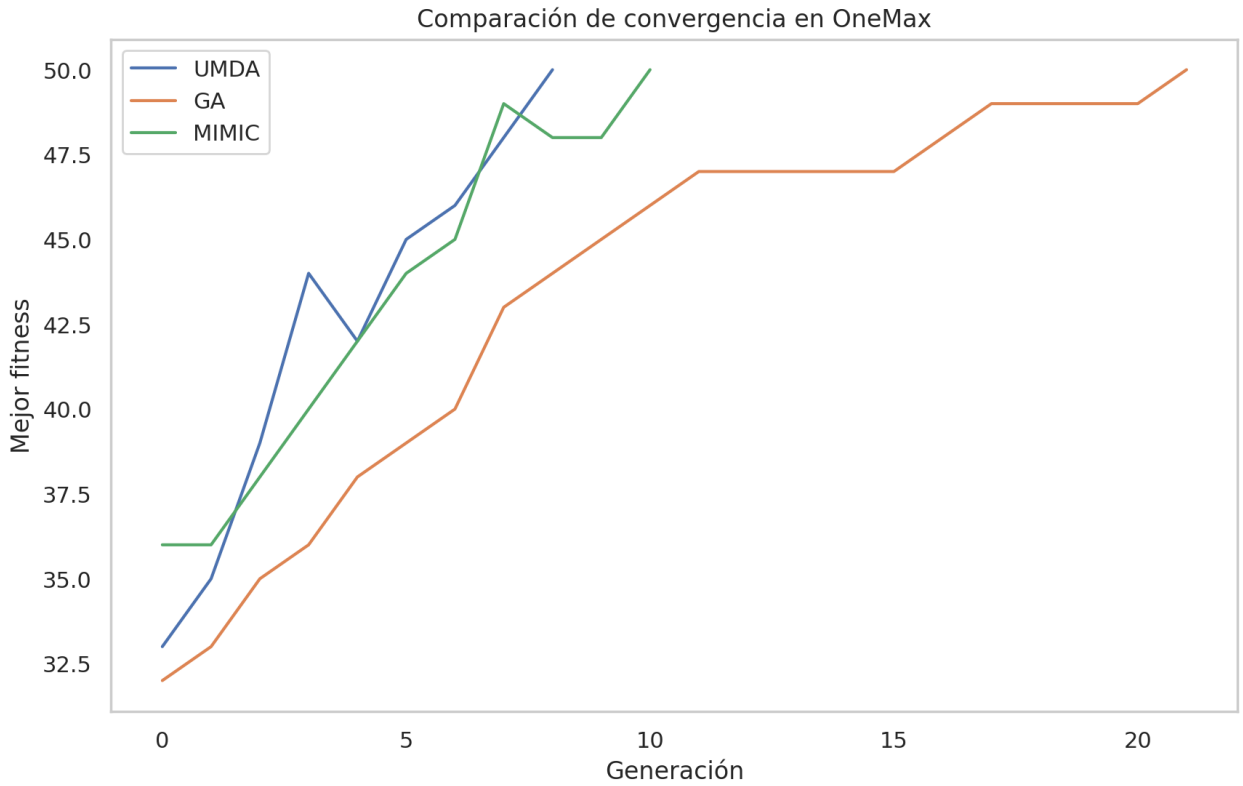

Figura 1: Curva de convergencia promedio

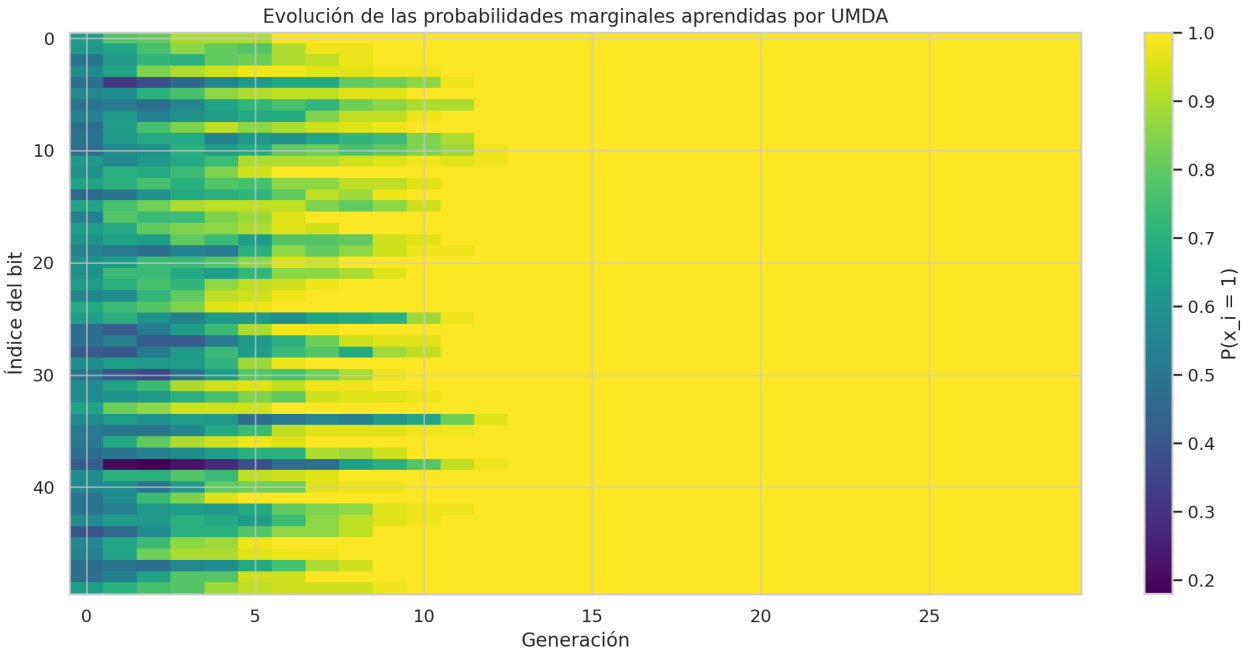

Figura 2: Mapa de calor de probabilidades marginales (UMDA)

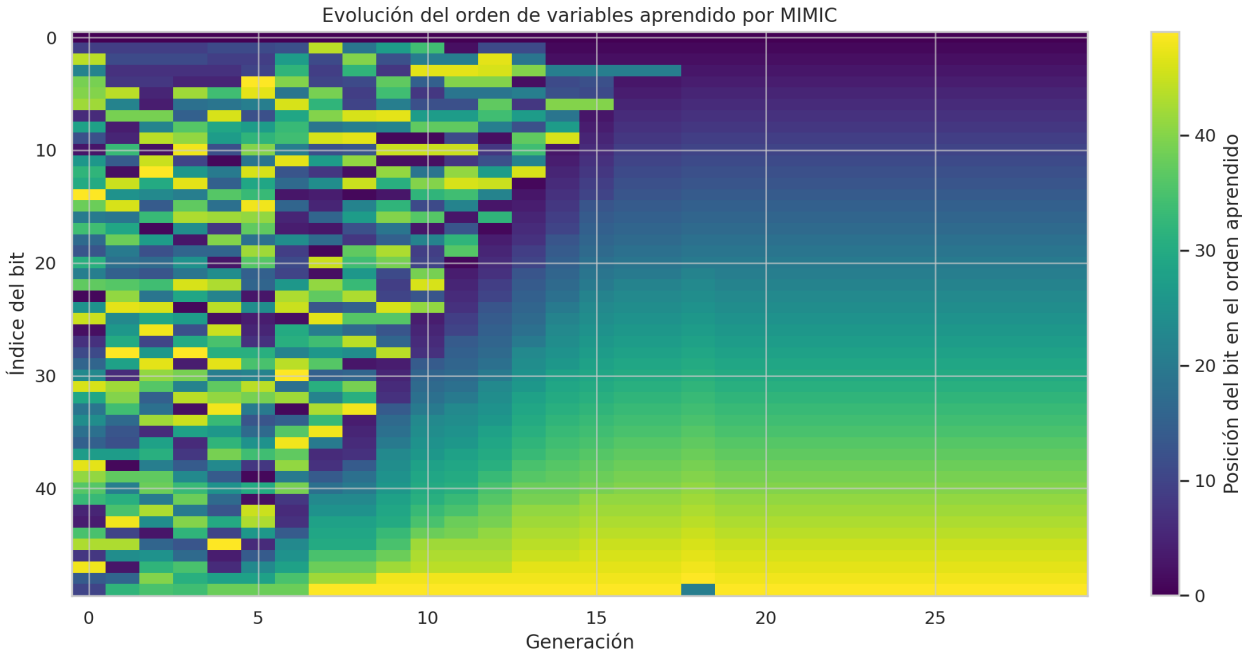

Figura 3: Evolución del orden de variables (MIMIC)

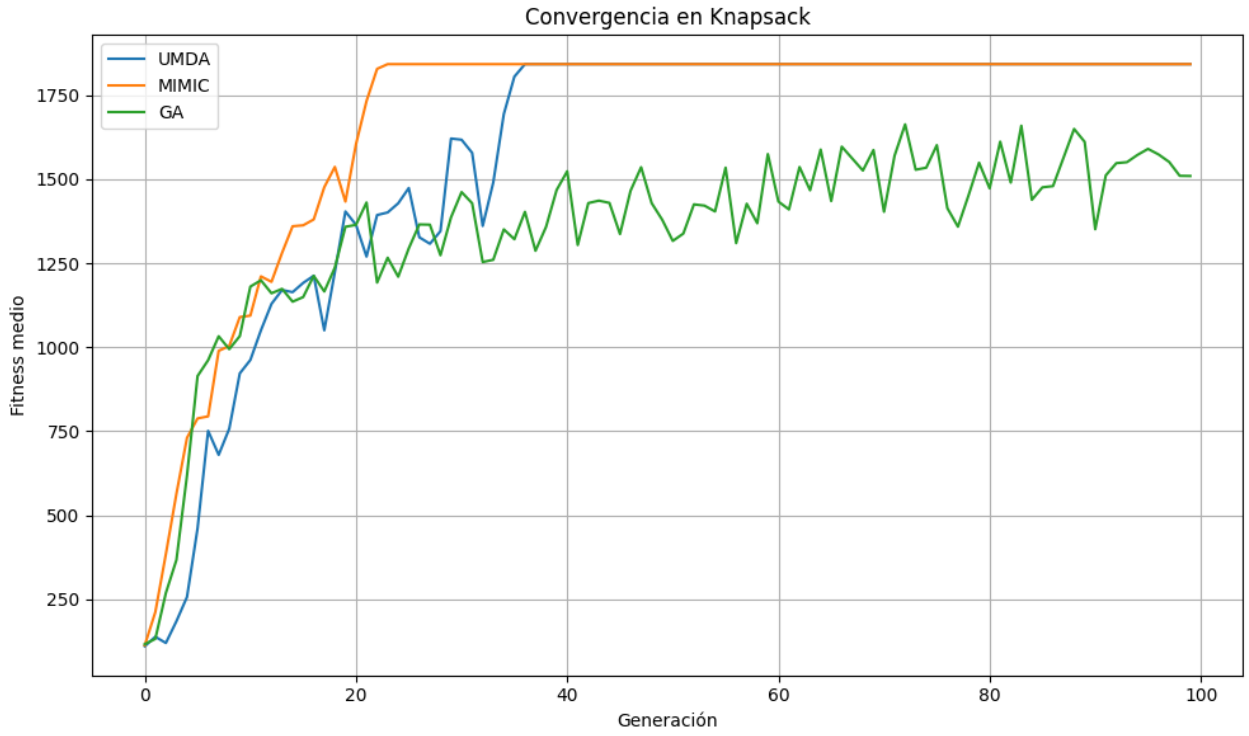

Figura 4: Convergencia en Knapsack — evolución del fitness medio por generación.

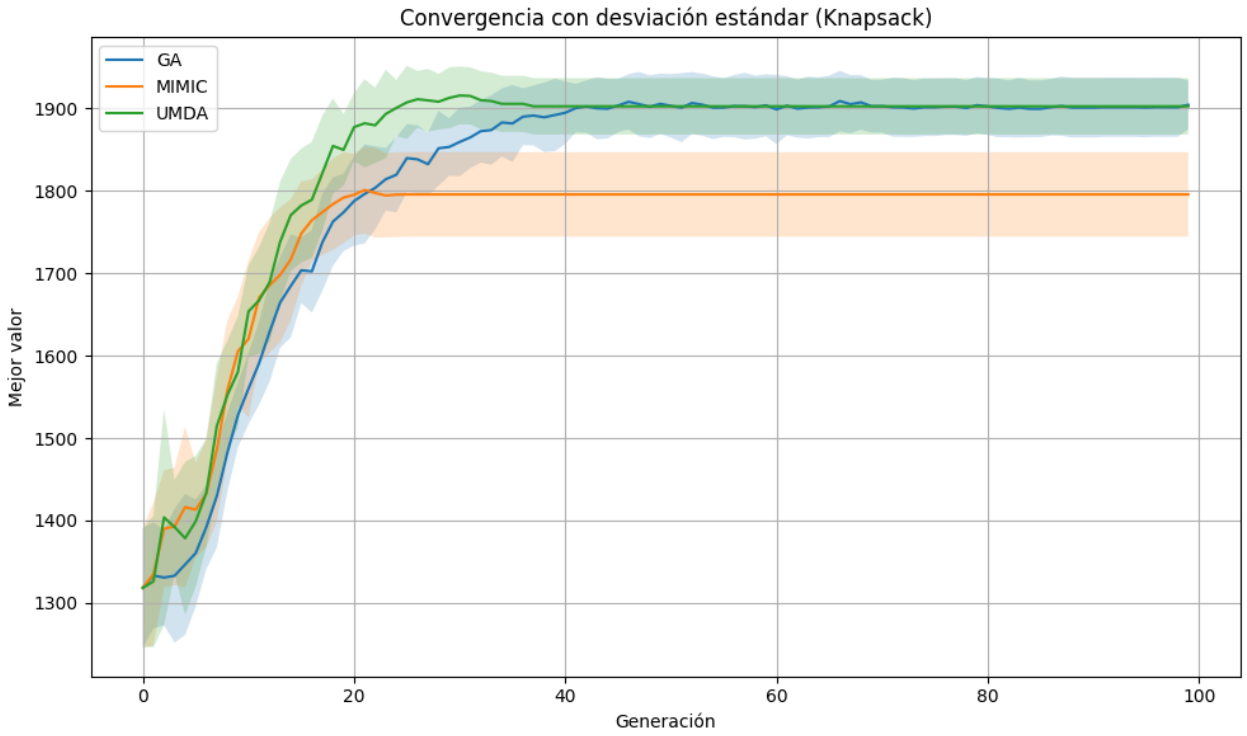

Figura 5: Convergencia media con desviación estándar — variabilidad del mejor fitness alcanzado.