A Comparative Analysis of Univariate and Bivariate EDAs in Binary Optimization Problems

Lucía Martín-Núñez

May 19, 2025

1 Introduction

Evolutionary algorithms are metaheuristics inspired by the process of natural selection to solve complex problems. Among their most promising variants are Estimation of Distribution Algorithms (EDAs) [Larrañaga and Lozano, 2002, Mühlenbein and Paaß, 1996], which form a family of population-based techniques that, unlike traditional genetic algorithms (GAs), do not use crossover or mutation operators. Instead, they learn a probabilistic model of the most promising solutions and generate new individuals by sampling from it. The interest in EDAs has increased due to their ability to exploit probabilistic models, avoiding the need for manually designed operators and enabling their application to problems with complex structures.

This report presents a comparative analysis of two discrete EDA variants:

- **Univariate Marginal Distribution Algorithm (UMDA)**, which assumes complete independence among variables.
- **Mutual Information Maximization for Input Clustering (MIMIC)**, which captures potential dependencies between variables.

These are contrasted with a baseline genetic algorithm, using classic combinatorial optimization problems (OneMax and Knapsack) as testbeds.

2 Theoretical Foundations

2.1 UMDA

As proposed by [Mühlenbein, 1997], UMDA is based on the strong assumption that all variables in the solution vector are mutually independent. Thus, the probability of a candidate solution can be modeled as:

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

Step-by-step procedure:

1. Generate an initial population M of binary solutions (e.g., bit strings).
2. Evaluate each individual using an objective function.
3. Select the top N individuals (commonly the top 50%).
4. Estimate the probability $p(x_i = 1)$ for each position i, based on the selected individuals.
5. Sample new individuals using independent Bernoulli distributions.
6. Repeat the process until a stopping criterion is met.

Illustrative example: If 70% of the selected individuals have a '1' in the third bit, then each new individual will have a 70% chance of having a '1' at that position.

2.2 MIMIC

MIMIC, introduced by [Bonet et al., 1996], improves upon UMDA by modeling conditional dependencies between variables via ordered chains:

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

Here, π denotes a permutation of the variables. MIMIC estimates conditional entropies to find an ordering that minimizes the total entropy of the model.

Step-by-step procedure:

1. Generate an initial population.
2. Select the top-performing individuals.
3. Compute pairwise conditional entropies.
4. Determine the optimal variable ordering π.
5. Estimate conditional probabilities accordingly.
6. Generate new individuals via sequential sampling.

Illustrative example: If bits 3 and 4 are strongly correlated, MIMIC will place one after the other in the ordering π and estimate $p(x_3 \mid x_4)$ accordingly.

2.3 Comparative Table: GA vs UMDA vs MIMIC

Feature	GA	UMDA	MIMIC
Probabilistic model	No	Marginal independence	Ordered bivariate dependencies
Generation mechanism	Crossover and mutation	Bernoulli sampling	Conditional sequential sampling
Knowledge representation	Implicit (population)	Probability vector	Ordered conditional model
Independence assumption	No	Total	Partial
Computational cost	Medium	Low	Medium-High

3 Benchmark Problems

To empirically evaluate the performance of UMDA, MIMIC, and a basic Genetic Algorithm (GA), two well-known combinatorial optimization problems in the binary domain were selected:

3.1 OneMax

OneMax is a simple yet widely used benchmark in evolutionary computation. The objective is to maximize the number of ones in a binary vector:

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

The global optimum is the all-ones vector. Despite its simplicity, OneMax is particularly useful for analyzing convergence behavior and stability of evolutionary algorithms.

3.2 Binary Knapsack

The classic 0-1 Knapsack problem involves selecting a subset of items with weights w_i and values v_i, such that the total value is maximized without exceeding a maximum capacity W:

$$\max \sum_{i=1}^{n} v_i x_i \quad \text{subject to } \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}$$

This problem serves to assess how algorithms handle constraints and trade-offs between exploration and feasibility. It is especially suitable for evaluating robustness and adaptability.

4 Implementation Considerations

Initially, the `EDAspy` library [Soloviev et al., 2024] was considered for implementation. However, compatibility issues related to package dependencies and custom objective function definitions hindered its effective integration.

As a result, full custom implementations of UMDA, MIMIC, and GA were developed from scratch in Python. This allowed complete control over the evolutionary process, including selection, model estimation, sampling, and metric instrumentation.

The implementation leveraged standard Python libraries such as `NumPy`, `Pandas`, and `Matplotlib`, and adopted a modular design with reusable components:

- Binary solution encoding and problem-specific fitness functions.
- Unified experimental loop with fixed seeds for reproducibility.
- Comparative plotting utilities and export of results to `.csv`.

All code and results are publicly available to support reproducibility. The GitHub repository[1] includes:

- Full implementations of UMDA, MIMIC, and GA.
- Utilities for metric tracking and visualization (fitness, diversity, variable ordering).
- Generation-wise `.csv` result files.
- Executable notebooks (`.ipynb`) and `.html` exports for inspection.

5 Experiments on OneMax

5.1 Methodology

Population size	100
Selection rate	0.5 (top 50%)
Generations	100 (maximum)
Repetitions	Single run per algorithm
Language	Python

Each algorithm was run on 50-bit binary strings. The goal was to maximize the number of 1s, a separable and additive objective.

Experimental Setup

All algorithms used a population of 100 individuals and a selection rate of 50%. Chromosome length was fixed at $n = 50$. Multiple repetitions were not employed, as the analysis focused on single-run comparative convergence behavior.

Genetic Algorithm (GA):

- **Selection:** elitist truncation (top 50%).
- **Crossover:** one-point.
- **Mutation:** bitwise with probability 0.01.

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

UMDA: As per Section 2, marginal probabilities were computed for each bit and new individuals were sampled independently using Bernoulli distributions.

MIMIC: Conditional entropy was used to determine variable orderings and dependencies, from which new individuals were generated sequentially.

Metric Collection

See Section 4 for implementation details. Additional instrumentation for UMDA and MIMIC included:

- Marginal probability evolution (UMDA).
- Variable ordering stability (MIMIC).

Metrics collected per generation:

- Best fitness achieved.
- Average fitness of the population.
- Diversity (mean Hamming distance).

5.2 Results

Algorithm	Mean Fitness	Std. Deviation	Convergence Generation
UMDA	47.2	7.17	9
MIMIC	48.0	7.65	11
GA	47.1	6.33	22

Table 1: Performance on OneMax

Fitness: MIMIC achieved the highest mean fitness, suggesting stronger exploration through conditional modeling. UMDA performed comparably well, aided by the separable structure of OneMax. GA yielded slightly lower performance.

Variance: GA exhibited the lowest variance, reflecting steady evolutionary dynamics. UMDA and MIMIC were more variable due to probabilistic sampling.

Convergence: UMDA converged fastest (generation 9), consistent with theoretical bounds for separable problems [Krejca and Witt, 2020]. MIMIC converged at generation 11, and GA at generation 22.

Synthesis: Both EDAs outperformed GA in convergence speed and solution quality. MIMIC showed marginally better robustness; UMDA showed excellent speed on separable problems.

$$\lambda + \mu\sqrt{n} + n \log n, \quad \text{when } \lambda = \mathcal{O}(\mu) \text{ or } \mu = \mathcal{O}(\log n)$$

Visual Analysis

- Figure 1: convergence curves.
- Figure 2: evolution of UMDA's marginal probabilities.
- Figure 3: variable ordering stabilization in MIMIC.

6 Experiments on Knapsack

Experimental Setup

Random instance with:

- **Items:** 50
- **Weights:** integers in [1, 20]
- **Values:** integers in [10, 100]
- **Capacity:** 40% of total weight

Each algorithm was run for 100 generations, with population size 100 and 50% selection rate.

Algorithm	Mean Fitness	Std. Deviation	Best Solution
UMDA	1842.00	458.34	1876
MIMIC	1842.00	385.48	1842
GA	1509.35	291.10	1936

Table 2: Performance on Knapsack

Results Analysis

- **UMDA:** high average fitness, but also highest variability, reflecting instability under constraint handling. Best solution was 1876.
- **MIMIC:** equal average fitness, but lower variance. Best solution was 1842. Highest global average fitness (1669.97).
- **GA:** lowest average fitness, yet found the best individual (1936). Most stable but least consistent.

Synthesis: MIMIC is the most robust and consistent. UMDA performs well but fluctuates more. GA shows potential for exceptional solutions, though less reliable.

Visual Analysis

- Figure 4: fitness evolution per generation.
- Figure 5: best fitness and deviation.

7 Future Work

A natural extension is to apply these algorithms to the MaxSAT problem, a canonical benchmark in combinatorial optimization. MaxSAT seeks to maximize the number of satisfied clauses in a CNF formula, such as:

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

This *NP*-complete problem poses additional challenges due to multiple local optima and complex variable dependencies. Studying how UMDA and MIMIC adapt to such landscapes could reveal important insights about conditional modeling advantages.

8 Conclusions

This work presented a comparative evaluation of three evolutionary algorithms—UMDA, MIMIC, and a classic GA—on binary optimization problems (OneMax and Knapsack). The experiments reveal:

- **UMDA** excels in separable problems like OneMax, with fast convergence but more erratic behavior on constrained problems.
- **MIMIC** offers robustness across both problems, consistently handling dependencies and yielding stable results.
- **GA**, though less effective on average, produced the best single solution in Knapsack, showing promise in highly multimodal landscapes.

EDAs generally outperformed the genetic baseline, particularly in convergence speed and structure exploitation. These findings support further application of EDAs in real-world scenarios where problem structure is not fully known, and conditional modeling may be advantageous over traditional evolutionary methods.

References

[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.

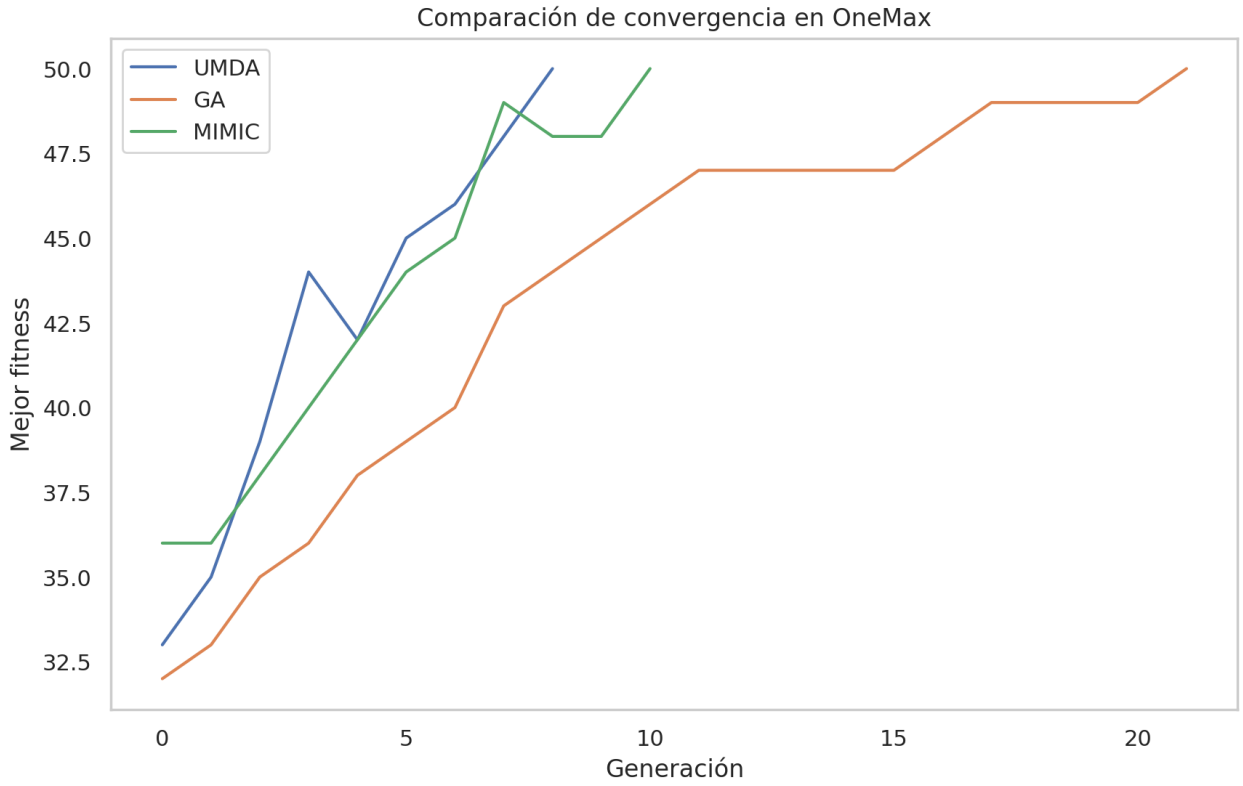

Figure 1: Average convergence curve

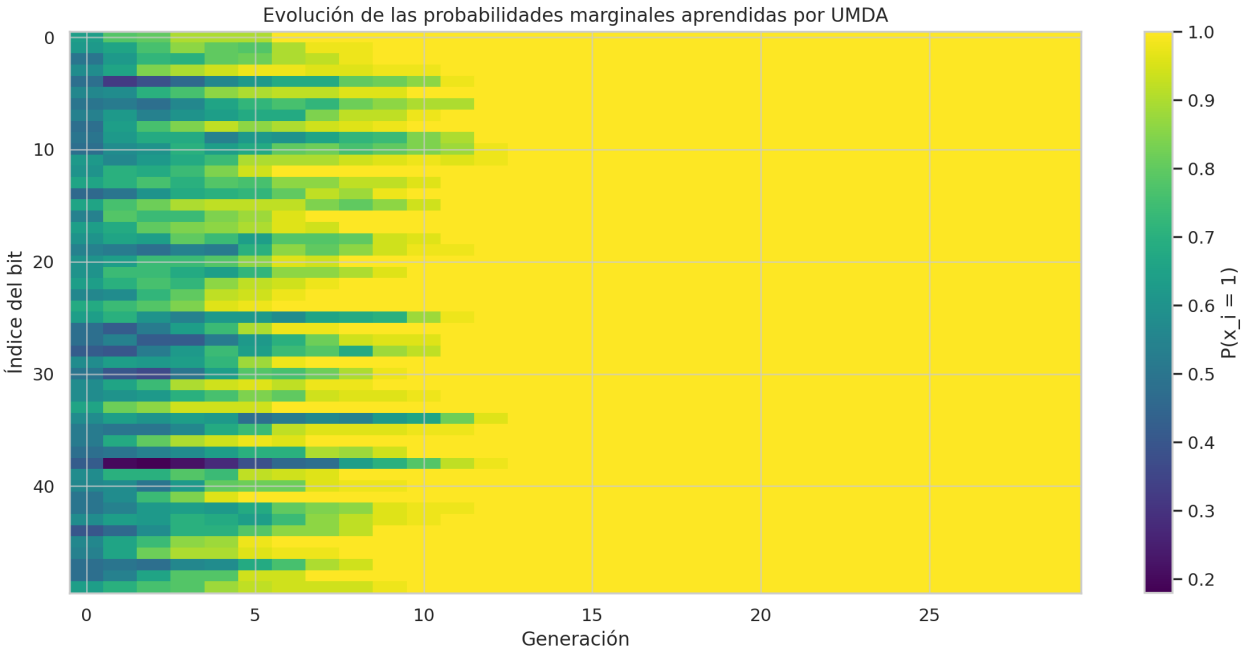

Figure 2: UMDA marginal probability heatmap

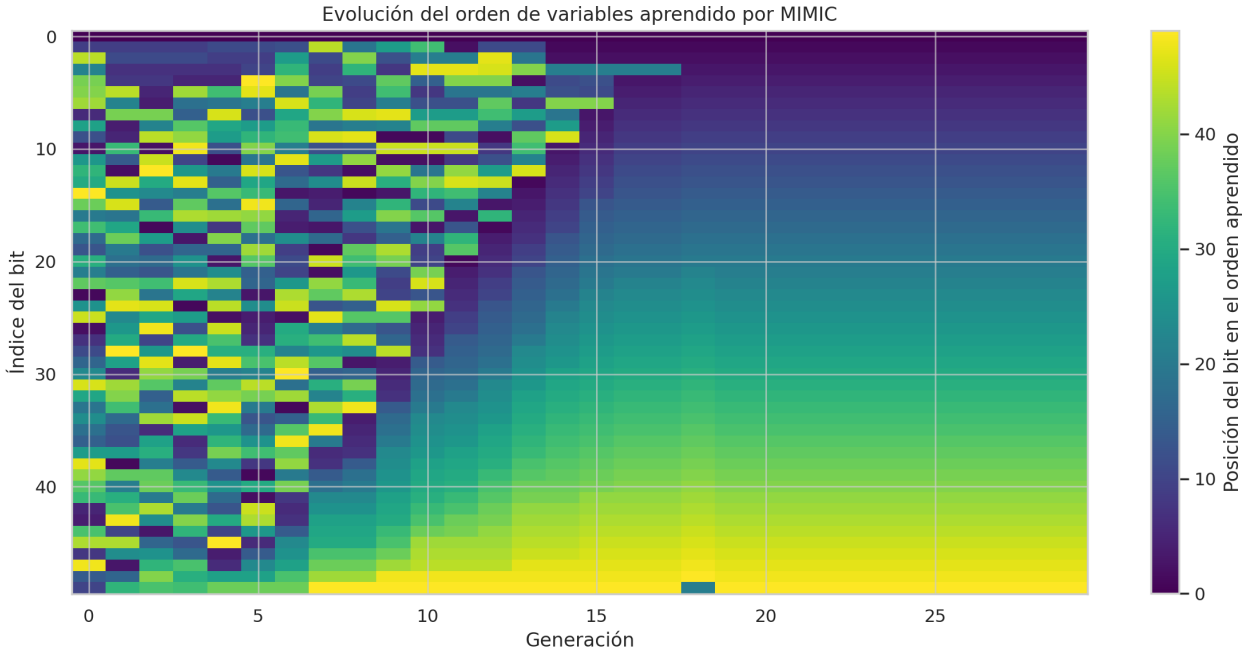

Figure 3: MIMIC variable ordering evolution

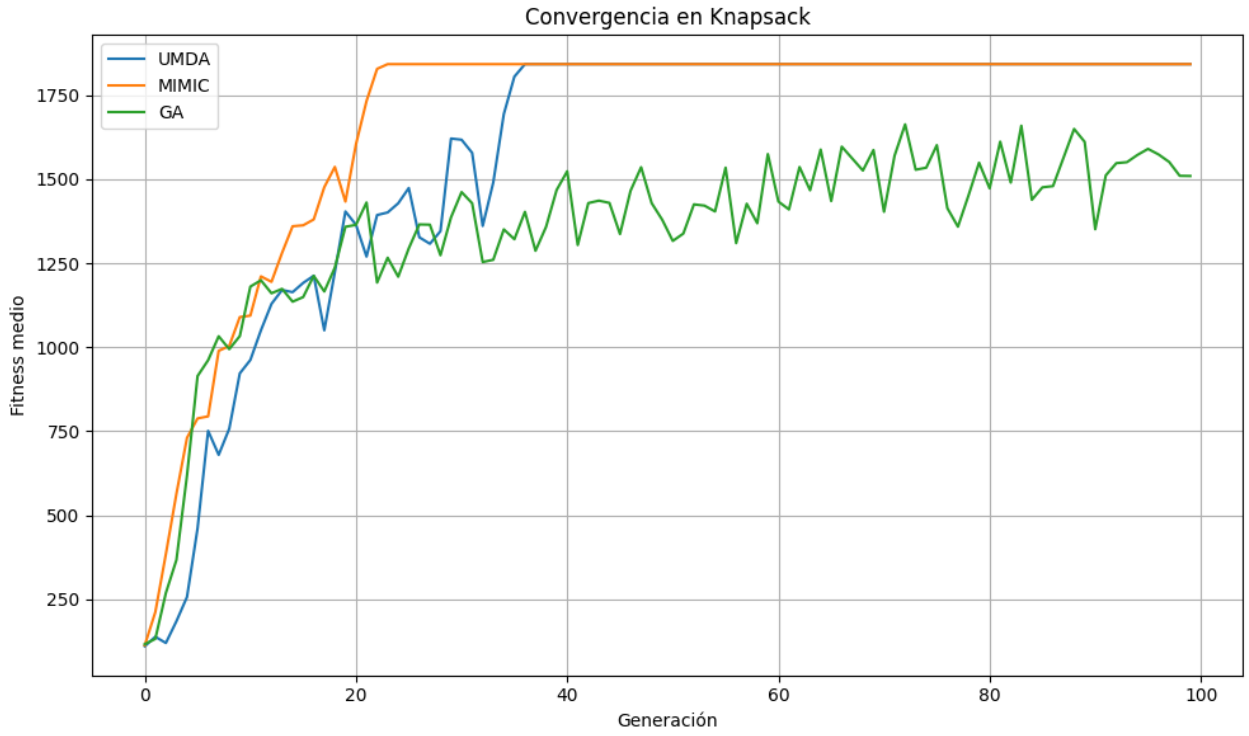

Figure 4: Knapsack convergence – mean fitness per generation

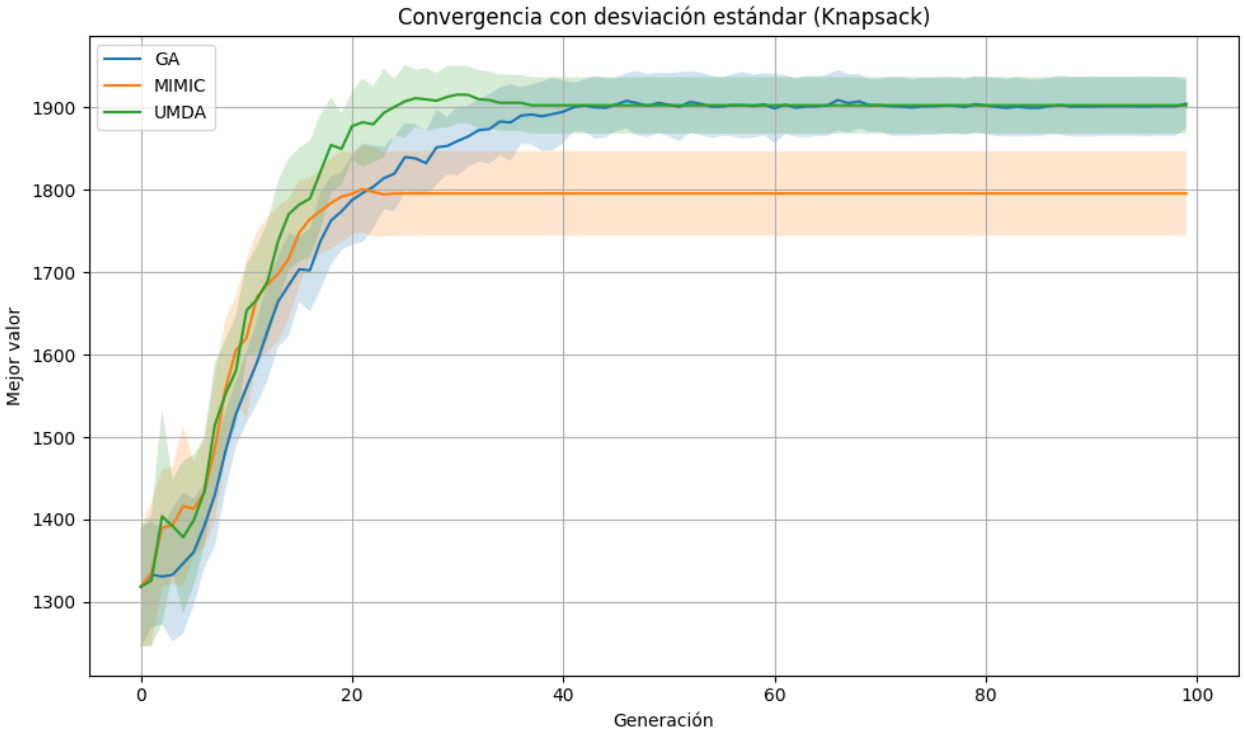

Figure 5: Convergence with standard deviation – best fitness variability