



Técnicas de Soft Computing para Aprendizaje y optimización. Redes
Neuronales y Metaheurísticas, programación evolutiva y bioinspirada
2024-2025

MASTER CIENCIA DE DATOS
UNIVERSIDAD DE GRANADA

Revisión de los Algoritmos Genéticos

MIGUEL GARCÍA LÓPEZ

Índice

1. Introducción	3
2. Contexto	3
2.1. Orígenes	3
2.2. Definición	4
2.3. Operadores principales	4
2.4. Selección	6
2.5. Cruce	6
2.6. Mutación	7
3. Informe bibliométrico	8
4. Hibridaciones	11
4.1. Hibridaciones colaborativas	11
4.1.1. Colaborativas <i>teamwork</i>	11
4.1.2. Colaborativas <i>relay</i>	11
4.2. Hibridaciones integrativas	12
4.2.1. Integrativas <i>teamwork</i>	12
4.2.2. Integrativas <i>relay</i>	12
5. Bibliografía	13

Índice de figuras

1. Número de artículos desde 2010 – 2024 relacionados con los algoritmos genéticos en <i>Scopus</i>	8
2. Publicaciones cerradas vs abiertas sobre GAs	9

3.	Instituciones que más publican sobre GAs	10
4.	Tipo de publicaciones sobre las GAs	10



1. Introducción

En esta revisión se va a proceder a explicar qué son los **algoritmos genéticos**, de dónde vienen y cómo surgen a lo largo del tiempo como una solución bio-inspirada para solucionar problemas de optimización con codificación binaria. Se explicará cómo estos algoritmos surgieron y fueron “evolucionando” con el tiempo, adaptándose a nuevos problemas y creando variaciones del algoritmo original.

Se recabará información a través del buscador de *Scopus* para realizar un análisis bibliométrico, con el que se analizará la producción científica y académica y se medirá el impacto, la calidad y la evolución de publicaciones, autores, revistas e instituciones en el área específica de los algoritmos genéticos.

Se explicará, además, el funcionamiento básico del algoritmo genético original y el de algunas de sus variantes e hibridaciones. Para ello se explicarán los operadores que lo conforman así como el *workflow* o procesos que sigue el algoritmo y cómo interactúan los operadores entre sí para hacer funcionar al algoritmo de optimización.

2. Contexto

Los algoritmos genéticos están inspirados en la selección natural y se emplean tanto en problemas de optimización con restricciones como en aquellos sin ellas. Esta metaheurística modifica de manera repetida una población de soluciones individuales, seleccionando soluciones “padre” que generarán la siguiente generación de soluciones en la siguiente iteración del algoritmo.

En su forma más básica, un algoritmo genético opera sobre una población de soluciones potenciales a un problema dado. Cada solución potencial, frecuentemente llamada individuo o cromosoma, está representada como una cadena de símbolos, que puede ser binaria, numérica o simbólica. [1].

2.1. Orígenes

Los algoritmos genéticos o **GAs** adquirieron popularidad en la década de 1970, especialmente en 1975 con la publicación del libro de John Holland [2]. Este tipo de metaheurísticas se diseñan tomando como inspiración la selección natural. Un conjunto de fenotipos (soluciones) evoluciona a lo largo de generaciones para emular el cruce entre especies, es decir, el cruce de soluciones mediante un intercambio común de cromosomas, lo que da lugar a nuevos individuos con características de ambos padres. A lo largo del tiempo, este tipo de algoritmos fueron incorporando nuevas características y, aunque inicialmente fueron concebidos para resolver problemas discretos, también existen versiones

que optimizan problemas continuos [3].

2.2. Definición

Los algoritmos genéticos definen las soluciones como vectores numéricos, en el caso de un problema binario (**GAs** originales) la solución es un vector de 0s y 1s, en el caso de un problema sobre un dominio continuo, la solución es un vector de números reales. Cada individuo $x \in P$ se representa como una cadena de genes:

$$x = (g_1, g_2, \dots, g_n)$$

donde $g_i \in \Sigma$ y Σ es el alfabeto de genes que puede ser:

- Binario: $\Sigma = \{0, 1\}$
- Entero: $\Sigma = \mathbb{Z}$
- Real: $\Sigma = \mathbb{R}$

El algoritmo trata de minimizar o maximizar una función objetivo, esta es conocida como la función *fitness*. En esta se representa una métrica de cómo de bien lo está haciendo el algoritmo (o el error) y se trata de optimizar para guiar la evolución de la población.

Se comienza con una población de soluciones totalmente aleatoria partiendo de una distribución (puede ser normal o uniforme). A partir de ahí se evalúan los individuos y se les proporciona un *score* o evaluación a cada uno. Con esa evaluación se ordenan y se eligen dos padres (o varios, dependiendo de la variante) mediante un proceso de selección, como el método de la ruleta.

Esos padres son cruzados entre sí mediante el operador de cruce para obtener uno o más hijos que compartan características de los padres. Dada una probabilidad definida como p se le aplicará el operador de mutación a los hijos para favorecer la diversidad y exploración.

A partir de ahí, se introducen los hijos en la población, si se pueden introducir, y se vuelve a iterar. La condición de parada puede ser llegar a un valor de *fitness*, condición de tiempo o condición de iteraciones máximas.

2.3. Operadores principales

Se procede a definir los operadores principales de los algoritmos genéticos, al menos los básicos o más usados, ya que cubrir todos los operadores de todas las variaciones es inabarcable.

Algorithm 1 Algoritmo Genético con Elitismo

Require: Tamaño de población N , tasa de cruce C , tasa de mutación M , porcentaje de elitismo E , condición de terminación

Ensure: La mejor solución encontrada

```

1: Inicializar población aleatoria  $P$  con  $N$  individuos
2: while no se cumpla la condición de terminación do
3:   Evaluar fitness de cada individuo en  $P$ 
4:   Ordenar  $P$  de mejor a peor según fitness
5:    $P_{\text{elite}} \leftarrow$  primeros  $E \times N$  individuos de  $P$ 
6:   Inicializar  $P_{\text{descendencia}} \leftarrow \emptyset$ 
7:   for  $i$  desde 1 hasta  $N$  do
8:     Seleccionar padres  $p_1, p_2$  de  $P$  usando selección por ruleta/torneo
9:     if  $\text{rand}() < C$  then
10:       $h \leftarrow \text{Cruce}(p_1, p_2)$ 
11:     else
12:       $h \leftarrow$  copia de  $p_1$  o  $p_2$  (aleatorio)
13:     endif
14:     if  $\text{rand}() < M$  then
15:       $h \leftarrow \text{Mutación}(h)$ 
16:     endif
17:     Agregar  $h$  a  $P_{\text{descendencia}}$ 
18:   end for
19:   Reemplazar los  $E \times N$  peores individuos en  $P_{\text{descendencia}}$  con  $P_{\text{elite}}$ 
20:    $P \leftarrow P_{\text{descendencia}}$ 
21: end while
22: return mejor individuo en  $P$ 

```

2.4. Selección

La selección se puede llevar a cabo por varios métodos. Uno de los más utilizados es **selección por torneo** (eq: 1). En la selección por torneo, los individuos compiten entre sí en grupos pequeños. Cada grupo (torneo) tiene varios competidores. El competidor con mejor *fitness* dentro del grupo tiene más probabilidades de ganar. Los ganadores son seleccionados para participar en el proceso de cruce. El proceso se repite hasta que se completa la *pool* de apareamiento con los ganadores de los torneos [4].

$$P(\text{parent}_i) = \frac{\text{fitness}(\text{parent}_i)}{\sum_{j=1}^N \text{fitness}(\text{parent}_j)} \quad (1)$$

2.5. Cruce

El operador de cruce (*crossover*) tiene como propósito principal explorar nuevas regiones del espacio de búsqueda combinando las características favorables de dos soluciones “padres” para generar nuevas soluciones “hijas” potencialmente mejores. Algunas versiones realizan cruce siempre, mientras que otras proponen una probabilidad para que dos soluciones puedan cruzarse, de forma que ocurra más esporádicamente.

One-point crossover (binario): Se elige al azar un punto en los cromosomas de ambos progenitores y se designa como “punto de cruce”. Los bits a la derecha de ese punto se intercambian entre los dos cromosomas parentales. El resultado son dos descendientes, cada uno con información genética de ambos progenitores [5]. Dados dos padres (los cromosomas) $P_1 = (p_1^1, p_2^1, \dots, p_n^1)$ y $P_2 = (p_1^2, p_2^2, \dots, p_n^2)$, se selecciona un punto de cruce k aleatorio. Los hijos H_1 y H_2 se generan como:

$$H_1 = (p_1^1, p_2^1, \dots, p_k^1, p_{k+1}^2, \dots, p_n^2) \quad (2)$$

$$H_2 = (p_1^2, p_2^2, \dots, p_k^2, p_{k+1}^1, \dots, p_n^1) \quad (3)$$

Blend crossover (continuo): Dado dos números reales para cada uno de los genes de los padres al hijo se le asignará un número aleatorio entre ese rango de gen para cada gen que conforme al vector cromosómico [6]. Dados dos padres $P_1 = (p_1^1, p_2^1, \dots, p_n^1)$ y $P_2 = (p_1^2, p_2^2, \dots, p_n^2)$, para cada gen i , se calcula un intervalo $[c_{\min}, c_{\max}]$, donde:

$$c_{\min} = \min(p_i^1, p_i^2) - \alpha \cdot d \quad (4)$$

$$c_{\max} = \max(p_i^1, p_i^2) + \alpha \cdot d \quad (5)$$

$$d = |p_i^1 - p_i^2| \quad (6)$$

El valor del gen i en el hijo H se genera aleatoriamente dentro del intervalo $[c_{\min}, c_{\max}]$:

$$H_i = \text{rand}(c_{\min}, c_{\max}) \quad (7)$$

Por supuesto existen muchos más tipos de cruces tanto para versiones en dominio continuo como en dominio discreto y binario, pero los principales operadores y que más suelen utilizarse son los descritos.

2.6. Mutación

El operador de mutación tiene como propósito principal favorecer la exploración del espacio de búsqueda introduciendo perturbaciones aleatorias en las soluciones hijo. Este operador es más delicado que el de cruce y por lo general se propone un valor bajo de probabilidad de mutación, ya que demasiadas perturbaciones pueden afectar a los resultados. Un desajuste entre el ratio de exploración y explotación a favor de la exploración puede dar como resultado soluciones con un *fitness* muy malo.

Mutación binaria: Este operador básico y clásico consiste en cambiar un *bit* arbitrario de un genotipo o solución de un algoritmo genético binario a su estado inverso dada una probabilidad de mutación [7].

$$x'_i = \begin{cases} 1 - x_i & \text{con probabilidad } p_m, \\ x_i & \text{con probabilidad } 1 - p_m. \end{cases} \quad (8)$$

Mutación continua: Este tipo de mutación se utiliza especialmente en problemas en los que se busca una exploración más amplia del espacio de búsqueda, ya que la distribución de *Cauchy* tiene colas pesadas y puede generar valores aleatorios alejados del centro con mayor probabilidad en comparación con una distribución normal. La mutación se define como:

$$m(x_i) = \begin{cases} (2r)^{\frac{1}{\eta+1}} - 1 & \text{si } p \leq 0,5, \\ 1 - (2r)^{\frac{1}{\eta+1}} & \text{si } p > 0,5, \end{cases} \quad (9)$$

donde:

- r es un número aleatorio en el intervalo $[0, 1]$,

- η es una variable de control que ajusta la distribución,
- p es una probabilidad aleatoria en $[0, 1]$.

Esta función de mutación está diseñada para generar una exploración más amplia del espacio de búsqueda, con la posibilidad de generar valores más alejados del centro de la distribución.

Por supuesto y tal como se mencionó en el apartado de *crossover*, existen multitud de propuestas en cuanto a operadores de mutación.

3. Informe bibliométrico

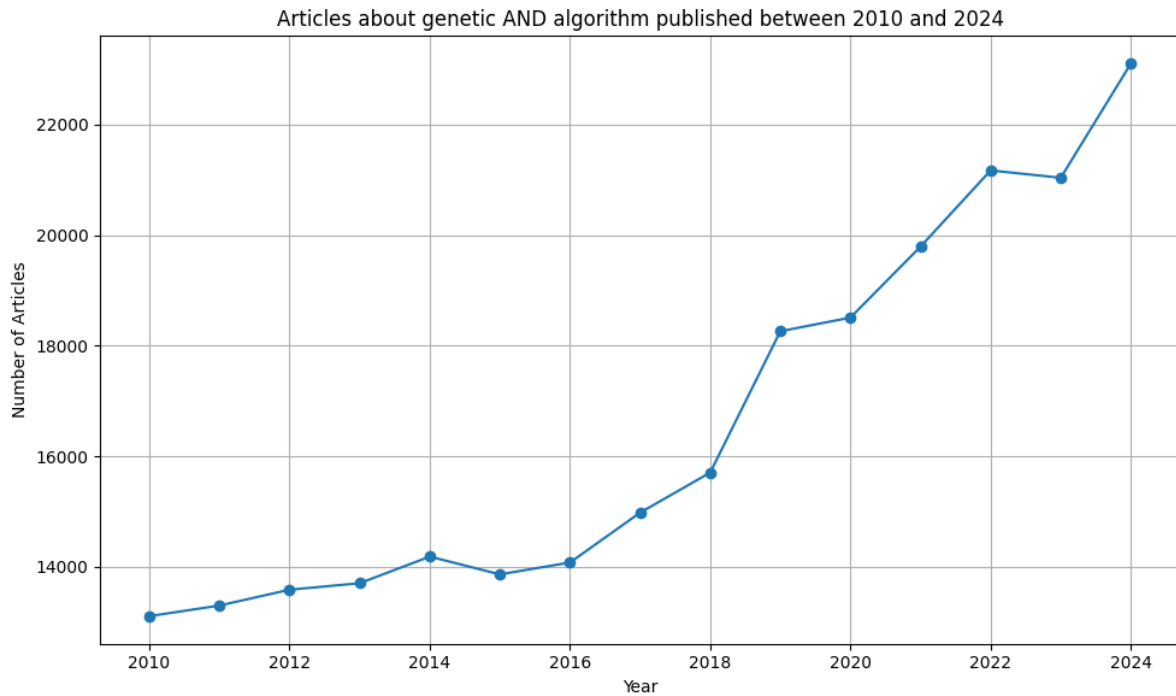


Figura 1: Número de artículos desde 2010 – 2024 relacionados con los algoritmos genéticos en *Scopus*.

Como puede verse en la gráfica de la figura 1, los algoritmos genéticos son, pese a su longeva existencia, cada vez más populares. El número de artículos por año crece y crece cada vez más sin ninguna tendencia evidente a bajar, lo que parece indicar que los algoritmos genéticos son más populares que nunca y se sigue innovando en este sector.

Estos datos han sido obtenidos mediante el buscador de *Scopus*. Utilizando una herramienta como *OpenAlex* se han obtenido datos complementarios. Por ejemplo, se puede observar que la gran mayoría de las publicaciones son (fig 2) cerradas (se accede mediante

pasarelas de pago o acceso institucional). Otros datos interesantes se pueden obtener de este portal, como cuáles son las instituciones más prolíficas (fig 3) en relación a los **GAs** y cuáles son los tipos de publicaciones más comunes (fig 4) (un 88 % son artículos).

Open Access vs Closed Access

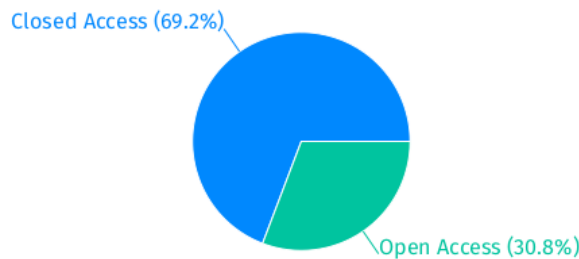


Figura 2: Publicaciones cerradas vs abiertas sobre **GAs**.

A partir de los artículos más citados en Scopus relacionados con Genetic Algorithms (**GAs**), los temas más candentes parecen estar relacionados con:

1. Optimización y mejora de algoritmos genéticos

- Revisión del estado del arte de los **GAs** y sus mejoras a lo largo del tiempo.
- Optimización de hiperparámetros en modelos de aprendizaje automático, clave para la implementación eficiente de **GAs**.

2. Metaheurísticas bioinspiradas y nuevas variantes

- Desarrollo de nuevos algoritmos de optimización como *Marine Predators Algorithm*, *Equilibrium Optimizer* y *Arithmetic Optimization Algorithm*.
- Técnicas para mejorar la convergencia y evitar estancamientos en óptimos locales.

3. Aplicaciones en biología y bioinformática

- Uso de **GAs** para predecir estructuras de proteínas con aprendizaje profundo.
- Aplicaciones en la clasificación evolutiva de sistemas *CRISPR-Cas*.

4. Optimización multiobjetivo y herramientas avanzadas

- Desarrollo de herramientas como *Pymoo*, una biblioteca en Python para optimización multiobjetivo con **GAs** y otras metaheurísticas.

Top 5 Contributing Institutions

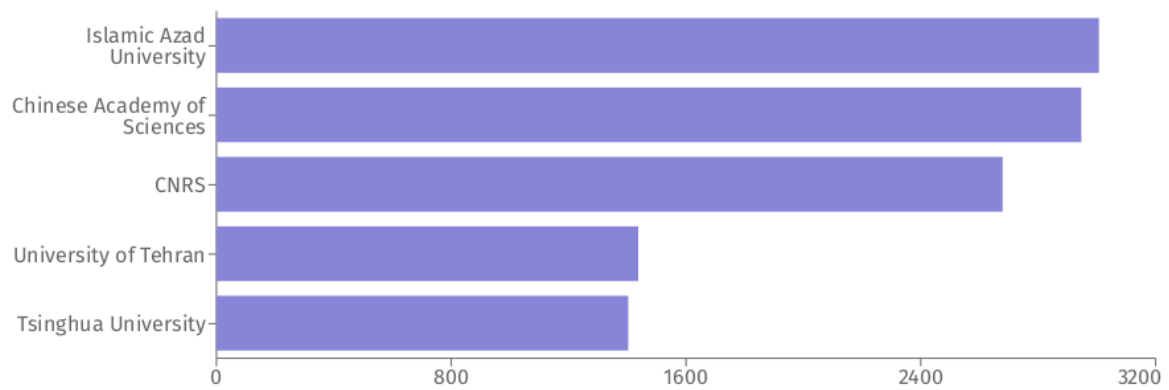


Figura 3: Instituciones que más publican sobre **GAs**.

Publication Types Distribution

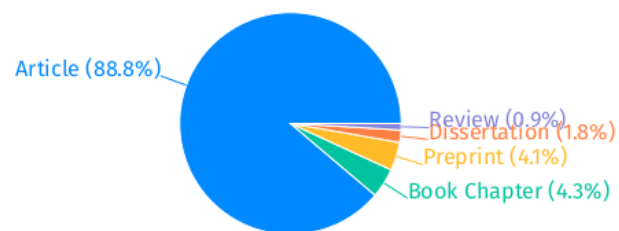


Figura 4: Tipo de publicaciones sobre las **GAs**.

4. Hibridaciones

Los algoritmos híbridos entre genéticos y otras técnicas de búsqueda son muy comunes y ampliamente utilizados. Los algoritmos genéticos se combinan con otros métodos de optimización, como búsqueda local o enfriamiento simulado, con el objetivo de mejorar la eficiencia y precisión.

El objetivo detrás de este tipo de algoritmos es utilizar a las **GA**s como método principal de búsqueda y exploración. Tras ir poco a poco convergiendo a una solución, se suelen utilizar métodos de búsqueda local o enfriamiento para mejorar y refinar la fase de explotación del algoritmo.

A continuación se detallan algunos de estos métodos [8] de hibridación y se explicarán las ventajas de cada uno y algunos ejemplos dentro de estos métodos de hibridación.

4.1. Hibridaciones colaborativas

Estas estrategias implican la interacción entre algoritmos independientes que intercambian información durante su ejecución.

4.1.1. Colaborativas *teamwork*

Varios algoritmos se ejecutan en paralelo, compartiendo información periódicamente.

1. Ejemplo 1: Los **GA** distribuidos (**DGAs**) dividen la población en subpoblaciones (islas), cada una procesada por un **GA** independiente. Las migraciones entre islas introducen diversidad. Por ejemplo, en [9], se utilizan subpoblaciones con distintos parámetros (alta mutación para exploración, baja para explotación).
2. Ejemplo 2: En [10], se combinan **GA**, búsqueda tabú y búsqueda local. Los **GA** generan soluciones en zonas no exploradas usando una memoria adaptativa que registra el historial de búsqueda de otros algoritmos.

4.1.2. Colaborativas *relay*

Los algoritmos se ejecutan secuencialmente en una canalización.

1. Ejemplo 1: Un **GA** global seguido de un **GA** local. El primero explora el espacio de búsqueda, y el segundo refina las mejores soluciones encontradas.

2. Ejemplo 2: Un **GA** se combina con el método Nelder-Mead: el **GA** realiza una exploración global, y Nelder-Mead mejora las soluciones prometedoras. El método de Nelder-Mead es un método numérico utilizado para encontrar el mínimo o el máximo de una función objetivo en un espacio multidimensional. Es un método de búsqueda directa (basado en la comparación de funciones) y suele aplicarse a problemas de optimización no lineal para los que pueden no conocerse las derivadas.

4.2. Hibridaciones integrativas

Un algoritmo actúa como componente de otro, integrándose en su funcionamiento interno.

4.2.1. Integrativas *teamwork*

Un metaheurística se incrusta dentro de otra como componente clave.

1. Ejemplo 1: Algoritmos meméticos [11], donde un **GA** se combina con búsqueda local.
2. Ejemplo 2: Micro-GA (μ **GA**) [12], usado como operador de refinamiento. Un **GA** con población pequeña (ej. 5 individuos) mejora soluciones específicas del **GA** principal, aprovechando su capacidad para seguir crestas en espacios complejos.

4.2.2. Integrativas *relay*

Un **GA** realiza funciones específicas dentro de otra metaheurística dominante.

1. Ejemplo 1: Un **GA** *steady-state* genera soluciones candidatas para un *simulated annealing* (**SA**). El **SA** decide aceptar o rechazar las soluciones, integrando exploración (**GA**) y explotación (**SA**).
2. Ejemplo 2: En [13], un μ **CHC** (variante de **GA** con alta presión selectiva) actúa como operador de perturbación en una búsqueda local iterada, introduciendo diversidad sin perder calidad.

Las hibridaciones de algoritmos genéticos representan un campo en constante evolución dentro de la computación evolutiva y la optimización metaheurística. El éxito demostrado por estas técnicas híbridas ha llevado a un creciente interés en desarrollar nuevas combinaciones y estrategias que puedan abordar problemas cada vez más complejos. La tendencia actual apunta hacia la integración con técnicas de aprendizaje profundo y otros métodos de inteligencia artificial.

5. Bibliografía

- [1] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262631857.
- [2] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd. Ann Arbor, MI: University of Michigan Press, 1975, second edition, 1992.
- [3] A. Eiben y J. Smith, *Introduction to Evolutionary Computing* (Natural Computing Series), 2nd. Berlin, Heidelberg: Springer, 2015, pág. 30, S2CID 20912932, ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8.
- [4] B. L. Miller, “Genetic Algorithms, Tournament Selection, and the Effects of Noise,” en,
- [5] Z. C. Dagdia y M. Mirchev, “Chapter 15 - When Evolutionary Computing Meets Astro- and Geoinformatics,” en *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, P. Škoda y F. Adam, eds., Elsevier, 2020, págs. 283-306, ISBN: 978-0-12-819154-5. DOI: 10.1016/B978-0-12-819154-5.00026-6. URL: <https://www.sciencedirect.com/science/article/pii/B9780128191545000266>.
- [6] Purdue University College of Engineering, *Lecture 4: Real-Coded Genetic Algorithms*, Lecture notes, Accessed on April 27, 2024. URL: <https://engineering.purdue.edu/~sudhoff/ee630/Lecture04.pdf>.
- [7] S. Mirjalili y S. Mirjalili, “Genetic algorithm,” *Evolutionary algorithms and neural networks: Theory and applications*, págs. 43-55, 2019.
- [8] C. García-Martínez, F. J. Rodríguez y M. Lozano, “Genetic Algorithms,” en *Handbook of Heuristics*, R. Martí, P. M. Pardalos y M. G. C. Resende, eds. Cham: Springer International Publishing, 2018, págs. 431-464, ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_28. URL: https://doi.org/10.1007/978-3-319-07124-4_28.
- [9] F. Herrera y M. Lozano, “Gradual distributed real-coded genetic algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 4, n.º 1, págs. 43-63, 2000. DOI: 10.1109/4235.843494.
- [10] E.-G. Talbi y V. Bachelet, “COSEARCH: A parallel cooperative metaheuristic,” *J. Math. Model. Algorithms*, vol. 5, págs. 5-22, abr. de 2006. DOI: 10.1007/s10852-005-9029-7.
- [11] P. Moscato y C. Cotta, “A Gentle Introduction to Memetic Algorithms,” en ene. de 2003, vol. 57, págs. 105-144, ISBN: 1-4020-7263-5. DOI: 10.1007/0-306-48056-5_5.
- [12] S. Kazarlis, S. Papadakis, J. Theoharis y V. Petridis, “Microgenetic algorithms as generalized hill-climbing operators for GA optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 5, n.º 3, págs. 204-217, 2001. DOI: 10.1109/4235.930311.
- [13] M. Lozano y C. García-Martínez, “Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report,” *Computers & Operations Research*, vol. 37, págs. 481-497, mar. de 2010. DOI: 10.1016/j.cor.2009.02.010.