

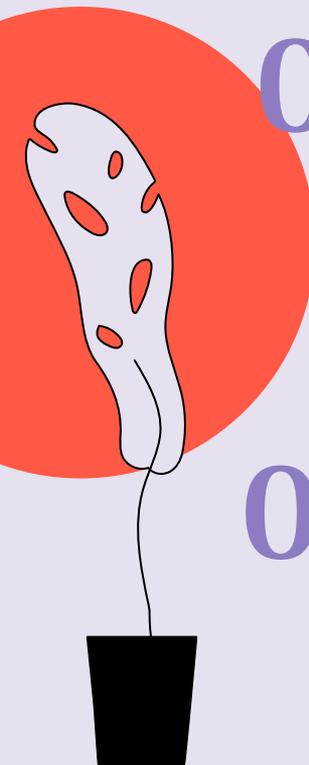
Métodos Numéricos 2

Algoritmos evolutivos y sus aplicaciones

Karina Valladares y Rodrigo Morales



Contenidos a tratar



01

Antecedentes

Conceptos básicos necesarios.

02

Algoritmos genéticos

Funcionamiento, aplicaciones y límites.

03

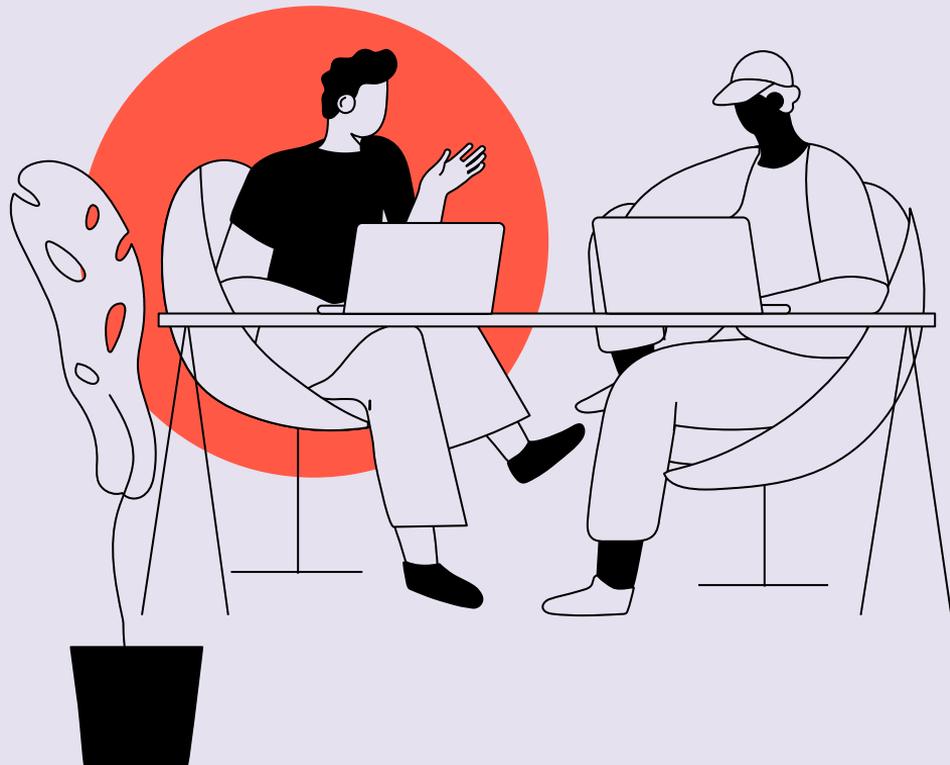
Evolución diferencial

Funcionamiento, aplicaciones y límites.

04

Ejemplos

Implementaciones de los algoritmos y sus resultados.



01

Antecedentes

Optimización y algoritmos de
optimización evolutivos



Optimización matemática

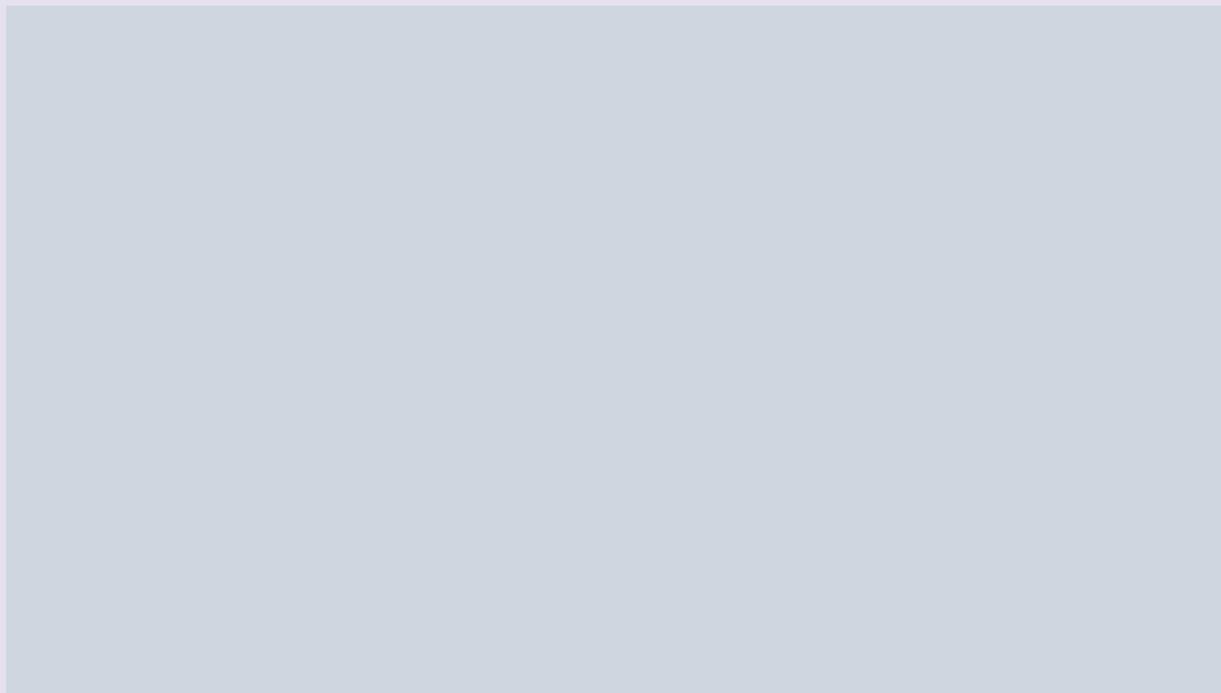
- La optimización matemática es la selección del mejor elemento, con respecto a algún criterio, de un conjunto de elementos posibles.
- En el caso más simple, un problema de optimización consiste en maximizar o minimizar una función real eligiendo sistemáticamente valores de entrada (tomados de un conjunto permitido) y computando el valor de la función.
- Un programa matemático es un problema de optimización de la forma

$$\text{Maximizar } f(x) : x \in X, g(x) \leq 0, h(x) \leq 0$$

en donde X es un subconjunto de \mathbb{R}^n y está en el dominio de las funciones f, g, h que se mapean en espacios reales. Las relaciones de g y h son llamadas **restricciones** y la función f es la **función objetivo**



Algoritmos Evolutivos



Algoritmos de optimización evolutivos

En su forma más común, la adaptación es un proceso biológico en el que los organismos evolucionan al reordenar su material genético para la supervivencia en ambientes que los retan. Dichas interacciones pueden ser retratadas por modelos matemáticos no lineales.

La idea central de estos es mantener un conjunto de individuos que representan a una posible solución del problema. Con dicho gremio, los elementos compiten entre sí y se mezclan. Gracias al principio de selección natural, se asegura de que el resultado final sea únicamente los mejor adaptados.



-
- En un algoritmo evolutivo, tras parametrizar el problema en una serie de variables, (x_1, x_2, \dots, x_n) , se codifican en una población de cromosomas. Sobre esta población se aplican uno o varios operadores genéticos y se fuerza una presión selectiva.
 - Un algoritmo evolutivo puede ser implementado de forma independiente del problema, o a lo sumo, con un conocimiento básico de este, lo cual los hace algoritmos robustos.
 - Son útiles para cualquier problema, pero a la vez débiles, pues no están especializados en ningún problema concreto siendo los genéticos empleados los que en gran parte confieren la especificidad al método empleado



Aplicaciones de los Algoritmos Evolutivos

Los algoritmos evolutivos tienen una alta demanda ya que sirven para optimización, planificación, diseño y problemas administrativos. Estos problemas se encuentran en todas las industrias. Si se analizan los resultados obtenidos de la base de datos de WoS, se observa que los algoritmos evolutivos son usados principalmente en:

- Ingeniería Eléctrica/Electrónica,
 - Inteligencia Artificial,
 - Métodos Teóricos en las ciencias de la computación,
 - Sistemas de Control de Automatización,
 - Sistemas de Informática
 - Investigación de Operaciones
- 

Area of applications	GA	GP	DE	ES	EP	ALL
Engineering electrical electronic	26,961	1364	3902	516	755	33,498
Computer science artificial intelligence	23983	3376	4294	655	634	32,942
Computer science theory methods	13,253	2460	2071	451	330	18,535
Computer science interdisciplinary applications	11,355	888	1669	216	177	14,305
Automation control systems	9217	437	1027	120	176	10,977
Computer science information systems	8318	673	996	136	170	10,293
Operations research management science	7397	282	791	106	74	8650
Engineering mechanical	6197	–	–	–	–	6197
Telecommunications	5850	–	725	–	98	6673
Energy fuels	5325	–	835	–	149	6309
Computer science software engineering	–	688	–	111	79	878
Mathematical computational biology	–	329	–	–	–	329
Engineering civil	–	304	–	–	–	304
Engineering multi-disciplinary	–	–	785	–	–	785
Physics applied	–	–	–	125	–	125
Mathematics applied	–	–	–	98	–	98

GA Genetic algorithm, *GP* genetic programming, *DE* differential evolution, *ES* evolution strategy, *EP* evolutionary programming, *ALL* sum of the papers for all listed algorithms in given area of applications

02

Algoritmos genéticos

Funcionamiento, aplicaciones,
desventajas y límites.



Orígenes de la teoría

A partir de los estudios realizados por John Henry Holland, en los años 1970, surgió una de las líneas con futuro brillante dentro de la rama de la inteligencia artificial.

Holland se inspiró en la evolución biológica y su base genética y molecular, dando origen a los algoritmos genéticos, cuya idea general es hacer evolucionar a una población de individuos sometiéndose a acciones aleatorias, como mutaciones y recombinaciones genéticas, así como también a una selección de acuerdo con criterios particulares



Conceptos cruciales



Fenotipo

Conjunto de posibles soluciones al problema



Cromosoma

Cadena que codifica la información de la solución



Generaciones

Evoluciones distintas de cada cromosoma



¿Cómo funciona?

- Dado un problema específico de optimización a resolver, el AG utiliza el fenotipo, codificadas de alguna manera y de una función de aptitud que permite evaluar cuantitativamente a cada candidata a solución.
- Estas candidatas se suelen generar aleatoriamente, o bien pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG depure las opciones válidas hasta escoger la mejor.
- Cada una de las soluciones potenciales es evaluada por la función de aptitud, una ecuación matemática, que le da una calificación para saber qué tan “buena” es con respecto a las demás soluciones.
- Las candidatas prometedoras se conservan y se les permite reproducirse.
- El proceso se repite hasta alcanzar uno de los criterios de paro.

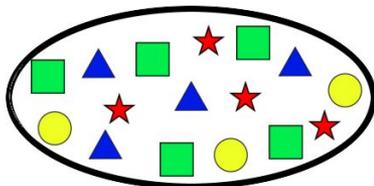


Criterios de paro para el algoritmo

Normalmente se usan dos criterios: correr el AG un número máximo de iteraciones (generaciones) o detenerlo cuando no haya cambios en la población. Mientras no se cumpla la condición de término se hace lo siguiente:

- **Selección:** después de evaluar la aptitud de cada cromosoma, se seleccionan los más aptos para ser cruzados en la siguiente generación.
 - **Cruzamiento:** opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.
 - **Mutación:** modifica al azar parte del cromosoma de los individuos.
 - **Reemplazo:** se seleccionan los mejores individuos para conformar la población de la generación siguiente.
- 

**SUSTITUCIÓN
DE
POBLACIÓN**

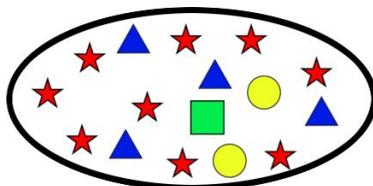


**POBLACIÓN INICIAL /
NUEVA GENERACIÓN**

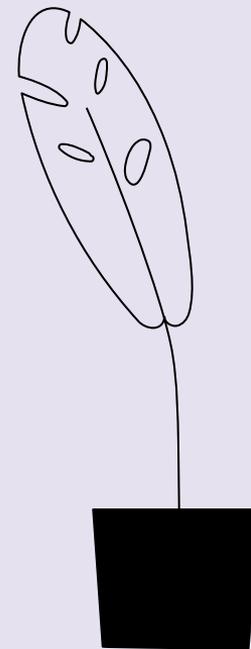
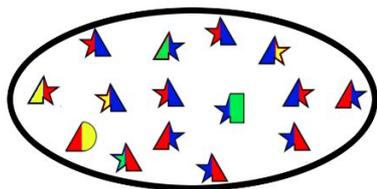
EVALUACIÓN

★ = 5 ▲ = 4
● = 3 ■ = 2

SELECCIÓN



**CRUZAMIENTO
MUTACIÓN**



1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---



1	0	1	1	1	0	1	0
---	---	---	---	---	---	---	---

Mutación

Papá

1	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

Mamá

0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

Hijo 1

1	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---

Hijo 2

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

Cruzamiento

Pseudocódigo

1. Determinar la función objetivo (OF)
2. Asignar el número de generación a 0 ($t = 0$)
3. Crear aleatoriamente individuos para la población inicial $P(t)$
4. Evaluar a los individuos en la población $P(t)$ utilizando la OF
5. Mientras no se cumpla el criterio de paro, hacer:
 - a. $T = t+1$
 - b. Seleccionar a los individuos de la población $P(t)$ de $P(t-1)$
 - c. Evaluar los individuos en $P(t)$ usando la OF
6. Regresar a los mejores individuos encontrados en las evoluciones



Desventajas y límites

- Para problemas de alta complejidad la función de evaluación puede tornarse demasiado costosa en términos de tiempo y recursos, por lo que el algoritmo se vuelve demasiado complejo y largo.
- Podría no llegar a converger en una solución óptima o bien terminar en una convergencia en uno de los óptimos locales o puntos arbitrarios cercanos a la solución óptima.
- No poseen una buena escalabilidad con la complejidad, por ejemplo para sistemas que están compuestos por muchas variables, componentes o elementos, el espacio de búsqueda crece de forma exponencial.
- No es recomendable utilizarlos para problemas que buscan respuesta a problemas que convergen en soluciones simples como sí y no

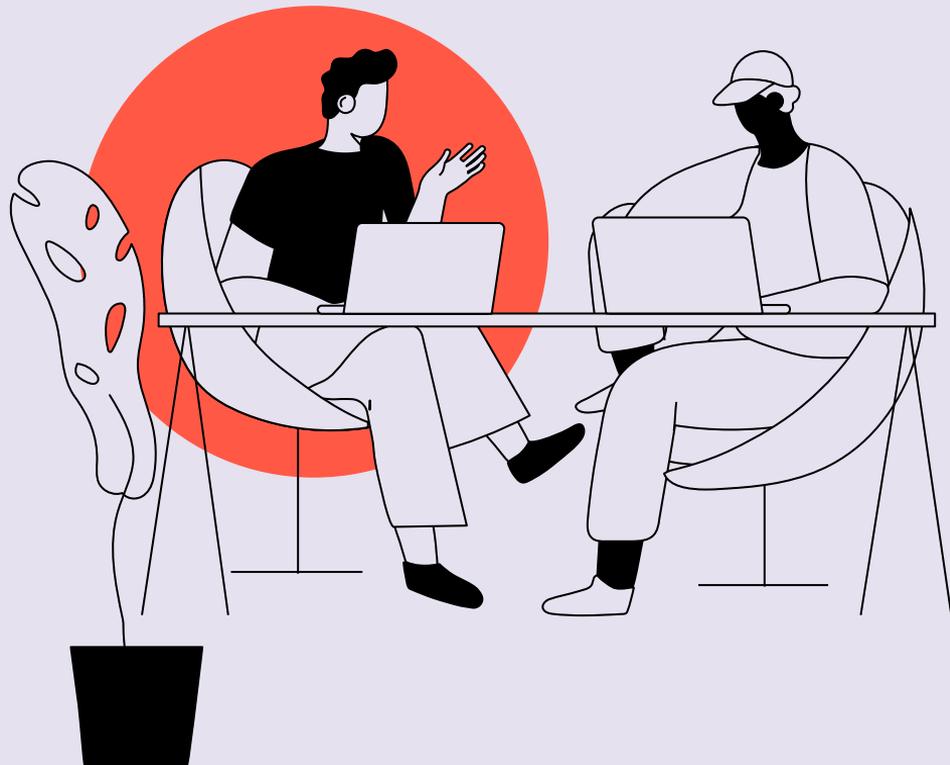


Aplicaciones



Rutas o caminos	Permiten encontrar la ruta más corta o más rápida
Finanzas	Permiten descubrir reglas de inversión que indican cuándo entrar y salir de un mercado
Robótica	Aprendizaje y comportamiento de robots





03

Evolución diferencial

Funcionamiento, aplicaciones,
desventajas y límites.



¿Qué es?

La *Evolución Diferencial* (DE, por sus siglas en inglés) es un tipo de algoritmo evolutivo que se usa principalmente en optimización de funciones en un espacio continuo. Se ha discutido también una versión del algoritmo para problemas de combinatorias.

El primer artículo sobre ED fue publicado por Kenneth Price y Rainer Storn en 1995 con el nombre:

Differential Evolution -- a simple and efficient adaptive scheme for global optimization over continuous spaces

El método fue concebido inicialmente para resolver el problema del ajuste del polinomio de Tchebychev. En estos días la Evolución Diferencial representa una de las corrientes principales en la investigación de la computación evolutiva.



Ventajas

Las principales ventajas de los DE sobre los algoritmos genéticos tradicionales son:

- Fáciles de usar.
- Utilización eficiente de la memoria.
- Baja complejidad computacional (En problemas grandes escala mejor).
- Menos esfuerzo computacional (Convergencia más rápida).



Algoritmo

La idea principal de la Evolución Diferencial es calcular la diferencia entre dos individuos escogidos aleatoriamente de la población. La DE determina el gradiente de una función dentro de un área dada y no solo en un punto, por lo cual previene que la solución se quede en un extremo local. Veinte años de investigación han dado como resultado un algoritmo estándar.

Lo primero es que el algoritmo asume que las variables del problema están incluidas en cada entrada de un vector de número reales. Entonces, la longitud n del vector representa la cantidad de variables del problema. Además, sea **NP** el número de padres que componen a la población.



Definamos el vector x_i^g donde i es el índice del individuo dentro de la población con $i \in \{1, \dots, NP\}$ y g es el número de generación. Además, cada vector tiene múltiples variables por lo que usamos m como el índice de la variable del vector correspondiente con $m \in \{1, \dots, n\}$. Entonces, cada variable puede ser descrita así: $x_{i,m}^g$.

Es importante recordar que el dominio de las variables está acotado, puede ser con los valores mínimos y máximos de todos los posibles valores.



Fase 1: Inicialización

Inicialización: La primera generación se genera aleatoriamente, considerando los valores mínimos y máximos previamente definidos:

$$x_{i,m}^g = x_m^{min} + (x_m^{max} - x_m^{min}) \cdot \epsilon \sim U(0, 1)$$



Fase 2: Mutación

Mutación: Construimos NP *noisy random vectors*, los cuales son a partir de 3 vectores distintos elegidos aleatoriamente, llamados *target vectors*, t_a, t_b, t_c . Los *noisy vectors* se calculan así:

$$n_i^g = t_c + F \cdot (t_a - t_b)$$

$F \in [0, 2]$ es un parámetro de la tasa de mutación.



Fase 3: Recombinación

Recombinación: Ya con los *noisy random vectors* calculados, la recombinación se hace de manera aleatoria al comparar con los vectores orginales. Con esto se busca los *trial vectors*. Cuyas variables se calculan de la siguiente manera:

$$Tr_{i,m}^g = \begin{cases} n_{i,m}^g & \text{si } \epsilon \sim U(0, 1) < CR \\ n_{i,m}^g & \text{de lo contrario} \end{cases}$$

El parámetro CR controla la tasa de recombinación. Lo importante es notar que los vectores de prueba son una combinación entre los orginales y los *noisy vectors* ya que se calculan variable a variable.



Fase 4: Selección

Selección: Por último, realizamos la selección comparando los vectores originales con los de prueba. Por lo que los vectores de las siguientes generaciones son los que tengan mejor valor de la función *fitness*:

$$x_i^{g+1} = \begin{cases} t_i^g & \text{si } fit(t_i^g) > fit(x_i^g) \\ x_i^g & \text{de lo contrario} \end{cases}$$

Este proceso se realiza hasta alcanzar o un valor de la función *fitness* deseado o un máximo de generaciones.



Pseudocódigo

Algorithm 2 Differential Evolution

```
1: determine objective function (OF)
2: assign number of generation to 0 ( $t=0$ )
3: randomly create individuals in initial population  $P(t)$ 
4: while termination criterion is not satisfied do
5:    $t=t+1$ 
6:   for each  $i$ -th individual in the population  $P(t)$  do
7:     randomly generate three integer numbers:
8:      $r_1, r_2, r_3 \in [1; \text{population size}]$ , where  $r_1 \neq r_2 \neq r_3 \neq i$ 
9:     for each  $j$ -th gene in  $i$ -th individual ( $j \in [1; n]$ ) do
10:       $v_{i,j} = x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j})$ 
11:      randomly generate one real number  $rand_j \in$ 
12:       $[0; 1)$ 
13:      if  $rand_j < CR$  then  $u_{i,j} := v_{i,j}$ 
14:      else
15:         $u_{i,j} := x_{i,j}$ 
16:      end if
17:    end for
18:    if individual  $u_i$  is better than individual  $x_i$  then
19:      replace individual  $x_i$  by child  $u_i$  individual
20:    end if
21:  end for
22: end while
23: return the best individual in population  $P(t)$ 
```

Aplicaciones en la vida real

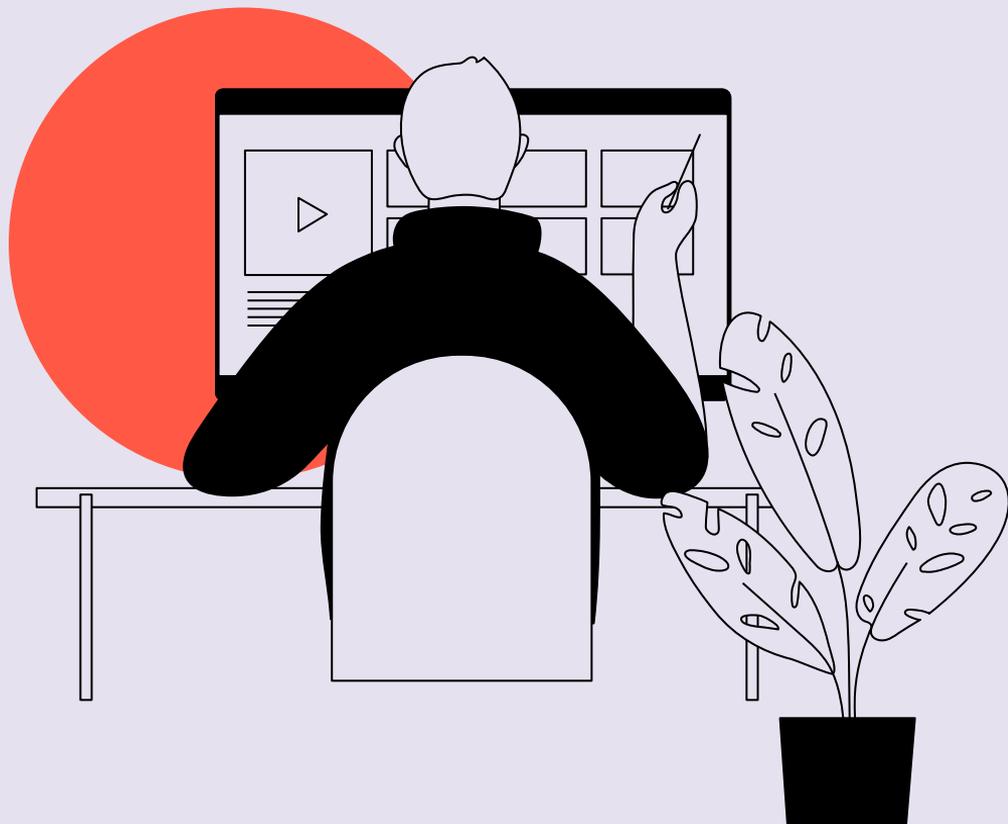
- Por ejemplo, se presenta un aplicación multiobjetivo para encontrar el tamaño óptimo de una celda fotovoltaica con almacenamiento de batería.
- Se usa un algoritmo multiobjetivo para optimizar una novedosa combinación de refrigeración, calefacción y sistema de almacenamiento de energía de aire comprimido basado en energía.
- El algoritmo DE se aplica para la optimización del diseño de parques eólicos con el objetivo de maximizar la potencia de salida.



04

Ejemplos

Implementaciones de un algoritmo genético y de una evolución diferencial.



Ejemplo GA

En ejemplo Maximizamos tres funciones de dos variables con restricciones sobre el dominio.

La primera función fue:

$$f(x, y) = 15x + 30y + 4xy - 2x^2 - 4y^2, \text{ con } x + 2y \leq 30, x \geq 0, y \geq 0$$

La segunda función:

$$f(x, y) = 3x + 5y, \text{ con } 3x + 2y \leq 18, x \geq 0, y \geq 0$$

Por último:

$$f(x, y) = 5x - x^2 + 8y - 2y^2, \text{ con } x + 2y \leq 6, x \geq 0, y \geq 0$$

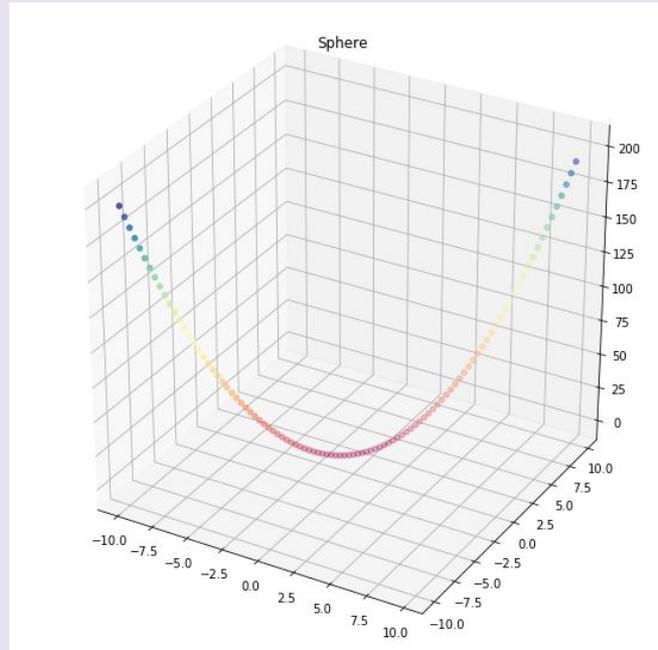
Para cada función se hicieron tres intentos para comparar las diferencias.



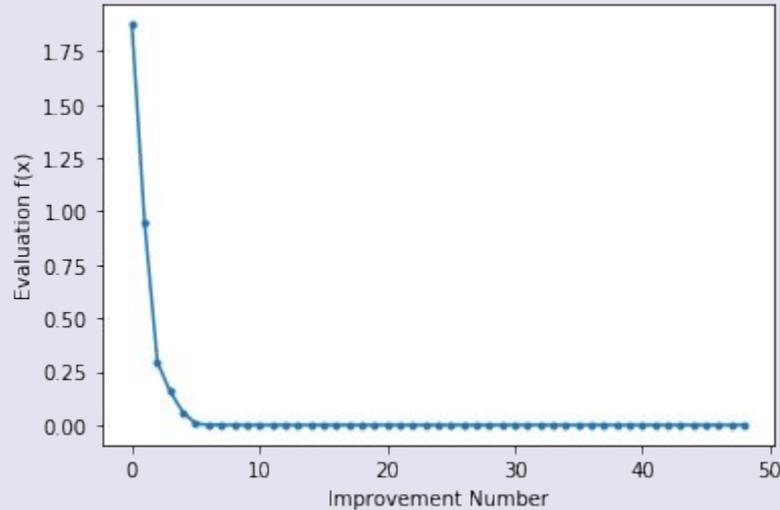
Maximización		Intento 1	Intento 2	Intento 3
Función 1	Solución	(11.8, 9.1)	(11.1, 9.3)	(12.5, 8.7)
	Solución Teórica	(12, 9)	(12, 9)	(12, 9)
	Resultado	269.8	266.04	268.24
	Resultado Teórica	270	270	270
	Error	-0.07%	-1.47%	-0.65%
Función 2	Solución	(0.1, 8.7)	(2.3, 5.5)	(5, 1.5)
	Solución Teórica	(0,9)	(0,9)	(0,9)
	Resultado	43.8	34.4	22.5
	Resultado Teórica	45	45	45
	Error	-2.67%	-23.56%	-50.00%
Función 3	Solución	(1, 1.5)	(0.7, 1.9)	(0.5, 2)
	Solución Teórica	(7/3, 11/6)	(7/3, 11/6)	(7/3, 11/6)
	Resultado	11.5	10.99	10.25
	Resultado Teórica	14.16	14.16	14.16
	Error	-18.79%	-22.39%	-27.61%

Ejemplo DE

Para este caso, se hizo un ejemplo de optimización con una función cuadrática de dos variables $x^2 + y^2$:



Se usaron los siguientes parámetros: **NP** = 10, el espacio de búsqueda o dominio de las variables es $[-5,5] \times [-5,5]$. El máximo de generaciones que se harán son 100, la tasa de mutación $F = 0.5$ y la tasa de recombinación **CR** = 0.7



Referencias consultadas

García Gutiérrez, J. (s. f.). *Análisis e implementación de algoritmos evolutivos para la optimización de simulaciones en ingeniería*. Escuela Universitaria Politécnica.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge.

Slowik, A. y Kwasnicka, H. (2020). *Evolutionary algorithms and their applications to engineering problems*. *Neural Computing and Applications* (2020) 32:12363–12379
[https://doi.org/10.1007/s00521-020-04832-8\(0123456789\(\),-volV\)\(0123456789,-\(\).volV\)](https://doi.org/10.1007/s00521-020-04832-8(0123456789(),-volV)(0123456789,-().volV))



Demostración en vivo de Python



Gracias

