

Practica 3. Algoritmo Evolutivos: Estrategias Evolutivas y Evolución Diferencial.

Algoritmos Bioinspirados

Diego Castillo Reyes
Marthon Leobardo Yañez Martinez
Aldo Escamilla Resendiz

28 de abril de 2024

Índice general

1. Introducción

Los Algoritmos Evolutivos son una familia de métodos de optimización inspirados en el proceso de selección natural y evolución biológica. Dos enfoques destacados dentro de esta familia son las **Estrategias Evolutivas** y la **Evolución Diferencial**, cada uno con sus particularidades y aplicaciones.

Estrategias Evolutivas (ES)

Las Estrategias Evolutivas son técnicas que se centran principalmente en la optimización numérica continua. En estas estrategias, la población de soluciones evoluciona a través de la mutación y la selección. La mutación se realiza generalmente mediante la adición de ruido normalmente distribuido a los vectores de parámetros, lo que permite una exploración efectiva del espacio de búsqueda. Una característica distintiva de las ES es el uso de mecanismos de autoadaptación para ajustar los parámetros de la mutación, como la tasa de mutación, basándose en el éxito de las generaciones anteriores.

Evolución Diferencial (DE)

La Evolución Diferencial es otro método robusto para optimizar problemas de optimización numérica. Se caracteriza por su sencillez y eficacia, especialmente en problemas multimodales (con múltiples óptimos locales). En DE, la nueva generación se crea añadiendo la diferencia ponderada entre dos o más soluciones de la población actual a una tercera solución. Este método se basa en operadores simples como la mutación diferencial, la recombinación y la selección. La mutación diferencial es particularmente útil para mantener la diversidad genética dentro de la población, lo que ayuda a explorar de manera efectiva el espacio de búsqueda.

Ambos métodos, aunque similares en su inspiración evolutiva, difieren en sus mecanismos de mutación y adaptación, lo que los hace adecuados para diferentes tipos de problemas de optimización. La elección entre Estrategias Evolutivas y Evolución Diferencial a menudo depende del problema específico, la naturaleza del espacio de búsqueda y las preferencias del investigador o ingeniero.

2. Desarrollo

Para el desarrollo de esta práctica se programaron los siguientes códigos.

2.1 Estrategia Evolutiva (μ, λ)

Usa una estrategia evolutiva en la que solo los descendientes (no los padres) son considerados para la generación siguiente, lo que puede ayudar a evitar la convergencia prematura hacia mínimos locales subóptimos.

```
1 # La estrategia evolutiva de este programa es ( , ) donde solo los descendientes son
   considerados para la siguiente generacion.
2 from matplotlib import pyplot as plt
3 import numpy as np
4 from functools import reduce
5 from operator import mul
6
7 # Funciones a optimizar
8 def rosenbrock(x):
9     result = 0
10    for i in range(len(x) - 1):
11        result += 100 * ((x[i] ** 2) - x[i + 1]) ** 2 + (1 - x[i]) ** 2
12    return result
13
14 def ackley(x):
15    summation1 = sum(xi**2 for xi in x)
16    summation2 = sum(np.cos(2 * np.pi * xi) for xi in x)
17    exp1 = np.exp(-0.2 * np.sqrt((1 / len(x)) * summation1))
18    exp2 = np.exp((1 / len(x)) * summation2)
19    result = (- 20) * exp1 - exp2 + np.e + 20
20    return result
21
22 def griewank(x):
23    summation = sum(xi**2 for xi in x)
24    producer = reduce(mul, np.cos(x / np.sqrt(np.arange(1, len(x) + 1))))
25    result = (1 / 4000) * summation - producer + 1
26    return result
27
```

```

28 def rastrigin(x):
29     summation = sum(xi**2 - 10 * np.cos(2 * np.pi * xi) for xi in x)
30     result = summation + 10 * len(x)
31     return result
32
33
34 # Estrategias evolutivas
35 def randSolution(interval, dimension):
36     limInf, limSup = interval
37     solution = np.round(np.random.uniform(limInf, limSup, dimension), 2)
38     return solution
39
40 def clip(x, interval): # revisa los limites de las variables de decision
41     limInf, limSup = interval
42     return max(min(x, limSup), limInf)
43
44 def mutation(sigmaT, x, interval):
45     newX = []
46     for i in range(len(x)):
47         newX.append(clip(np.round(x[i] + sigmaT * np.random.normal(0, 1), 2), interval))
48     return newX
49
50 def crossover(parents, dimension):
51     newX = []
52     for i in range(dimension):
53         index = np.random.randint(0, len(parents))
54         newX.append(parents[index][i])
55     return newX
56
57
58 def estrategiaEvolutiva(Gmax, dimension, interval, fun, mu, lamb, c, sigma):
59     x = randSolution(interval, dimension) #solucion base/solución inicial (la podria tomar del
60     conocimiento del problema)
61     fx = fun(x)
62     print('SOLUCION INICIAL', x, fx)
63     bestSolution = []
64     sigmas = []
65     successes = 0
66     ps = 0
67
68     for gen in range(Gmax): # Numero maximo de generaciones
69         if gen == 0:
70             # Crea padres
71             parents = []
72             for _ in range(mu):
73                 individual = mutation(sigma, x, interval)
74                 fitness = np.round(fun(individual), 4)
75                 parents.append([individual, fitness])
76             print(f"Padres: {parents}")
77         else:
78             # Selecciona los nuevos padres a partir de los hijos generados en la generación
79             anterior
80             # Ordena los hijos por su fitness y selecciona los mejores para ser padres
81             parents = sorted(offspring, key=lambda child: child[1])[:mu]
82
83             # Crear hijos a partir de los padres
84             offspring = []
85             for _ in range(lamb):
86                 # Seleccionar aleatoriamente a dos padres para el cruce
87                 # Generar indices aleatorios
88                 parent_indices = np.random.choice(len(parents), 2, replace=False)
89                 # Usar indices para obtener los individuos
90                 p1, p2 = parents[parent_indices[0]][0], parents[parent_indices[1]][0]
91                 child = crossover([p1, p2], dimension)
92                 child = mutation(sigma, child, interval)
93                 offspring.append([child, np.round(fun(child), 4)])
94
95             # Actualizar la mejor solución
96             if offspring[-1][1] < fun(x):
97                 x = offspring[-1][0]

```

```

97         successes += 1
98
99         print(f"Generacion {gen} Descendientes:\n{offspring}")
100
101         bestSolution.append(fun(x))
102         sigmas.append(sigma)
103
104         #Actualizar ps: frecuencia relativa de mutaciones exitosas.
105         if gen % (10 * dimension) == 0: # calcula la proporción de éxito cada 10*n generaciones
106             ps = successes / (gen + 1)
107             #if gen%n == 0: # n mas grande ps se mantiene mas generaciones
108             if ps > 1/5:
109                 sigma = sigma / c # no hay tantos éxitos por lo tanto explora regiones con tamaños
de paso más grande
110             elif ps < 1/5:
111                 sigma = sigma * c # encuentra región prometedora por lo tanto refina la solución
actual(explotacion)
112             elif ps == 1/5: #caso contrario sigma queda con el mismo valor
113                 sigma = sigma
114
115         return bestSolution, sigmas
116
117 Gmax = 1000
118 np.random.seed(123)
119 dimension = 10
120 fun = rosenbrock
121
122 interval = (-2.048, 2.048) if fun == rosenbrock else \
123             (-32.768, 32.768) if fun == ackley else \
124             (-600, 600) if fun == griewank else \
125             (-5.12, 5.12) if fun == rastrigin else None
126
127 mu = 20
128 lamb = 30 # Debe ser mayor que mu
129 sigma = 0.5 if fun == rosenbrock else \
130           2.0 if fun == ackley else \
131           20 if fun == griewank else \
132           1.0 if fun == rastrigin else None
133 c = 0.817
134
135 best, sigmas = estrategiaEvolutiva(Gmax, dimension, interval, fun, mu, lamb, c, sigma)
136
137 print('BEST', best)
138 print('SIGMAS', sigmas)
139 plt.plot(range(0, len(best)), best, color = 'green', label='mejores')
140 plt.legend()
141 plt.show()
142 plt.plot(range(0, len(sigmas)), sigmas, label='sigmas')
143 plt.legend()
144
145 plt.show()

```

Este código es una implementación directa de una estrategia evolutiva que facilita la optimización en espacios de búsqueda complejos mediante la adaptación continua de los parámetros y el uso de operadores genéticos como la mutación y el crossover.

2.2 Estrategia Evolutiva ($\mu + \lambda$)

```

1  # La estrategia evolutiva de este programa es (  $\mu + \lambda$  ) donde solo los descendientes son
considerados para la siguiente generacion.
2  from matplotlib import pyplot as plt
3  import numpy as np
4  from functools import reduce
5  from operator import mul
6
7  # Funciones a optimizar
8  def rosenbrock(x):
9      result = 0
10     for i in range(len(x) - 1):
11         result += 100 * ((x[i] ** 2) - x[i + 1]) ** 2 + (1 - x[i]) ** 2

```

```

12     return result
13
14 def ackley(x):
15     summation1 = sum(xi**2 for xi in x)
16     summation2 = sum(np.cos(2 * np.pi * xi) for xi in x)
17     exp1 = np.exp(-0.2 * np.sqrt((1 / len(x)) * summation1))
18     exp2 = np.exp((1 / len(x)) * summation2)
19     result = (- 20) * exp1 - exp2 + np.e + 20
20     return result
21
22 def griewank(x):
23     summation = sum(xi**2 for xi in x)
24     producer = reduce(mul, np.cos(x / np.sqrt(np.arange(1, len(x) + 1))))
25     result = (1 / 4000) * summation - producer + 1
26     return result
27
28 def rastrigin(x):
29     summation = sum(xi**2 - 10 * np.cos(2 * np.pi * xi) for xi in x)
30     result = summation + 10 * len(x)
31     return result
32
33
34 # Estrategias evolutivas
35 def randSolution(interval, dimension):
36     limInf, limSup = interval
37     solution = np.round(np.random.uniform(limInf, limSup, dimension), 2)
38     return solution
39
40 def clip(x, interval): # revisa los limites de las variables de decision
41     limInf, limSup = interval
42     return max(min(x, limSup), limInf)
43
44 def mutation(sigmaT, x, interval):
45     newX = []
46     for i in range(len(x)):
47         newX.append(clip(np.round(x[i] + sigmaT * np.random.normal(0, 1), 2), interval))
48     return newX
49
50 def crossover(parents, dimension):
51     newX = []
52     for i in range(dimension):
53         index = np.random.randint(0, len(parents))
54         newX.append(parents[index][i])
55     return newX
56
57
58 def estrategiaEvolutiva(Gmax, dimension, interval, fun, mu, lamb, c, sigma):
59     x = randSolution(interval, dimension) #solucion base/solución inicial (la podria tomar del
60     conocimiento del problema)
61     fx = fun(x)
62     print('SOLUCION INICIAL', x, fx)
63     bestSolution = []
64     sigmas = []
65     successes = 0
66     ps = 0
67
68     for gen in range(Gmax): # Numero maximo de generaciones
69         if gen == 0:
70             # Crea padres
71             parents = []
72             for _ in range(mu):
73                 individual = mutation(sigma, x, interval)
74                 fitness = np.round(fun(individual), 4)
75                 parents.append([individual, fitness])
76             print(f"Padres: {parents}")
77         else:
78             # Selecciona los nuevos padres a partir de los mejores individuos en la generación
79             anterior
80             # Ordena los individuos por su fitness y selecciona los mejores para ser padres
81             parents = sorted(population, key=lambda child: child[1])[:mu]

```

```

81
82 # Crear hijos a partir de los padres
83 offspring = []
84 for _ in range(lamb):
85     # Seleccionar aleatoriamente a dos padres para el cruce
86     # Generar índices aleatorios
87     parent_indices = np.random.choice(len(parents), 2, replace=False)
88     # Usar índices para obtener los individuos
89     p1, p2 = parents[parent_indices[0]][0], parents[parent_indices[1]][0]
90     child = crossover([p1, p2], dimension)
91     child = mutation(sigma, child, interval)
92     offspring.append([child, np.round(fun(child), 4)])
93
94 population = parents + offspring
95 for _ in range(len(population)):
96     # Actualizar la mejor solución
97     if population[-1][1] < fun(x):
98         x = population[-1][0]
99         successes += 1
100
101 print(f"Generacion {gen} Descendientes:\n{offspring}")
102
103 bestSolution.append(fun(x))
104 sigmas.append(sigma)
105
106 #Actualizar ps: frecuencia relativa de mutaciones exitosas.
107 if gen % (10 * dimension) == 0: # calcula la proporción de éxito cada 10*n generaciones
108     ps = successes / (gen + 1)
109     #if gen%n == 0: # n mas grande ps se mantiene mas generaciones
110     if ps > 1/5:
111         sigma = sigma / c # no hay tantos éxitos por lo tanto explora regiones con tamaños
112         de paso más grande
113     elif ps < 1/5:
114         sigma = sigma * c # encuentra región prometedora por lo tanto refina la solución
115     elif ps == 1/5: # caso contrario sigma queda con el mismo valor
116         sigma = sigma
117     return bestSolution, sigmas
118
119 Gmax = 1000
120 np.random.seed(21)
121 dimension = 10
122 fun = rosenbrock
123
124 interval = (-2.048, 2.048) if fun == rosenbrock else \
125             (-32.768, 32.768) if fun == ackley else \
126             (-600, 600) if fun == griewank else \
127             (-5.12, 5.12) if fun == rastrigin else None
128
129 mu = 20
130 lamb = 30
131 sigma = 0.5 if fun == rosenbrock else \
132         2.0 if fun == ackley else \
133         20 if fun == griewank else \
134         1.0 if fun == rastrigin else None
135 c = 0.817
136
137 best, sigmas = estrategiaEvolutiva(Gmax, dimension, interval, fun, mu, lamb, c, sigma)
138
139 print('BEST', best)
140 print('SIGMAS', sigmas)
141 plt.plot(range(0, len(best)), best, color = 'green', label='mejores')
142 plt.legend()
143 plt.show()
144 plt.plot(range(0, len(sigmas)), sigmas, label='sigmas')
145 plt.legend()
146
147 plt.show()

```

2.3 Evolución Diferencial

```
1 import numpy as np
2 from functools import reduce
3 from operator import mul
4 import os
5 # import estrategiaEvolutiva1 as ee1
6
7 def clear_screen():
8     # Comprueba si el sistema operativo es Windows
9     if os.name == 'nt':
10         os.system('cls') # cls es el comando para limpiar la consola en Windows
11     else:
12         os.system('clear') # clear es el comando para limpiar la consola en Unix/Linux
13
14 # Funciones a optimizar
15 def rosenbrock(x):
16     result = 0
17     for i in range(len(x) - 1):
18         result += 100 * ((x[i] ** 2) - x[i + 1]) ** 2 + (1 - x[i]) ** 2
19     return result
20
21 def ackley(x):
22     summation1 = sum(xi**2 for xi in x)
23     summation2 = sum(np.cos(2 * np.pi * xi) for xi in x)
24     exp1 = np.exp(-0.2 * np.sqrt((1 / len(x)) * summation1))
25     exp2 = np.exp((1 / len(x)) * summation2)
26     result = (- 20) * exp1 - exp2 + np.e + 20
27     return result
28
29 def griewank(x):
30     summation = sum(xi**2 for xi in x)
31     producer = reduce(mul, np.cos(x / np.sqrt(np.arange(1, len(x) + 1))))
32     result = (1 / 4000) * summation - producer + 1
33     return result
34
35 def rastrigin(x):
36     summation = sum(xi**2 - 10 * np.cos(2 * np.pi * xi) for xi in x)
37     result = summation + 10 * len(x)
38     return result
39
40
41 # Definir el algoritmo de Evolución Diferencial
42 def differential_evolution(strategy, objective_function, dimensions, population_size, F, R, Gmax,
43     L, U):
44     # Ejemplo para 'rand/1/bin':
45     if strategy == 'rand/1/bin':
46         X, best = rand_1_bin(objective_function, dimensions, population_size, F, R, Gmax, L, U)
47         print(X)
48         print(f"Best: {best}")
49
50     # Ejemplo para 'rand/1/exp':
51     elif strategy == 'rand/1/exp':
52         X, best = rand_1_exp(objective_function, dimensions, population_size, F, R, Gmax, L, U)
53         print(X)
54         print(f"Best: {best}")
55
56     # Ejemplo para 'best/1/bin':
57     elif strategy == 'best/1/bin':
58         X, best = best_1_bin(objective_function, dimensions, population_size, F, R, Gmax, L, U)
59         print(X)
60         print(f"Best: {best}")
61
62     # Ejemplo para 'best/1/exp':
63     elif strategy == 'best/1/exp':
64         X, best = best_1_exp(objective_function, dimensions, population_size, F, R, Gmax, L, U)
65         print(X)
66         print(f"Best: {best}")
67
68     return
```



```

69 #crea poblacion dentro de los intervalos de la funcion objetivo
70 def start_poblacion(population_size, dimension, lower_limit, upper_limit):
71     poblacion = []
72
73     for _ in range(population_size):
74         individual = []
75         for _ in range(dimension):
76             individual.append(np.random.uniform(lower_limit, upper_limit))
77
78         poblacion.append(individual)
79
80     return poblacion
81
82 # Generacion de r1 != r2 != r3
83 def random_numbers(population_size, n):
84     indexes = np.random.randint(0, population_size, n)
85     while len(set(indexes)) != n:
86         indexes = np.random.randint(0, population_size, n)
87     return indexes
88
89 def clip(x, interval): # revisa los limites de las variables de decision
90     limInf, limSup = interval
91     return max(min(x, limSup), limInf)
92
93 # Ejecutar el experimento
94 def run_experiment(strategy, objective_function, dimensions, population_size, F, R, Gmax, runs, L,
95     U):
96     for _ in range(runs):
97         differential_evolution(strategy, objective_function, dimensions, population_size, F, R,
98             Gmax, L, U)
99     return
100
101 def rand_1_bin(objective_function, dimensions, population_size, F, R, Gmax, L, U):
102     X = start_poblacion(population_size, dimensions, L, U)
103     print("====Rand/1/bin====")
104     print(f"====Funcion objetivo: {objective_function.__name__}====")
105
106     print("====Poblacion inicial====")
107     print(f"{X} tamaño: {len(X)}")
108     best_individual = min(X, key=objective_function)
109     best = objective_function(best_individual)
110     for g in range(Gmax):
111         for i in range(population_size):
112             r1, r2, r3 = random_numbers(population_size, 3)
113             jrand = np.random.randint(0, dimensions)
114             ui = []
115             for j in range(dimensions):
116                 if np.random.rand() < R or j == jrand:
117                     u = X[r1][j] + F * (X[r2][j] - X[r3][j])
118                     u = clip(u, (L, U))
119                     ui.append(u)
120                 else:
121                     ui.append(X[i][j])
122
123             if objective_function(ui) <= objective_function(X[i]):
124                 X[i] = ui
125
126             if objective_function(X[i]) < best:
127                 best = objective_function(X[i])
128                 best_individual = X[i]
129
130     print(f"Generacion {g} Mejor individuo: {best_individual} Fitness: {best}")
131
132     return X, best
133
134 def rand_1_exp(objective_function, dimensions, population_size, F, R, Gmax, L, U):
135     X = start_poblacion(population_size, dimensions, L, U)
136     print("====Rand/1/exp====")
137     print(f"====Funcion objetivo: {objective_function.__name__}====")
138
139     print("====Poblacion inicial====")
140     print(f"{X} tamaño: {len(X)}")

```

```

136 best_individual = min(X, key=objective_function)
137 best = objective_function(best_individual)
138 for g in range(Gmax):
139     for i in range(population_size):
140         r1, r2, r3 = random_numbers(population_size, 3)
141         jrand = np.random.randint(0, dimensions)
142         ui = X[i][:]
143         j = 0
144         # n = 0
145         while True:
146             u = X[r1][j] + F * (X[r2][j] - X[r3][j])
147             u = clip(u, (L, U))
148             ui[j] = u
149             j += 1 %% dimensions # Incrementa j de manera circular
150             # jrand = (jrand + 1) % dimensions # Asegura que al menos uno cambie
151             # n += 1
152             if j < dimensions and np.random.rand() < R or j == jrand:
153                 break
154
155             if objective_function(ui) <= objective_function(X[i]):
156                 X[i] = ui
157
158             if objective_function(X[i]) < best:
159                 best = objective_function(X[i])
160                 best_individual = X[i]
161         print(f"Generacion {g} Mejor individuo: {best_individual} Fitness: {best}")
162
163     return X, best
164
165 def best_1_bin(objective_function, dimensions, population_size, F, R, Gmax, L, U):
166     X = start_poblation(population_size, dimensions, L, U)
167     print("=====Best/1/bin=====")
168     print(f"=====Funcion objetivo: {objective_function.__name__}=====")
169     print("=====Poblacion inicial=====")
170     print(f"{X} tamaño: {len(X)}")
171     best_individual = min(X, key=objective_function)
172     best = objective_function(best_individual)
173     for g in range(Gmax):
174         for i in range(population_size):
175             r2, r3 = random_numbers(population_size, 2)
176             jrand = np.random.randint(0, dimensions)
177             ui = []
178             for j in range(dimensions):
179                 if np.random.rand() < R or j == jrand:
180                     u = best_individual[j] + F * (X[r2][j] - X[r3][j])
181                     u = clip(u, (L, U))
182                     ui.append(u)
183                 else:
184                     ui.append(X[i][j])
185
186             if objective_function(ui) <= objective_function(X[i]):
187                 X[i] = ui
188
189             if objective_function(X[i]) < best:
190                 best = objective_function(X[i])
191                 best_individual = X[i]
192         print(f"Generacion {g} Mejor individuo: {best_individual} Fitness: {best}")
193
194     return X, best
195
196 def best_1_exp(objective_function, dimensions, population_size, F, R, Gmax, L, U):
197     X = start_poblation(population_size, dimensions, L, U)
198     print("=====Best/1/exp=====")
199     print(f"=====Funcion objetivo: {objective_function.__name__}=====")
200     print("=====Poblacion inicial=====")
201     print(f"{X} tamaño: {len(X)}")
202     best_individual = min(X, key=objective_function)
203     best = objective_function(best_individual)
204     for g in range(Gmax):

```

```

205     for i in range(population_size):
206         r2, r3 = random_numbers(population_size, 2)
207         jrand = np.random.randint(0, dimensions)
208         ui = X[i][:]
209         j = 0
210         # n = 0
211         while True:
212             u = best_individual[j] + F * (X[r2][j] - X[r3][j])
213             u = clip(u, (L, U))
214             ui[j] = u
215             j = (j + 1) %% dimensions # Incrementa j de manera circular
216             # jrand = (jrand + 1) % dimensions # Asegura que al menos uno cambie
217             # n += 1
218             if np.random.rand() < R or j == jrand:
219                 break
220
221             if objective_function(ui) <= objective_function(X[i]):
222                 X[i] = ui
223
224             if objective_function(X[i]) < best:
225                 best = objective_function(X[i])
226                 best_individual = X[i]
227         print(f"Generación {g} Mejor individuo: {best_individual} Fitness: {best}")
228
229     return X, best
230
231 # Configuración de parámetros según la imagen
232 population_size = 50
233 dimensions = 10
234 F = 0.6
235 R = 0.9
236 Gmax = 1000
237 runs = 1
238
239 np.random.seed(7)
240
241 # Estrategias de evolución diferencial
242 strategies = ['rand/1/bin', 'rand/1/exp', 'best/1/bin', 'best/1/exp']
243
244 # Funciones objetivo
245 objective_functions = [rosenbrock, ackley, griewank, rastrigin]
246
247 # Correr los experimentos
248 for strategy in strategies:
249     for objective_function in objective_functions:
250         if objective_function == rosenbrock:
251             interval = (-2.048, 2.048)
252         elif objective_function == ackley:
253             interval = (-32.768, 32.768)
254         elif objective_function == griewank:
255             interval = (-600, 600)
256         elif objective_function == rastrigin:
257             interval = (-5.12, 5.12)
258         lower_limit, upper_limit = interval
259         run_experiment(strategy, objective_function, dimensions, population_size, F, R, Gmax, runs
, lower_limit, upper_limit)
260         input("Presione Enter para continuar...")
261         clear_screen()

```

¿Cuáles fueron las versiones de algoritmos que no convergieron a la solución óptima?

En la función Griewank, todas las ejecuciones tienden a converger hacia soluciones con valores de fitness relativamente bajos. Esto sugiere que no existen evidencias claras que indiquen la falta de convergencia en alguna de las variantes del algoritmo evaluadas. Por tanto, aunque el rendimiento puede no ser óptimo, todos los enfoques del algoritmo parecen alcanzar un cierto nivel de estabilidad en sus resultados.

¿A qué versión de estrategia evolutiva le fue mejor?

En la función Rastrigin, todas las estrategias evolutivas exhiben un rendimiento parecido, convergiendo hacia soluciones con valores de fitness bajos. Sin embargo, se observa una excepción notable en la segunda ejecución utilizando la Evolución Diferencial con la estrategia (rand/1/exp), que logró alcanzar un fitness significativamente superior. Esto sugiere que esta configuración particular del algoritmo puede ser más efectiva bajo ciertas condiciones, des-

tacándose del resto en términos de eficacia.

¿Qué versión de evolución diferencial fue la mejor e indique por que cree que le fue mejor (en qué problemas).

En la función Rastrigin, las estrategias de Evolución Diferencial (DE) utilizando tanto (rand/1/bin) como (best/1/bin) mostraron un desempeño destacado, logrando en varias ejecuciones converger hacia soluciones con un fitness cercano a cero, que representa el óptimo para esta función. Esto indica que estas estrategias podrían ser especialmente efectivas para abordar problemas que presentan múltiples óptimos locales, como es el caso de Rastrigin, debido a su eficiente balance entre exploración y explotación del espacio de búsqueda.

Por otro lado, en la función Griewank, todas las estrategias evaluadas parecen converger hacia soluciones con fitness bajo, lo que no permite identificar diferencias significativas en el rendimiento entre las distintas variantes de evolución diferencial aplicadas. Sin embargo, la estrategia DE (best/1/exp) mostró un rendimiento ligeramente superior en algunas de las ejecuciones, sugiriendo que podría tener ciertas ventajas en determinadas configuraciones o escenarios de este problema.

Considerando todas las pruebas, cuál es el algoritmo que converge mas rápido (respecto a su valor de media) El algoritmo de Evolución Diferencial (DE) utilizando la estrategia (rand/1/bin) se destaca por su rapidez en la convergencia, especialmente cuando se aplica a la función Rastrigin. En múltiples ejecuciones, este algoritmo ha demostrado su capacidad para converger eficientemente hacia soluciones con un fitness cercano a cero, el cual es el valor óptimo para esta función. Esta eficacia en alcanzar rápidamente soluciones de alta calidad lo convierte en una opción atractiva para problemas de optimización complejos con múltiples óptimos locales.

Cuadro 1: Comparación de Métodos Evolutivos

Algoritmo Genético	Evolución Diferencial	Estrategias Evolutivas
Utiliza operadores de cruzamiento y mutación.	Se basa en la combinación de soluciones ponderadas.	Utiliza mutaciones con adaptación de parámetros.
Operan sobre una representación binaria o simbólica.	Opera sobre una representación real.	Opera generalmente sobre representaciones reales.
Selección basada en el rendimiento relativo.	Selección directa por torneos entre individuos y sus mutantes.	Uso de técnicas de selección como la (μ, λ) y $(\mu + \lambda)$.
Aplicación amplia en problemas de optimización combinatoria.	Eficiente en problemas de optimización sobre espacios continuos.	Enfocado en la auto-adaptación de las estrategias de búsqueda.

Cuadro 2: Comparación de estrategias en diferentes funciones

Problema	(m, λ)	(m + λ)	(r/1/b)	(r/1/e)	(b/1/b)	(b/1/e)
Ackley	3.985e-15	3.9856e-15	1.0302	0.5505	0.7049	0.388
Griewank	–	5.1222e-08	0	8.3599e-12	–	7.233e-08
Rastrigin	1.790e-12	40.709	4.07	31.20	11.286	29.971
Rosenbrock	2.059e-14	4.6960e-15	12.8	12.292	–	25.863

3. Resultados

4. Conclusiones

Escamilla Resendiz Aldo

Los algoritmos de Evolución Diferencial (DE) han mostrado ser herramientas potentes y versátiles para abordar funciones de optimización complejas como Rastrigin y Griewank. Específicamente, la estrategia DE (rand/1/bin) se ha destacado por su rapidez en la convergencia hacia el óptimo en la función Rastrigin, mientras que las estrategias DE (rand/1/bin) y DE (best/1/bin) también han demostrado ser efectivas en enfrentar problemas con múltiples

óptimos locales gracias a su capacidad para explorar y explotar eficientemente el espacio de búsqueda.

Castillo Reyes Diego

Todas las estrategias tendieron a converger a soluciones de baja fitness en la función Griewank, la DE (best/1/exp) ha mostrado un rendimiento ligeramente superior en ciertas ejecuciones. Esto sugiere que la elección de la estrategia DE puede ser crucial dependiendo de la naturaleza específica del problema de optimización a resolver.

Yañez Martinez Marthon

la importancia de elegir la estrategia adecuada de DE, adaptada a las características específicas del problema, para lograr un equilibrio óptimo entre exploración y explotación.