



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 4 INTELIGENCIA DE ENJAMBRE: COLONIA DE ABEJAS

INTEGRANTES:

- Gongora Hernández Jorge Alan
- Miranda Chávez Víctor Ulises
- Sandoval Gil Gael Sebastián

GRUPO: 5BM1

ASIGNATURA: ALGORITMOS BIONINSPIRADOS

PROFESORA: ABRIL VALERIA URIARTE ARCI

Fecha de entrega: 07/Ene/2023

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

1. Introducción

La programación genética es capaz de tomar una serie de programas totalmente aleatorios, no entrenados y ajenos a cualquier función objetivo que se tenga en mente, y hacer que se reproduzcan, muten y evolucionen hacia la verdad.

Piense en la programación genética como un proceso de optimización estocástica. Cada vez que se concibe una población inicial, y con cada paso de selección y evolución del proceso, se seleccionan individuos aleatorios de la generación actual para que sufran cambios aleatorios con el fin de entrar en la siguiente.

La colonia de abejas es un algoritmo de optimización basado en una metaheurística que simula el comportamiento de una colonia de abejas para resolver problemas de optimización. Al igual que en la programación genética, la colonia de abejas utiliza principios de la evolución natural para encontrar soluciones óptimas a problemas complejos.

Sin embargo, a diferencia de la programación genética, en la colonia de abejas las soluciones no se representan mediante cromosomas sino a través de una colonia de abejas virtuales que se mueven en el espacio de búsqueda y evalúan diferentes soluciones para encontrar la mejor solución global. En lugar de mutación y cruce, el algoritmo de colonia de abejas utiliza un proceso de exploración y explotación basado en el comportamiento de las abejas reales para encontrar soluciones óptimas.

1.1. Instrucciones

En el intervalo de valores $(-5,5)$ para los valores x y y , realizar la minimización de la función

$$x^2 + y^2 = r^2$$

Tomando en cuenta los siguientes requisitos:

- Número de abejas: 40
- Iteraciones: 50
- Límite: 10
- En cada iteración imprimir las soluciones asociadas a las abejas obreras.

2. Desarrollo

2.1. Código de la solución

```
1 import math
2 import random
3
4
5 # Definimos la función de aptitud (objetivo a minimizar)
6 def evaluate(solution):
7     x, y = solution
8     return x**2 + y**2
9
10
11 # Definimos la función que produce nuevas soluciones para las abejas obreras
12 def produce_worker_solutions(solutions, solution_limite):
13     # Produce nuevas soluciones para las abejas obreras
14     worker_solutions = []
15     worker_limite = []
```

```

16
17 for i, solution in enumerate(solutions):
18     x, y = solution
19     limite = solution_limites[i]
20     bandera = True
21     if limite <= 10:
22         r1 = random.uniform(-1, 1)
23         # Seleccionamos una soluci n al azar
24         xk, yk = random.choice(solutions)
25
26         eleccion = random.randint(0, 1)
27         if eleccion == 0:
28             xj = x + r1*(xk - x)
29             if evaluate((xj, solution[1])) < evaluate(solution):
30                 worker_solutions.append((xj, solution[1]))
31                 nuevo_limite = 0
32                 worker_limites.append(nuevo_limite)
33                 bandera = False
34         else:
35             yj = y + r1*(yk - y)
36             if evaluate((solution[0], yj)) < evaluate(solution):
37                 worker_solutions.append((solution[0], yj))
38                 nuevo_limite = 0
39                 worker_limites.append(nuevo_limite)
40                 bandera = False
41             limite += 1
42             solution_limites[i] = limite
43
44     return worker_solutions, worker_limites, solution_limites
45
46
47 # Definimos la funci n que selecciona las mejores soluciones
48 def select_best_solutions(solutions, solutions_limites):
49     evaluations = [evaluate(bee) for bee in solutions]
50
51     solutionsAgrup = list(zip(solutions, solutions_limites, evaluations))
52
53     # Ordenamos las soluciones por su aptitud y seleccionamos las mejores
54     solutionsAgrup.sort(key=lambda x: x[2])
55
56     return solutionsAgrup[:10]
57
58
59 # Definimos la funci n que calcula los valores de probabilidad para cada
    soluci n
60 def calculate_probabilities(solutions):
61     # Calculamos los valores de fit de acuerdo a la regla de minimizaci n
62     fit_list = []
63     for s in solutions:
64         f = evaluate(s)
65         newFit = 0
66         if f >= 0:
67             newFit = 1 / (1 + f)
68         else:
69             newFit = 1 + abs(f)
70         fit_list.append(newFit)
71
72     # Calculamos el valor de sigma como la suma de todos los valores de fit
73     sigma = sum(fit for fit in fit_list)
74
75     probabilities = []
76     for fit in fit_list:
77         # Calculamos la probabilidad de cada soluci n utilizando la f rmula
78         p1 = fit/sigma
79         probability = fit / sigma
80         probabilities.append(probability)

```

```

80     return probabilities
81
82
83 # Definimos la funci n que produce nuevas soluciones para las abejas
observadoras
84 def produce_observer_solutions(solutions, solutions_limites, probabilities):
85     # Produce nuevas soluciones para las abejas observadoras
86     observer_solutions = []
87     observer_limites = []
88     solutions_agrupado = list(zip(solutions, solutions_limites, probabilities))
89
90     for _ in range(20):
91         solution_elegida = random.choices(solutions_agrupado, probabilities)[0]
92         sol = solution_elegida[0]
93         x, y = solution_elegida[0]
94         limite = solution_elegida[1]
95         indiceAbeja = solutions_agrupado.index(solution_elegida)
96         bandera = True
97
98         if random.uniform(0, 1) < solution_elegida[2]:
99             if limite <= 10:
100                 r2 = random.uniform(-1, 1)
101                 # Seleccionamos una soluci n utilizando el m todo de la
ruleta
102                 xk, yk = random.choices(solutions, probabilities)[0]
103
104                 eleccion = random.randint(0, 1)
105                 if eleccion == 0:
106                     xj = x + r2*(x - xk)
107                     if evaluate((xj, sol[1])) < evaluate(sol):
108                         observer_solutions.append((xj, sol[1]))
109                         nuevo_limite = 0
110                         observer_limites.append(nuevo_limite)
111                         bandera = False
112                 else:
113                     yj = y + r2*(y - yk)
114                     if evaluate((sol[0], yj)) < evaluate(sol):
115                         observer_solutions.append((sol[0], yj))
116                         nuevo_limite = 0
117                         observer_limites.append(nuevo_limite)
118                         bandera = False
119                 limite += 1
120
121                 solution, limiteSolucion, probability = solutions_agrupado[
indiceAbeja]
122                 solutions_agrupado[indiceAbeja] = solution, limite, probability
123             else:
124                 continue
125
126     solutions, solutions_limites, prob = list(zip(*solutions_agrupado))
127     return observer_solutions, observer_limites, solutions_limites
128
129
130 # Definimos la funci n que produce nuevas soluciones aleatorias para las
abejas exploradoras
131 def produce_explorer_solutions():
132     # Generamos una nueva soluci n aleatoria en el intervalo (-5, 5) para x y
y
133     x = random.uniform(-5, 5)
134     y = random.uniform(-5, 5)
135     return (x, y), 0, 0
136
137
138 def ABC():
139     # Inicializamos la poblaci n de soluciones
140     uj = 5

```

```

141     lj = -5
142     solutions = [(lj + random.uniform(0, 1)*(uj - lj), lj +
143                 random.uniform(0, 1)*(uj - lj)) for _ in range(20)]
144     solutions_limit = [0 for _ in range(len(solutions))]
145     evaluations = [evaluate(bee) for bee in solutions]
146
147     solutions_agrup = list(zip(solutions, solutions_limit, evaluations))
148
149     # Inicializamos la mejor solución encontrada hasta el momento
150     best_solution = min(solutions_agrup, key=lambda x: x[2])
151
152     # Inicializamos el contador de ciclos a 1
153     cycle = 1
154
155     # Repetimos los siguientes pasos hasta que el contador de ciclos sea igual
156     # a 50
157     while cycle <= 50:
158         # Guardamos las mejores soluciones por iteración
159         file.write(f"\n{cycle}. Soluciones: \n{solutions}")
160
161         # Produce nuevas soluciones para las abejas obreras
162         worker_solutions, worker_limit, solutions_limit =
163         produce_worker_solutions(
164             solutions, solutions_limit)
165
166         # Seleccionamos las mejores soluciones
167         solutions, solutions_limit, evaluations = list(zip(
168             *select_best_solutions(worker_solutions + solutions, worker_limit
169             + solutions_limit)))
170
171         # Calculamos los valores de probabilidad para cada solución
172         probabilities = calculate_probabilities(solutions)
173
174         # Produce nuevas soluciones para las abejas observadoras
175         observer_solutions, observer_limit, solutions_limit =
176         produce_observer_solutions(
177             solutions, solutions_limit, probabilities)
178
179         solutions = list(solutions)
180         solutions_limit = list(solutions_limit)
181
182         # Seleccionamos las mejores soluciones
183         solutions_agrup = select_best_solutions(
184             observer_solutions + solutions, observer_limit +
185             solutions_limit)
186
187         solutions, solutions_limit, evaluations = list(zip(*solutions_agrup))
188         solutions = list(solutions)
189         solutions_limit = list(solutions_limit)
190
191         # Actualizamos la mejor solución encontrada hasta el momento
192         solutions_agrup = list(solutions_agrup)
193         solutions_agrup.append(best_solution)
194         best_solution = min(solutions_agrup, key=lambda x: x[2])
195
196         # Contamos cuántas soluciones han sido abandonadas por las abejas
197         # exploradoras
198         abandoned_count = 20 - len(solutions_agrup)
199
200         # Reemplazamos las soluciones abandonadas con nuevas soluciones
201         # aleatorias
202         solutions_agrup += [produce_explorer_solutions()
203                             for _ in range(abandoned_count)]
204
205         solutions, solutions_limit, evaluations = zip(*solutions_agrup)
206         solutions = list(solutions)

```

```

200     solutions_limite = list(solutions_limite)
201
202     # Incrementamos el contador de ciclos
203     cycle += 1
204
205     # Devolvemos la mejor solución encontrada
206     return best_solution
207
208
209 # Creamos nuestro archivo de resultados
210 file = open("resultados.txt", 'w')
211
212 # Ejecutamos el algoritmo y mostramos la mejor solución encontrada
213 best_solution = ABC()
214 file.write(f"\nLa mejor solución es: {best_solution}")
215
216 file.close()

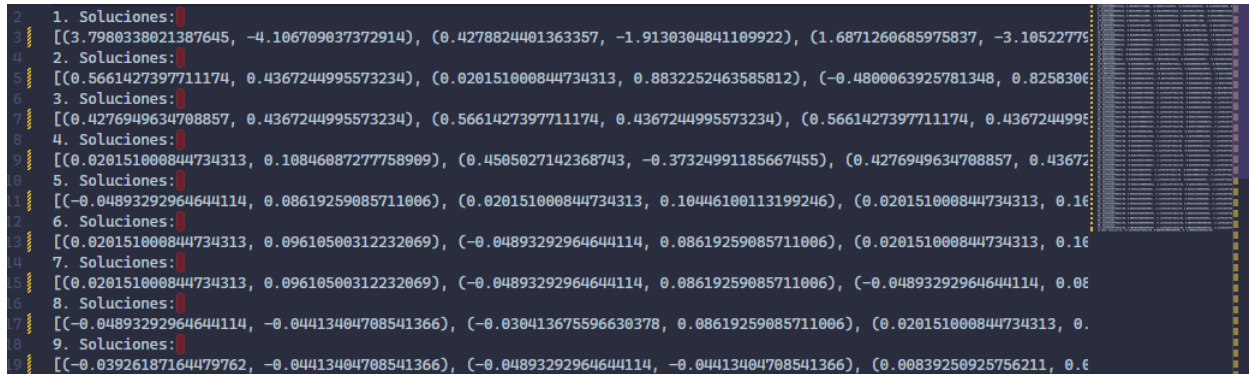
```

Listing 1: Algoritmo de colonia de abejas

2.2. Resultados

Las soluciones asociadas a las abejas obreras se imprimen en un archivo .txt llamado "resultados.txt". Lo anterior se hace al final de cada iteración del algoritmo.

A continuación se muestran capturas de pantalla de los resultados de las iteraciones realizadas:

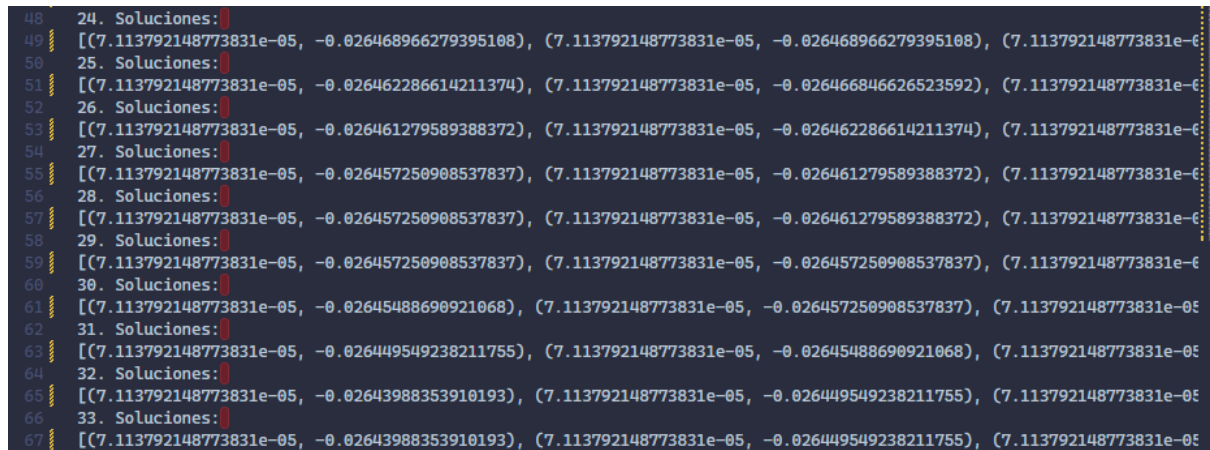


```

1  1. Soluciones:
2  [(3.7980338021387645, -4.106709037372914), (0.4278824401363357, -1.9130304841109922), (1.6871260685975837, -3.10522779
3
4  2. Soluciones:
5  [(0.5661427397711174, 0.4367244995573234), (0.020151000844734313, 0.8832252463585812), (-0.4800063925781348, 0.8258306
6
7  3. Soluciones:
8  [(0.4276949634708857, 0.4367244995573234), (0.5661427397711174, 0.4367244995573234), (0.5661427397711174, 0.4367244995
9
10 4. Soluciones:
11 [(0.020151000844734313, 0.1084608727758909), (0.4505027142368743, -0.37324991185667455), (0.4276949634708857, 0.43672
12
13 5. Soluciones:
14 [(-0.04893292964644114, 0.08619259085711006), (0.020151000844734313, 0.10446100113199246), (0.020151000844734313, 0.10
15
16 6. Soluciones:
17 [(0.020151000844734313, 0.09610500312232069), (-0.04893292964644114, 0.08619259085711006), (0.020151000844734313, 0.10
18
19 7. Soluciones:
20 [(0.020151000844734313, 0.09610500312232069), (-0.04893292964644114, 0.08619259085711006), (-0.04893292964644114, 0.08
21
22 8. Soluciones:
23 [(-0.04893292964644114, -0.04413404708541366), (-0.030413675596630378, 0.08619259085711006), (0.020151000844734313, 0.
24
25 9. Soluciones:
26 [(-0.03926187164479762, -0.04413404708541366), (-0.04893292964644114, -0.04413404708541366), (0.00839250925756211, 0.0

```

Figura 1: Resultado de las primeras 9 iteraciones.



```

48 24. Soluciones:
49 [(7.113792148773831e-05, -0.026468966279395108), (7.113792148773831e-05, -0.026468966279395108), (7.113792148773831e-05,
50
51 25. Soluciones:
52 [(7.113792148773831e-05, -0.026462286614211374), (7.113792148773831e-05, -0.026466846626523592), (7.113792148773831e-05,
53
54 26. Soluciones:
55 [(7.113792148773831e-05, -0.026461279589388372), (7.113792148773831e-05, -0.026462286614211374), (7.113792148773831e-05,
56
57 27. Soluciones:
58 [(7.113792148773831e-05, -0.026457250908537837), (7.113792148773831e-05, -0.026461279589388372), (7.113792148773831e-05,
59
60 28. Soluciones:
61 [(7.113792148773831e-05, -0.026457250908537837), (7.113792148773831e-05, -0.026461279589388372), (7.113792148773831e-05,
62
63 29. Soluciones:
64 [(7.113792148773831e-05, -0.026457250908537837), (7.113792148773831e-05, -0.026461279589388372), (7.113792148773831e-05,
65
66 30. Soluciones:
67 [(7.113792148773831e-05, -0.026457250908537837), (7.113792148773831e-05, -0.026461279589388372), (7.113792148773831e-05,

```

Figura 2: Resultado de las iteraciones 24-33.

```

80 40. Soluciones:
81 [(7.113792148773831e-05, -0.0264324586420702), (7.113792148773831e-05, -0.0264324586420702), (7.113792148773831e-05, -
82 41. Soluciones:
83 [(7.113792148773831e-05, -0.026432176000920942), (7.113792148773831e-05, -0.0264324586420702), (7.113792148773831e-05, -
84 42. Soluciones:
85 [(7.113792148773831e-05, -0.02643113554835394), (7.113792148773831e-05, -0.026432176000920942), (7.113792148773831e-05, -
86 43. Soluciones:
87 [(7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -0.02643113554835394), (7.113792148773831e-05, -
88 44. Soluciones:
89 [(7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -0.02643113554835394), (7.113792148773831e-05, -
90 45. Soluciones:
91 [(7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -
92 46. Soluciones:
93 [(7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -
94 47. Soluciones:
95 [(7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -
96 48. Soluciones:
97 [(7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, 0.004634246340329953), (7.113792148773831e-05, -
98 49. Soluciones:
99 [(7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, -
100 50. Soluciones:
101 [(7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, 0.0007835700654996759), (7.113792148773831e-05, -
102 La mejor solución es: ((7.113792148773831e-05, 0.0007835700654996759), 5, 6.190426514207621e-07)

```

Figura 3: Resultado de las últimas iteraciones y de la mejor solución.

3. Conclusiones:

En conclusión, la colonia de abejas es un algoritmo de optimización que utiliza principios de la evolución natural para encontrar soluciones óptimas a problemas complejos, pero se basa en una metaheurística diferente a la programación genética y utiliza un proceso de exploración y explotación basado en el comportamiento de las abejas reales en lugar de mutación y cruce.

Esta práctica no resultó tan difícil y se aprendió mucho durante el proceso. La colonia de abejas es una técnica de optimización muy útil y efectiva para resolver problemas complejos, y la práctica nos permitió comprender mejor cómo funciona y cómo se puede aplicar en la práctica. Aunque al principio puede parecer un poco intimidante, una vez que se entiende el proceso y se familiariza con los conceptos clave, se vuelve relativamente sencillo implementar y utilizar el algoritmo de colonia de abejas. En general, fue una experiencia muy enriquecedora y estamos seguros de que seguirá siendo de gran utilidad en el futuro.

4. Referencias:

1. De los Cobos Silva, S. G., Andrade, M. Á. G., García, E. A. R., Velázquez, P. L., Cornejo, M. A. (2014). Colonia de abejas artificiales y optimización por enjambre de partículas para la estimación de parámetros de regresión no lineal. *Revista de matematica: teoria y aplicaciones*, 21(1), 107-126.
2. Ortega, D. S. Colonia de abejas artificiales: Técnica metaheurística bioinspirada.
3. Martín-Moreno, R., Vega-Rodríguez, M. A. Algoritmo Multiobjetivo de Colonia de Abejas Artificiales aplicado al Problema de Orientación.