Proyecto. PSO de maximización para la solución del problema de selección de personal

Autores:
Victor Ulises Miranda Chávez
Jorge Alan Gongora Hernandez

Grupo: 5BM1

Fecha de entrega: 13/01/2023

1. Introducción

1.1. Planteamiento del problema

La selección de personal es un problema complejo que implica la elección de los candidatos más adecuados para un puesto específico. Una técnica emergente para abordar este problema es el uso de algoritmos de enjambre de partículas (PSO, por sus siglas en inglés). PSO es un algoritmo de optimización basado en la inteligencia artificial que simula el comportamiento de un enjambre de partículas en movimiento. Estas partículas representan soluciones posibles al problema y se mueven en el espacio de búsqueda para encontrar la solución óptima. En el caso de la selección de personal, se pueden utilizar características como la edad, la experiencia laboral, la educación y las habilidades para evaluar a los candidatos y encontrar el mejor ajuste para el puesto.

Uno de los beneficios de utilizar PSO es que puede manejar un gran número de criterios de selección y candidatos, lo que lo hace adecuado para manejar problemas complejos de selección de personal. Además, el algoritmo es capaz de adaptarse y aprender de los resultados anteriores, lo que permite mejorar constantemente la precisión de la selección. Sin embargo, es importante tener en cuenta que el uso de algoritmos de enjambre de partículas para la selección de personal debe ser utilizado junto con otras técnicas tradicionales, como entrevistas y pruebas, para obtener una visión más completa del candidato y asegurar una selección justa y equitativa.

2. Desarrollo

2.1. Lenguaje utilizado

- Python

2.2. Código:

```python
import random

import numpy as np

# Evalua la calidad de cada candidato
# Esta evaluacion es modificable por los deseos de la empresa
# Ajustamos los pesos de acuerdo con la prioridad que soliciten

def calcular_fitness(candidato):
    # Inicializa el fitness en cero
    fitness = 0
```

```
12
13      # A ade puntos por edad (m s joven es mejor)
14      fitness -= candidato[0] * 0.5
15
16      # A ade puntos por a os de experiencia (m s a os son mejores)
17      fitness += candidato[1] * 0.1
18
19      # A ade puntos por formaci n (m s alta es mejor)
20      fitness += candidato[2] * 0.3
21
22      # A ade puntos por habilidades
23      fitness += candidato[3] * 0.1
24
25      # Devuelve el fitness
26      return fitness
27
28  random.seed(0)
29
30  # Par metros del algoritmo
31  N_PARTICLES = 250 # N mero de part culas
32  N_DIMENSIONS = 4 # N mero de dimensiones del problema (edad, experiencia,
        nivel de formaci n y habilidades)
33  N_ITERATIONS = 170 # N mero de iteraciones del algoritmo
34  C1 = 0.5 # Constante de aceleraci n
35  C2 = 0.47 # Constante de aceleraci n
36  W = 0.8 # Inercia
37
38  print("SOLUCION AL PROBLEMA DE LA SELECCI N CON ENJAMBRE DE PARTICULAS \n")
39  print("No. de particulas: ", N_PARTICLES)
40  print("No. iteraciones: ", N_ITERATIONS)
41  print("")
42
43  # Crea un enjambre de part culas
44  enjambre = []
45  for i in range(N_PARTICLES):
46      # Genera valores aleatorios para cada caracter stica
47      edad = random.randint(50, 64)
48      a os_de_experiencia = 0.5 * edad
49      formacion = random.randint(1, 2)
50      habilidades = random.randint(1, 2)
51
52      # Incializamos las caractersisticas de nuestras particulas
53      posicion = [edad, a os_de_experiencia, formacion, habilidades]
54
55      velocidad = [random.uniform(-5, 5) for _ in range(N_DIMENSIONS)]
56
57      pbest = posicion
58
59      particula = [posicion, velocidad, pbest]
60      enjambre.append(particula)
61
62  # Inicializa la mejor soluci n global
63  global_best = enjambre[0][0]
64
65  # Ejecuta el algoritmo PSO
66  for t in range(N_ITERATIONS):
67      # Recorre cada part cula del enjambre
68      for i in range(N_PARTICLES):
69
70          particula = enjambre[i]
71          posicion = particula[0]
72          velocidad = particula[1]
73
74          # Actualizamos la velocidad de la part cula
75          velocidad = [W * v + C1 * random.uniform(0, 1) * (pbest - x) + C2 *
        random.uniform(0, 1) * (gbest - x) for v, pbest, x, gbest in zip(velocidad,
```

```
    particula[2], posicion, global_best)]
76
77          # Actualizamos la posici n de la part cula
78          agrupacion = zip(posicion, velocidad)
79          posicion = []
80          i = 0
81          for j, (x, v) in enumerate(agrupacion):
82              nuevaPos = x + v
83              # Verificamos si nuestra nueva posicion ha alcanzado los limites
    propuestos
84              # (los limites van de acuerdo a las condiciones que pida una
    empresa)
85              if i == 0:
86                  # No se contratan a gente de menos de 25 a os
87                  if nuevaPos < 25:
88                      posicion.append(25)
89                  # No se contrata a a gente de mas de 65 a os
90                  elif nuevaPos > 65:
91                      posicion.append(65)
92                  else:
93                      posicion.append(nuevaPos)
94              # Para la caracteristica de la experencia, realizamos una
    validacion
95              # que tenga sentido logico, es decir, que la experencia se
    proporcional
96              # a la edad.
97              elif i == 1:
98                  if nuevaPos > posicion[j - 1] - posicion[j - 1] * 0.7:
99                      posicion.append(posicion[j - 1] * 0.3)
100                 else:
101                     posicion.append(nuevaPos)
102             elif i == 2:
103                 # No se contrata a gente con mas de 4 formaciones
104                 if nuevaPos > 4:
105                     posicion.append(4)
106                 else:
107                     posicion.append(nuevaPos)
108             elif i == 3:
109                 # No se contrata a gente con mas de 5 habilidades
110                 if nuevaPos > 5:
111                     posicion.append(5)
112                 else:
113                     posicion.append(nuevaPos)
114             i += 1
115
116         # Actualizamos la mejor posici n de la part cula
117         if calcular_fitness(posicion) > calcular_fitness(particula[2]):
118             particula[2] = posicion
119
120         # Actualizamos el mejor resultado global si es necesario
121         if calcular_fitness(posicion)  > calcular_fitness(global_best):
122             global_best = posicion
123         else:
124             continue
125
126     # Imprimimos el mejor resultado obtenido en cada iteraci n
127     print(f'Iteraci n {t}: {global_best}')
128
129 # Imprime la mejor soluci n global
130 print("\nLa mejor combinacion de caracteristicas obtenida es: \n")
131 print("Edad: ", global_best[0])
132 print("Experiencia: ", global_best[1])
133 print("Formacion: ", global_best[2])
134 print("Habilidades: ", global_best[3])
```

2.3. Capturas de resultados:

```
SOLUCION AL PROBLEMA DE LA SELECCIÓN CON ENJAMBRE DE PARTICULAS

No. de particulas:  250
No. iteraciones:  170

Iteración 0: [44.80127873558759, 13.440383620676275, 4, 1.2391427669859472]
Iteración 1: [43.061144962429424, 12.918343488728826, 4, 1.4339863485897988]
Iteración 2: [41.52188135949131, 12.456564407847393, 4, 1.3435190673368131]
Iteración 3: [41.52188135949131, 12.456564407847393, 4, 1.3435190673368131]
Iteración 4: [39.969908033731286, 11.990972410119385, 4, -0.8566189546646981]
Iteración 5: [39.18176724038635, 11.754530172115905, 4, -2.400620982026222]
Iteración 6: [38.9634444289781, 11.68903332869343, 4, 1.2114842310594596]
Iteración 7: [37.74843177999122, 11.324529533997366, 1.176935504646967, 5]
Iteración 8: [38.216065877250216, 11.464819763175065, 4, 5]
Iteración 9: [38.216065877250216, 11.464819763175065, 4, 5]
Iteración 10: [38.216065877250216, 11.464819763175065, 4, 5]
Iteración 11: [38.216065877250216, 11.464819763175065, 4, 5]
Iteración 12: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 13: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 14: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 15: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 16: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 17: [36.99084394043341, 11.097253182130022, 4, 1.83736046857309]
Iteración 18: [33.94227368846475, 10.182682106539426, -0.37526522625371594, 5]
Iteración 19: [33.94227368846475, 10.182682106539426, -0.37526522625371594, 5]
Iteración 20: [36.640121858856006, 10.992036557656801, 4, 5]
Iteración 21: [34.93452830020196, 10.480358490060588, 1.9287369495982931, 3.986598394587529]
Iteración 22: [33.22722780741982, 9.968168342225946, 1.2619683015991765, 4.7658266059440395]
Iteración 23: [33.22722780741982, 9.968168342225946, 1.2619683015991765, 4.7658266059440395]
Iteración 24: [32.75237441160077, 9.82571232348023, 4, -0.8078854854278452]
Iteración 25: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
Iteración 26: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
Iteración 27: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
Iteración 28: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
```

Figura 1: Mejores particulas resultantes 0 - 28

```
Iteración 28: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
Iteración 29: [32.227641905953156, 9.668292571785946, 4, 1.0089770029949872]
Iteración 30: [29.60780556322589, 8.882341668967767, 4, 0.6553732299891768]
Iteración 31: [29.60780556322589, 8.882341668967767, 4, 0.6553732299891768]
Iteración 32: [29.60780556322589, 8.882341668967767, 4, 0.6553732299891768]
Iteración 33: [29.60780556322589, 8.882341668967767, 4, 0.6553732299891768]
Iteración 34: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 35: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 36: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 37: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 38: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 39: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 40: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 41: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 42: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 43: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 44: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 45: [28.44547834947214, 8.533643504841642, 4, 1.1657481974684343]
Iteración 46: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 47: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 48: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 49: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 50: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 51: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 52: [26.794633847371696, 8.038390154211509, 3.890117132896415, -1.9255089114762347]
Iteración 53: [25.725089685018666, 7.7175269055056, 4, -3.670028117167014]
Iteración 54: [25.725089685018666, 7.7175269055056, 4, -3.670028117167014]
Iteración 55: [25.725089685018666, 7.7175269055056, 4, -3.670028117167014]
Iteración 56: [25.725089685018666, 7.7175269055056, 4, -3.670028117167014]
Iteración 57: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 58: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 59: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 60: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 61: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 62: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
```

Figura 2: Mejores particulas resultantes 28 - 62

```
Iteración 62: [25.667547016644154, 7.700264104993246, 4, -3.384850614260543]
Iteración 63: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 64: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 65: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 66: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 67: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 68: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 69: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 70: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 71: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 72: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 73: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 74: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 75: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 76: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 77: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 78: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 79: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 80: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 81: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 82: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 83: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 84: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 85: [25.88367937816855, 7.765103813450565, 4, -1.1232882944573812]
Iteración 86: [25.524640858240232, 7.657392257472069, 4, 0.4958610035658799]
Iteración 87: [25.524640858240232, 7.657392257472069, 4, 0.4958610035658799]
Iteración 88: [25.524640858240232, 7.657392257472069, 4, 0.4958610035658799]
Iteración 89: [25.524640858240232, 7.657392257472069, 4, 0.4958610035658799]
Iteración 90: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 91: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 92: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 93: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 94: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 95: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 96: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
```

Figura 3: Mejores particulas resultantes 62 - 96

```
Iteración 96: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 97: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 98: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 99: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 100: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 101: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 102: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 103: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 104: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 105: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 106: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 107: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 108: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 109: [25.715667045952117, 7.714700113785635, 4, 3.315006002218077]
Iteración 110: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 111: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 112: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 113: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 114: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 115: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 116: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 117: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 118: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 119: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 120: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 121: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 122: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 123: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 124: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 125: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 126: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 127: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 128: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 129: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 130: [25.392592978848004, 7.6177778936544005, 4, 5]
```

Figura 4: Mejores particulas resultantes 96 - 130

```
Iteración 130: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 131: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 132: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 133: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 134: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 135: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 136: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 137: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 138: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 139: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 140: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 141: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 142: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 143: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 144: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 145: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 146: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 147: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 148: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 149: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 150: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 151: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 152: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 153: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 154: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 155: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 156: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 157: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 158: [25.392592978848004, 7.6177778936544005, 4, 5]
Iteración 159: [25, 7.5, 4, 5]
Iteración 160: [25, 7.5, 4, 5]
Iteración 161: [25, 7.5, 4, 5]
Iteración 162: [25, 7.5, 4, 5]
Iteración 163: [25, 7.5, 4, 5]
Iteración 164: [25, 7.5, 4, 5]
Iteración 165: [25, 7.5, 4, 5]
```

Figura 5: Mejores particulas resultantes 130 - 165

```
Iteración 159: [25, 7.5, 4, 5]
Iteración 160: [25, 7.5, 4, 5]
Iteración 161: [25, 7.5, 4, 5]
Iteración 162: [25, 7.5, 4, 5]
Iteración 163: [25, 7.5, 4, 5]
Iteración 164: [25, 7.5, 4, 5]
Iteración 165: [25, 7.5, 4, 5]
Iteración 166: [25, 7.5, 4, 5]
Iteración 167: [25, 7.5, 4, 5]
Iteración 168: [25, 7.5, 4, 5]
Iteración 169: [25, 7.5, 4, 5]

La mejor combinacion de caracteristicas obtenida es:

Edad:  25
Experiencia:  7.5
Formacion:  4
Habilidades:  5
```

Figura 6: Mejores particulas resultantes 165 - 169

3. Conclusiones:

En resumen, el algoritmo de Enjambre de Partículas (Particle Swarm Optimization, PSO) es una técnica de optimización inspirada en el comportamiento de las colonias de animales. El algoritmo utiliza un enjambre de partículas que se mueven a través del espacio de búsqueda en busca de una solución óptima. Cada partícula representa una posible solución y tiene una velocidad y una posición asociadas. El algoritmo se basa en el intercambio de información entre las partículas para mejorar la búsqueda global y se actualiza en cada iteración. PSO es una técnica eficaz para resolver problemas de optimización y se utiliza en una variedad de campos, incluyendo la inteligencia artificial, la robótica, y la ingeniería.