Inteligência Artificial

Thiago Henrique Leite da Silva, RA: 139920

AULA7: Exercício prático algoritmos bioinspirados

1) Qual a diferença entre AG e PSO?

Estes algoritmos são bem semelhantes, mas possuem algumas diferenças importantes de serem observadas, uma destas diferenças entre estes dois algoritmos é o contexto em que eles se baseiam, os Algoritmos Genéticos se baseiam na computação evolutiva, como vimos nas aulas da semana passada, baseados nos conceitos da teoria evolutiva de Darwin, nós aplicamos isto a computação para melhorar nossas soluções. Já o PSO se baseia na inteligência de enxames.

A inteligência de enxames é um paradigma mais recente neste ramo dos algoritmos bioinspirados, e é considerada uma extensão dos algoritmos evolutivos, onde o AG se encaixa. Levando em consideração um enxame de abelhas, nós conseguimos observar um grande trabalho coletivo entre os indivíduos, o que proporciona um ganho muito grande pro conjunto de abelhas, e, se tratando do PSO, é neste comportamento social coletivo que estamos interessados computacionalmente falando, ao contrário do algoritmo genético, onde nos baseamos na adaptação genética destes indivíduos.

Outra diferença a ser destacada, é o campo de atuação, os algoritmos genéticos performam melhor em ambientes discretos, já os algoritmos PSO são mais adequados a ambientes discretos.

2) Aplique PSO para maximizar a função $f(x) = 1 + 2x – x$

Considere os seguintes parâmetros:

w = 0.70 c1 = 0.20 c2 = 0.60 r1 = 0.4657 r2 = 0.5319

n = 5 iterações = 3

X = [0.4657, 0.8956, 0.3877, 0.4902, 0.5039]
V = [0.5319, 0.8185, 0.8331, 0.7677, 0.1708]

Inicialização das posições:
Opcional: Multiplica-se por 10 para gerar partículas > 1 e subtrai 0.5 para gerar partículas negativas:

X(0) = 10 * [X – 0.5]
X(0) = 10 *{ [0.4657, 0.8956, 0.3877, 0.4902, 0.5039] – 0.5}
X(0) = [-0.343, 3.956, -1.123, -0.098, 0.039]

Inicialização da velocidade:
V(0) = [V – 0.5]
V(0) = {[0.5319, 0.8185, 0.8331, 0.7677, 0.1708] – 0.5}
V(0) = {[0.0319, 0.3185, 0.3331, 0.2677, -0.3292]}

1ª Iteração:
X(1) = X(0) = [-0.343, 3.956, -1.123, -0.098, **0.039**]
V(1) = V(0) = [0.0319, 0.3185, 0.3331, 0.2677, -0.3292]
F(1) = 1 + 2*X(1) –X(1)^2
F(1) = [0.1976, -6.7368, -2.5061, 0.7942, **1.0755**]

Local best position – LBP (Pbest) = [-0.343, 3.956, -1.123, -0.098, 0.039]
Global best fitness = 1.0755
Global best position – GBP (Gbest) = 0.039

2ª Iteração:
V(i+1) = $v_{i+1} = wv_i + c_1 r_1 (P_{best} - X_i) + c_2 r_2 (G_{best} - X_i)$

X(i+1) = $X_{i+1} = X_i + v_{i+1}$

F(2) = 1 + 2*X(2) –X(2)^2
F(2) =

Local best position –LBP (Pbest) =
Global best fitness =
Global best position – GBP (Gbest) =

3ª Iteração:
...

Soulução

2ª Iteração:

- V(2) = $v_{i+1} = wv_i + c_1 r_1 (P_{best} - X_i) + c_2 r_2 (G_{best} - X_i)$
- V(2) = [0.1442, -1.0271, 0.604, 0.2311, -0.2304]

- X(2) = [-0.1988, 2.9289, -0.519, 0.1331, -0.1914]

- F(2) = 1 + 2*X(2) – X(2)²
- F(2) = [0.5629, -1.7207, -0.3074, 1.2485, 0.5806]

- Local best position (Pbest): [-0.1988, 2.9289, -0.519, 0.1331, -0.1914]
- Global best fitness: 1.2485
- Global best position (Gbest): 0.1331

3ª Iteração:

- V(2) = $v_{i+1} = wv_i + c_1 r_1 (P_{best} - X_i) + c_2 r_2 (G_{best} - X_i)$
- V(2) = [0.2069, -1.6112, 0.6309, 0.1618, -0.0577]

- X(2) = [0.0081, 1.3177, 0.1119, 0.2949, -0.2491]

- F(2) = 1 + 2*X(2) – X(2)²
- F(2) = [1.0161, 1.8991, 1.2113, 1.5028, 0.4397]
-
- Local best position (Pbest): [0.0081, 1.3177, 0.1119, 0.2949, -0.2491]
- Global best fitness: 1.8991
- Global best position (Gbest): 1.3177

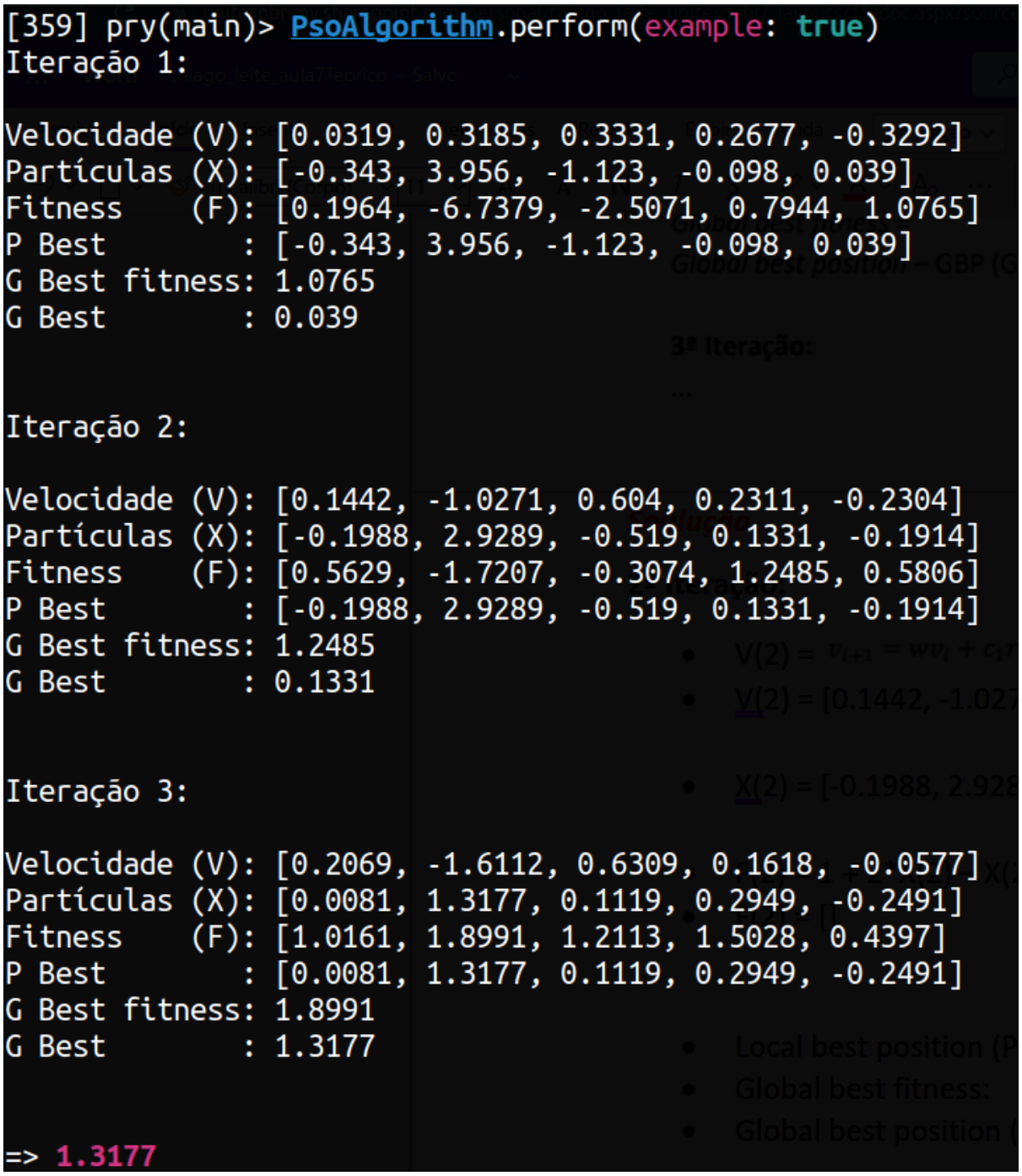
Para chegar aos valores, utilizei meu próprio algoritmo implementado:

Valores de entrada:

```ruby
# Valores vistos em aula
def set_particles
  @particles = [0.4657, 0.8956, 0.3877, 0.4902, 0.5039]
end

def set_velocity
  @velocity = [0.5319, 0.8185, 0.8331, 0.7677, 0.1708]
end
```

Saída obtida:



Obs.: No exemplo fornecido, alguns valores de F(1) não estavam condizentes com os valores corretos se aplicássemos a partícula na função.

3) Continue a execução do ACO apresentado na aula considerando um grafo com 5 vértices e a distância entre eles dada por:

	A	B	C	D	E
A	0	2	10	8	3
B	1	0	2	5	7
C	9	1	0	3	6
D	10	4	3	0	2
E	2	7	5	1	0

Parâmetros: alpha = 1, beta = 1, rho = 0.5, Tau(0) = 2

Obs: Pode considerar o custo total F(S) na atualização do feromônio igual ao custo da aresta.

Obs2: Pode fazer as equações numa planilha do Excel, não precisa ser manual

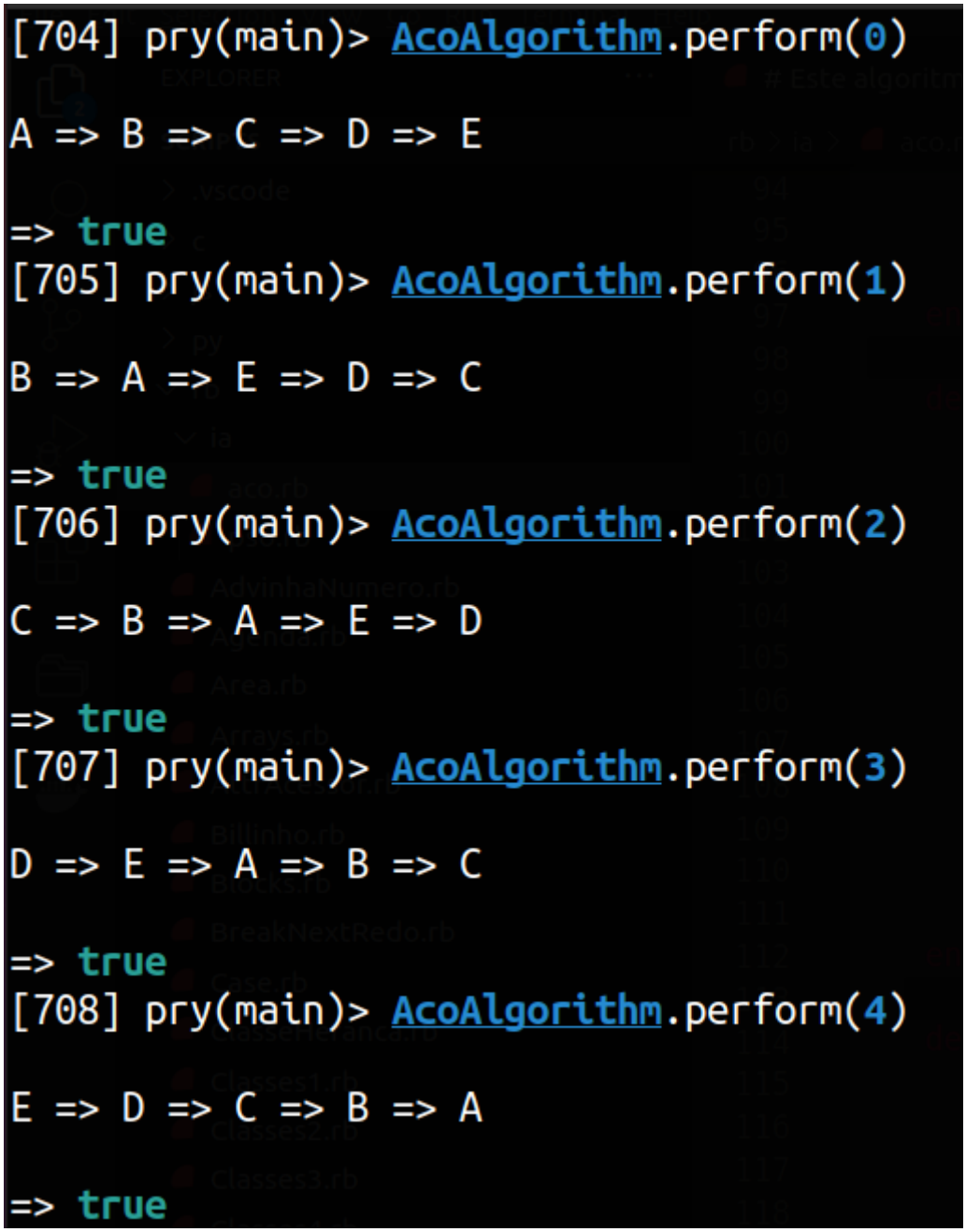
Na aula foi apresentada uma simulação saindo do vértice A, agora, faça simulações saindo dos vértices B, C, D e E. Apresente a rota gerada por cada simulação e atualize o feromônio para todas as restas.

$$p_{i,j}^{k}(t)\begin{cases}\frac{\alpha\tau_{i,j}\cdot\beta\eta_{i,j}}{\sum_j \alpha\tau_{i,j}\cdot\beta\eta_{i,j}} & j\epsilon J^k \\ 0 & c.c\end{cases}$$

$$\tau_{i,j\ (t+1)=}(1-\rho)\cdot\tau_{i,j}+\rho\Delta\tau_{i,j}$$

$$\Delta\tau_{i,j}=\begin{cases}\frac{1}{f(S)} & (i,j)\epsilon S \\ 0 & c.c.\end{cases}$$

Para realizar este exercício, implementei um algoritmo específico para ele, com isso, consigo definir o vértice inicial ao qual queremos iniciar o caminho, e ele nos retorna qual o melhor caminho a ser seguido com base nos parâmetros fornecidos, que deixei fixo no código. Sendo assim, segue os resultados obtidos:



As entradas numéricas que vão de 0..5 correspondem as letras A, B, C, D e E, respectivamente.

Agora, mostrando a tabela de Feromônios que é atualizada a cada novo caminho.

```
[772] pry(main)> AcoAlgorithm.perform

Caminho (1): A => B => C => D => E

[0, 1.25, 1.05, 1.06, 1.17]
[2, 0, 2, 2, 2]
[2, 2, 0, 2, 2]
[2, 2, 2, 0, 2]
[2, 2, 2, 2, 0]


Caminho (2): B => A => E => D => C

[0, 1.25, 1.05, 1.06, 1.17]
[1.5, 0, 1.25, 1.1, 1.07]
[2, 2, 0, 2, 2]
[2, 2, 2, 0, 2]
[2, 2, 2, 2, 0]


Caminho (3): C => B => A => E => D

[0, 1.25, 1.05, 1.06, 1.17]
[1.5, 0, 1.25, 1.1, 1.07]
[1.06, 1.5, 0, 1.17, 1.08]
[2, 2, 2, 0, 2]
[2, 2, 2, 2, 0]


Caminho (4): D => E => A => B => C

[0, 1.25, 1.05, 1.06, 1.17]
[1.5, 0, 1.25, 1.1, 1.07]
[1.06, 1.5, 0, 1.17, 1.08]
[1.05, 1.13, 1.17, 0, 1.25]
[2, 2, 2, 2, 0]
```

```
Caminho (5): E => D => C => B => A

[0, 1.25, 1.05, 1.06, 1.17]
[1.5, 0, 1.25, 1.1, 1.07]
[1.06, 1.5, 0, 1.17, 1.08]
[1.05, 1.13, 1.17, 0, 1.25]
[1.25, 1.07, 1.1, 1.5, 0]

=> 5
```