UNIVERSIDADE ESTADUAL PAULISTA

Programa de Pós-Graduação em Ciência da Computação

Computação Inspirada pela Natureza - **Trabalho 3** - Professor Fabricio Breve

Giovanna Carreira Marinho

Introdução

Baseados no funcionamento e inteligência de enxames, o *Particle Swarm Optimization* (PSO) e *Ant Colony Optimization* (ACO) constituem dois algoritmos poderosos e aplicados em diferentes problemas. As seções a seguir apresentam uma aplicação desses algoritmos: PSO para minimização de uma função e ACO para busca de menor rota.

Para o desenvolvimento dos algoritmos, foi utilizada a linguagem de programação *Python*, assim como alguns pacotes de bibliotecas disponíveis. O ambiente de desenvolvimento escolhido foi o *Google Colab*. As implementações podem ser acessadas pelo seguinte link: https://github.com/Giovannacm/nature-inspired-computing.

Particle Swarm Optimization - PSO

Inspirado no comportamento de enxames (como pássaros e peixes) o PSO – *Particle Swarm Optimization* é utilizado em vários problemas de busca e otimização. Em enxames, os indivíduos aprendem com sua própria experiência e com a experiência dos outros, se avaliando e comparando aos seus vizinhos e imitando os vizinhos superiores a eles próprios.

A seção a seguir apresenta o uso do PSO para minimização da função $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ no intervalo $[-5, 5]$. Para isso, uma população de n indivíduos é criada a partir de uma lista de dicionários com os seguintes campos: `'id'` (identificador do indivíduo), `'viz'` (lista de vizinhos do indivíduo, calculada de forma circular), `'cord'` (posição/coordenada do indivíduo, inicializada aleatoriamente), `'vel'` (velocidade do indivíduo, inicializada aleatoriamente) e `'p'` (melhor solução do indivíduo). A função do PSO foi desenvolvida com base no pseudocódigo da Figura 1, apresentado em aula.

Figura 1 – Pseudocódigo PSO.

```
procedimento [X] = PS(max_it, AC1, AC2, Vmax, Vmin)
  inicializar X   // normalmente x_i, ∀_i, é inicializado aleatoriamente
  inicializar V   // aleatoriamente, v_i ∈[v_min, v_max]
  t ← 1;
  enquanto t < max_it faça
    para i de 1 até N faça          // para cada partícula
      se g(x_i) > g(p_i)
         então p_i = x_i            // melhor desempenho individual
      fim-se
      g = i                         // arbitrário
      para j = índice dos vizinhos // para todos os vizinhos
        se g(p_j) > g(p_g)
          então g = j               // índice do melhor vizinho
        fim-se
      fim-para
      v_i ← v_i + φ1 ⊗ (p_i − x_i) + φ2 ⊗ (p_g − x_i)
      v_i ∈ [v_min, v_max]
      x_i ← x_i + v_i
    fim-para
    t ← t + 1
  fim-enquanto
fim-procedimento
```

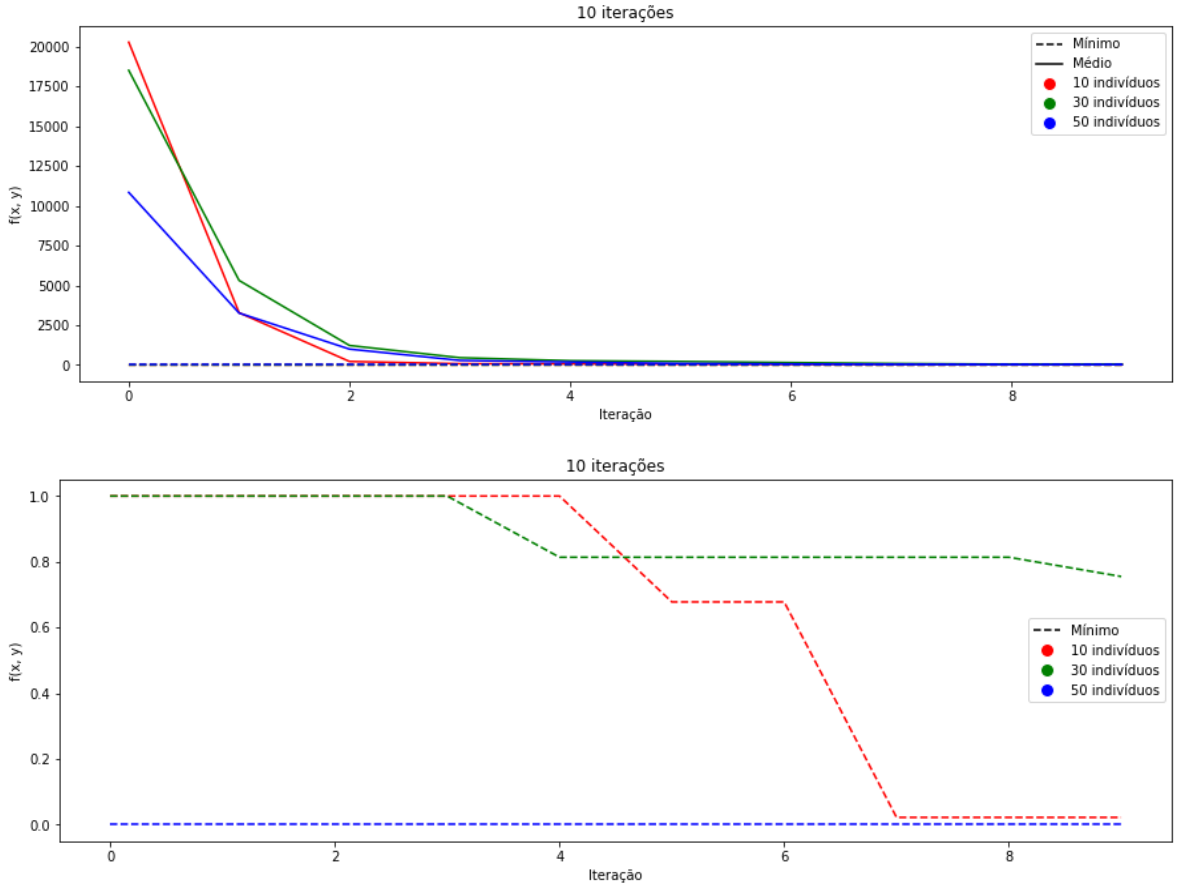
g() é a função de aptidão
g é o índice do melhor vizinho
$\mathbf{p}_i$ é o melhor desempenho da partícula i
$\mathbf{p}_g$ é o melhor desempenho do melhor vizinho de i

Foram realizados dois testes: 10 e 30 iterações. Para cada um, três populações de tamanhos diferentes foram utilizadas: 10, 30 e 50 indivíduos. Sendo aplicada as seguintes configurações:

- v_{min}: -2
- v_{max}: 2
- AC_1, AC_2: 2.05, ambos utilizados no cálculo de φ_1 e φ_2

A Figura 2 apresenta o resultado da execução do PSO em 10 iterações. É possível observar o valor mínimo e médio da população em relação às iterações. As três populações de tamanhos diferentes são diferenciadas pelas cores. O segundo gráfico apresenta apenas os valores mínimos.

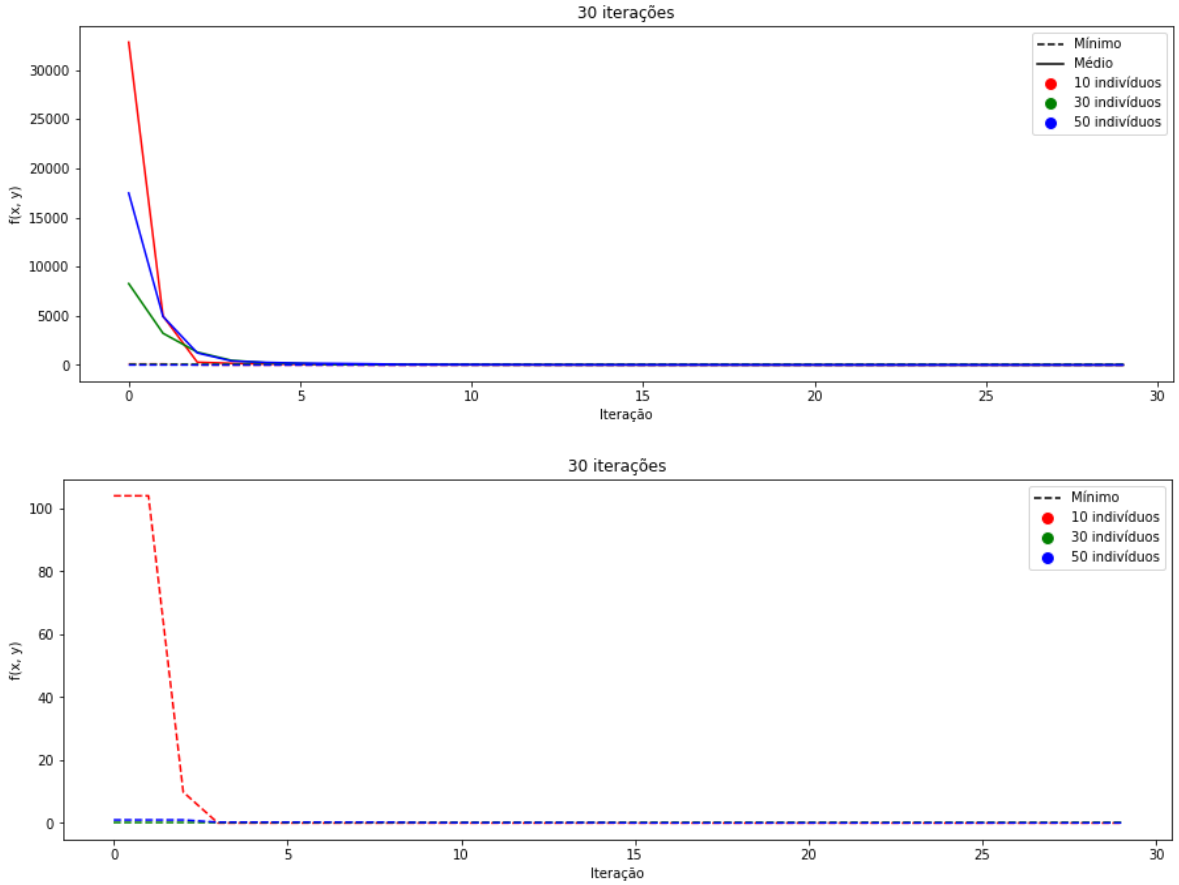
Figura 2 – 10 iterações do PSO para diferentes tamanhos de população.



Analisando os valores obtidos, observa-se que a quantidade de indivíduos envolvidos na busca da melhor solução foi relevante para a convergência. A solução com mais indivíduos (50) foi aquela que convergiu mais rápido, obtendo o valor 0. As outras ainda não convergiram, mas analisando o comportamento observa-se que a partir de mais algumas iterações o valor mínimo seria alcançado. Como foi feita apenas uma execução do algoritmo e ele se baseia em valores aleatórios, uma nova execução pode resultar em outra configuração da solução.

A Figura 3 apresenta o resultado da execução do PSO em 30 iterações. Nesse exemplo, as populações que envolveram mais indivíduos apresentaram melhor desempenho. Diferentemente do resultado anterior, a maior quantidade de iterações permitiu que as outras duas soluções alcançassem o valor mínimo (0).

Figura 3 – 30 iterações do PSO para diferentes tamanhos de população.



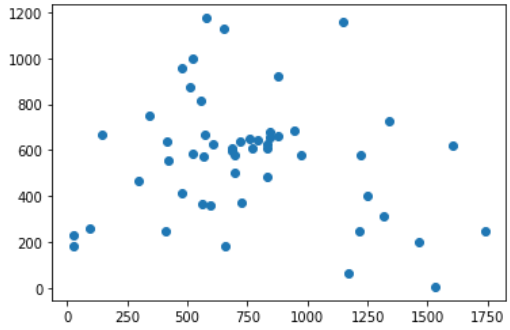
Comparando os resultados com aqueles obtidos pelos Algoritmos Genéticos, observa-se que o PSO permitiu uma convergência mais rápida e um resultado mais preciso. Em termos de implementação, o primeiro é mais simples e com menos valores a serem ajustados. Já o segundo possuí muitos detalhes a serem considerados.

Ant Colony Optimization - ACO

Baseado no comportamento observado em colônias de formigas, o *Ant Colony Optimization* (ACO) é utilizado em problemas de otimização, controle distribuído, agrupamento de dados e outros. Sabe-se que as formigas possuem glândulas abdominais que excretam feromônios utilizados em diversas tarefas. O princípio da auto-organização das colônias, sentidos das formigas, vida comunitária, a busca por trajetos mais curtos são conceitos levados em consideração na implementação.

Essa última seção aplica o ACO para encontrar o menor caminho no problema do Caixeiro Viajante, com 52 cidades dispersas no plano cartesiano, apresentadas na Figura 4.

Figura 4 – Cidades envolvidas no problema.



Para o desenvolvimento e mapeamento da solução, as cidades e suas distâncias (distância euclidiana) foram representadas em uma estrutura de dados de matriz de adjacências (grafo). Uma população de 54 “formigas” foi criada. Cada formiga é representada por uma lista que armazena seu trajeto percorrido em cada iteração durante a execução do algoritmo. Cada formiga foi inicializada em uma cidade. Além disso, uma matriz de adjacências secundária foi necessária para armazenar o respectivo valor do feromônio. Por fim, a função do ACO foi desenvolvida com base no pseudocódigo da Figura 5, apresentado em aula.

Figura 5 – Pseudocódigo ACO.

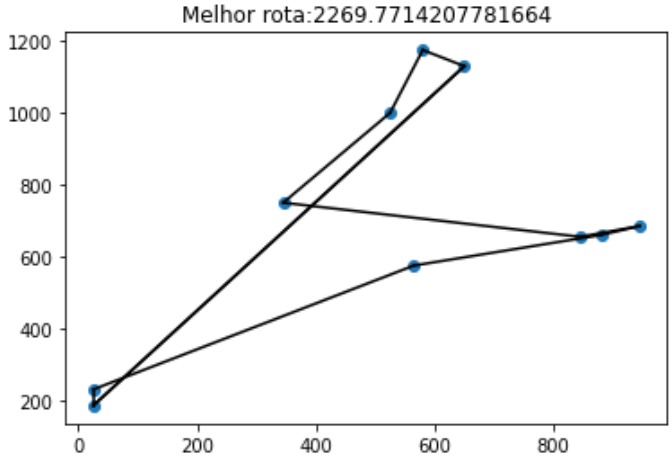
procedimento $[melhor] =$ AS−TSP$(\max_it, \alpha, \beta, \rho, N, e, Q, \tau_0, b)$
  inicializar τ_{ij} // normalmente inicializados com o mesmo τ_0
  colocar cada formiga k em uma cidade selecionada aleatoriamente
  seja *melhor* a melhor rota encontrada desde o início do algoritmo
    e L_{melhor} seu tamanho
  $t \leftarrow 1$;
  enquanto $t <$ max_*it* **faça**
    para i de 1 até N **faça** // para cada formiga
      // e é o número de cidades no grafo
      construa uma rota $T^k(t)$ aplicando o seguinte passo $(e-1)$ vezes:
      na cidade i, escolha a próxima cidade j com probabilidades $p_{ij}^k(t)$
    fim-para
    avaliar o tamanho da rota construída por cada formiga
    se uma rota mais curta for encontrada
      então atualizar *melhor* e L_{melhor}
    fim-se
    para cada aresta **faça**
      Atualizar as trilha de feromônio aplicando a seguinte regra:
        $\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) + b.\Delta\tau_{ij}^b(t)$, onde
        $\Delta\tau_{ij}(t) = \sum_k \Delta\tau_{ij}^k(t)$, $k = 1, \ldots, N$;
        $\Delta\tau_{ij}^k = \begin{cases} Q/L^k(t) & \text{se } (i,j) \in T^k(t) \\ 0 & \text{caso contrário} \end{cases}$, e
        $\Delta\tau_{ij}^b = \begin{cases} Q/L_{melhor} & \text{se } (i,j) \in melhor \\ 0 & \text{caso contrário} \end{cases}$.
    fim-para
    $t \leftarrow t + 1$
  fim-enquanto
fim-procedimento

A execução do algoritmo foi feita com as seguintes configurações:

- α: 1
- β: 5
- ρ: 0.5
- Q: 100
- τ_0: 10^{-6}
- b: 5

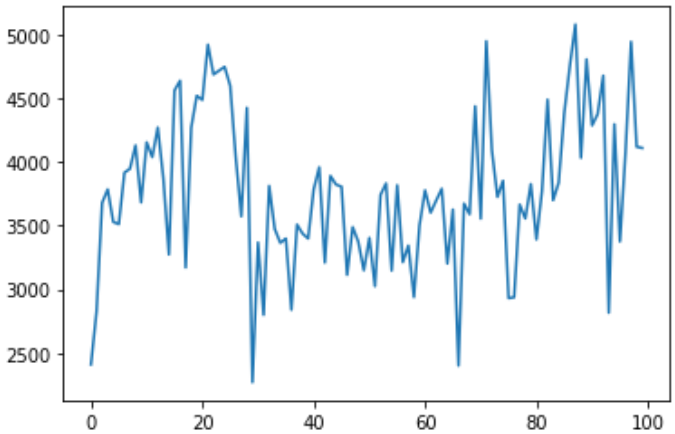
A Figura 6 apresenta a rota de um teste inicial do algoritmo para as primeiras 10 cidades do conjunto de dados. O ACO foi executado com as mesmas configurações apresentadas anteriormente. A execução foi feita com 100 iterações.

Figura 6 – Resultado para as 10 primeiras cidades do conjunto de dados com 100 iterações.



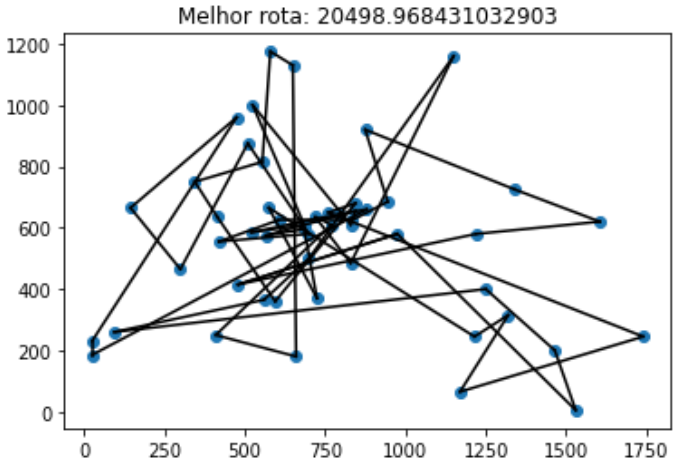
A melhor rota obtida (9, 8, 7, 2, 4, 3, 5, 0, 6, 1, 9) apresentou custo de 2269.77. Analisando as melhores distâncias obtidas durante as execuções, na Figura 7, observa-se que a melhor solução foi obtida na iteração 29.

Figura 7 – Histórico das melhores soluções durante as 100 iterações para as 10 cidades.



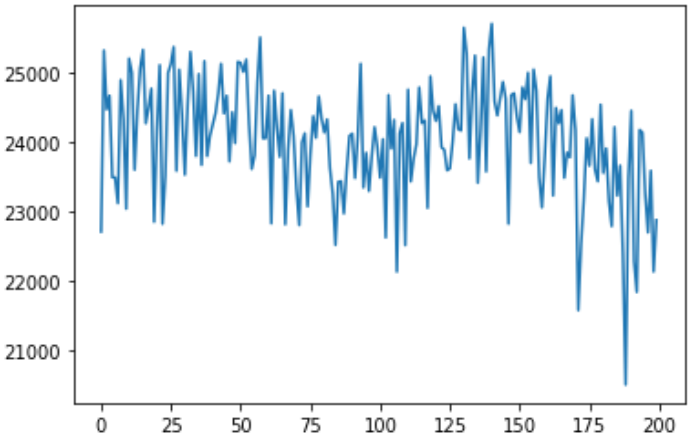
A Figura 8 apresenta a rota final da execução do algoritmo para todas as 52 cidades, com as configurações apresentadas anteriormente e 200 iterações. Tanto no exemplo anterior quanto nesse, pelo fato do algoritmo se basear em valores aleatórios, outra execução pode implicar em um resultado diferente.

Figura 8 – Resultado para as 52 cidades do conjunto de dados com 200 iterações.



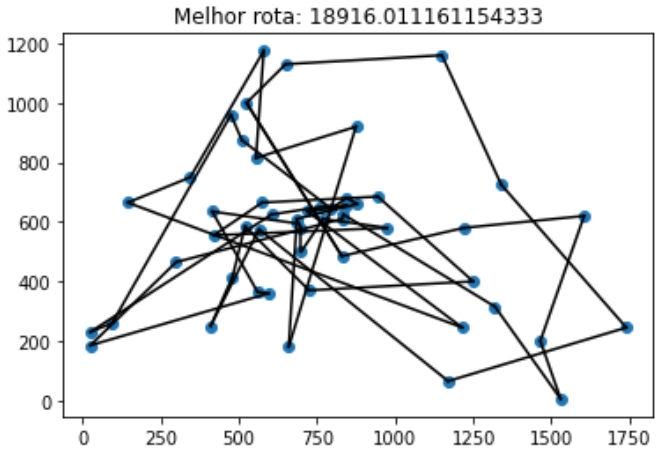
A rota escolhida como a melhor (22, 11, 10, 50, 42, 3, 23, 48, 30, 33, 18, 20, 16, 40, 6, 1, 5, 37, 21, 35, 7, 47, 4, 0, 34, 15, 31, 25, 26, 46, 51, 39, 14, 49, 17, 2, 44, 8, 9, 28, 29, 36, 38, 45, 32, 43, 19, 41, 27, 12, 13, 24, 22) obteve custo de 20498.97. Analisando o histórico de melhores soluções encontradas em cada iteração, observa-se que a melhor solução para o problema foi obtida na iteração 188.

Figura 9 – Histórico das melhores soluções durante as 200 iterações para as 52 cidades.



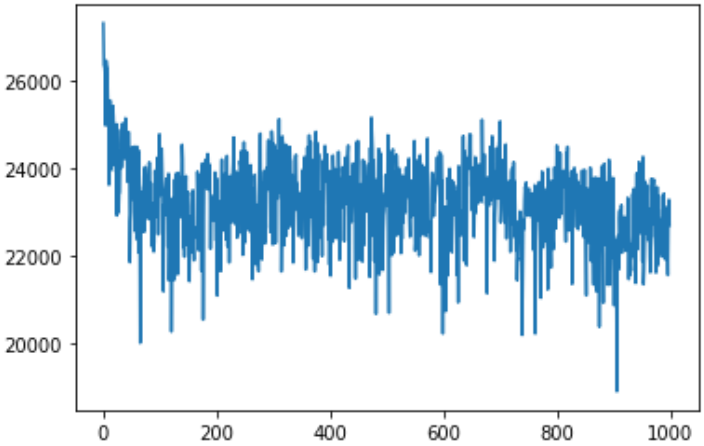
Aumentando a quantidade de iterações para 1000, a melhor rota (45, 11, 10, 12, 13, 26, 23, 47, 20, 1, 49, 19, 17, 34, 28, 42, 44, 8, 2, 16, 25, 38, 4, 39, 48, 6, 41, 40, 18, 37, 14, 5, 35, 33, 43, 36, 24, 30, 31, 3, 27, 15, 21, 22, 29, 0, 46, 51, 50, 32, 9, 7, 45) obteve um custo menor de 18916.01. A Figura 10 apresenta a rota definida por esse resultado.

Figura 10 – Resultado para as 52 cidades do conjunto de dados com 1000 iterações.



Observando o histórico de melhores soluções, esta foi obtida na iteração 906, como mostra a Figura 11.

Figura 11 – Histórico das melhores soluções durante as 1000 iterações para as 52 cidades.



Com o aumento do número de iterações, verifica-se que um trajeto com menor custo foi encontrado.