UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

PSO e ACO

Computação Inspirada pela Natureza

Aluno: Samuel de Souza Lopes

Professor: Dr. Fabricio Aparecido Breve

2018

1. Introdução e Objetivos

O método de enxame de partículas, ou PSO, é um ramo da inteligência artificial que otimiza um problema iterativamente ao tentar melhorar a solução candidata com respeito a uma dada medida de qualidade. Já o método de otimização por colônia de formigas, ou ACO, é um método baseado em probabilidade criado para solucionar problemas computacionais que envolvem procura de caminhos em grafos. Ambos os métodos são bioinspirados, sendo bastante utilizados em diversas aplicações como otimização de funções e solução do problema do caixeiro viajante (DE CASTRO, 2006).

Neste contexto, o presente trabalho objetiva mostrar duas aplicações relacionadas a PSO e ACO, a saber: minimização de funções e solução do problema do caixeiro viajante. Na seção 2 os algoritmos foram detalhados, bem como os testes são expostos. Por fim, na seção 3 foram feitas conclusões acerca do trabalho.

2. Aplicações

Nesta seção são detalhadas as implementações de duas aplicações relacionadas a PSO e ACO, a saber: minimização de funções e solução do problema do caixeiro viajante. A linguagem de programação e o ambiente de desenvolvimento utilizados foram *Python* e *PyCharm*, respectivamente.

Ao final de cada implementação, foram feitos testes e os resultados obtidos foram discutidos. Vale ressaltar que os algoritmos implementados são estocásticos, ou seja, com diferentes execuções os gráficos e valores ótimos encontrados podem variar. As especificações do computador utilizado para fazer os testes de execução dos algoritmos implementados é:

- **Processador:** Intel® Core™ i3-3217U CPU @ 1.80GHz × 4;
- **Memória RAM:** 4 GB;
- **Placa de Vídeo:** Intel® Ivybridge Mobile (integrada)
- **Sistema Operacional:** Ubuntu 16.04 LTS.

2.1. Minimização de funções

O objetivo desta aplicação é implementar, utilizando o PSO, um programa para minimizar a seguinte função e o seu respectivo domínio *Df*, conforme mostra a Figura 1. A solução ótima de f(x,y) é 0.

Figura 1: Função f(x,y).

$$f(x,y)=(1-x)^2+100(y-x^2)^2, Df=\{(x,y)\in\mathbb{R}: x\in[-5,+5] e\ y\in[-5,+5]\}$$

Fonte: Elaborada pelo próprio autor.

Detalhamento do algoritmo

A função principal do programa possui seis parâmetros, a saber: número máximo de iterações - max_it, constante de aceleração 1 - ac1, constante de aceleração 2 - ac2, número de partículas - numero_particulas, velocidade mínima da partícula - vmin e velocidade máxima da partícula - vmax. A matriz x armazena os valores das partículas ao longo das iterações, sendo que cada linha de x é uma coordenada do domínio de f(x,y). Enquanto o número máximo de iterações não for atingido, os passos a seguir são executados: Para cada partícula, o melhor desempenho individual de cada partícula é obtido e o índice do melhor vizinho é calculado, sendo que a vizinhança local foi utilizada. Ao final da iteração, a velocidade de cada partícula é atualizada, sendo que a matriz v é somada a matriz x.

Experimentos e Testes

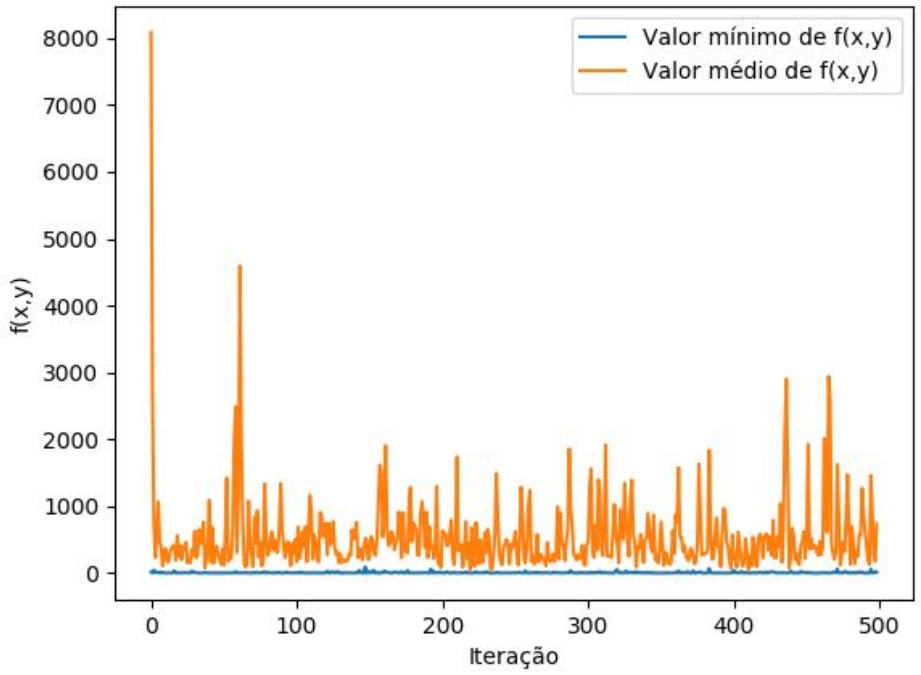
Ao final da implementação do algoritmo, testes foram feitos, sendo divididos em 2 etapas, a saber:

1. Gráfico que mostra os valores de aptidão médios e máximos das posições do espaço de busca armazenados em x ao longo das iterações;
2. Gráfico que mostra a evolução dos valores de aptidão médios e máximos obtidos pelas partículas ao longo das iterações.

Na etapa 1, o algoritmo foi executado em busca dos valores de aptidão máximos e médios das posições do espaço de busca armazenados na matriz x ao longo das iterações. Para

tanto, foram utilizadas 10 partículas, o número de iterações utilizado foi 500 e as velocidades mínima e máxima de movimentação das partículas utilizadas foram -2 e 2, respectivamente. Na Figura 2, é mostrado o gráfico obtido, sendo que o eixo x é o número de iterações e o eixo y é o valor de aptidão máximo, em azul, ou médio, em laranja, encontrado em x a cada iteração.

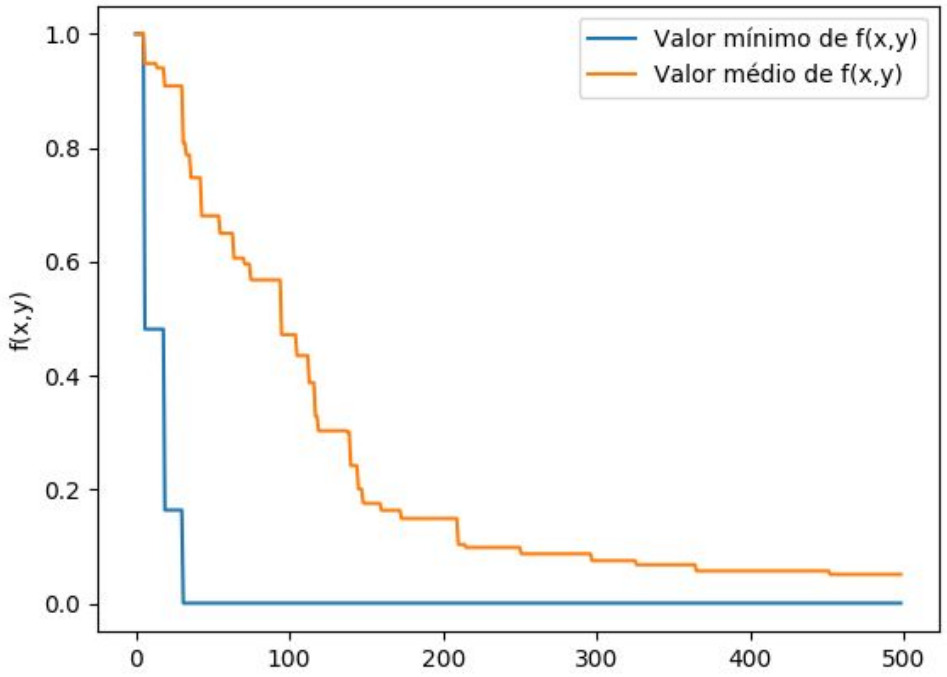
Figura 2: Gráfico do valor mínimo e médio da aptidão dos elementos de x em cada iteração.



Fonte: Elaborada pelo próprio autor.

Já na etapa 2, o algoritmo foi executado em busca de se verificar a evolução do melhor valor e do valor médio de aptidão encontrado pelas partículas ao longo das iterações, sendo que em laranja estão os valores médios de aptidão das partículas e em azul estão as melhores aptidões das partículas em cada iteração.

Figura 3: Gráfico dos valores de aptidão médios e máximos das partículas a cada iteração.



Fonte: Elaborada pelo próprio autor.

2.2. Solução do problema do Caixeiro Viajante

O objetivo desta aplicação é implementar, utilizando ACO, um algoritmo que consiga obter o menor caminho no problema do Caixeiro Viajante com 52 cidades.

Detalhes da implementação do algoritmo

A função principal do programa possui dez parâmetros, a saber:

- **cidade** - matriz com 52 linhas e 2 colunas, em que as linhas armazenam as coordenadas das 52 cidades;
- **max_it** - número máximo de iterações, utilizado como critério de parada;
- **alfa** - peso definido pelo usuário;
- **beta** - peso definido pelo usuário;
- **ro** - taxa de decaimento do feromônio;
- **n** - número de formigas;
- **e** - número de cidades, sendo que 52 cidades foram utilizadas;
- **q** - peso definido pelo usuário;
- **feromonio_inicial** - valor utilizado para inicializar a matriz de feromônios;

- **b** - número de formigas elitistas.

Inicialmente, a matriz feromônio, cuja função é armazenar a quantidade de feromônio colocada em cada aresta do grafo, é inicializada com todos os elementos possuindo o valor do parâmetro feromonio_inicial. Enquanto o número máximo de iterações não for atingido, os passos a seguir são executados: Primeiro, de maneira probabilística e utilizando a matriz de feromônios, são calculadas as rotas das formigas. A cidade a ser visitada pela formiga é calculada utilizando roleta, sendo que quanto maior a probabilidade de uma cidade vizinha que ainda não foi visitada ser visitada, maior é a sua porção na roleta. Após isso, verifica-se as rotas calculadas e caso alguma possua distância menor que a melhor rota das iterações anteriores, as variáveis melhor e Lmelhor que armazenam a melhor rota e a distância da melhor rota são atualizadas. Ao final da iteração, a matriz feromônio é atualizada conforme visto na disciplina.

Experimentos e Testes

Ao final da implementação do algoritmo, testes foram feitos, sendo divididos em 2 etapas, a saber:

1. Gráfico que mostre a distância total do menor caminho encontrado ao longo das iterações;
2. Gráfico mostrando as cidades em suas respectivas coordenadas e o traçado do menor caminho encontrado.

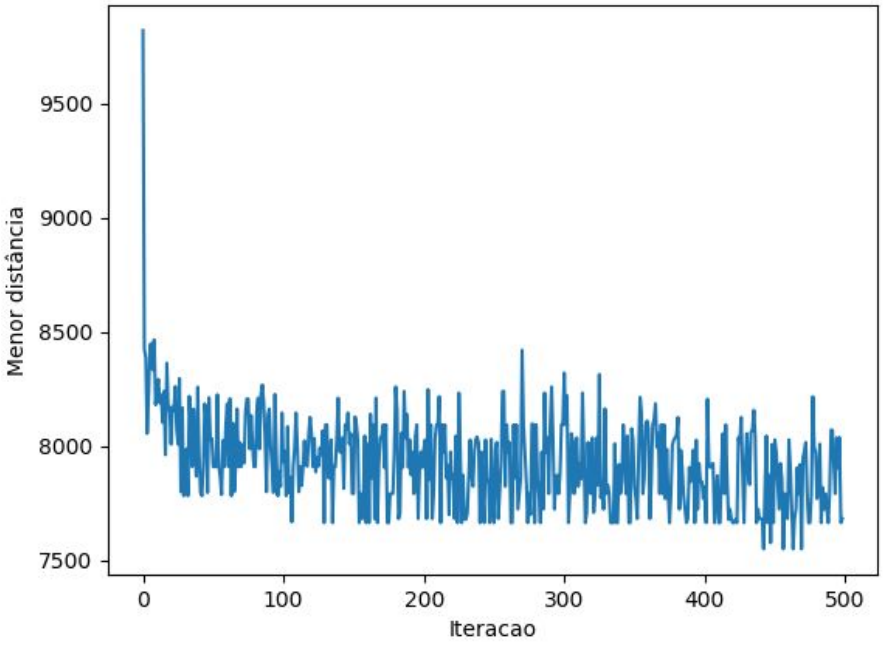
Na etapa 1, o algoritmo foi executado em busca da rota mais curta obtida em cada iteração, sendo que os parâmetros de entrada utilizados estão expostos na Tabela 1. A Figura 4 mostra a variação da maior aptidão dos elementos da população em cada iteração, enquanto que a Figura 5 mostra a evolução da distância da rota mais curta, armazenada em Lmelhor, do algoritmo.

Tabela 1: Parâmetros de entrada utilizados no algoritmo ACO.

Parâmetro de entrada	Valor
max_it	500
alfa	1
beta	5
ro	0.5
n	52
e	52
q	100
feromonio_inicial	10^{-6}
b	5

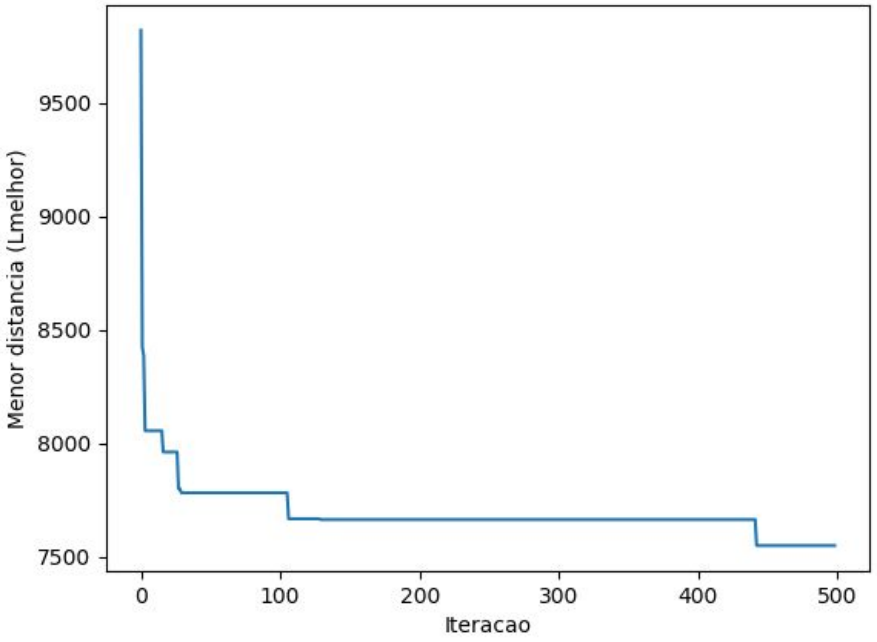
Fonte: Elaborado pelo próprio autor.

Figura 4: Variação da maior aptidão dos elementos da população em cada iteração.



Fonte: Elaborado pelo próprio autor.

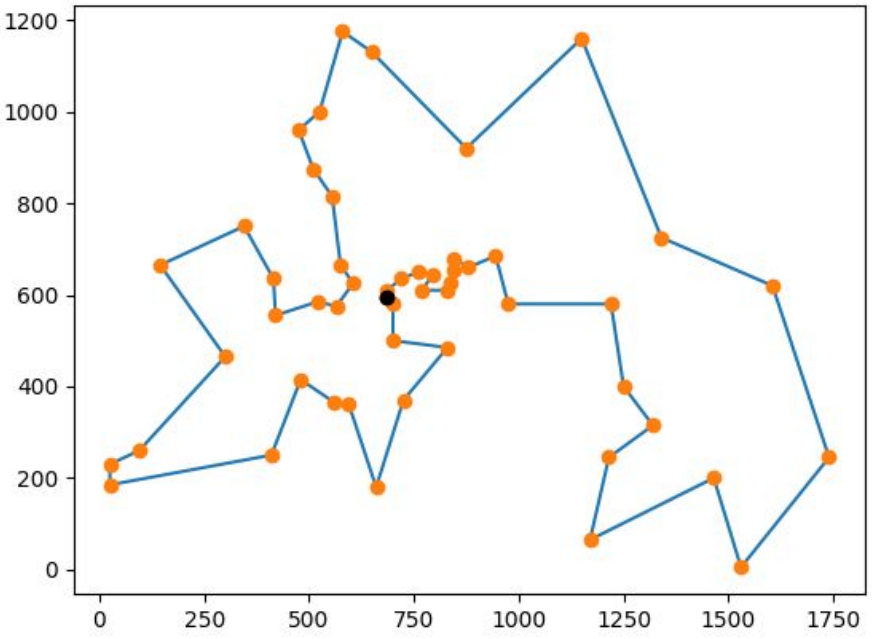
Figura 6: Evolução do valor da variável Lmelhor ao longo das iterações.



Fonte: Elaborado pelo próprio autor.

Já na etapa 2, a menor rota encontrada é mostrada na Figura 6, sendo que os pontos do gráfico indicam as cidades. A distância total desta rota é 7548.9927.

Figura 7: Rota com a menor distância obtida pelo ACO.



Fonte: Elaborado pelo próprio autor.

3. Conclusão

Na Aplicação 1, conforme mostra a Figura 2, o valor de aptidão médio das posições das partículas oscilam ao longo das iterações. Tal fato indica que o algoritmo explora o espaço de busca até a última iteração, objetivando encontrar soluções próximas a solução ótima a cada iteração. Com relação a convergência, a Figura 3 mostra que tanto o valor de aptidão médio quanto o valor máximo de aptidão das partículas convergiram para valores próximos a solução ótima do problema em poucas iterações. Assim, pode-se concluir que o algoritmo gera boas soluções em poucas iterações já que, em 20 testes do algoritmo, sendo que cada teste executou 500 iterações, o valor médio obtido foi 0.00375, com desvio padrão 0.00345.

Já na Aplicação 2, uma boa solução para o problema foi obtida com menos de 500 iterações, conforme mostra o gráfico da Figura 5. Por sua vez, por meio do comportamento do gráfico mostrado na Figura 4, é possível concluir que o algoritmo procura encontrar melhores rotas ao longo de todas as iterações. A distância total da melhor rota obtida, mostrada na Figura 5, é 7548.9927 Km. Assim, comparando com a solução ótima do problema, que é 7544.3659 Km, o solução obtida é bem próxima da solução ótima.

Diante do exposto, pode-se concluir que o PSO pode ser utilizado quando o propósito é encontrar soluções próximas a solução ótima para os problemas de otimização de funções, assim com o ACO pode ser utilizado para solucionar o problema do caixeiro viajante, de maneira rápida, simples de implementar e com resultados satisfatórios.

4. Referências Bibliográficas

DE CASTRO, Leandro Nunes. **Fundamentals of natural computing: basic concepts, algorithms, and applications**. CRC Press, 2006.