Implementação de Otimização por Colônia de Formigas para o Problema do Caixeiro-Viajante

Lucas Machado Cogrossi[1], **João Pedro Nunes Gonçalves** [1]

[1]Departamento de Ciência da Computação
Universidade Estadual do Centro Oeste (Unicentro)
Guarapuava, PR – Brasil

contact@lucascogrossi.com, joao.pedronunes266@gmail.com

Abstract. *This paper presents the implementation of the Ant Colony Optimization (ACO) algorithm to solve the Traveling Salesman Problem (TSP). The algorithm was developed in Python and simulates the collective behavior of ants searching for food, using pheromone trails to guide the search for optimal solutions. The method employs probabilistic transition rules based on pheromone intensity and heuristic information (inverse of distance), along with pheromone evaporation and deposition mechanisms. The implementation allows visual tracking of the best path per iteration and the overall best path, demonstrating the progressive refinement of solutions. Experimental results show that ACO efficiently finds high-quality approximate solutions for the TSP.*

Resumo. *Este artigo apresenta a implementação do algoritmo Ant Colony Optimization (ACO) para resolver o Problema do Caixeiro Viajante (PCV). O algoritmo, desenvolvido em Python, simula o comportamento coletivo de formigas em busca de alimento, utilizando trilhas de feromônio para guiar a busca por soluções ótimas. O método emprega regras de transição probabilística baseadas na intensidade de feromônio e informação heurística (inverso da distância), além de mecanismos de evaporação e deposição de feromônio. A implementação permite acompanhar visualmente o melhor caminho de cada iteração e o melhor caminho global, evidenciando o refinamento gradual das soluções. Resultados experimentais indicam que o ACO encontra soluções aproximadas de alta qualidade de forma eficiente.*

1. Introdução

O Problema do Caixeiro Viajante (PCV) é um clássico problema de otimização combinatória na área de ciência da computação e pesquisa operacional [Cook 2011]. Ele consiste em determinar o caminho mais curto que permite a um vendedor visitar um conjunto de cidades exatamente uma vez e retornar à cidade de origem. Apesar de sua formulação simples, o problema é computacionalmente desafiador, pois o número de possíveis rotas cresce factorialmente com o número de cidades, tornando soluções exatas inviáveis para instâncias grandes [Naga 1999].

Devido à complexidade do problema, técnicas de otimização aproximada são frequentemente empregadas. Entre essas técnicas, o algoritmo Ant Colony Optimization (ACO) se destaca por sua capacidade de explorar grandes espaços de soluções de forma eficiente, inspirando-se no comportamento coletivo de colônias de formigas [**?**]. O ACO

foi proposto por Marco Dorigo em 1992 e desde então tem sido amplamente aplicado em problemas de otimização combinatória.

Na natureza, formigas são capazes de encontrar o caminho mais curto entre sua colônia e uma fonte de alimento através da deposição e detecção de feromônios. Caminhos mais curtos tendem a acumular mais feromônio ao longo do tempo, pois mais formigas conseguem percorrê-los, criando um mecanismo de retroalimentação positiva que guia a colônia para soluções cada vez melhores [**?**].

A escolha do Ant Colony Optimization se justifica pela sua eficácia comprovada no PCV, sendo este o problema para o qual o algoritmo foi originalmente desenvolvido [**?**]. Além disso, a implementação permite comparar o melhor caminho de cada iteração com o melhor caminho encontrado em todas as iterações, oferecendo uma visualização intuitiva do processo de otimização.

2. Desenvolvimento

A implementação do algoritmo Ant Colony Optimization para o Problema do Caixeiro Viajante foi realizada em Python utilizando a biblioteca `Pygame` para visualização das soluções ao longo das iterações. Nesta seção, são detalhados os principais componentes do algoritmo, incluindo a representação das soluções, regra de transição probabilística, mecanismos de feromônio e parâmetros do algoritmo.

2.1. Representação da Solução

Cada formiga constrói uma solução completa para o PCV, representada por uma lista ordenada de inteiros, onde cada inteiro corresponde a uma cidade específica. Por exemplo, para um conjunto de 10 cidades, uma solução pode ser representada como:

```
[0, 3, 7, 2, 5, 1, 8, 4, 6, 9]
```

Esta representação garante que cada cidade seja visitada exatamente uma vez, respeitando a restrição do Problema do Caixeiro Viajante. A ordem dos elementos define a rota que será percorrida pela formiga, incluindo o retorno à cidade inicial.

2.2. Regra de Transição Probabilística

A escolha da próxima cidade a ser visitada por uma formiga é feita de forma probabilística, combinando a intensidade do feromônio na aresta e a informação heurística (inverso da distância). A probabilidade de uma formiga na cidade i escolher a cidade j como próximo destino é dada por:

$$P_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in \mathcal{N}_i} [\tau_{ik}]^{\alpha} \cdot [\eta_{ik}]^{\beta}} \tag{1}$$

onde:

- τ_{ij} é a quantidade de feromônio na aresta entre as cidades i e j;
- $\eta_{ij} = \frac{1}{d_{ij}}$ é a informação heurística, sendo d_{ij} a distância entre as cidades;
- α é o parâmetro que controla a importância relativa do feromônio;
- β é o parâmetro que controla a importância relativa da heurística;

- $\mathcal{N}_i$ é o conjunto de cidades ainda não visitadas pela formiga.

A seleção da próxima cidade utiliza o método da **roleta**, onde cidades com maior probabilidade têm mais chance de serem escolhidas, mas a decisão não é determinística, permitindo exploração do espaço de soluções.

2.3. Evaporação de Feromônio

Para evitar a convergência prematura e permitir que o algoritmo "esqueça" decisões ruins, os níveis de feromônio são reduzidos a cada iteração através do processo de evaporação:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \tag{2}$$

onde $\rho \in (0, 1]$ é a taxa de evaporação. Este mecanismo garante que trilhas menos utilizadas percam gradualmente sua atratividade, direcionando a busca para regiões mais promissoras do espaço de soluções.

2.4. Deposição de Feromônio

Após todas as formigas construírem suas soluções, o feromônio é depositado nas arestas percorridas. A quantidade de feromônio depositada é inversamente proporcional ao comprimento da rota encontrada:

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \tag{3}$$

onde Q é uma constante e L_k é o comprimento total da rota construída pela formiga k. Desta forma, formigas que encontram rotas mais curtas depositam mais feromônio, reforçando caminhos de alta qualidade.

A atualização do feromônio em cada aresta é dada por:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{4}$$

onde m é o número total de formigas.

2.5. Parâmetros do Algoritmo

Os principais parâmetros do ACO utilizados na implementação são:

- **Número de Formigas (m)**: quantidade de formigas que constroem soluções em cada iteração;
- **Taxa de Evaporação (ρ)**: controla a velocidade de "esquecimento" das trilhas de feromônio;
- **Alpha (α)**: peso dado à intensidade do feromônio na regra de transição;
- **Beta (β)**: peso dado à informação heurística (distância) na regra de transição;
- **Constante Q**: fator de escala para o depósito de feromônio.

2.6. Fluxo Geral do Algoritmo

O algoritmo segue o fluxo clássico do Ant Colony Optimization:

1. Inicialização da matriz de feromônios com valores uniformes.
2. Cada formiga constrói uma solução completa usando a regra de transição probabilística.
3. Cálculo da distância total para cada solução construída.
4. Evaporação dos feromônios em todas as arestas.
5. Deposição de feromônio nas arestas percorridas pelas formigas.
6. Atualização do melhor caminho da iteração atual e do melhor caminho global.
7. Repetição dos passos 2 a 6 até o critério de parada ser atingido.

O pseudocódigo abaixo ilustra a implementação principal do ACO para o PCV:

Listing 1. Pseudocódigo do Ant Colony Optimization para o PCV

```
ACO_TSP(Cidades, NumFormigas, TxEvaporacao, Alpha, Beta, Q,
    NumIteracoes):

    # Inicializa o
    MatrizDistancias <- CalcularDistancias(Cidades)
    MatrizFeromonio <- InicializarFeromonio(NumCidades, valor=1.0)
    MelhorGlobal <- Nenhum
    MelhorDistanciaGlobal <- Infinito

    para iteracao = 1 at  NumIteracoes fa a:

        ToursFormigas <- []
        DistanciasFormigas <- []

        # Cada formiga constr i uma solu  o
        para k = 1 at  NumFormigas fa a:
            CidadeInicial <- EscolherAleatoriamente(Cidades)
            Tour <- [CidadeInicial]
            NaoVisitadas <- Cidades - {CidadeInicial}

            enquanto NaoVisitadas n o vazio fa a:
                CidadeAtual <- UltimaCidade(Tour)
                ProximaCidade <- SelecionarProximaCidade(
                    CidadeAtual, NaoVisitadas,
                    MatrizFeromonio, MatrizDistancias,
                    Alpha, Beta
                )
                Adicionar ProximaCidade em Tour
                Remover ProximaCidade de NaoVisitadas
            fim enquanto

            Distancia <- CalcularDistanciaTotal(Tour, MatrizDistancias)
            Adicionar Tour em ToursFormigas
            Adicionar Distancia em DistanciasFormigas

            # Atualiza melhor global
            se Distancia < MelhorDistanciaGlobal ent o
                MelhorDistanciaGlobal <- Distancia
                MelhorGlobal <- Tour
```

```
            fim se
        fim para

        # Evapora  o do ferom nio
        para cada aresta (i,j) fa a:
            MatrizFeromonio[i][j] <- (1 - TxEvaporacao) *
               MatrizFeromonio[i][j]
        fim para

        # Deposi  o de ferom nio
        para cada formiga k fa a:
            DeltaFeromonio <- Q / DistanciasFormigas[k]
            para cada aresta (i,j) em ToursFormigas[k] fa a:
                MatrizFeromonio[i][j] <- MatrizFeromonio[i][j] +
                   DeltaFeromonio
                MatrizFeromonio[j][i] <- MatrizFeromonio[j][i] +
                   DeltaFeromonio
            fim para
        fim para

        # (Opcional) Visualiza  o do progresso
        MelhorIteracao <- MelhorTour(ToursFormigas, DistanciasFormigas)
        MostrarCaminho(MelhorIteracao, MelhorGlobal)

    fim para

    retornar MelhorGlobal, MelhorDistanciaGlobal
```

A visualização do algoritmo permite observar simultaneamente o melhor caminho de cada iteração (na metade inferior da tela, em verde) e o melhor caminho encontrado até o momento (na metade superior, em vermelho), proporcionando uma compreensão intuitiva da evolução das soluções ao longo do tempo.

3. Resultados

A Tabela 1 apresenta os resultados de cinco execuções distintas do algoritmo Ant Colony Optimization, utilizando diferentes configurações dos hiperparâmetros (Taxa de Evaporação, Número de Formigas, Alpha e Beta) para resolver o Problema do Caixeiro Viajante com 10 cidades, executado por 500 iterações em todos os casos.

Table 1. Resultados das Execuções do Ant Colony Optimization

N° Execução	Tx. Evaporação	N° Formigas	Alpha	Beta	N° Iterações	Melhor Distância
1	0.5	20	1.0	2.0	500	802.47
2	0.3	30	1.0	3.0	500	756.82
3	0.7	20	1.0	2.0	500	845.31
4	0.5	50	2.0	5.0	500	721.95
5	0.3	40	1.5	4.0	500	698.43

A análise da Tabela 1 permite observar a influência dos parâmetros na qualidade das soluções encontradas:

- A **Execução 5**, com taxa de evaporação baixa (0.3), maior número de formigas (40) e valores balanceados de Alpha (1.5) e Beta (4.0), obteve a **melhor distância**

(698.43), indicando que a combinação de persistência do feromônio com maior exploração do espaço de soluções foi eficaz.
- A **Execução 3**, que utilizou a maior taxa de evaporação (0.7), resultou na **pior distância** (845.31), confirmando que uma evaporação muito rápida pode prejudicar a convergência, pois as trilhas de feromônio não persistem tempo suficiente para guiar as formigas.
- O parâmetro **Beta** mostrou-se importante: valores mais altos (como 4.0 e 5.0 nas Execuções 4 e 5) favoreceram a exploração de caminhos mais curtos, dando maior peso à informação heurística.
- O **número de formigas** também influenciou positivamente: as execuções com mais formigas (40 e 50) encontraram melhores soluções, pois maior diversidade de caminhos explorados aumenta a probabilidade de encontrar rotas de alta qualidade.

O acompanhamento visual confirmou que o algoritmo converge de caminhos iniciais aleatórios e complexos para rotas progressivamente mais otimizadas. A convergência típica ocorreu entre 100 e 300 iterações, após as quais melhorias se tornaram incrementais.

4. Conclusão

A implementação do algoritmo Ant Colony Optimization para o Problema do Caixeiro Viajante permitiu compreender de forma prática os princípios da inteligência de enxame e sua aplicação em problemas de otimização combinatória. Durante o desenvolvimento, foi possível observar como a escolha dos parâmetros — taxa de evaporação, número de formigas, alpha e beta — influencia diretamente na qualidade das soluções encontradas e na velocidade de convergência do algoritmo.

O mecanismo de feromônio, inspirado no comportamento real de colônias de formigas, mostrou-se eficaz para balancear exploração (busca por novas soluções) e intensificação (refinamento de soluções promissoras). A taxa de evaporação adequada é crucial: valores muito altos impedem a formação de trilhas estáveis, enquanto valores muito baixos podem levar à estagnação em ótimos locais.

O acompanhamento visual do algoritmo, implementado com `Pygame`, facilitou a compreensão do processo de otimização, permitindo comparar o melhor caminho da iteração atual com o melhor caminho encontrado em todas as iterações, mostrando claramente o refinamento gradual das soluções através do acúmulo de feromônio nos melhores caminhos.

De maneira geral, a experiência de implementar o ACO foi positiva, evidenciando a eficácia do algoritmo para o PCV e sua elegância conceitual ao traduzir um comportamento biológico em uma meta-heurística computacional poderosa. A implementação prática consolidou conceitos teóricos de inteligência de enxame e demonstrou a aplicabilidade desses métodos em problemas reais de otimização.

5. References

References

Cook, W. J. (2011). *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press.

Naga, P. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Computers Operations Research*, 26(1):1–13.