Universidade Federal
de São João del-Rei

Departamento de Ciência da Computação

Vítor Augusto Niess Soares Fonseca

Relatório Algoritmo Colônia de Formigas Para Caixeiro Viajante

São João del-Rei, Agosto de 2024

Sumário

1 Introdução

Os algoritmos de colônia de formigas (ACOs) são uma classe de algoritmos inspirados no comportamento coletivo das formigas em busca de alimentos. Esses algoritmos baseiam-se na comunicação indireta entre os agentes (formigas) por meio de feromônios, uma substância química que as formigas depositam no solo para marcar os caminhos percorridos. A partir dessa comunicação, as formigas conseguem encontrar o caminho mais curto entre a colônia e a fonte de alimento, processo que pode ser adaptado para resolver problemas complexos de otimização.

O problema do caixeiro viajante (PCV) é um problema clássico de otimização combinatória que consiste em determinar a rota mais curta que permite a um caixeiro viajante visitar um conjunto de cidades exatamente uma vez, retornando à cidade de origem. Este problema é representado por uma matriz de distâncias entre as cidades, e a solução ótima é a rota que minimiza a distância total percorrida. O PCV é conhecido por ser NP-difícil, o que implica que não existe uma solução algorítmica eficiente que encontre a solução ótima em tempo polinomial para instâncias de grande porte.

Neste trabalho, é proposta a utilização do algoritmo de colônia de formigas para resolver o problema do caixeiro viajante. Especificamente, será investigado como a combinação de fatores como a influência do feromônio, a visibilidade e a atualização da matriz de feromônio contribuem para a eficiência do algoritmo em encontrar soluções próximas do ótimo para o problema descrito. Além disso, será avaliada a evolução das melhores soluções ao longo do tempo, com o intuito de compreender o comportamento do algoritmo durante o processo de otimização.

2 Metodologia

Para abordar o problema do caixeiro viajante utilizando o algoritmo de colônia de formigas (Ant Colony Optimization - ACO), foram implementadas um conjunto de funções em Python que compõem o núcleo do algoritmo. As principais etapas do algoritmo incluem a inicialização da matriz de feromônio, a construção dos caminhos pelas formigas, a atualização da matriz de feromônio e a execução do ciclo iterativo até a convergência ou até que o número máximo de iterações seja atingido.

2.1 Carregamento da Matriz de Distâncias

A função `carregar_matriz_distancias(filename)` é responsável por carregar a matriz de distâncias entre as cidades a partir de um arquivo de texto. Esta matriz é utilizada para calcular as distâncias percorridas pelas formigas em suas jornadas.

2.2 Inicialização da Matriz de Feromônio

A função `inicializar_feromonio(num_cidades)` cria a matriz de feromônio, que é inicialmente preenchida com um valor muito pequeno (10^{-16}). Esta matriz será atualizada ao longo das iterações, com base nos caminhos percorridos pelas formigas e nas respectivas distâncias.

2.3 Cálculo das Probabilidades de Escolha

A função `calcular_probabilidade(cidade_atual, feromonio, distancias, cidades_nao_v alfa, beta)` calcula a probabilidade de uma formiga escolher a próxima cidade a ser visitada, com base na quantidade de feromônio depositado e na visibilidade (inverso da distância). A função considera o parâmetro α, que controla a influência do feromônio, e β, que controla a influência da visibilidade.

2.4 Construção do Caminho

A função `construir_caminho(feromonio, distancias, num_cidades)` gera uma permutação dos vértices (cidades) para cada formiga, utilizando as probabilidades calculadas na função anterior. A formiga inicia sua jornada em uma cidade aleatória e, em seguida, escolhe as próximas cidades com base nas probabilidades até que todas as cidades tenham sido visitadas.

2.5 Cálculo da Distância Total

A função `calcular_distancia_total(caminho, distancias)` calcula a distância total percorrida por uma formiga ao visitar todas as cidades na ordem especificada pelo vetor `caminho`. Esta função é utilizada para avaliar a qualidade dos caminhos construídos.

2.6 Atualização da Matriz de Feromônio

A função `atualizar_feromonio(feromonio, caminhos, distancias, taxa_evaporacao, Q)` é responsável por atualizar a matriz de feromônio após cada iteração. O feromônio em cada aresta ij é atualizado com base na equação:

$$T_{ij} = (1-\rho) \cdot T_{ij} + \sum_{k} \frac{Q}{L_k}$$

onde ρ é a taxa de evaporação, Q é uma constante, e L_k é a distância total percorrida pela k-ésima formiga. A soma é realizada apenas para as arestas que fazem parte dos caminhos percorridos pelas formigas.

2.7 Execução do Algoritmo de Colônia de Formigas

A função `ACO(distancias, num_formigas, max_it)` implementa o ciclo iterativo do algoritmo de colônia de formigas. Ela inicializa a matriz de feromônio, constrói os caminhos para todas as formigas em cada iteração, atualiza a matriz de feromônio e mantém o melhor caminho encontrado. A função retorna o melhor caminho e a menor distância encontrada após a última iteração.

2.8 Plotagem da Evolução das Melhores Distâncias

A função `plotar_evolucao(melhores_distancias)` gera um gráfico que mostra a evolução da melhor distância encontrada ao longo das iterações. Este gráfico é útil para visualizar a convergência do algoritmo e a melhoria das soluções ao longo do tempo.

3 Resultados e Discussões

Para avaliar o desempenho do algoritmo de colônia de formigas na resolução do problema do caixeiro viajante, foram realizados diversos testes utilizando diferentes configurações de parâmetros. Inicialmente, foi utilizada a seguinte configuração:

- `max_it = 150`: número máximo de iterações.
- `alfa = 1`: influência do feromônio.
- `beta = 5`: influência do custo do caminho.
- `taxa_evaporacao = 0.5`: taxa de evaporação do feromônio.
- `Q = 100`: constante utilizada na atualização do feromônio.

Com essa configuração inicial, os primeiros testes foram realizados utilizando a entrada do arquivo `lau15`, cujo gráfico de desempenho pode ser observado na Figura 1. A solução ótima foi encontrada rapidamente, sendo uma distância de 291, geralmente entre uma ou duas iterações.

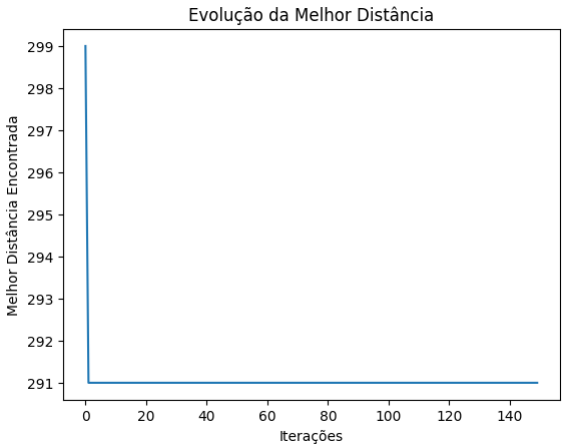

Figura 1: Desempenho do algoritmo utilizando o arquivo lau15.

Para os testes subsequentes, foi utilizado o arquivo `sgb128`. Nestes testes, o algoritmo apresentou uma boa solução, com uma distância de 21741 para o melhor caminho encontrado, conforme ilustrado na Figura 2. No entanto, foi observado que a solução se estagnou em um ótimo local, sem melhorias ao longo das iterações restantes.

Mesmo após realizar ajustes nos parâmetros, não foi possível evitar essa estagnação. Por exemplo, em uma das iterações, o algoritmo alcançou uma distância de 22632, conforme mostrado na Figura 3, mas permaneceu preso em um ótimo local sem conseguir progredir para soluções melhores.

Esses resultados indicam que, embora o algoritmo de colônia de formigas seja eficaz em encontrar soluções boas rapidamente, ele pode se deparar com desafios ao escapar de ótimos locais em instâncias mais complexas, como a do arquivo `sgb128`. Ajustes finos nos parâmetros ou estratégias adicionais podem ser necessários para melhorar o desempenho em tais casos.

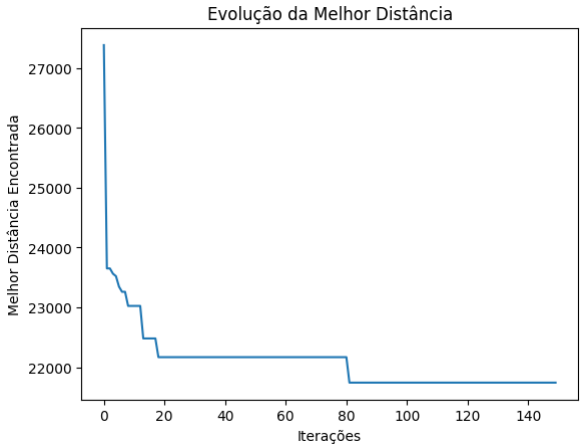

Figura 2: Desempenho do algoritmo utilizando o arquivo sgb128, com uma solução de 21741 como distância do melhor caminho.

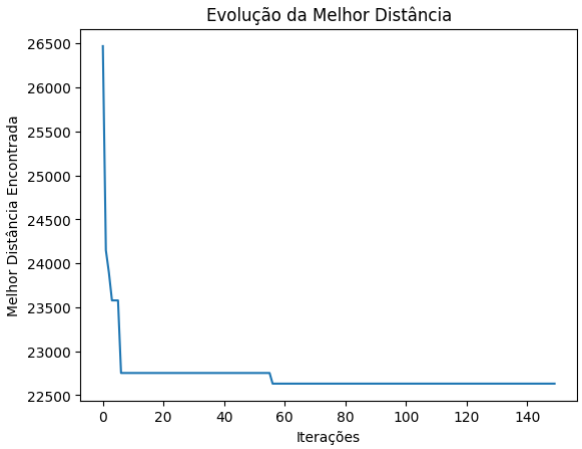

Figura 3: Desempenho do algoritmo utilizando o arquivo sgb128, com uma solução de 22632 como distância do melhor caminho.

4 Considerações Finais

Os testes realizados com diferentes configurações de parâmetros demonstraram que a eficácia do algoritmo pode ser significativamente influenciada pela escolha dos mesmos. A configuração inicial, ao ser aplicada ao problema com menos cidades, como o `lau15`, mostrou-se extremamente eficaz, encontrando a solução ótima em poucas iterações. No entanto, ao aplicar a mesma configuração a problemas mais complexos, como o `sgb128`, o algoritmo encontrou uma boa solução rapidamente, mas ficou estagnado em ótimos locais, mostrando a necessidade de ajustes mais refinados.

Observou-se que, mesmo com alterações nos parâmetros, a estagnação não pôde ser completamente evitada, o que sugere que o algoritmo, em sua configuração atual, pode não ser suficientemente robusto para escapar de ótimos locais em instâncias mais complexas. Futuras melhorias podem incluir o ajuste dinâmico dos parâmetros durante a execução do algoritmo ou a implementação de estratégias adicionais para evitar estagnação, como a introdução de diversidade na população de soluções.

Em suma, este trabalho não só ampliou a compreensão sobre o uso de algoritmos de colônia de formigas para a resolução de problemas de otimização combinatória, mas também destacou a importância da escolha adequada dos parâmetros. O conhecimento adquirido é valioso para

futuras pesquisas e aplicações, proporcionando uma base sólida para o desenvolvimento de estratégias mais eficazes em problemas de otimização mais desafiadores.