PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS (PUC GOIÁS)

I DESAFIO EM OTIMIZAÇÃO COM METAHEURÍSTICAS

Equipe: Correntinópolis - TOBA

Integrantes: Alysson Victor Almeida Souza e Davi Saraiva Coelho.

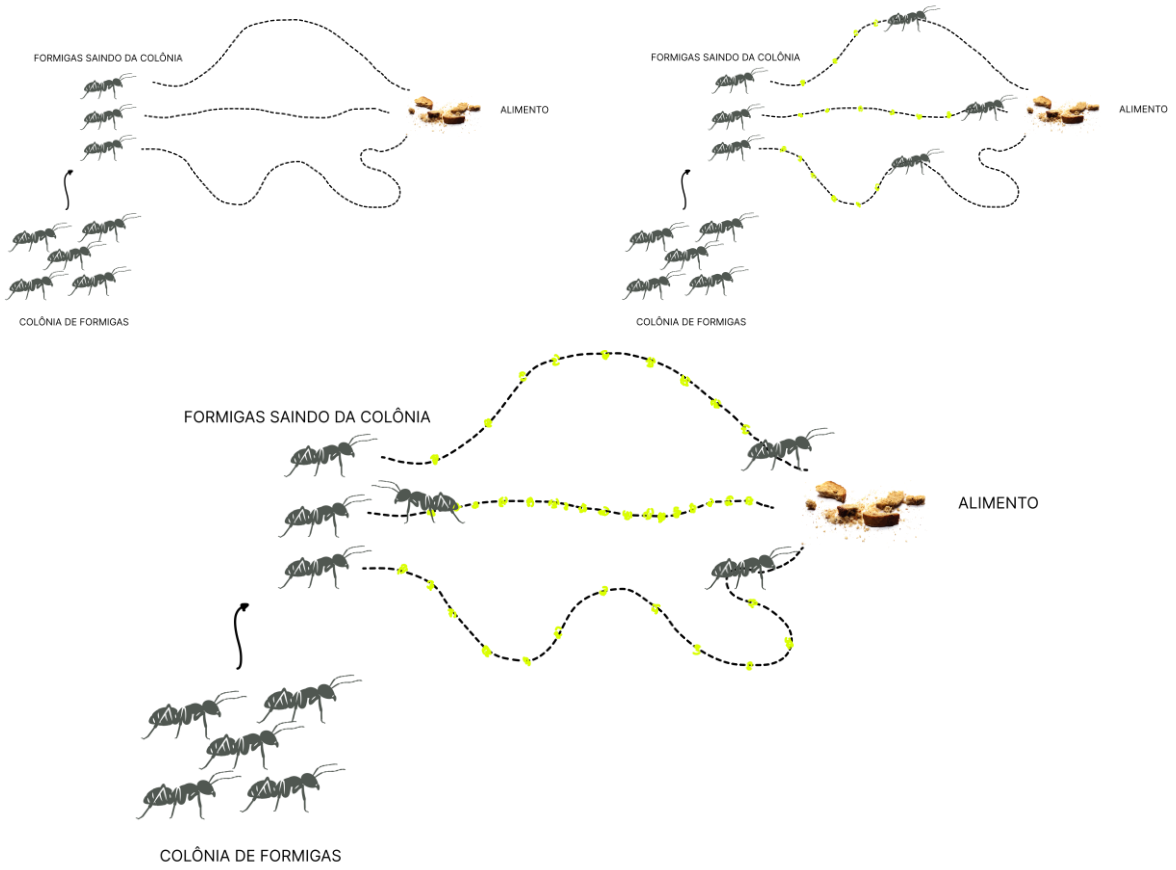
Introdução

O Problema do Caixeiro Viajante (PCV) é um problema de otimização combinatória amplamente conhecido por sua característica de ser NP-difícil, ou seja, sua solução exata não é viável com o poder de processamento dos computadores atuais. Porém, existem métodos que nos permitem alcançar resultados extremamente satisfatórios utilizando algoritmos baseados em meta-heurísticas, como o Algoritmo Genético (GA), Otimização por Colônia de Formigas (ACO), entre outros. O algoritmo escolhido para a realização deste desafio foi o ACO, que pode ser utilizado para resolver problemas de otimização combinatória estáticos e dinâmicos. O PCV é um problema estático, levando em conta que as instâncias estão definidas e não são alteradas durante a execução da solução (DORIGO, 2006).

A Otimização por Colônia de Formigas foi introduzida por Marco Dorigo em sua tese de PhD, onde através da análise do comportamento das formigas foi percebido um padrão em que o inseto saia da colônia a procura de alimento e depositava feromônio enquanto realizava a busca, assim, ao alcançar seu objetivo e retornar à colônia, a formiga deixa um rastro de feromônios no caminho trilhado. O rastro de feromônios é evaporado com o tempo, porém outra formiga que está saindo da colônia tende a ser atraída pelo feromônio, fazendo com que o caminho que foi percorrido pela formiga anterior tenha seus feromônios reforçados pela atual.

Representação da Otimização por Colônia de Formigas

As imagens abaixo representam a distribuição dos feromônios nas rotas com o passar do tempo. É perceptível que a rota mais curta tende a acumular feromônios em uma velocidade superior que as demais rotas.



Com base nisso, para construir o ACO é necessário definir certos parâmetros, sendo eles: **a quantidade de iterações, número de formigas, fator de ampliação, α, β, ρ** e Q .

A quantidade de iterações define quantas vezes cada uma das n formigas irão construir uma solução, e baseada nessas soluções, realizaremos a evaporação e atualização dos feromônios. No nosso algoritmo, uma solução é definida como: uma formiga sair da cidade inicial (determinada como cidade 0), percorrer todas as outras cidades sem passar duas vezes por nenhuma delas e retornar à inicial. Diante disso, uma vez que a formiga está na cidade i , a escolha do caminho será feita através de uma roleta que utilizará as probabilidades da formiga sair da cidade i e ir para as cidades adjacentes.

O cálculo das probabilidades é feito utilizando a seguinte equação:

$$p_{ij}^{k} = \frac{[T_{ij}]^{\alpha}[N_{ij}]^{\beta}}{\sum_{z \in n_i^k}[T_{ij}]^{\alpha}[N_{ij}]^{\beta}}$$

Na equação acima, P_{ij}^{k} representa a probabilidade da formiga k escolher a rota ij . A variável T_{ij} representa a quantidade de feromônios na aresta ij , já N_{ij} representa a atratividade do caminho ij que definimos como o inverso da distância (quanto menor a distância, maior a atratividade) e é calculada como $N_{ij} = \frac{1}{Dij}$ onde D_{ij} é a distância de i para j . O parâmetro **α** controla a influência do feromônio na probabilidade de transição e o **β** é responsável pela influência da atratividade na probabilidade de transição.

Após todas as formigas construírem suas soluções, utilizamos **ρ** como parâmetro da taxa de evaporação e atualização dos feromônios, e $\Delta T_{ij}^{k} = \frac{Q}{d_k}$que é calculada utilizando Q como constante de atualização do feromônio e d_k como a distância percorrida pela formiga. Na equação de atualização também utilizamos m representando o total de formigas:

$$T_{ij} = (1-\rho)T_{ij} + \left(\sum_{k=1}^{m} \Delta T_{ij}^{k}\right)$$

Variações Implementadas e Testes

Com o intuito de acelerar a convergência das formigas para os melhores caminhos e afastá-las dos piores, realizamos testes no algoritmo e chegamos à conclusão que uma evaporação dupla do pior caminho encontrado pela formiga na respectiva iteração poderia ser uma boa alternativa. Esta segunda evaporação é definida pelo fator de ampliação, que é uma constante definida para esse objetivo. Com isso, a evaporação dos feromônios é realizada em todas as cidades e as rotas que foram mais recorrentes acabam recebendo mais feromônios, finalizando dessa forma a primeira iteração.

Pseudocódigo:

1. Inicializar:

1.1 - Defina o número de formigas, número de iterações e parâmetros alfa, beta, p e Q

1.2 - leia quantidade de cidades

1.3 - Construção do grafo

1.4 - inicialize Feromônios e Probabilidades

2. Para cada iteração até o critério de parada:

2.1 - Para cada formiga faça

2.1.1 - construa uma solução usando a regra de transição probabilística

2.1.2 - calcule o comprimento da solução

2.1.3 - armazene a solução

2.1.4 - atualize a melhor solução global

2.2 Atualização de feromônios

2.2.1 - evapore os feromônios

2.2.2 - evapore novamente os feromônios do pior caminho

2.2.3 - atualize os feromônios locais

fim

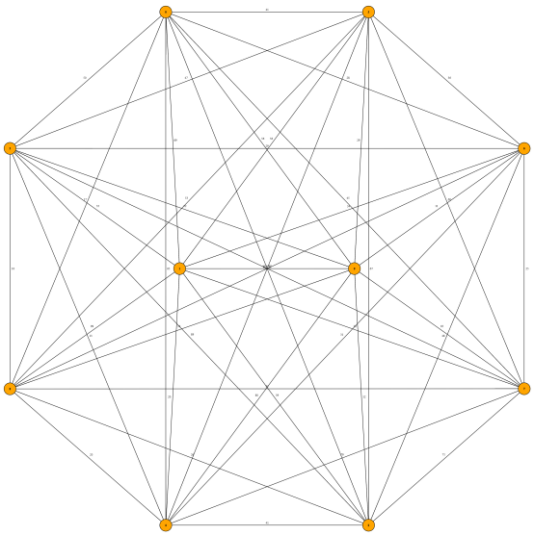
3. retorne a melhor solução encontrada

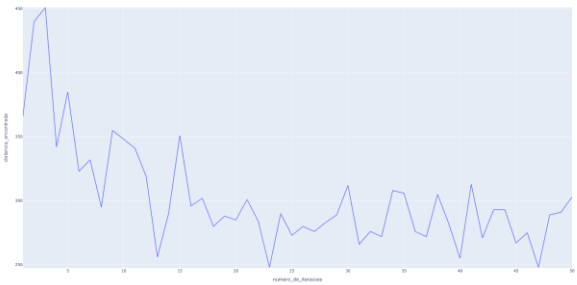
fim procedimento

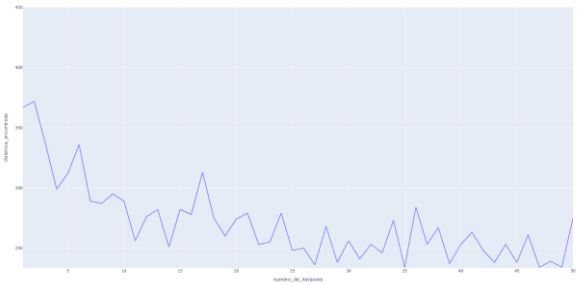
Exemplo de grafo

Diante disso, quanto maior for a quantidade de iterações, mais otimizada será a matriz de feromônios e as rotas feitas pelas formigas serão cada vez melhores. Entretanto, experimentos realizados provaram que, ainda que as iterações façam uma diferença significativa, uma grande quantidade de formigas é extremamente relevante para o alcance de uma boa solução, como no exemplo a seguir em que temos um Grafo com muitas possibilidades de caminhos para se encontrar uma solução.

O grafo utilizado possui 10 vértices e 45 arestas, e a menor distância é 226. Os gráficos mostram as distâncias encontradas conforme as formigas finalizavam uma iteração.

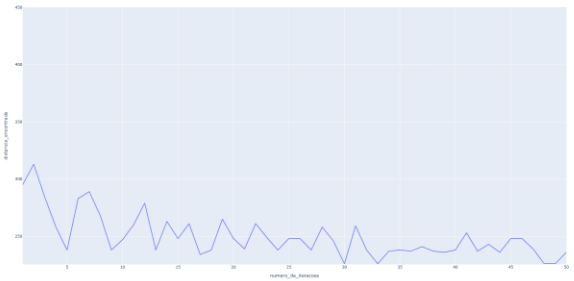


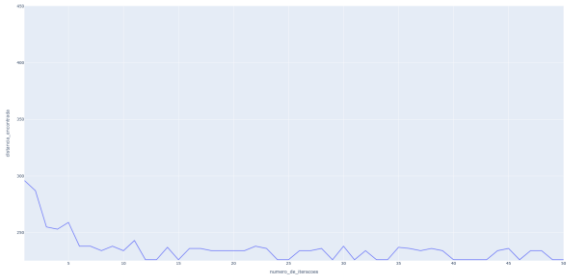


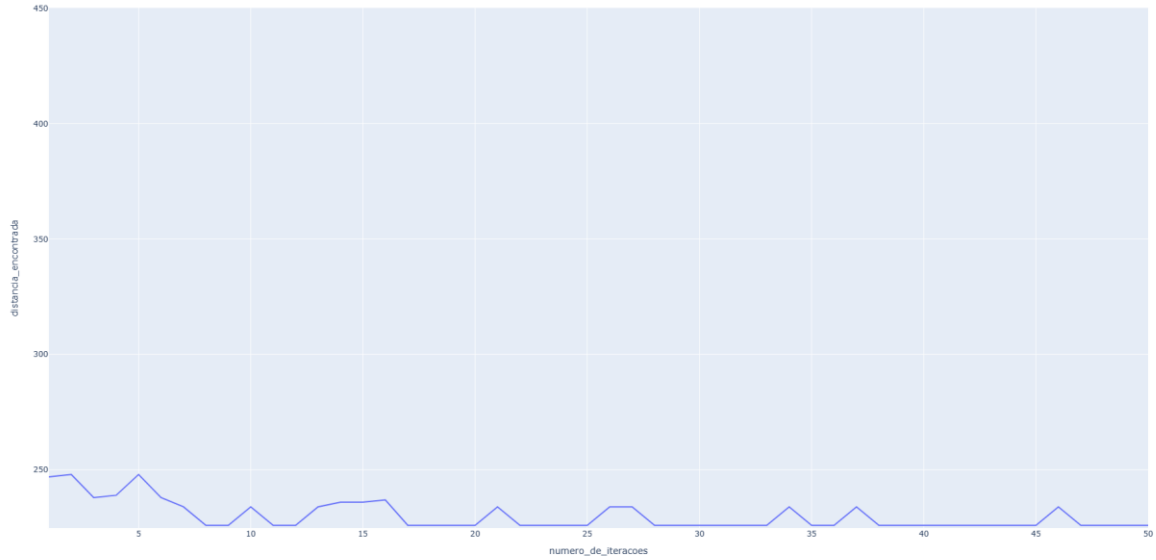


O primeiro teste é mostrado no gráfico 1, onde foi utilizada apenas uma formiga e a menor distância encontrada foi 248. Já no segundo, foram utilizadas três formigas e as menor distância encontrada foi 234.

A partir deste ponto, testamos os resultados para uma quantidade maior de formigas, sendo os seguintes gráficos de 10, 50 e 100 formigas respectivamente:







No terceiro gráfico, já é possível notar resultados muito mais promissores, tendo em vista que na primeira iteração já foi possível encontrar um caminho muito melhor do que nos casos em que menos formigas eram utilizadas, além de que o caminho ótimo foi encontrado após 30 iterações. No último teste já foi possível alcançar o caminho ótimo após 8 iterações, e a partir deste momento as formigas tenderam a percorrer caminhos ótimos com mais frequência.

Análise de Alterações nos Parâmetros

Diante desses testes, foi possível chegar à conclusão que um balanceamento do número de iterações e quantidade de formigas deve ser feito, pois uma quantidade exagerada de iterações se mostra desnecessária à medida que a quantidade de formigas é aumentada.

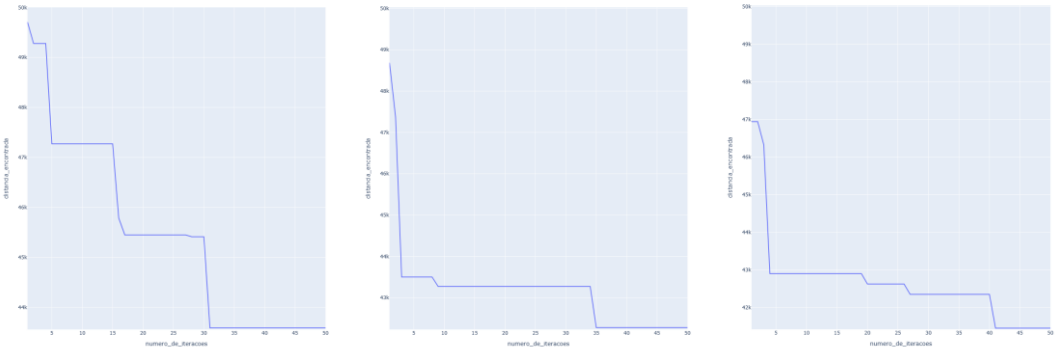
Também foram realizados testes para definir os parâmetros **α, β, ρ** e Q . Ao realizar testes com grafos que possuíam mais de 100 cidades, começamos a ter problemas para encontrar soluções viáveis, pois grande parte das formigas convergiam para caminhos onde as distâncias encontradas estavam longe das almejadas. Assim, após realizar testes e mudanças nos parâmetros, decidimos manter o **β** = 2, tornando assim a atratividade mais relevante e fazendo com que as formigas percorressem caminhos com peso menor. Outra mudança que foi feita e que

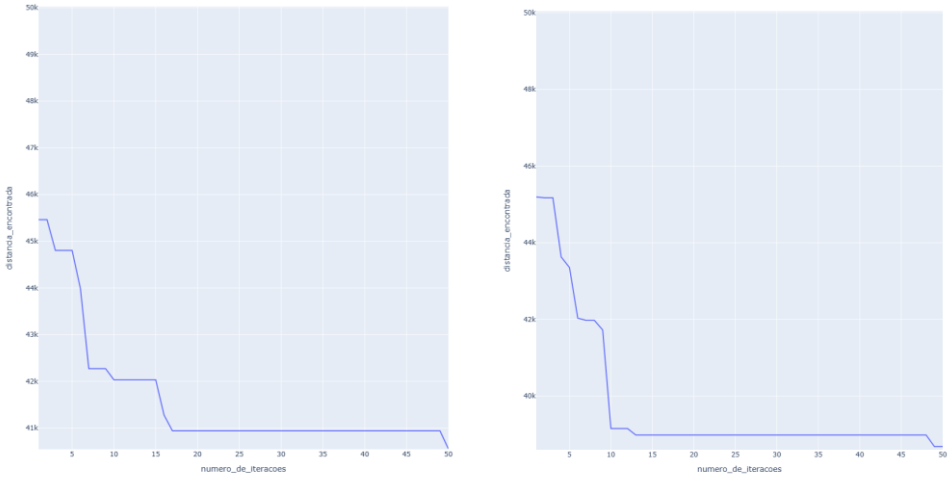
trouxe resultados significativos foram mudanças no cálculo da probabilidade, decidimos alterar a forma como calculávamos N_{ij} no numerador e denominador da equação. No numerador alteramos para $\frac{2}{D_{ij}}$ e no somatório do denominador alteramos para $\frac{0.75}{D_{ij}}$.

Ambiente de Desenvolvimento e Testes de Tempo

O algoritmo foi executado em um Computador com um processador Intel Core i7 8565U, 8 GB de memória RAM e SSD 400GB. O sistema operacional utilizado durante o desenvolvimento foi o Windows 11 utilizando o Visual Studio Code como Interface de Desenvolvimento (IDE) na linguagem de programação C/C++.

Para ter uma métrica, realizamos testes em um grafo com 144 vértices, seguem as comparações a seguir para 10, 20, 50, 100, 200:





Os tempo de execução do código foi: 0.7350 segundos, 1.8360 segundos, 4.5330 segundos, 5.7950 segundos e 11.2250 segundos respectivamente.

Referências bibliográficas:

DORIGO, M.; STÜTZLE, T. **Ant Colony Optimization**. IEEE Comput. Intell. Mag., vol. 1, no. 4, pp. 33, Nov. 2006.

DORIGO, M.; CARO G. D.; GAMBARDELLA, L. M. **Ant Algorithms for Discrete Optimization**, Artificial Life, 5,2, p. 137–172, 1999.

Maru, 2020. P, NP, **NP Hard and NP Complete Problem** | Reduction | NP Hard and NP Compete | Polynomial Class.