Universidade Federal Do Paraná

Comparação entre os algoritmos ABC e ACO no contexto do problema do caixeiro viajante TSP

Arthur Barreto Godoi | arthur.godoi@ufpr.br

Gustavo Gabriel Ripka | gustavo.ripka@ufpr.br

Pedro Henrique Gurski De Oliveira | pedrogurski@ufpr.br

Ulisses Curvello Ferreira | ulissesferreira@ufpr.br

Curitiba, 10 de dezembro de 2025

Disciplina: CI1170 - Tópicos em Computação Bioinspirada

Professor: Eduardo Jaques Spinosa

Sumário

1. Introdução

Este trabalho apresenta uma análise comparativa entre dois algoritmos bioinspirados de otimização: o Artificial Bee Colony (ABC) e o Ant Colony Optimization (ACO), aplicados ao clássico Problema do Caixeiro Viajante (TSP). Inicialmente realizados experimentos com instâncias de 30 cidades, avaliando qualidade de solução, tempo de execução e convergência.

1.1. Objetivos

Os objetivos deste trabalho são:

- Implementar e comparar os algoritmos ABC e ACO para o TSP;
- Avaliar o desempenho de ambos os algoritmos em termos de qualidade de solução e tempo computacional;
- Investigar o impacto de ajustes paramétricos no desempenho do ABC;
- Analisar os padrões de convergência de cada algoritmo;
- Identificar vantagens e limitações de cada abordagem.

2. Artificial Bee Colony (ABC)

2.1. Inspiração Biológica

A lógica desse algoritmo vem da maneira como as abelhas se organizam para encontrar comida. O que faz a colmeia ser eficiente não é o esforço de uma abelha só, mas o fato de que elas conversam e trocam informações o tempo todo sobre onde estão as melhores flores. É uma inteligência coletiva: se uma abelha acha um lugar bom, o grupo inteiro fica sabendo e foca o trabalho ali. Para isso funcionar no código, a colônia é dividida em três grupos:

Empregadas: Responsáveis por explorar fontes de alimento já conhecidas e memorizadas.
Observadoras: Avaliando os dados das empregadas para escolher o próximo destino.
Batedoras: Procuram novas áreas aleatoriamente assim que as fontes se esgotam.

2.2. Descrição do Algoritmo

No contexto do Problema do Caixeiro Viajante, cada "fonte de alimento"representa uma rota possível. O algoritmo busca o caminho mais curto através de um ciclo repetitivo dividido em três etapas:

1. **Fase das Empregadas**: Cada abelha empregada trabalha em uma rota específica e tenta modificá-la levemente (uma perturbação local) para encontrar um atalho. Se essa nova versão for melhor que a anterior, a abelha atualiza sua memória com o caminho mais curto.

2. **Fase das Observadoras**: Nesta etapa, as observadoras selecionam quais rotas serão exploradas com base na qualidade delas. Rotas mais curtas têm maior probabilidade de serem escolhidas. Isso garante que o algoritmo não perca tempo com caminhos ruins e foque seus recursos em refinar as soluções mais promissoras.

3. **Fase das Batedoras**: Se uma rota não apresenta melhorias após várias tentativas (atingindo um limite predefinido), ela é abandonada. A abelha então assume o papel de batedora e gera um caminho totalmente novo e aleatório. Esse mecanismo é essencial para evitar que o algoritmo fique "preso"em caminhos razoáveis, forçando a exploração de novas áreas do mapa em busca do melhor resultado global.

2.3. Ant Colony Optimization (ACO)

2.4. Inspiração Biológica

A lógica aqui vem do jeito que as formigas reais conseguem achar o caminho mais curto entre o ninho e a comida sem precisar de um mapa. Elas usam o feromônio, que é uma trilha química que deixam no chão.

Funciona assim: quando uma formiga acha um caminho curto, ela vai e volta mais rápido, acumulando mais "cheiro"naquele rastro. As outras formigas sentem esse rastro mais forte e começam a segui-lo. Com o tempo, o grupo inteiro acaba convergindo para o melhor caminho através dessa comunicação química simples, mas muito eficiente.

2.5. Descrição do Algoritmo

No código, o ACO simula esse processo criando "formigas artificiais"para percorrer as cidades do PCV. A construção da solução segue estes passos:

1. **Construção da Rota**: Em vez de escolherem a próxima cidade ao acaso, as formigas olham para dois fatores: a distância (heurística) e a quantidade de feromônio acumulada naquela aresta. Quanto mais feromônio uma rota tiver e mais curta ela for, maior a chance da formiga passar por ali.

2. **Deposição de Feromônio**: Assim que todas as formigas terminam suas rotas, as trilhas que formaram os caminhos mais curtos recebem um "bônus"de feromônio. É isso que diz para as formigas da próxima rodada que aquela região é promissora.

3. **Evaporação**: Para evitar que o algoritmo fique "viciado"em um caminho que parecia bom no começo (mas não era o melhor de todos), o feromônio evapora um pouco a cada ciclo. Isso apaga rastros antigos e ruins, forçando o enxame a estar sempre buscando rotas novas ou reforçando apenas as que continuam sendo curtas. É o que mantém o equilíbrio entre refinar o que já foi achado e continuar explorando o mapa.

3. Problema Estudado e Abordagem Computacional

3.1. Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é como planejar a rota de entrega mais barata possível. Imagine que você tem uma lista de cidades para visitar: você precisa passar por todas, apenas uma vez cada, e terminar a viagem voltando para onde começou. O desafio é fazer isso percorrendo a menor distância total.

3.2. Instâncias de Teste

Para verificar se os algoritmos funcionam de verdade, precisamos criar um ambiente de teste que fosse além de um exemplo simples. O objetivo não era apenas ver se o ABC ou o ACO resolviam o problema, mas sim entender o limite de cada um e como eles reagem quando o mapa começa a ficar grande demais.

Em vez de usar apenas um mapa estático, optmos por testar os algoritmos em cenários variados para entender como eles se comportam conforme a dificuldade aumenta. As instâncias foram geradas de forma euclidiana, distribuindo cidades aleatoriamente.

Para tornar a análise robusta, desenvolvemos um script de automação em Bash que executa uma bateria de testes granulares. Esse script varia o número de cidades (de 10 a 30) e ajusta a população de agentes (abelhas e formigas de 10 a 90). No total, o sistema realiza dezenas de combinações automáticas, salvando os tempos e custos em um arquivo CSV para posterior análise estatística. Isso permitiu observar não apenas qual algoritmo chega na melhor rota, mas qual deles escala melhor quando o mapa fica mais complexo.

3.3. Implementação

Toda a lógica dos algoritmos e a geração de gráficos foram desenvolvidas em Python3. Utilizamos a biblioteca NumPy para realizar os cálculos de matriz de distância e feromônio de forma rápida, e o Matplotlib para desenhar as rotas finais e os gráficos de convergência.

A estrutura do código foi montada para ser modular: o script Python recebe os parâmetros (quantidade de cidades e agentes) diretamente do terminal, o que facilitou a integração com o script de testes automatizados e garantiu que cada execução fosse

medida de forma isolada, tanto em termos de qualidade da solução quanto de tempo computacional.

4. Resultados

4.1. Experimento 1

O ACO produziu soluções 36,2% melhores que o ABC inicial, porém com tempo aproximadamente duas vezes maior.

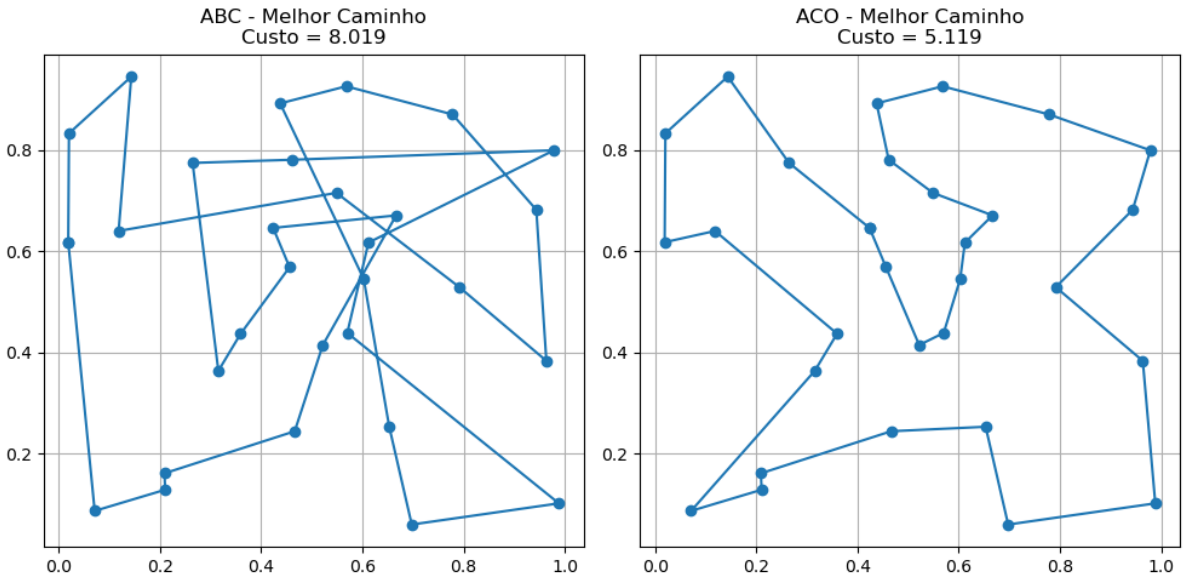

Figura 1: Comparação do custo entre os dois algoritmos no primeiro teste

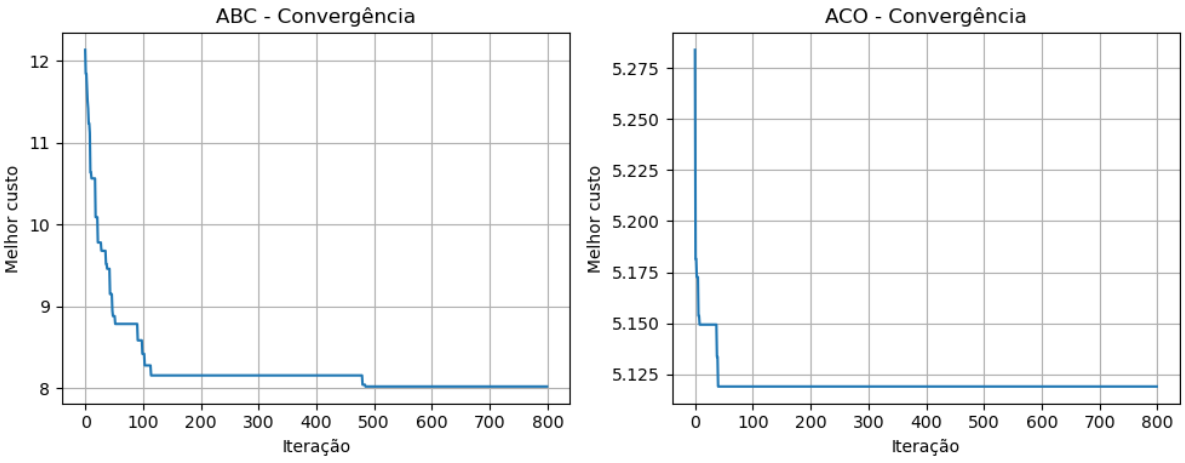

Figura 2: Diferença de convergência entre os dois algoritmos no primeiro teste

4.2. Experimento 2

A versão otimizada do ABC melhorou sua qualidade em 33,6% e tornou-se 14,6 vezes mais rápida que o ACO.

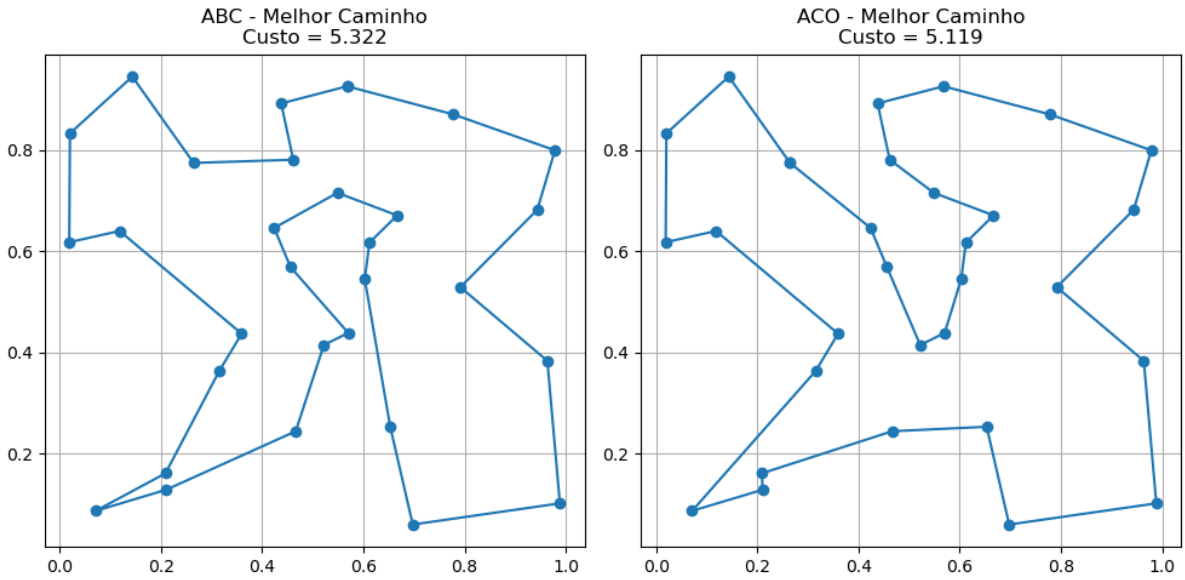

Figura 3: Comparação do custo entre os dois algoritmos no segundo teste

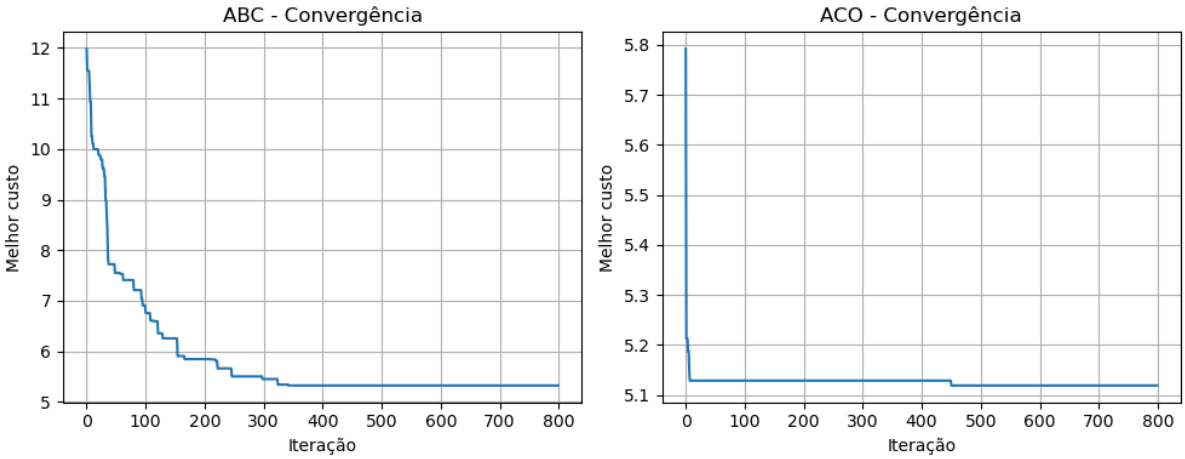

Figura 4: Diferença de convergência entre os dois algoritmos no segundo teste

4.3. Análise de Convergência

O ABC apresentou convergência inicial rápida, enquanto o ACO convergiu de forma mais gradual e estável.

4.4. Teste de Estresse

Realizamos um teste de estresse, onde foi aumentando o número de cidades, abelhas e formigas conforme os testes.

Ao analisarmos o esforço comparado ao tempo gasto de cada algoritmo, podemos notar que o algoritmo ACO levou 8x mais tempo do que o algoritmo ABC. Conforme a imagem abaixo:

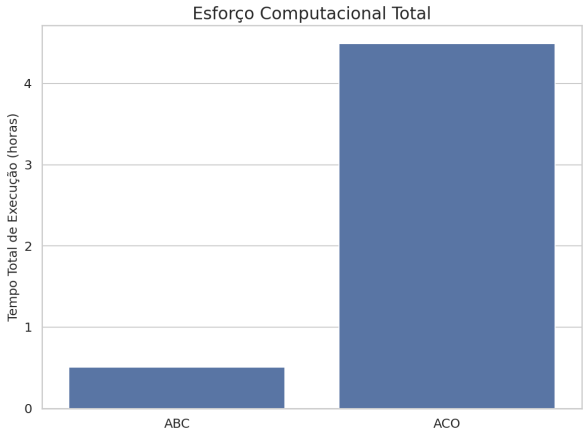

Figura 5: Diferença de Esforço Computacional entre os dois algoritmos em relação ao tempo

Podemos observar no gráfico abaixo que o algoritmo ABC tem maiores dificuldades ao encontrar a melhor solução para o problema conforme o número de cidades aumentam. Para valores baixos, os dois algoritmos mostram uma igualdade nos resultados. Para melhores resultados, podemos aumentar o número de abelhas ou o número de iterações.

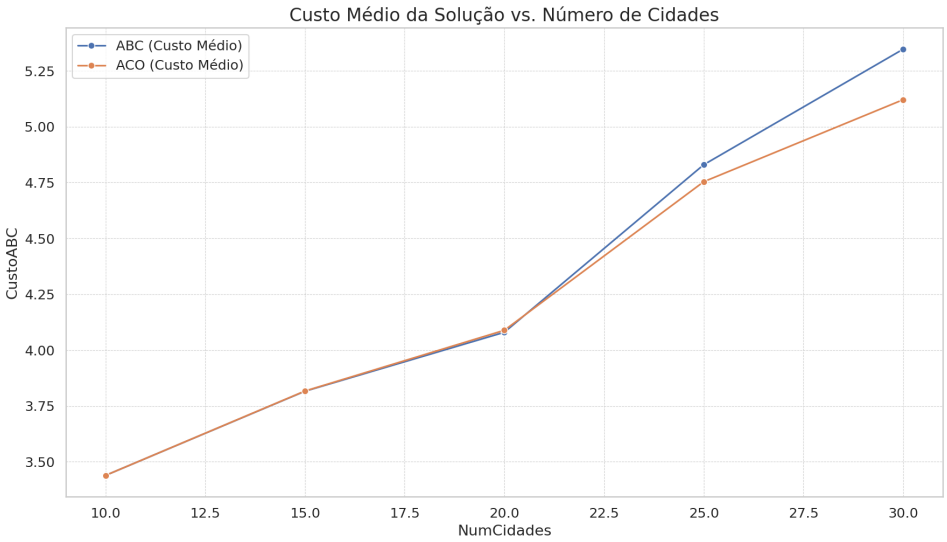

Figura 6: Comparação de Custo Médio da Solução comparado ao número crescente de cidades dos algoritmos

Analisando o gráfico abaixo, podemos ver que o tempo de execução médio do algoritmo ACO foi muito maior e crescente, conforme o número de cidades aumentam, comparado ao algoritmo ABC que teve tempos relativamentes parecidos, com uma leve crescente.

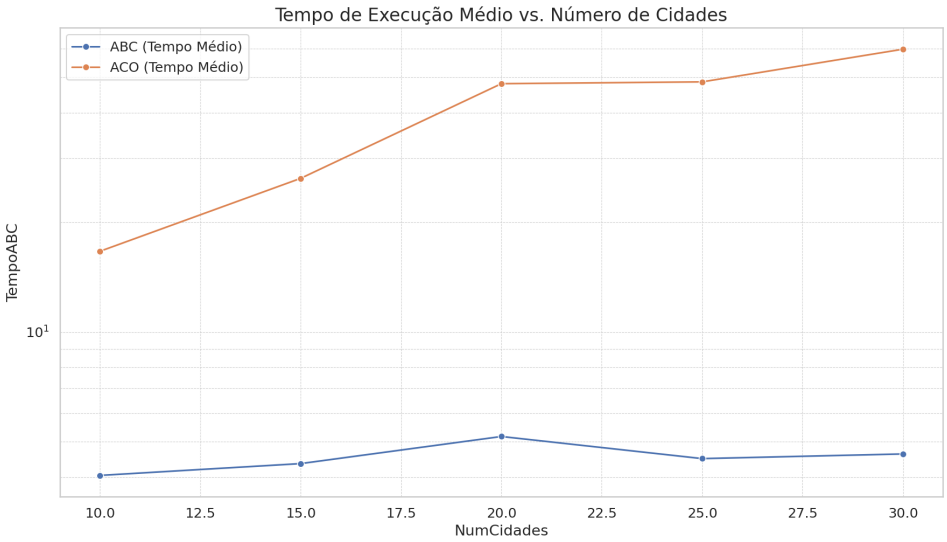

Figura 7: Comparação de Tempo de Execução Médio de cada algoritmo comparado ao número de cidades

5. Conclusões

5.1. Principais Conclusões

O ABC otimizado tornou-se competitivo com o ACO, sendo muito mais eficiente em tempo, porém, mais custoso computacionalmente (dificuldade em achar um ótimo global para número de cidades muito grandes). Ambos mostraram desempenho bons quando parametrizados corretamente.

Podemos concluir que, em cenários que não se há muito tempo, o algoritmo ABC é o ideal, pois é rápido e nos traz resultados satisfatórios. Em cenários contrários, onde se há tempo para realizar o cálculo e é exigido uma precisão maior, é aconselhável o algoritmo ACO, onde nos traz um custo menor, mas levando um bom tempo para o resultado.

5.2. Trabalhos Futuros

Para trabalhos futuros, pretendemos estender os experimentos para instâncias maiores, realizar análise estatística e investigar algoritmos híbridos. Podemos realizar aprimoramentos no algoritmo ACO e efetuar novos testes.

Referências

[1] Glaucus Augustus. *Inteligência de Exame e o Algoritmo das Abelhas.* Monografia, IME-USP, 2009. Disponível em: https://www.ime.usp.br/~gold/cursos/2009/mac5758/GlaucusSwarm.pdf

[2] Oliveira, Pedro Jorge de A. *Localização e Mapeamento Simultâneos via Otimização por Exame.* Dissertação de Mestrado, UERJ, 2019. Disponível em: https://www.pel.uerj.br/bancodissertacoes/Dissertacao_Pedro_Jorge_de_Oliveira.pdf

[3] Vaz, Eduardo A. *Uso de Inteligência de Abelhas no Planejamento da Expansão do Sistema de Transmissão.* Projeto de Diplomação, UFRGS, 2011. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/33097/000787144.pdf

[4] Oliveira, Ioná M. S. *Inteligência de Enxames na Otimização de Recargas de Reatores Nucleares.* Tese de Doutorado, COPPE-UFRJ, 2013. Disponível em: https://www.nuclear.ufrj.br/images/Tese_Iona_Maghali.pdf

[5] McCaffrey, James. *Bee Colony Algorithms para Problemas Difíceis.* MSDN Magazine, 2015. Disponível em: https://learn.microsoft.com/pt-br/archive/msdn-magazine/2011/april/msdn-magazine-natural-algorithms-use-bee-colony-algorithms-to-solve-impossible-problems

[6] McKee, Amberle. *Algoritmos de Inteligência de Exame em Python.* Datacamp, atualizado em 10 out. 2024. Disponível em: https://www.datacamp.com/pt/tutorial/swarm-intelligence

[7] Pinho de Sá, Luiz Henrique. *Inteligência de Enxames: Abelhas.* Slideshare, 2015. Disponível em: https://pt.slideshare.net/slideshow/inteligncia-de-enxames-abelhas-127384604/127384604