UNIVERSIDADE ESTADUAL DO PIAUÍ
CENTRO DE TECNOLOGIA E URBANISMO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Jardson Costa Silva

ANÁLISE ENTRE ALGORITMO GENÉTICO E OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS CAPACITADOS

TERESINA
2025

Jardson Costa Silva

ANÁLISE ENTRE ALGORITMO GENÉTICO E OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS CAPACITADOS

Monografia de Trabalho de Conclusão de Curso apresentado na Universidade Estadual do Piauí – UESPI como parte dos requisitos para conclusão do Curso de Bacharelado em Ciência da Computação.

Orientador: Dr. Constantino Augusto Dias Neto

TERESINA

2025

S586a Silva, Jardson Costa.
Análise entre algoritmo genético e otimização por colônia de formigas aplicados ao problema de roteamento de veículos capacitados / Jardson Costa Silva. - 2025.
72 f.: il.

Monografia (graduação) - Universidade Estadual do Piauí - UESPI, Bacharelado em Ciência da Computação, Campus Poeta Torquato Neto, Teresina-PI, 2025.
"Orientador: Prof. Dr. Constantino Augusto Dias Neto".

1. Problema de Roteamento de Veículos Capacitado (PRVC). 2. Algoritmo Genético. 3. Otimização por Colônia de Formigas (OCF). 4. Otimização de Rotas. I. Dias Neto, Constantino Augusto . II. Título.

CDD 004.2

Ficha elaborada pelo Serviço de Catalogação da Biblioteca da UESPI
Francisca Carine Farias Costa (Bibliotecário) CRB-3ª/1637

ANÁLISE ENTRE ALGORITMO GENÉTICO E OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS APLICADOS AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS CAPACITADOS

Jardson Costa Silva

Monografia de Trabalho de Conclusão de Curso apresentado na Universidade Estadual do Piauí – UESPI como parte dos requisitos para conclusão do Curso de Bacharelado em Ciência da Computação.

Documento assinado digitalmente
gov.br
CONSTANTINO AUGUSTO DIAS NETO
Data: 07/07/2025 14:34:13-0300
Verifique em https://validar.iti.gov.br

Dr. Constantino Augusto Dias Neto, Dsc.
Orientador

Nota da Banca Examinadora: 9,0

Banca Examinadora:

Documento assinado digitalmente
gov.br
CONSTANTINO AUGUSTO DIAS NETO
Data: 07/07/2025 14:33:08-0300
Verifique em https://validar.iti.gov.br

Dr. Constantino Augusto Dias Neto, Dsc.
Presidente

Documento assinado digitalmente
gov.br
MAURICIO REGO MOTA DA ROCHA
Data: 07/07/2025 18:21:34-0300
Verifique em https://validar.iti.gov.br

Dr. Maurício Rêgo da Mota Rocha, Dsc.
Membro

Documento assinado digitalmente
gov.br
LILIAM BARROSO LEAL
Data: 10/07/2025 11:00:51-0300
Verifique em https://validar.iti.gov.br

Dra. Liliam Barroso Leal, Dsc.
Membro

Dedico este trabalho a todas as pessoas residentes em cidades pequenas, que, apesar das adversidades, buscam incansavelmente uma formação acadêmica.

AGRADECIMENTOS

Ao concluir esta importante etapa da minha formação acadêmica, gostaria de expressar minha sincera gratidão a todos que tornaram este percurso possível.

Primeiramente, agradeço a Deus pela vida, saúde e força concedidas durante esta trajetória, permitindo-me superar os desafios e chegar até aqui.

À minha família, pelo apoio incondicional, compreensão nos momentos de ausência e por sempre acreditarem no meu potencial, mesmo quando eu duvidava. Em especial à minha mãe Ivanete Costa, que sempre foi meu pilar de sustentação, exemplo de força e determinação, cujo amor, incentivo constante e sacrifícios tornaram possível cada conquista em minha vida. A meu pai Edivaldo de Sousa, que mesmo sem estudos formais, me ensinou o valor do trabalho árduo e da honestidade através de seus sacrifícios diários, garantindo que eu pudesse ter as oportunidades que ele não teve.

À minha irmã Cida (Maria Aparecida), pelas conversas descontraídas, pelo apoio emocional e por ser minha confidente nos momentos mais desafiadores deste caminho.

À minha tia Vera Lúcia e meus primos Élio Lopes e Carliene Lopes, que me apoiaram desde o princípio e foram fundamentais para que eu pudesse chegar até aqui. Seus exemplos de superação e determinação sempre foram uma inspiração.

Aos meus tios José Sousa e Raimundo Matos, cujo apoio e generosidade no início desta etapa foram essenciais para que eu pudesse dar os primeiros passos rumo à realização deste sonho.

Ao meu orientador, Prof. Dr. Constantino Augusto Dias Neto, pela dedicação, paciência e valiosas contribuições que foram fundamentais para a realização deste trabalho. Seus conhecimentos e orientações transformaram significativamente minha visão acadêmica.

Aos professores do curso de Ciência da Computação, que compartilharam generosamente seus conhecimentos e experiências ao longo destes anos, proporcionando-me uma formação sólida e inspiradora.

A todos que, direta ou indiretamente, fizeram parte desta caminhada e contribuíram para a conclusão desta etapa, meu sincero muito obrigado.

“Não é o mais forte que sobrevive, nem o mais inteligente,
mas o que melhor se adapta às mudanças.”
(Leon C. Megginson, 1963)

RESUMO

O mundo moderno demanda crescente eficiência operacional para que produtos sejam entregues com agilidade e qualidade, proporcionando uma experiência superior aos clientes. Nesse contexto, a otimização de rotas representa um desafio central para empresas do setor logístico. O Problema de Roteamento de Veículos Capacitado (PRVC) destaca-se como uma das variantes mais relevantes, exigindo soluções que minimizem custos operacionais e respeitem restrições de capacidade. Este trabalho apresenta uma análise comparativa entre o Algoritmo Genético (AG) e a Otimização por Colônia de Formigas (OCF) aplicados ao PRVC. Os algoritmos foram implementados na linguagem Kotlin e os testes foram conduzidos sobre três instâncias-padrão da literatura, avaliando critérios como qualidade das soluções (distância total e desvio percentual em relação ao ótimo conhecido), tempo de execução, estabilidade estatística e comportamento de convergência. Os resultados mostraram que o AG apresentou melhor desempenho em instâncias de porte médio, enquanto a OCF se destacou em instâncias maiores pela robustez e menor variação entre execuções. O estudo contribui para o entendimento comparativo entre essas metaheurísticas e oferece subsídios para sua aplicação em problemas logísticos reais.

Palavras-chaves: problema de roteamento de veículos; algoritmo genético; otimização por colônia de formigas; otimização de rotas.

ABSTRACT

The modern world demands increasing operational efficiency to ensure products are delivered quickly and with quality, providing a superior customer experience. In this context, route optimization represents a central challenge for companies in the logistics sector. The Capacitated Vehicle Routing Problem (CVRP) stands out as one of the most relevant variants, requiring solutions that minimize operational costs while respecting capacity constraints. This work presents a comparative analysis between the Genetic Algorithm (GA) and Ant Colony Optimization (ACO) applied to the CVRP. The algorithms were implemented in Kotlin, and tests were conducted on three standard benchmark instances from the literature, evaluating criteria such as solution quality (total distance and percentage deviation from the known optimum), execution time, statistical stability, and convergence behavior. The results showed that GA performed better on medium-sized instances, while ACO stood out on larger instances due to its robustness and lower variation between runs. This study contributes to the comparative understanding of these metaheuristics and provides insights for their application to real-world logistics problems.

Keywords: vehicle routing problem; genetic algorithm; ant colony optimization; route optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
CAE	Cruzamento Alternado de Arestas
CC	Cruzamento por Ciclo
CG	Cruzamento Guloso
CH	Cruzamento Heurístico
CO	Cruzamento por Ordem
CSC	Cruzamento Sequencial Construtivo
GAP	Desvio percentual em relação ao ótimo conhecido
OCF	Otimização por Colônia de Formigas
PCV	Problema do Caixeiro Viajante
PMC	Cruzamento Parcialmente Mapeado
PRV	Problema de Roteamento de Veículos
PRVC	Problema de Roteamento de Veículos Capacitados
SE	Seleção Elitista
SR	Seleção por Ranking
SRR	Seleção por Roleta
ST	Seleção por Torneio

SUMÁRIO

1 INTRODUÇÃO

A logística moderna enfrenta desafios significativos devido ao crescimento exponencial do comércio eletrônico nos últimos anos. De acordo com Gomes et al. (2023), a gestão logística tornou-se um componente crucial para o sucesso das empresas que operam no varejo, especialmente aquelas no setor de *e-commerce*. Com o contínuo aumento da popularidade das compras online, as empresas precisam implementar práticas logísticas eficientes para satisfazer os clientes, reduzir custos e manter uma vantagem competitiva (Raj et al., 2024). Para Wang et al. (2020), essa evolução transformou o papel da logística na cadeia de suprimentos, impondo novas exigências em termos de eficiência e agilidade nas entregas. Nesse contexto, é necessário desenvolver métodos eficazes de otimização para lidar com a complexidade crescente das operações logísticas. Um desses métodos é o Problema de Roteamento de Veículos (PRV), uma extensão do Problema do Caixeiro Viajante (PCV), amplamente estudado devido à sua relevância em otimização logística. O PRV visa determinar a rota mais eficiente para uma frota de veículos atender a um conjunto de clientes distribuídos geograficamente, minimizando custos, como consumo de combustível e tempo de viagem (Laporte, 2010).

No entanto, o PRV é um problema NP-difícil, o que significa que o tempo necessário para encontrar uma solução exata aumenta de forma significativa à medida que o número de clientes e veículos cresce. Isso torna o problema inviável de ser resolvido por métodos exatos em grande escala. Isso tem motivado o desenvolvimento de abordagens que podem gerar soluções próximas do ótimo em menos tempo, o que é importante em contextos de tomada de decisão rápida, como na logística urbana e no setor de vendas online. (Ahmed et al., 2023). Nesse contexto, destacam-se métodos baseados em meta-heurísticas, como o Algoritmo Genético (AG) e a Otimização por Colônia de Formigas (OCF) (Kumari et al., 2023).

Nos últimos anos, o avanço dessas meta-heurísticas inspiradas na biologia se intensificou, com suas aplicações se expandindo em diversos domínios, como o mercado digital, onde a otimização das rotas é fundamental para garantir entregas mais ágeis e precisas. Essas soluções proporcionam uma flexibilidade e escalabilidade que métodos tradicionais não conseguem alcançar, resultando em operações mais ágeis e economicamente vantajosas (Deshmukh; Dorle, 2015). A adoção de soluções baseadas em AG e OCF permite que empresas que operam em grandes centros urbanos superem desafios como o congestionamento de tráfego, restrições de circulação e mudanças inesperadas na demanda, ajustando suas operações de forma eficiente. Isso melhora o uso da frota e diminui gastos, além de elevar a qualidade do serviço oferecido aos clientes.

Diante desse contexto, este trabalho propõe uma análise comparativa entre as abordagens AG e OCF na resolução do Problema de Roteamento de Veículos Capacitados (PRVC).

1.1 Motivação e Justificativa

O comércio eletrônico tem demonstrado um crescimento expressivo em escala global, com as vendas online alcançando aproximadamente 5,8 trilhões de dólares em 2023, e projeções indicando que esse valor pode ultrapassar 8 trilhões de dólares até 2027 (Statista, 2023). Nesse cenário de expansão acelerada, a otimização das rotas de entrega torna-se crucial para reduzir custos com combustível, diminuir o tempo de entrega e melhorar a experiência do cliente, visto que a eficiência no transporte impacta diretamente as despesas operacionais e a percepção de valor dos consumidores. Além disso, conforme argumentado por Toth e Vigo (2014), o uso de métodos computacionais para resolver o PRV proporciona economias substanciais nos custos de transporte, além de contribuir para processos de planejamento mais rápidos e eficazes, reforçando a relevância de abordagens baseadas em algoritmos de otimização no contexto atual.

Além dos benefícios econômicos, a otimização de rotas apresenta importantes implicações ambientais. Segundo Zhang et al. (2015), algoritmos eficientes de roteamento podem reduzir significativamente o consumo de combustível e as emissões de carbono, alcançando reduções de até 15% na pegada ambiental das operações logísticas. Os autores demonstraram que existe uma relação de compromisso entre objetivos econômicos e ambientais que pode ser equilibrada através de algoritmos meta-heurísticos apropriados.

Portanto, a adoção de algoritmos como o AG e a OCF para a otimização das rotas no PRV se justifica pela sua habilidade em gerar soluções eficientes em um tempo reduzido, uma característica essencial para a dinâmica da logística moderna. Nesse contexto, Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021) demonstrou, por meio de experimentos, que o AG é capaz de encontrar soluções próximas ao valor ótimo. Além disso, conforme demonstrado no estudo de Huang et al. (2018), a OCF se mostrou eficaz na resolução de problemas de roteamento de veículos em instâncias de grande escala, proporcionando soluções de alta qualidade dentro de um tempo de execução razoável.

A análise comparativa desses dois algoritmos é relevante para determinar qual abordagem apresenta melhor desempenho em diferentes cenários do PRVC, considerando fatores como a dimensão do problema, restrições de capacidade e distribuição geográfica dos clientes. Os resultados dessa análise podem oferecer orientações valiosas para pesquisadores e profissionais do setor logístico na escolha da meta-heurística mais adequada para suas necessidades específicas, contribuindo para avanços significativos na eficiência operacional das empresas, melhor gestão dos recursos e redução de custos.

1.2 Objetivos

Esta seção apresenta o objetivo geral do trabalho e os desdobra em objetivos específicos necessários para sua realização.

1.2.1 Objetivo Geral

Analisar comparativamente o desempenho do AG e da OCF na resolução do PRVC, implementando ambos os algoritmos e identificar a abordagem mais eficiente em diferentes cenários de teste.

1.2.2 Objetivos Específicos

- **Analisar o Problema de Roteamento de Veículos e sua variante capacitada:** Estudar suas características, restrições e aplicações para estabelecer a base teórica do trabalho.
- **Implementar o Algoritmo Genético e a Otimização por Colônia de Formigas:** Desenvolver implementações destes algoritmos para resolver o problema de roteamento de veículos capacitado.
- **Realizar análise comparativa entre Algoritmo Genético e Otimização por Colônia de Formigas:** Executar testes controlados usando instâncias *benchmark* da para avaliar o desempenho dos algoritmos implementados.
- **Analisar os resultados:** Comparar os algoritmos em termos de qualidade das soluções, tempo de execução, eficiência computacional e robustez.

1.3 Organização do Trabalho

Este trabalho está estruturado em cinco capítulos. O **Capítulo 1** apresenta a introdução, contextualizando o problema de roteamento de veículos e sua relevância no cenário atual, além de definir os objetivos e a justificativa do estudo.

O **Capítulo 2** aborda a fundamentação teórica, apresentando conceitos sobre otimização de rotas, o Problema de Roteamento de Veículos e suas variantes, bem como os algoritmos meta-heurísticos utilizados neste trabalho: Algoritmo Genético e Otimização por Colônia de Formigas.

O **Capítulo 3** descreve a metodologia adotada, detalhando a implementação dos algoritmos, os parâmetros utilizados, as métricas de avaliação e as instâncias de teste selecionadas.

O **Capítulo 4** apresenta e discute os resultados obtidos nas comparações entre os algoritmos, com base nos testes realizados.

Por fim, o **Capítulo 5** apresenta as conclusões, as contribuições do trabalho, suas limitações e sugestões para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica necessária para a compreensão do trabalho desenvolvido. Inicialmente, é apresentada uma visão geral sobre otimização de rotas, destacando sua importância no contexto atual e seus principais desafios. Em seguida, são abordados os conceitos fundamentais do PRV e suas principais variantes, com ênfase no PRVC. A fundamentação teórica aqui apresentada serve como base para o desenvolvimento da solução proposta, que utiliza AG e OCF para resolver PRVC.

2.1 Otimização de rotas

A otimização de rotas busca determinar o trajeto mais eficiente e econômico para veículos atingirem seus destinos. Essa metodologia tem grande importância no setor de transportes e logística, pois reduz tempos de viagem e custos operacionais (Abousaeidi; Fauzi; Muhamad, 2016). Para implementar um sistema de otimização de rotas, é fundamental documentar todos os percursos possíveis, possibilitando simulações e análises de diferentes cenários. Com o uso de sistemas computacionais, o planejamento de rotas tornou-se digital e utiliza algoritmos especializados para otimização, visando objetivos variados, como a minimização do tempo de viagem e custos, além da maximização da produtividade (Woroniuk; Marinov, 2013).

É importante notar que a otimização de rotas vai além de encontrar o caminho mais curto; ela considera múltiplas variáveis que impactam a eficiência, como bloqueios temporários, condições de tráfego e restrições do cliente (Souza et al., 2019). Assim, a otimização contribui diretamente para a eficiência logística, permitindo entregas mais racionais e eficientes.

A otimização, em termos gerais, envolve tornar um sistema ou decisão o mais eficaz possível, frequentemente por meio de algoritmos. Desenvolver algoritmos de otimização é um campo de pesquisa dinâmico, pois problemas reais exigem soluções ótimas ou quase ótimas em prazos viáveis. Avanços nesse campo têm se concentrado na criação de novas estratégias de busca e no aperfeiçoamento de procedimentos existentes, seja com modificações ou combinações de métodos (Akpinar, 2016).

2.2 Problema de Roteamento de Veículos (PRV)

Proposto inicialmente por Dantzig e Ramser (1959), no contexto da otimização da distribuição de gasolina, o PRV é um problema de otimização combinatória que busca determinar as melhores rotas para uma frota de veículos que precisa atender um conjunto de clientes.

O PRV pode ser visto como uma generalização do clássico Problema do Caixeiro Vi-

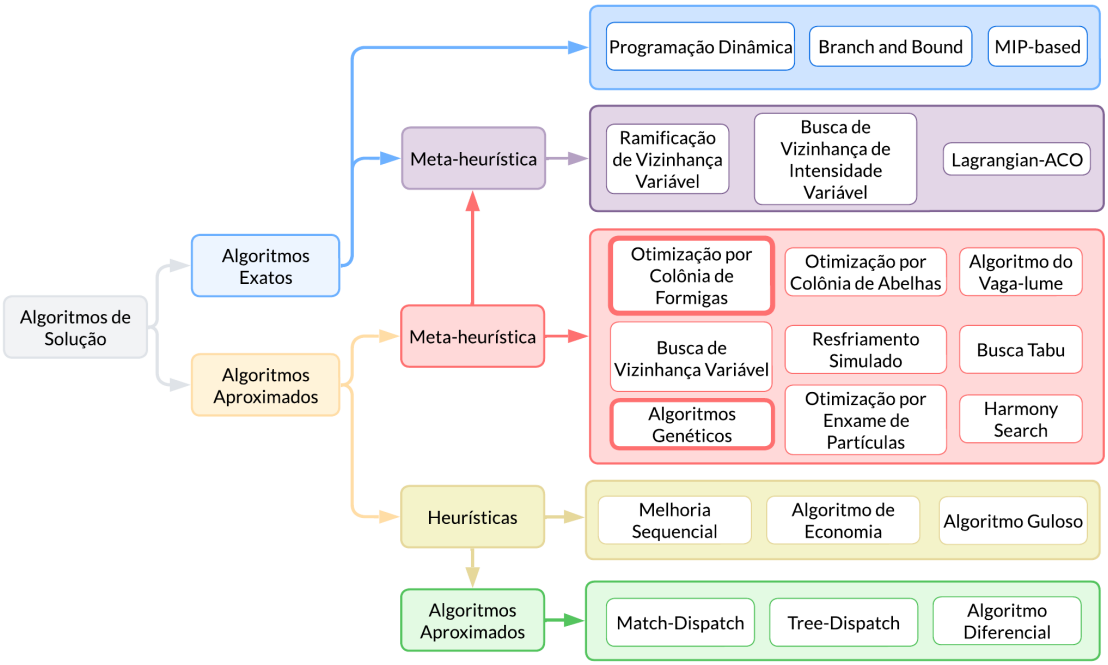
ajante (PCV), porém, enquanto o PCV busca determinar a rota mais curta para que um único vendedor visite um conjunto de cidades e retorne ao ponto de origem, o PRV envolve múltiplos veículos que partem de um ou mais depósitos e devem atender diferentes clientes. A complexidade do PRV levou ao desenvolvimento de diversas variantes para atender cenários específicos do mundo real (Laporte, 2010). Como discutido por Elshaer e Awad (2020), estas variantes permitem que o PRV seja aplicado de forma eficaz em diferentes contextos logísticos, otimizando operações e reduzindo custos operacionais. Sharma, Routroy e Yadav (2018) apresentam as principais variantes do PRV:

- **Problema de Roteamento de Veículos Capacitado**: Considera uma frota com capacidade uniforme que atende demandas a partir de um depósito comum. Sua principal característica é a restrição de capacidade fixa dos veículos, visando minimizar o custo total das rotas.
- **Problema de Roteamento de Veículos com Múltiplos Depósitos**: Os veículos partem de diferentes pontos de origem. A solução envolve duas etapas: a designação dos clientes aos depósitos e o planejamento das rotas para cada depósito.
- **Problema de Roteamento de Veículos Periódico**: Considera um horizonte de planejamento de múltiplos dias, permitindo diferentes frequências de visitas aos clientes.
- **Problema de Roteamento de Veículos com Janelas de Tempo**: Adiciona restrições temporais onde cada cliente deve ser atendido em intervalos específicos, sendo comum em entregas de produtos perecíveis e serviços agendados.
- **Problema de Roteamento de Veículos Verde**: Busca equilibrar custos econômicos e impactos ambientais, como a redução de emissões de gases.
- **Problema de Roteamento de Veículos com Coleta e Entrega**: Considera operações onde os veículos realizam tanto entregas quanto coletas, podendo ocorrer simultaneamente ou em sequência.

A escolha da variante mais adequada depende das características específicas do problema real, considerando restrições operacionais, objetivos organizacionais e requisitos dos clientes.

Nesse sentido, para resolver estas diferentes variantes do PRV, diversos algoritmos e métodos foram desenvolvidos ao longo dos anos, incluindo tanto métodos exatos quanto aproximados (Elshaer; Awad, 2020). A Figura 1 apresenta uma classificação hierárquica dos principais algoritmos utilizados para resolver o PRV e suas variantes.

Figura 1 – Diferentes abordagens de solução para PRV



Fonte: Matijević (2022)

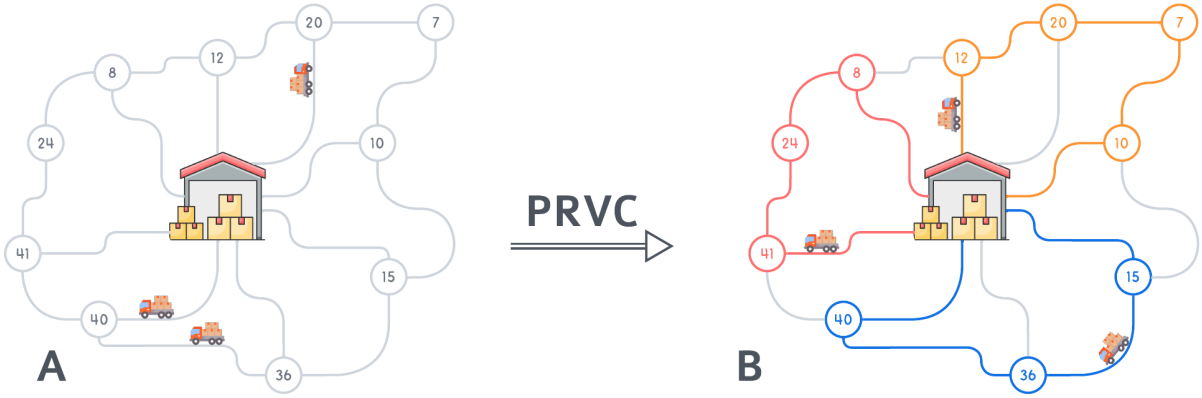
2.2.1 Problema de Roteamento de Veículos Capacitados (PRVC)

O PRVC consiste em um único depósito e um conjunto de clientes que precisam ser atendidos a partir desse depósito, utilizando um conjunto de K veículos homogêneos e com capacidade limitada. O principal objetivo é determinar as rotas que minimizam o custo total, o que implica na alocação de clientes a veículos e na definição das sequências de visita para cada rota, visando reduzir a distância total percorrida pelos veículos. Para um conjunto de N clientes e um depósito, o PRVC pode ser representado por um grafo $G = (N, E)$, onde $N = \{0, ..., n\}$ representa o conjunto de nós e $E = \{(i, j) : i, j \in N\}$ o conjunto de arestas. O nó 0 designa o depósito, que serve como ponto de partida e chegada dos veículos. Os demais nós correspondem aos clientes, cada um com uma demanda conhecida d_i, e cada cliente deve ser atendido por apenas um veículo. A distância de viagem entre os nós i e j é indicada por $d_{ij} > 0$, e cada veículo possui uma capacidade específica Q_k. A demanda total dos clientes designados a uma rota não deve ultrapassar a capacidade do veículo. Dessa forma, o PRVC pode ser formulado de modo que X_{kij} seja igual a 1 se o veículo k viaja do nó i para o nó j e 0 em caso contrário (Akpinar, 2016 apud Lin et al., 2009).

No PRVC, a alocação eficiente de clientes aos veículos e a definição das rotas mais econômicas são fatores centrais para o sucesso do planejamento logístico. A Figura 2 apresenta a transformação de um problema de roteamento em sua solução otimizada. Na Figura 2 (A), é mostrada a disposição inicial dos clientes, representados por círculos numerados que indicam suas respectivas demandas, conectados ao depósito central. Na 2 (B), é apresentada uma solução

para o PRVC, onde três veículos com capacidade máxima de 100 unidades realizam entregas a partir do depósito central, com rotas identificadas por cores distintas (vermelho, laranja e azul), demonstrando a otimização do atendimento aos clientes.

Figura 2 – PRVC com uma solução válida.



Fonte: Adaptado de Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021)

2.3 Algoritmos Genéticos

Os Algoritmos Genéticos (AG) são métodos de busca e otimização baseados nos mecanismos de seleção natural e genética. Introduzidos por Holland (1975), os AGs trabalham com uma população de possíveis soluções que evoluem de acordo com regras de seleção específicas para um estado otimizado (Mitchell, 1998).

2.3.1 Fundamentos Biológicos

O funcionamento dos AGs é inspirado nos princípios da evolução natural proposta por Charles Darwin. Na natureza, indivíduos de uma população competem entre si por recursos como alimento, água e território. Além disso, membros da mesma espécie frequentemente competem para atrair companheiros para reprodução. Os indivíduos que obtêm mais sucesso em sobreviver e se reproduzir terão relativamente mais descendentes na próxima geração. A combinação das boas características dos ancestrais pode produzir descendentes mais aptos, cuja adequação é maior que a dos pais. Dessa forma, as espécies evoluem para se tornarem mais adaptadas ao seu ambiente (Sivanandam; Deepa, 2008).

2.3.2 Estrutura Básica

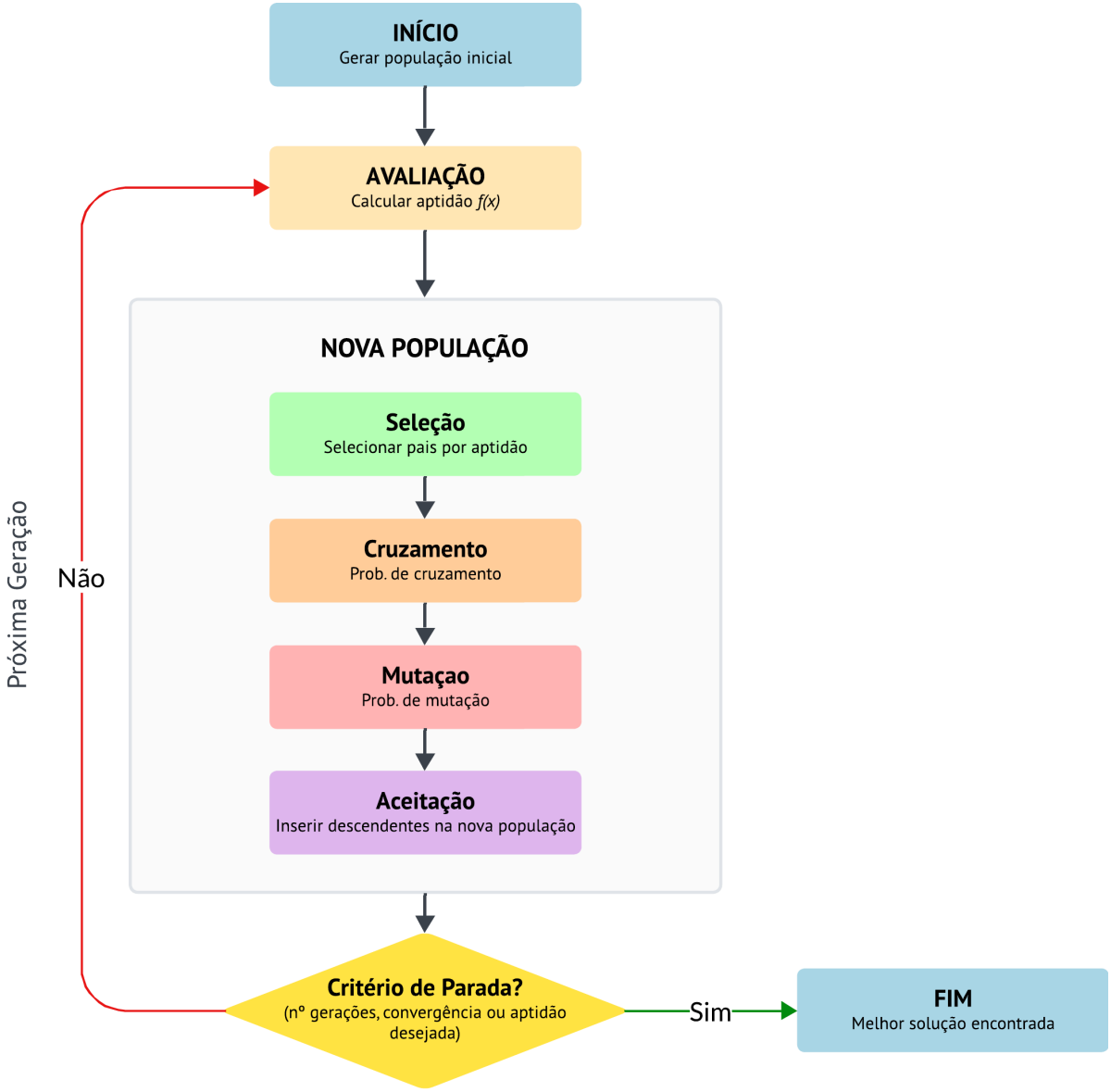
Um AG típico possui os seguintes componentes principais:

- Uma representação genética para soluções potenciais do problema

- Uma forma de criar uma população inicial de soluções potenciais
- Uma função de avaliação que classifica as soluções em termos de sua adequação
- Operadores genéticos que alteram a composição dos descendentes
- Valores dos parâmetros que o algoritmo utiliza (tamanho da população, probabilidades de aplicar operadores genéticos, etc.)

O processo básico de um AG pode ser visualizado no fluxograma da Figura 3.

Figura 3 – Fluxograma do processo iterativo de um AG.



Fonte: Adaptado de Mitchell (1998)

De modo geral, o AG, conforme descrito por Mohammed, Ahmad e Mostafa (2012), começa com a geração de uma população inicial de soluções candidatas. Em seguida, cada solução é avaliada quanto à sua aptidão, determinando o quão bem ela resolve o problema. Após

essa avaliação, o AG passa para o ciclo de evolução, onde são aplicadas diversas operações para formar uma nova população. Na etapa de seleção, são escolhidos pares de soluções (ou ”pais”) com maior aptidão para o cruzamento. No cruzamento, esses pais combinam seus ”genes”para gerar novos descendentes, introduzindo variação na busca por soluções melhores. Em seguida, a etapa de mutação realiza pequenas alterações aleatórias nos descendentes, o que ajuda a explorar novas áreas do espaço de soluções e evita a convergência prematura. Após essas modificações, na etapa de aceitação, os descendentes são inseridos na nova população. Ao final de cada ciclo, é verificado o critério de parada, que pode ser um número máximo de gerações ou a convergência das soluções. Se o critério de parada for atendido, o processo termina, e a melhor solução encontrada é apresentada como resultado. Caso contrário, o AG continua para a próxima geração, repetindo o processo de avaliação e evolução.

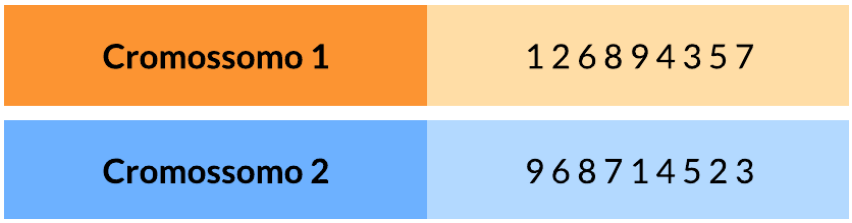
2.3.3 Aplicação de Algoritmos Genéticos no Problema de Roteamento de Veículos Capacitados

A aplicação de AG no PRVC exige uma abordagem diferente da tradicional. Como o PRVC é um problema sequencial, uma única solução não pode ser representada como um conjunto de *bits*. A representação adequada para esse tipo de problema é uma sequência de números inteiros, que representa os vértices no grafo do problema (Ochelska-Mierzejewska; Poniszewska-Marańda; Marańda, 2021). A partir desta representação fundamental, são necessárias outras adaptações nos operadores genéticos para garantir soluções viáveis para o problema. Na literatura, diferentes abordagens de representação e operadores genéticos têm sido propostas para adequar o AG às particularidades do PRVC.

2.3.3.1 Representações Cromossômicas para o Problema de Roteamento de Veículos Capacitados

Os cromossomos são transformados em um conjunto de rotas e submetidos a um processo evolutivo iterativo até que seja atingido um número mínimo possível de agrupamentos de rotas ou a condição de término seja satisfeita (Azad; Hasin, 2019). Em Mohammed, Ahmad e Mostafa (2012) é apresentado uma codificação onde os genes representam localizações de entrega numeradas sequencialmente. Cada cromossomo simboliza uma sequência de números que define uma rota válida, conforme o exemplo ilustrado na Figura 4.

Figura 4 – Codificação cromossômica com sequência direta de entregas

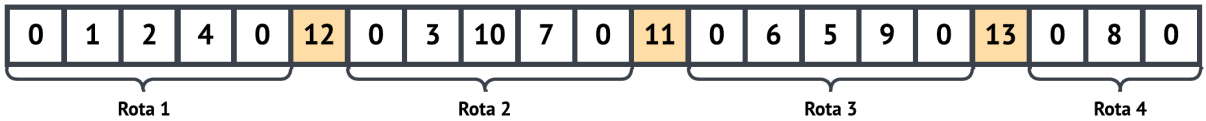


Fonte: Mohammed, Ahmad e Mostafa (2012)

Uma abordagem alternativa é proposta por Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021), onde é utilizada uma representação modular. Nesta abordagem, o número 0 representa o depósito, enquanto os clientes são numerados de 1 a n (sendo n o total de entregas a serem realizadas). Os veículos são identificados por números de $n+1$ a $m-1$, onde m é o total de entidades numeradas, de modo que a quantidade de veículos na frota é $m-n$, com um veículo sempre omitido da contagem.

A representação modular organiza a solução em blocos, com cada bloco representando uma rota específica. Cada bloco começa e termina com o número 0, indicando o ponto de partida e retorno ao depósito, enquanto os números dos clientes atendidos por um veículo são inseridos entre esses zeros na ordem de atendimento. Todos os blocos são unidos para formar a solução completa. No entanto, o primeiro bloco não precisa incluir o número do veículo na posição inicial. Se, após essa construção, dois números de veículos aparecerem adjacentes ou se a sequência final terminar com um número de veículo, isso indica que nem todos os veículos foram usados na solução, podendo esses elementos ser removidos, desde que a capacidade dos veículos seja uniforme. A Figura 5 ilustra um exemplo de codificação cromossomos com a representação apropriada para o PRV com 10 entregas e 4 veículos.

Figura 5 – Codificação cromossômica com representação modular por rotas



Fonte: Adaptado de Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021)

2.3.4 Operadores Genéticos

Dentre os operadores genéticos - seleção, cruzamento e mutação - o operador de cruzamento é considerado crucial no projeto e implementação de AGs, tendo sido desenvolvidos diversos tipos para diferentes problemas de otimização combinatória (Ahmed et al., 2023). O desempenho destes operadores no PRVC depende de como são adaptados para lidar com as restrições específicas do problema.

2.3.4.1 Seleção

A seleção é um componente crucial no AG, tendo diferentes métodos para sua implementação. Segundo Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021), existem vários métodos que podem ser utilizados para selecionar os indivíduos que participarão da próxima geração, todos seguindo o princípio de que indivíduos mais aptos devem ter maior chance de serem escolhidos. Os principais métodos são:

- **Seleção por Roleta (SRR):** Os indivíduos recebem uma probabilidade de seleção proporcional ao seu valor de aptidão.
- **Seleção Elitista (SE):** Garante que os melhores indivíduos da geração atual sempre passem para a próxima geração.
- **Seleção por Ranking (SR):** A probabilidade de seleção é baseada na posição (*ranking*) do indivíduo quando a população é ordenada por aptidão.
- **Seleção por Torneio (ST):** Realiza competições entre indivíduos selecionados aleatoriamente.

Para adaptar estes métodos ao PRVC, que é um problema de minimização, Ahmed et al. (2023) propõem utilizar uma função de aptidão (*fitness*) que realiza o inverso do valor da função objetivo, garantindo assim que rotas com menor custo total tenham maior probabilidade de serem selecionadas. Esta função é definida como:

$$f_i = \frac{1}{1+o_i} \quad ; \quad o_i \text{ é o valor da função objetivo do cromossomo } i \tag{2.1}$$

2.3.4.2 Cruzamento

Ahmed et al. (2023) dividem os operadores de cruzamento em duas principais categorias: cruzamentos cegos (*blind crossovers*) e cruzamentos baseados em distância (*distance-based crossovers*). Baseado nesta classificação, diferentes técnicas foram desenvolvidas:

- **Cruzamentos Cegos:**

  - **Cruzamento Parcialmente Mapeado (PMC):** Trabalha com o mapeamento entre segmentos dos cromossomos pais.
  - **Cruzamento por Ordem (CO):** Foca em preservar a ordem dos clientes de um dos pais.
  - **Cruzamento por Ciclo (CC):** Mantém as posições absolutas dos elementos.
  - **Cruzamento Alternado de Arestas (CAE):** Opera alternando entre os pais na construção das rotas.

- **Cruzamentos Baseados em Distância:**
  - **Cruzamento Guloso (CG):** Utiliza estratégia gulosa na seleção dos próximos clientes.
  - **Cruzamento Heurístico (CH):** Incorpora informações dos custos das rotas.
  - **siglaCMHCruzamento Heurístico Modificado:** Versão aprimorada do CH com análise mais ampla.
  - **Cruzamento Sequencial Construtivo (CSC):** Constrói as rotas de forma sequencial considerando tanto a estrutura dos pais quanto os custos.

Com base em extensivos testes computacionais, Ahmed et al. (2023) demonstraram que os operadores baseados em distância tendem a apresentar resultados superiores na resolução do PRVC, com destaque especial para o CSC em termos de qualidade das soluções obtidas.

2.3.4.3 Mutação

Segundo Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021), os operadores de mutação para o PRVC não precisam ser completamente reinventados em relação aos operadores tradicionais, mas sim adaptados para trabalhar com a representação baseada em números inteiros. Dois tipos principais de mutação são propostos:

- **Mutação por Troca:** Seleciona aleatoriamente dois valores inteiros na solução e troca suas posições, podendo estes valores representar tanto localizações quanto veículos
- **Mutação por Remoção e Reinserção:** Remove um valor de uma posição e o reinsere em outro local da solução

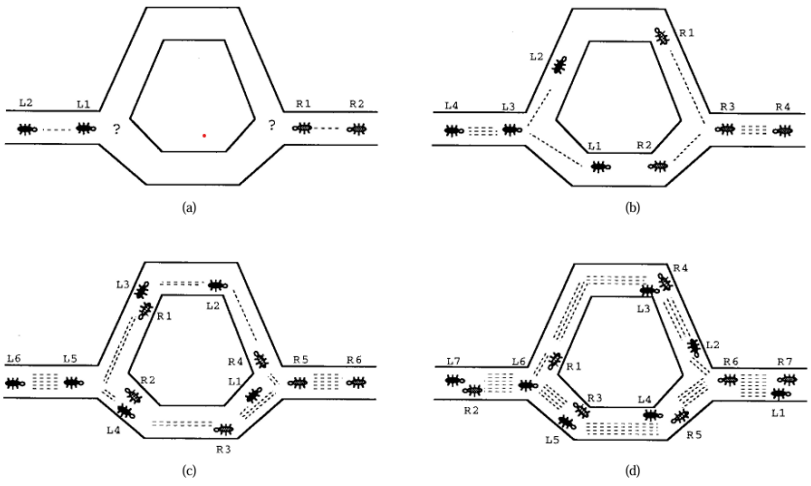
De acordo com Mohammed, Ahmad e Mostafa (2012), a mutação é aplicada nos cromossomos após o cruzamento por meio de mudanças aleatórias, sendo a troca de genes a operação mais comum devido à codificação por permutação utilizada no problema. Ahmed et al. (2023) destacam que em um problema com m veículos, são realizadas m trocas, garantindo que o cromossomo resultante seja sempre válido.

2.4 Otimização por Colônia de Formigas

A Otimização por Colônia de Formigas (OCF) é uma técnica meta-heurística de otimização que se inspira no comportamento de forrageamento de algumas espécies de formigas. Proposta inicialmente por Dorigo no início dos anos 90, a OCF tem se destacado como uma abordagem eficiente para resolver problemas de otimização combinatória. O princípio fundamental desta técnica baseia-se no comportamento das formigas que, durante a busca por alimento, depositam feromônios no solo para marcar caminhos favoráveis que devem ser seguidos por outros membros da colônia. Este mecanismo de comunicação indireta, conhecido como estigmergia [1], permite que as formigas encontrem os caminhos mais curtos entre o ninho e as fontes de alimento (Dorigo; Birattari; Stutzle, 2006).

A Figura 6 demonstra o processo natural de otimização de caminhos realizado pelas formigas. Na Figura 6 (A), as formigas chegam a um ponto de decisão onde devem escolher entre dois caminhos possíveis. Como ilustrado na Figura 6 (B), as formigas inicialmente se distribuem aleatoriamente entre os caminhos superior e inferior. Considerando que as formigas se movem a uma velocidade aproximadamente constante, aquelas que percorrem o caminho inferior, por ser mais curto, alcançam o ponto de decisão oposto mais rapidamente, conforme mostrado na Figura 6 (C). Ao retornarem pelo mesmo trajeto, essas formigas reforçam a trilha de feromônio. Como resultado, na Figura 6 (D), observa-se maior acúmulo de feromônio no caminho mais curto, representado pela maior quantidade de linhas tracejadas.

Figura 6 – Formigas descobrindo melhor caminho.



Fonte: Dorigo e Gambardella (1997)

A OCF deu origem a diversos algoritmos específicos, como o *Ant System* e suas variantes mais conhecidas, o *MAX-MIN Ant System* e o *Ant Colony System* (Dorigo; Birattari; Stutzle,

[1] Estigmergia é um mecanismo de coordenação indireta em que os agentes interagem por meio de modificações no ambiente, como os rastros de feromônio deixados pelas formigas. O termo foi cunhado por Pierre-Paul Grassé em 1959 (Marsh; Onof, 2007).

2006). Embora existam diferentes variações, os algoritmos baseados em OCF tipicamente compartilham processos principais.

2.4.1 Funcionamento da Otimização por Colônia de Formigas

O processo de busca por alimento das formigas é caracterizado pela deposição de feromônios ao longo de seus trajetos, estabelecendo um sistema de comunicação entre os membros da colônia. Este sistema faz com que rotas mais utilizadas acumulem maior quantidade de feromônio, aumentando sua atratividade para outras formigas. Entretanto, existe um processo natural de evaporação deste feromônio ao longo do tempo, criando um mecanismo que incentiva a exploração de novos caminhos pela colônia (Cai et al., 2022).

Para implementar computacionalmente este comportamento, o algoritmo utiliza um mecanismo probabilístico que simula o processo de decisão das formigas (Stodola et al., 2014). A probabilidade de uma formiga escolher um determinado caminho é calculada considerando tanto a quantidade de feromônio quanto uma informação heurística. Mazzeo e Loiseau (2004) define esta probabilidade por meio da seguinte fórmula:

$$P^k(i,j) = \begin{cases} \frac{[\tau(i,j)]^\alpha [\eta(i,j)]^\beta}{\sum\limits_{(i,z)\notin Tabu_k} [\tau(i,z)]^\alpha [\eta(i,z)]^\beta}, & \text{se } (i,j) \notin Tabu_k \\ 0, & \text{se } (i,j) \in Tabu_k \end{cases} \tag{2.2}$$

Onde $\tau(i,j)$ representa o nível de feromônio entre os pontos i e j, $\eta(i,j)$ é a informação heurística (geralmente definida como o inverso da distância), α e β são parâmetros que estabelecem a influência relativa de η versus τ, e $Tabu_k$ é uma lista de movimentos proibidos para a formiga k (Mazzeo; Loiseau, 2004).

Após cada iteração t ser completada, ou seja, quando todas as formigas completaram suas soluções, os níveis de feromônio são atualizados:

$$\tau(i,j) = \phi\tau(i,j) + \Delta\tau(i,j) \tag{2.3}$$

Onde ϕ é um coeficiente que representa o nível de persistência do feromônio e $\Delta\tau(i,j)$ representa as contribuições de todas as formigas que escolheram o movimento (i,j) em sua solução (Mazzeo; Loiseau, 2004).

2.4.2 Otimização por Colônia de Formigas Aplicada ao Problema de Roteamento de Veículos Capacitados

A OCF, quando aplicada a problemas de roteamento, trabalha com um grafo onde os vértices e arestas representam as possíveis conexões da rede. Em cada aresta, as formigas artificiais depositam e atualizam uma substância chamada feromônio, que serve como um mecanismo de

comunicação indireta entre elas (Dorigo; Birattari; Stutzle, 2006). Quando adaptada ao PRVC, esta abordagem deve considerar as restrições fundamentais do problema: a demanda total atendida em cada rota não pode exceder a capacidade do veículo, cada cliente deve ser atendido exatamente uma vez, e todas as rotas devem começar e terminar no depósito (Cai et al., 2022).

Na abordagem proposta por Mazzeo e Loiseau (2004), as formigas podem construir suas soluções de duas maneiras:

1. **Construção Sequencial**: as rotas são criadas uma por vez, onde cada veículo é preenchido até sua capacidade máxima antes de iniciar a construção da próxima rota.
2. **Construção Paralela**: todas as rotas são desenvolvidas simultaneamente, com os clientes sendo distribuídos entre os veículos disponíveis a cada iteração.

Para a seleção dos clientes em cada rota, são propostas duas regras de transição:

1. **Regra Proporcional Aleatória**: utiliza a regra probabilística tradicional da OCF onde a probabilidade de escolha considera tanto o nível de feromônio da aresta quanto a distância entre os pontos.
2. **Regra Pseudo-Proporcional Aleatória**: combina seleção aleatória com a melhor opção disponível. Dado um parâmetro q_0 ($0 \leq q_0 \leq 1$) e um número aleatório q gerado no intervalo $[0,1]$, o próximo cliente j é escolhido segundo:

$$j = \begin{cases} \max\limits_{u \in \Gamma(i)} [\tau(i,u)^{\alpha}][\eta(i,u)]^{\beta}, & \text{se } q \leq q_0 \text{ (intensificação)} \\ J, & \text{se } q > q_0 \text{ (exploração)} \end{cases} \tag{2.4}$$

Onde J é selecionado aleatoriamente de acordo com a regra proporcional aleatória e $\Gamma(i)$ representa o conjunto de clientes ainda não visitados.

Uma vez que um cliente é selecionado, ele é removido da lista para garantir que não seja visitado novamente. O cliente selecionado é adicionado à rota desde que a demanda acumulada não exceda a capacidade do veículo. Caso a inclusão do próximo cliente viole a restrição de capacidade, a formiga retorna ao depósito, iniciando uma nova rota (Mazzeo; Loiseau, 2004).

Após a construção das soluções, Mazzeo e Loiseau (2004) propõem diferentes estratégias de atualização do feromônio, que podem ser classificadas em dois tipos:

1. **Atualização Global**: Realizada após cada iteração completa do algoritmo, pode ocorrer de três formas:

- Considerando todas as soluções encontradas na iteração:

$$\tau(i,j) = \phi\tau(i,j) + \Delta\tau(i,j) \tag{2.5}$$

  Onde ϕ é o coeficiente de persistência do feromônio e $\Delta\tau(i,j)$ representa as contribuições de todas as formigas que escolheram o movimento (i,j).

- Utilizando apenas as soluções das melhores formigas (formigas elite):

$$\tau(i,j) = \phi\tau(i,j) + \sum_{\mu=1}^{\sigma-1} \Delta\tau_\mu + \sigma\Delta\tau^*(i,j) \tag{2.6}$$

  Onde σ é o número de formigas elite, $\Delta\tau_\mu$ é a contribuição da μ-ésima melhor formiga e $\Delta\tau^*$ é a contribuição da melhor solução encontrada.

- Baseando-se apenas na melhor solução encontrada

2. **Atualização Local**: Realizada imediatamente após cada movimento de uma formiga, com objetivo de diversificar as soluções por meio da redução do nível de feromônio nas arestas utilizadas:

$$\tau(i,j) = \phi\tau(i,j) + (1-\phi)\Delta\tau(i,j) \tag{2.7}$$

   Os autores propõem diferentes formas de determinar $\Delta\tau(i,j)$:

   - *Q-learning*: inspirado no método de aprendizagem automática, definido como $\Delta\tau^k(i,j) = \gamma \max_{z\in\Gamma(j)} \tau(j,z)$ com $0 \leq \gamma < 1$
   - Feromônio inicial: utiliza o valor inicial de feromônio $\Delta\tau(i,j) = \tau_0$
   - Proporcional à distância: $\Delta\tau(i,j) = \frac{\tau_0}{d(i,j)}$
   - Evaporação: $\Delta\tau(i,j) = 0$

Em qualquer uma dessas estratégias, a atualização do feromônio guia o processo de busca, onde a escolha da estratégia impacta diretamente o equilíbrio entre intensificação (exploração de regiões promissoras) e diversificação (busca em novas áreas do espaço de soluções) do algoritmo.

2.5 Trabalhos Relacionados

Esta seção apresenta e analisa os trabalhos mais relevantes da literatura que abordam a resolução do PRVC utilizando o AG e a OCF. É realizada uma análise crítica destes trabalhos, destacando suas contribuições, limitações e comparando-os com a presente proposta.

Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021) apresentaram uma análise do desempenho de diferentes operadores do AG para o PRVC, propondo uma abordagem modular para a representação cromossômica que permite organizar as rotas em blocos. Por meio

de cinco experimentos principais, os autores demonstraram que operadores baseados em arestas (*edge-based*) são superiores aos baseados em vértices (*vertex-based*) quando há tempo suficiente de processamento, embora os operadores baseados em vértices convirjam mais rapidamente para soluções aceitáveis em execuções curtas.

Othman et al. (2018) investigaram a aplicação do algoritmo de OCF para resolver o PRV. Os autores realizaram um estudo aprofundado sobre os parâmetros de controle do algoritmo OCF, analisando especificamente a influência de quatro parâmetros principais: número de formigas (nAnt), alfa (α), beta (β) e rho (ρ). Após extensivos experimentos, identificaram que a configuração ótima para o problema estudado foi com 20 formigas, $\alpha = 1$, $\beta = 1$ e $\rho = 0{,}05$. O estudo demonstrou a importância do ajuste correto destes parâmetros, obtendo uma distância média de 1057,839 km com desvio padrão de 25,913 km em 20 execuções independentes. A pesquisa contribui significativamente para a compreensão da sensibilidade do OCF em relação a seus parâmetros e confirma a eficácia do método para resolver o PRV, fornecendo uma base sólida para comparações com outras meta-heurísticas.

Haroun, Jamal e Hicham (2015) realizaram um estudo comparativo entre AG e OCF aplicados ao PCV, uma versão simplificada do PRVC. Os autores testaram ambos os algoritmos em três instâncias *benchmark* com diferentes tamanhos e complexidades, além de um caso real de aplicação em logística urbana. Os resultados demonstraram que o AG é mais eficiente para problemas de pequeno a médio porte, convergindo rapidamente para soluções ótimas, enquanto o OCF apresenta resultados superiores em problemas de maior dimensão, embora exija mais recursos computacionais. Os autores observaram que a taxa de erro aumenta com a complexidade do problema para ambos os algoritmos, mas a diferença entre suas taxas de erro torna-se maior em problemas complexos, favorecendo o OCF. Este estudo fornece *insights* importantes sobre o comportamento comparativo dos dois algoritmos em diferentes cenários, contribuindo para nossa análise no contexto específico do PRVC.

Em um trabalho mais recente, Ahmed et al. (2023) realizaram um estudo comparativo sobre oito operadores de cruzamento para o PRVC, classificando-os em cruzamentos cegos (PMC, CO, CC e CAE) e cruzamentos baseados em distância (CG, CH, CMH e CSC). Por meio de experimentos em diferentes instâncias do problema, os autores demonstraram que os cruzamentos baseados em distância são significativamente superiores aos cruzamentos cegos, com o CSC apresentando o melhor desempenho geral entre todos os operadores analisados.

2.5.1 Comparação com o Trabalho Proposto

O presente trabalho se diferencia dos anteriores nos seguintes aspectos:

1. **Análise comparativa direta:** Realizamos uma comparação direta entre implementações puras de AG e OCF aplicadas ao PRVC, sem hibridizações ou adaptações que poderiam mascarar as características fundamentais de cada método.

2. **Avaliação sistemática de parâmetros:** Nossa pesquisa investiga sistematicamente como diferentes configurações de parâmetros afetam o desempenho de cada algoritmo no contexto específico do PRVC, identificando as sensibilidades paramétricas próprias deste problema.

3. **Avaliação em cenários escalonáveis:** Testamos os algoritmos em instâncias que variam gradualmente em tamanho e complexidade, permitindo identificar pontos de inflexão onde um método se torna mais vantajoso que o outro, oferecendo diretrizes claras para escolha algorítmica baseada nas dimensões do problema.

4. **Métricas multidimensionais:** Além da qualidade da solução, analisamos aspectos operacionais como consumo de memória, padrões de convergência e estabilidade estatística entre execuções, proporcionando uma visão mais completa sobre o desempenho prático de cada algoritmo.

3 METODOLOGIA

Este capítulo apresenta a metodologia adotada para o desenvolvimento do trabalho, que consiste em quatro etapas principais: ambiente de desenvolvimento, especificação dos algoritmos, parametrização dos algoritmos e metodologia de avaliação comparativa.

3.1 Ambiente de Desenvolvimento

O desenvolvimento dos algoritmos foi realizado na linguagem *Kotlin*, utilizando a *IDE* IntelliJ IDEA. A execução dos testes, bem como a leitura e escrita dos resultados, também foi conduzida nesse ambiente. Para a análise estatística e geração dos gráficos, foram utilizados notebooks interativos com suporte a *Kotlin*, exclusivamente para o processamento dos resultados obtidos.

Para manipulação de dados tabulares, utilizou-se a biblioteca *Kotlin DataFrame*. A criação dos gráficos — como curvas de convergência, *boxplots* e visualizações das rotas — foi feita com a biblioteca *Lets-Plot*, inspirada no paradigma *ggplot*. A leitura de arquivos e a automação das análises utilizaram os recursos padrão da linguagem *Kotlin* e da API *java.io*.

O código-fonte completo, incluindo a implementação dos algoritmos e todos os recursos utilizados para análise de resultados e geração de gráficos, está disponível publicamente em um repositório no GitHub (COSTA, 2025). Para fins de ilustração e para apresentar a estrutura principal dos algoritmos, fragmentos selecionados do código-fonte também podem ser consultados no Apêndice A deste trabalho.

3.2 Especificação dos Algoritmos

3.2.1 Algoritmo Genético (AG)

O algoritmo genético foi implementado com uma estrutura baseada em população e evolução de gerações. Cada indivíduo representa uma solução viável do problema, composta por uma lista de rotas que respeitam a capacidade dos veículos. A evolução da população ocorre por meio dos operadores clássicos de algoritmos genéticos: seleção, cruzamento e mutação.

A seleção foi implementada por meio do método de torneio, permitindo que os indivíduos mais aptos tenham maior chance de reprodução. O operador de cruzamento utilizado foi o CSC, que cria novas soluções construindo rotas válidas a partir dos pais.

A mutação aplicada é do tipo mutação por troca. Nessa abordagem, dois genes aleatórios do cromossomo — que representam clientes — são selecionados e têm suas posições trocadas.

Essa operação garante a manutenção da viabilidade estrutural da solução, e busca introduzir diversidade genética à população, contribuindo para evitar a convergência prematura.

Por fim, o algoritmo também utiliza elitismo, garantindo que o melhor indivíduo da geração anterior seja preservado. A aptidão de cada indivíduo é calculada através de uma função que considera principalmente a distância total percorrida, mas também penaliza fortemente soluções que violam a capacidade dos veículos. Esta função de aptidão é inversamente proporcional ao custo total, privilegiando as soluções que minimizam as distâncias e respeitam as restrições de capacidade.

3.2.2 Otimização por Colônia de Formigas (OCF)

No nosso OCF, cada formiga artificial constrói uma solução completa ao selecionar os próximos clientes a serem visitados com base em uma regra de transição probabilística, que equilibra a influência dos feromônios e da distância entre os nós.

As soluções são construídas respeitando a capacidade dos veículos, com cada rota sendo completada até que não seja mais possível adicionar clientes sem violar a restrição. A atualização dos feromônios é realizada de duas formas: uma atualização local, aplicada a cada movimento durante a construção da solução; e uma atualização global, onde os melhores caminhos encontrados reforçam as trilhas de feromônio. A estratégia de atualização global incorpora elitismo, priorizando as melhores soluções da iteração ou do histórico global. Ao final de cada iteração, a solução mais promissora é comparada com a melhor encontrada até o momento.

3.3 Parametrização dos algoritmos

Para a execução eficiente do AG e OCF, foi necessário definir parâmetros que balanceassem a qualidade das soluções e o esforço computacional. A parametrização foi estabelecida com base em valores conhecidos na literatura e, em seguida, refinada através de um processo de calibração. Nesses testes preliminares, realizados em uma instância de pequeno porte não utilizada na avaliação final, o principal objetivo foi ajustar os parâmetros para que ambos os algoritmos apresentassem um tempo de execução semelhante, garantindo assim uma base de comparação justa.

3.3.1 Parâmetros do algoritmo genético

O Algoritmo Genético foi configurado com os parâmetros apresentados na Tabela 1:

O tamanho da população foi definido como 1000 indivíduos para garantir uma ampla diversidade genética. O tamanho do torneio de 3 foi escolhido para promover uma pressão seletiva adequada, permitindo que soluções de qualidade sejam selecionadas com maior probabilidade, sem eliminar completamente a diversidade.

Parâmetro	**Valor**
Tamanho da população	1000
Número máximo de gerações	2000
Tamanho do torneio	3
Taxa de cruzamento	0,50
Taxa de mutação	0,10

Tabela 1 – Parâmetros utilizados no Algoritmo Genético

A taxa de cruzamento de 0,50 estabelece que metade dos indivíduos da população passará pelo processo de recombinação genética, enquanto a taxa de mutação de 0,10 introduz variações pontuais suficientes para evitar a convergência prematura, permitindo a exploração de novas regiões do espaço de busca.

Essa configuração final se mostrou a mais equilibrada durante a etapa de calibração.

3.3.2 Parâmetros da Otimização por Colônia de Formigas

Para o algoritmo de Otimização por Colônia de Formigas, foram utilizados os parâmetros apresentados na Tabela 2:

Parâmetro	**Valor**
Número de formigas	50
Número máximo de iterações	2000
Fator de importância do feromônio (α)	2,0
Fator de importância da informação heurística (β)	2,0
Taxa de evaporação do feromônio (ρ)	0,1
Parâmetro de exploração (q_0)	0,5
Número de formigas elitistas	5
Taxa de atualização local	0,1

Tabela 2 – Parâmetros utilizados na Otimização por Colônia de Formigas

O número de formigas foi estabelecido em 50, um valor que proporciona diversidade de soluções sem comprometer demasiadamente o desempenho computacional. Os parâmetros α e β, ambos definidos como 2, garantem um equilíbrio entre a intensidade do feromônio e a visibilidade (informação heurística baseada na distância), permitindo que as formigas considerem igualmente ambas as informações ao construir suas soluções.

A taxa de evaporação de 0,1 promove um esquecimento gradual das piores soluções, enquanto o parâmetro q_0 de 0,5 estabelece um balanço entre exploração (escolha probabilística) e intensificação (escolha gulosa). O algoritmo também implementa um mecanismo elitista,

onde as 5 melhores formigas recebem privilégios na atualização do feromônio, reforçando os melhores caminhos encontrados. A combinação destes valores foi selecionada por alinhar o desempenho do OCF ao do AG em termos de esforço computacional.

3.3.3 Critérios de Parada

Ambos os algoritmos foram executados com um mesmo critério de parada, adotando um limite máximo de 2000 iterações — chamadas de gerações no AG e ciclos na OCF. Essa padronização foi aplicada com o objetivo de assegurar uma base justa de comparação entre os métodos.

A definição desse parâmetro foi validada por meio de testes preliminares com instâncias de referência. Nessas execuções, o limite de 2000 iterações emergiu como o ponto ideal onde os tempos de execução de ambos os algoritmos se tornavam compatíveis, e as curvas de convergência indicavam que melhorias adicionais seriam marginais.

3.4 Instâncias de testes

Para avaliar e comparar o desempenho dos algoritmos, foram selecionados três instâncias do conjunto proposto por Augerat, amplamente utilizado na literatura para o PRVC. Cada caso envolve um único depósito, um conjunto de clientes com demandas fixas e veículos homogêneos com capacidade limitada. A categorização segue a abordagem de Ochelska-Mierzejewska, Poniszewska-Marańda e Marańda (2021), que classifica os problemas em pequeno, médio e grande porte, conforme o número de clientes e veículos. A Tabela 3 apresenta os dados utilizados nos testes comparativos.

Categoria	**Instância**	**Clientes**	**Veículos**	**Cap. Veículos**	**Ótimo**
Pequena	A-n33-k5	32	5	100	661
Média	B-n50-k7	49	7	100	741
Grande	P-n101-k4	100	4	400	681

Tabela 3 – Instâncias selecionadas para os testes comparativos

Essas instâncias seguem a nomenclatura padrão X-nY-kZ, onde:

- **X**: tipo da instância (A, B ou P), indicando diferentes distribuições espaciais dos clientes;
- **nY**: número total de nós (Y clientes + 1 depósito);
- **kZ**: número de veículos disponíveis (Z).

Com base nessa classificação, cada conjunto apresenta características específicas que permitem avaliar diferentes aspectos dos algoritmos:

- Conjunto A: instâncias com distribuição aleatória de clientes em um plano euclidiano
- Conjunto B: instâncias com clientes agrupados em regiões, simulando cenários de distribuição urbana
- Conjunto P: instâncias com características específicas de roteamento, incluindo distribuição geográfica agrupada e maiores capacidades de veículos

Os clientes estão distribuídos em um plano euclidiano bidimensional, e as distâncias são calculadas como a distância em linha reta entre os pontos. Essas instâncias estão disponíveis publicamente no repositório CVRPLIB da PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro, 2025).

3.5 Métricas de Comparação

Para garantir uma análise abrangente e objetiva, foram definidas métricas em três dimensões principais:

3.5.1 Qualidade da solução

- **Distância total das rotas:** valor absoluto da distância percorrida por todos os veículos ao atenderem todos os clientes.
- **Desvio percentual em relação ao ótimo conhecido (GAP):** indica o quanto a solução obtida se afasta do valor ótimo conhecido da instância. É calculado pela fórmula:

$$\text{GAP} = \left(\frac{\text{Solução obtida} - \text{Ótimo conhecido}}{\text{Ótimo conhecido}} \right) \times 100$$

  Quanto menor o valor do GAP, mais próxima está a solução do ótimo.
- **Factibilidade:** verificação se todas as restrições do problema foram respeitadas, especialmente a capacidade máxima dos veículos e o atendimento completo a todos os clientes.

3.5.2 Eficiência computacional

A eficiência computacional foi medida principalmente pelo tempo de execução necessário para cada algoritmo atingir sua solução final. Os tempos foram registrados originalmente em milissegundos para maior precisão nas medições e posteriormente convertidos para segundos na apresentação dos resultados. Esta medida é particularmente importante para avaliar a aplicabilidade prática dos métodos em cenários reais, onde o tempo de resposta pode ser um fator crítico.

3.5.3 Comportamento de convergência

O comportamento de convergência foi analisado através de três aspectos complementares:

- **Taxa de melhoria ao longo das iterações:** verificação da velocidade com que cada algoritmo aproxima-se da solução final, permitindo identificar métodos que convergem mais rapidamente.
- **Estabilidade da solução:** análise da variabilidade dos resultados em múltiplas execuções, quantificada pelo desvio padrão das soluções obtidas.
- **Robustez:** avaliação da capacidade de encontrar boas soluções em diferentes cenários, independentemente das características específicas das instâncias.

3.5.4 Procedimento de Avaliação

Para garantir a confiabilidade dos resultados e possibilitar uma comparação objetiva, cada algoritmo foi executado 30 vezes para cada instância, seguindo as recomendações estatísticas para obtenção de amostras significativas.

Para cada instância de teste (i), foram calculados:

- Média dos desvios em relação ao valor ótimo (μ_i)
- Desvio padrão dos resultados (σ_i)
- Melhor solução encontrada ($best_i$)
- Tempo médio de execução (t_i)

Esta metodologia de avaliação proporcionou uma comparação objetiva e multidimensional entre os algoritmos, considerando não apenas a qualidade das soluções encontradas, mas também sua consistência e eficiência computacional. A análise conjunta destas métricas permitiu avaliar de forma abrangente o desempenho de cada algoritmo nas diferentes categorias de problemas.

4 RESULTADOS E DISCUSSÕES

Neste capítulo, são apresentados os resultados obtidos com os algoritmos implementados, bem como uma análise comparativa entre eles. As análises contemplam métricas relacionadas à qualidade das soluções, eficiência computacional e comportamento de convergência, permitindo avaliar o desempenho de cada abordagem em diferentes cenários.

4.0.1 Instância Pequena – A-n33-k5

Os resultados comparativos entre o AG e a OCF para a instância A-n33-k5 são apresentados na Tabela 4. Ambos os algoritmos obtiveram soluções próximas ao ótimo conhecido. O AG alcançou a melhor distância (662,26 contra 663,49), resultando em um GAP inferior (0,19% frente a 0,38%). No entanto, a OCF demonstrou maior consistência: obteve uma menor média das distâncias (695,58 contra 709,48), menor desvio padrão (14,21 contra 27,27) e um GAP médio mais baixo (5,23% frente a 7,33%). Em termos de eficiência, a OCF também se destacou, com tempo médio de execução aproximadamente 45,3% menor que o registrado pelo AG (9,34s contra 17,07s).

Métrica	**AG**	**OCF**
Ótimo conhecido	661,00	
Melhor distância	662,26	663,49
Média das distâncias	709,48	695,58
Desvio padrão	27,27	14,21
GAP melhor solução (%)	0,19	0,38
GAP médio (%)	7,33	5,23
Tempo médio (seg)	17,07	9,34

Tabela 4 – Comparativo AG e OCF – Instância Pequena

O comportamento de convergência na melhor execução de cada algoritmo é apresentado na Figura 7. Observa-se que o AG (linha vermelha) apresenta uma queda mais acentuada nas iterações iniciais, partindo de aproximadamente 1400 e atingindo a faixa dos 700 antes da iteração 200. Após essa fase inicial, o algoritmo entra em um estado de estabilização, com poucas melhorias subsequentes.

Por outro lado, a OCF (linha azul) inicia de um valor inferior (cerca de 900) e exibe uma convergência mais gradual, com uma fase de estagnação entre as iterações 200 e 600, seguida por pequenas melhorias. Ambos os algoritmos apresentam ganhos próximos ao final da execução. Vale destacar que o AG atinge valores mais baixos durante boa parte do processo, o que evidencia sua capacidade de exploração rápida do espaço de busca nesta instância.

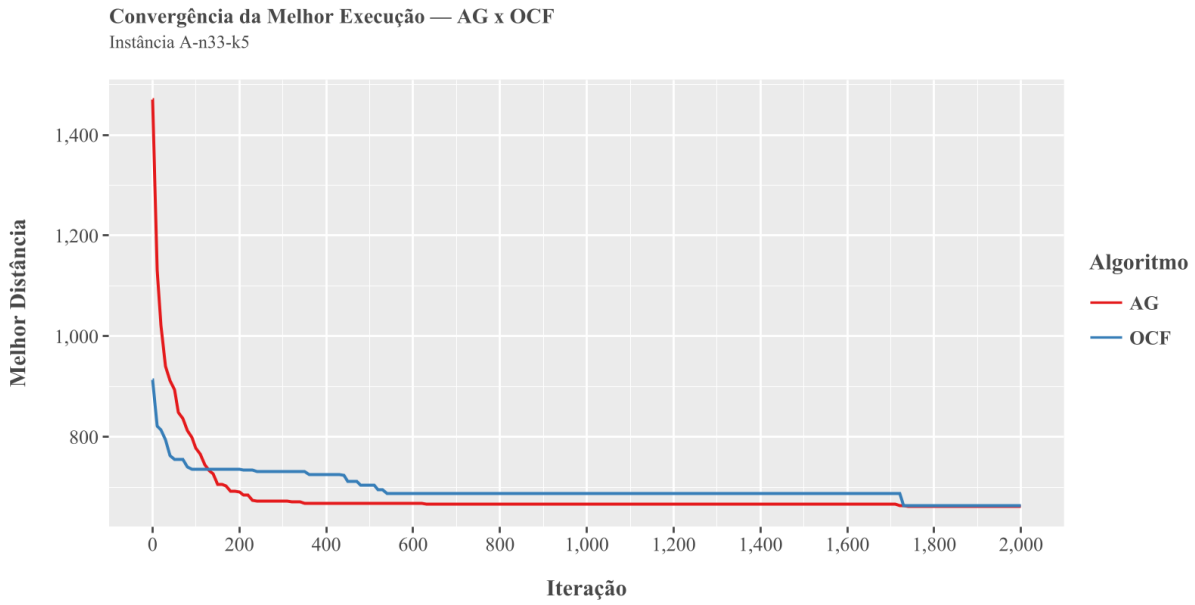


Figura 7 – Convergência da Melhor Execução – Instância Pequena

O *boxplot* das 30 execuções, apresentado na Figura 8, evidencia diferenças na estabilidade das soluções obtidas por cada algoritmo. O AG (vermelho) apresenta maior dispersão, com mediana em torno de 713, primeiro quartil em 688 e terceiro quartil em 720. Em contraste, a OCF (azul) exibe uma distribuição mais concentrada, com mediana próxima de 698 e quartis entre 685 e 705.

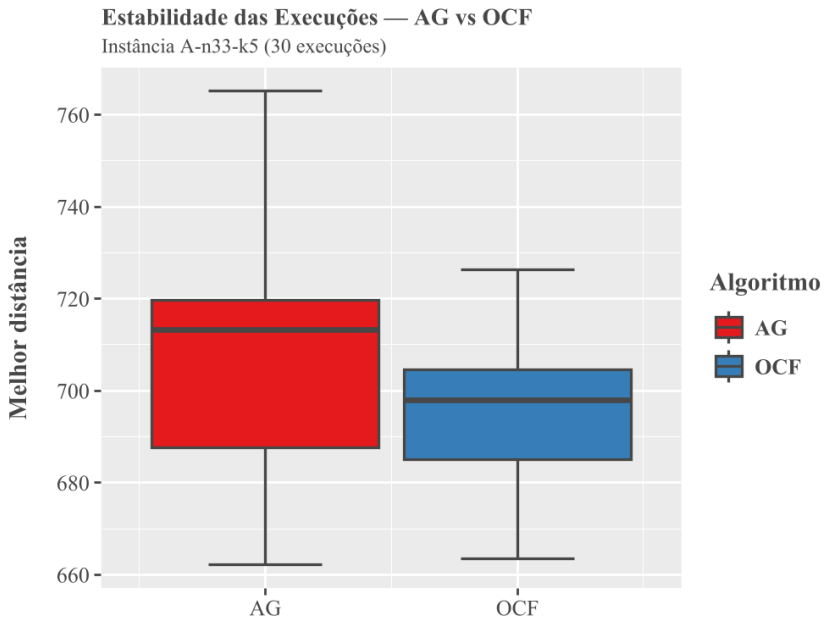
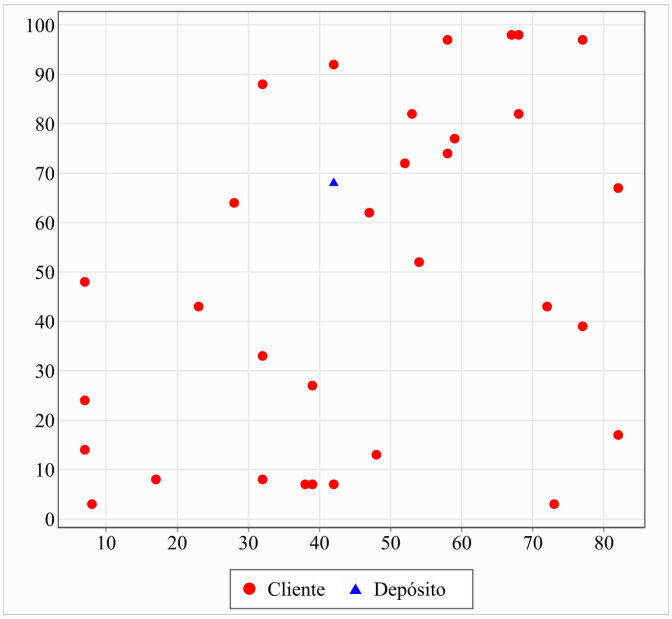


Figura 8 – Estabilidade das Execuções – Instância Pequena

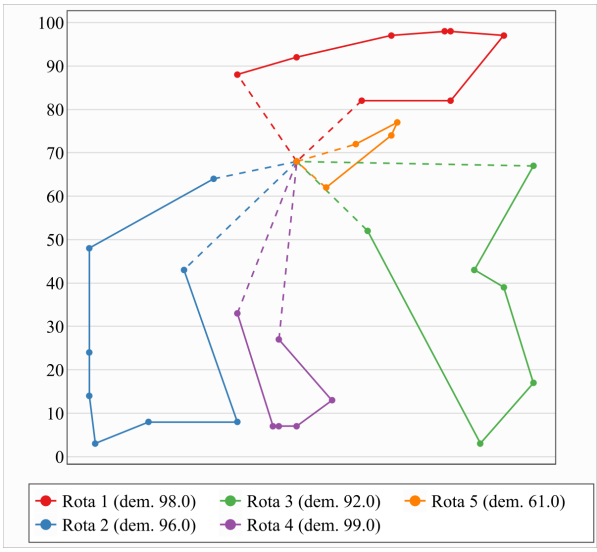
Nesse sentido, observa-se que ambos os algoritmos apresentam valores mínimos e máximos semelhantes, porém o intervalo interquartil mais estreito da OCF indica menor variabilidade

entre as execuções. Esse comportamento sugere que a OCF oferece maior previsibilidade nos resultados, enquanto o AG apresenta maior variação na qualidade das soluções encontradas.

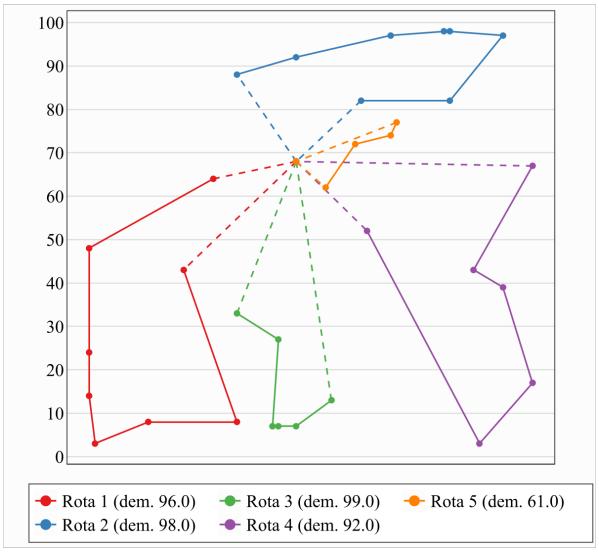
Como mostrado nas Figuras 9(a), (b) e (c), ilustram-se, respectivamente, a distribuição dos clientes e do depósito, a melhor solução do AG e a melhor solução da OCF. Nota-se que ambas as abordagens geraram cinco rotas respeitando a capacidade máxima de 100 unidades, com demandas variando entre 61 e 99 dessas unidades. As soluções mostram uma boa setorização geográfica e baixa sobreposição entre rotas, indicando que os algoritmos foram eficazes em distribuir os atendimentos de maneira equilibrada.



(a) Clientes e depósito



(b) Melhor solução do AG



(c) Melhor solução da OCF

Figura 9 – Comparação das Melhores Soluções – Instância Pequena

Em síntese, na instância A-n33-k5, a OCF demonstrou vantagem sobre o AG em termos de consistência e estabilidade (ou seja, suas 30 execuções resultaram em distâncias mais agru-

padas e comportamentos mais previsíveis) e tempo de execução, enquanto o AG obteve leve superioridade na melhor solução individual (menor distância total obtida em uma única execução). Essa análise reforça a complementaridade entre as abordagens, sugerindo que a escolha do método mais adequado depende das prioridades da aplicação: se o foco for a obtenção da melhor solução possível em uma única execução, o AG pode ser preferível; se a prioridade for a estabilidade e o menor tempo computacional, a OCF mostra-se mais vantajosa em instâncias de menor porte.

4.0.2 Instância Média – B-n50-k7

Em relação à instância B-n50-k7, a Tabela 5 apresenta um resumo comparativo dos resultados entre AG e OCF. O AG demonstrou desempenho superior na qualidade das soluções, com melhor distância de 747,86 (GAP de 0,93%) contra 772,39 (GAP de 4,24%) da OCF. A média das distâncias do AG também foi ligeiramente inferior (781,79 vs 788,75). No entanto, a OCF apresentou estabilidade substancialmente maior, com desvio padrão de apenas 7,65, o que representa aproximadamente 70% menor que o valor obtido pelo AG (25,41). Além disso, em termos de eficiência computacional, a OCF manteve vantagem, com tempo médio de 19,70 segundos, aproximadamente 15% menor que o AG (23,27 segundos).

Métrica	**AG**	**OCF**
Ótimo conhecido	741,00	
Melhor distância	747,86	772,39
Média das distâncias	781,79	788,75
Desvio padrão	25,41	7,65
GAP melhor solução (%)	0,93	4,24
GAP médio (%)	5,50	6,44
Tempo médio (seg)	23,27	19,70

Tabela 5 – Comparativo AG e OCF – Instância Média

O gráfico da Figura 10 ilustra os padrões de convergência da melhor execução (aquela com menor distância final) de ambos os algoritmos na instância de médio porte. A análise evidencia comportamentos distintos de convergência entre o AG e a OCF.

O AG (linha vermelha) apresenta uma queda inicial expressiva, saindo de valores superiores a 2200 e alcançando a faixa dos 800 nas primeiras 100 iterações, seguida por uma fase de estabilização com pequenas flutuações.

A OCF (linha azul) inicia em um valor menor (cerca de 1000) e também converge rapidamente para a faixa dos 800, mas com uma curva menos acentuada. É interessante notar que após a convergência inicial, ambos os algoritmos apresentam comportamentos muito similares, estabilizando em valores próximos. A OCF mostra uma ligeira melhoria próximo à iteração 1300. Nesta instância, a diferença entre os valores finais de convergência dos dois algoritmos é mínima.

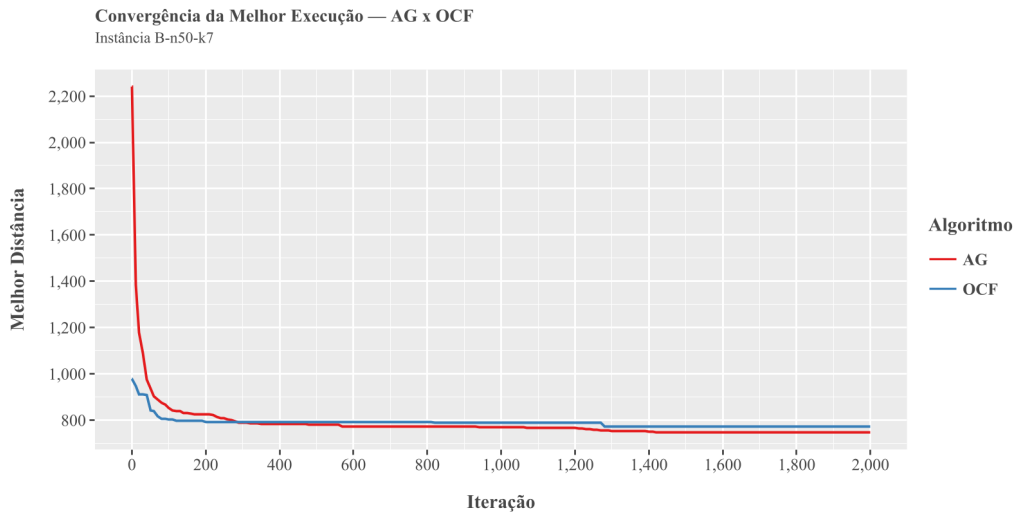


Figura 10 – Convergência da Melhor Execução – Instância Média

Observa-se na Figura 11 a comparação da estabilidade na instância de médio porte, representada pelos *boxplots* dos dois algoritmos.

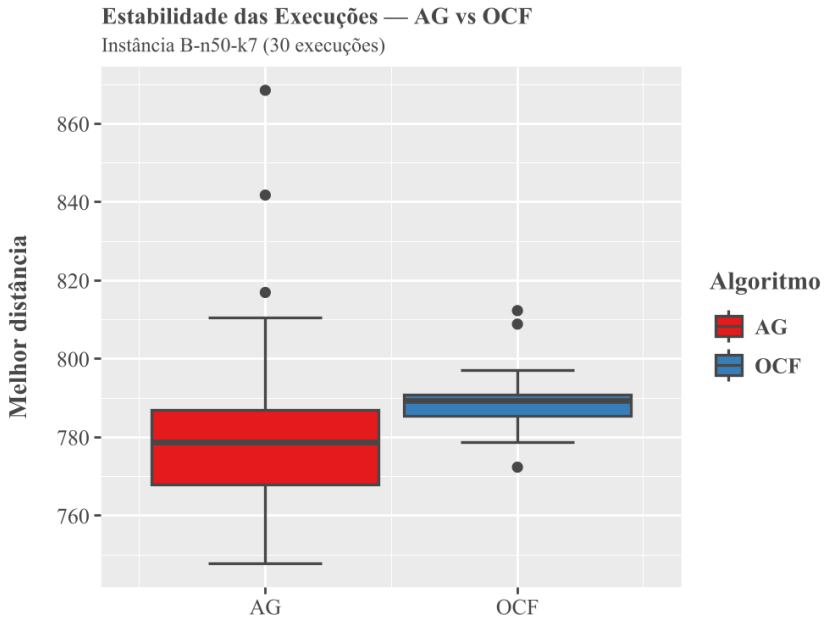


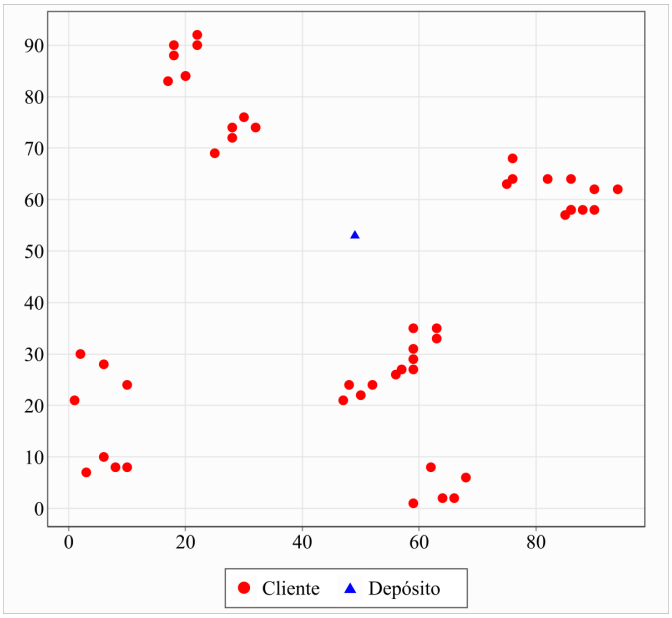
Figura 11 – Estabilidade das Execuções – Instância Média

Observa-se uma diferença clara nos padrões de distribuição: o AG (em vermelho) exibe uma dispersão maior, com mediana em torno de 779 e quartis entre 768 e 787, além de apresentar *outliers* que chegam a ultrapassar 810. A OCF (em azul) mostra uma distribuição significativamente mais estreita, com mediana próxima de 789 e quartis mais próximos entre si (785 e 791).

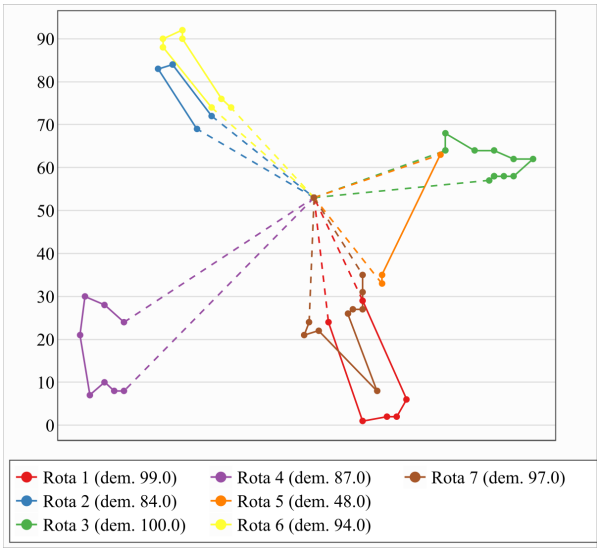
Embora a mediana da OCF seja ligeiramente superior, indicando soluções de qualidade média um pouco inferior, sua variabilidade é consideravelmente menor, como evidenciado pelo intervalo interquartil mais estreito. Este comportamento sugere um compromisso interessante: o

AG oferece possibilidade de encontrar soluções de melhor qualidade, mas com menor garantia de estabilidade, enquanto a OCF proporciona resultados mais previsíveis, porém com tendência a convergir para soluções de qualidade levemente inferior na instância média.

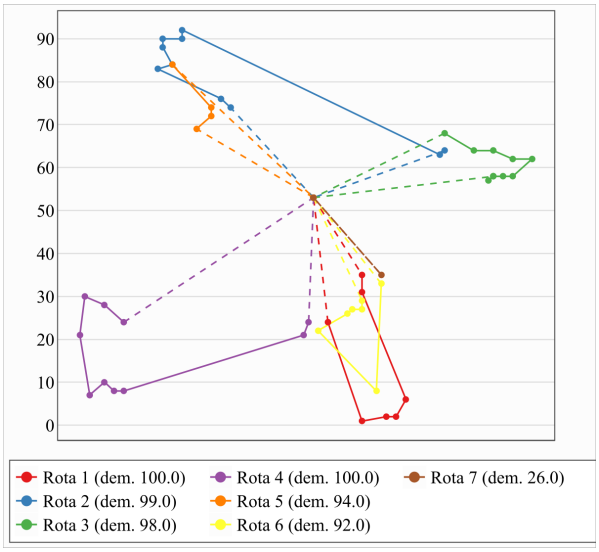
A representação gráfica na Figura 12 apresenta, respectivamente em (a), (b) e (c), a distribuição dos clientes e depósito, a melhor solução do AG e a melhor solução da OCF para a instância de médio porte.



(a) Clientes e depósito



(b) Melhor solução do AG



(c) Melhor solução da OCF

Figura 12 – Comparação das Melhores Soluções – Instância Média

As melhores soluções revelam diferenças na configuração espacial das rotas. Ambos os algoritmos geraram sete rotas respeitando os limites de capacidade, mas com características de setorização distintas. A solução do AG apresenta sobreposições na região central e trajetórias

paralelas que sugerem redundâncias. A OCF, por sua vez, mostra agrupamentos mais irregulares, com algumas rotas percorrendo áreas distantes de seus núcleos principais.

Em conclusão, para a instância B-n50-k7, o AG demonstrou superioridade na qualidade das soluções geradas, produzindo rotas com distâncias totais menores e mais próximas do ótimo conhecido. Por outro lado, a OCF destacou-se pela maior estabilidade entre execuções e menor tempo computacional. Esta análise sugere que, para problemas de porte médio, o AG pode ser mais indicado quando a prioridade é a qualidade da solução, enquanto a OCF oferece vantagens em cenários onde estabilidade e tempo de processamento são fatores críticos.

4.0.3 Instância Grande – P-n101-k4

Em relação à instância de maior porte, a Tabela 6 apresenta os resultados, evidenciando uma diferença expressiva no desempenho dos algoritmos.

Métrica	**AG**	**OCF**
Ótimo conhecido	681,00	
Melhor distância	868,27	760,56
Média das distâncias	1016,68	819,20
Desvio padrão	69,34	31,88
GAP melhor solução (%)	27,50	11,68
GAP médio (%)	49,29	20,29
Tempo médio (seg)	38,63	67,00

Tabela 6 – Comparativo AG e OCF – Instância Grande

A OCF obteve resultados significativamente superiores em termos de qualidade das soluções, com uma melhor distância de 760,56 (GAP de 11,68%) contra 868,27 (GAP de 27,50%) do AG. A média das distâncias da OCF (819,20) também foi substancialmente inferior à do AG (1016,68), representando uma redução de aproximadamente 19%. O desvio padrão da OCF (31,88) foi aproximadamente 54% menor que o do AG (69,34), indicando maior estabilidade. No entanto, o tempo médio de execução da OCF (67,00 segundos) foi significativamente superior ao do AG (38,63 segundos), representando um aumento de aproximadamente 73%.

O gráfico da Figura 13 ilustra as curvas de convergência da melhor execução (aquela com menor distância final) de ambos os algoritmos na instância de maior porte, revelando comportamentos significativamente distintos.

O AG (linha vermelha) apresenta uma redução contínua e gradual da distância ao longo de todas as iterações, partindo de valores superiores a 3000 e atingindo menos de 900 ao final, sem estabilização evidente. Este comportamento sugere que o algoritmo poderia se beneficiar de mais iterações para alcançar convergência completa.

Em contraste, a OCF (linha azul) inicia com um valor inicial muito inferior (cerca de 1200) e exibe uma redução inicial acentuada até aproximadamente a iteração 100, seguida por

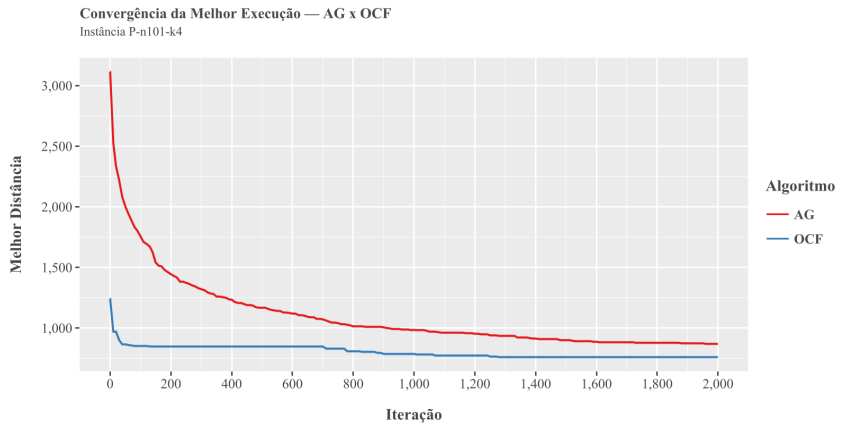


Figura 13 – Convergência da Melhor Execução – Instância Grande

um longo período de estabilidade. A partir da iteração 800, a OCF realiza melhorias pontuais, mas mantém uma clara vantagem sobre o AG durante todo o processo. A diferença na solução final é bastante expressiva, evidenciando a superioridade da OCF em instâncias de maior complexidade. Este comportamento indica que a OCF consegue explorar o espaço de busca de forma mais efetiva em problemas complexos, encontrando rapidamente boas soluções iniciais.

Para a instância de maior porte, a Figura 14 apresenta os *boxplots* utilizados na comparação da estabilidade dos algoritmos.

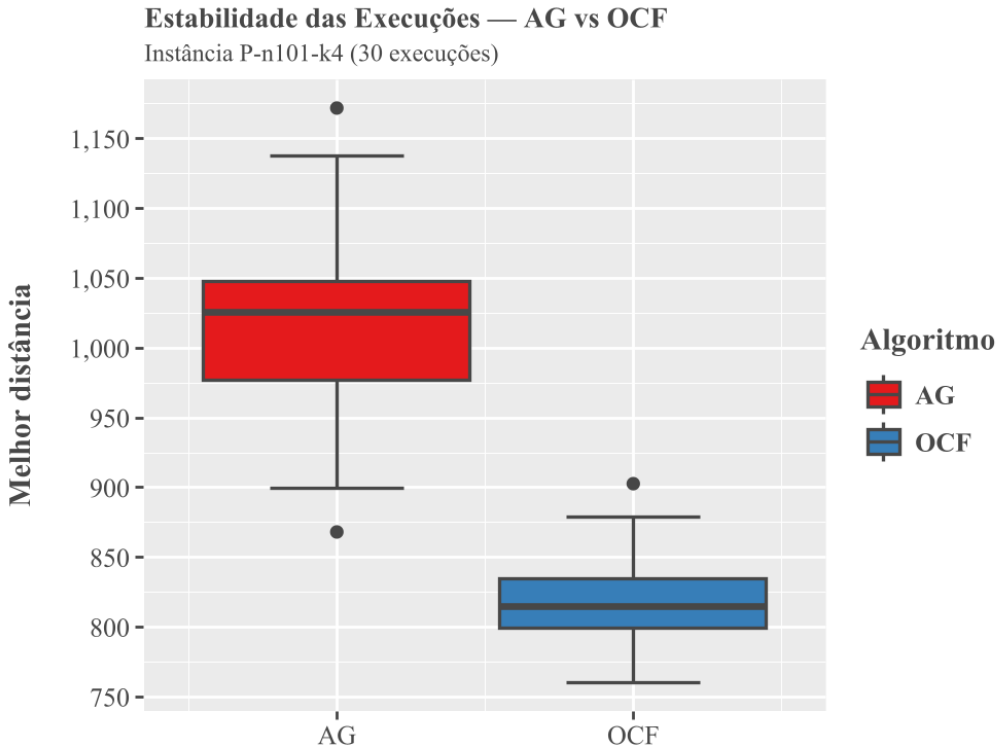


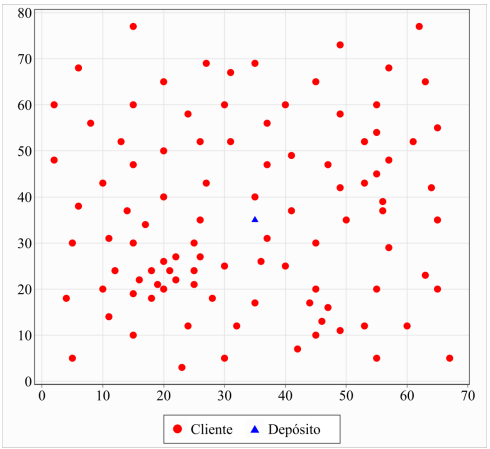
Figura 14 – Estabilidade das Execuções – Instância Grande

A diferença na posição das caixas é notável, com o AG (em vermelho) apresentando

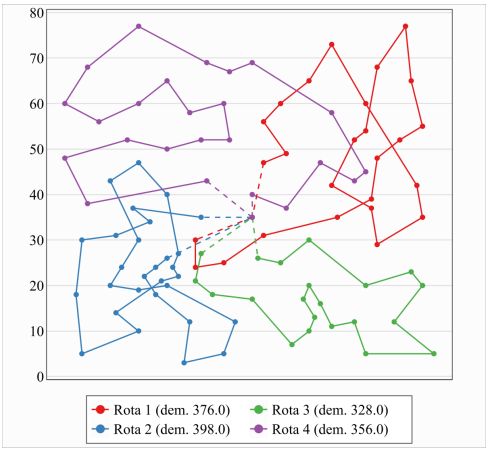
valores consideravelmente mais altos, com mediana em torno de 1026, primeiro quartil em 977 e terceiro quartil em 1048. A OCF (em azul) exibe distribuição com valores substancialmente menores, com mediana aproximada de 814 e quartis em 799 e 835.

Além da significativa diferença nas medidas de tendência central, também é evidente a maior variabilidade no AG, com um intervalo interquartil mais amplo. Ambos os algoritmos apresentam *outliers*, sendo um acima de 1150 no AG e outro acima de 900 na OCF. Esta análise demonstra claramente a superioridade da OCF na instância de maior porte, não apenas em termos de qualidade das soluções, mas também em estabilidade, apresentando menor variação entre as execuções.

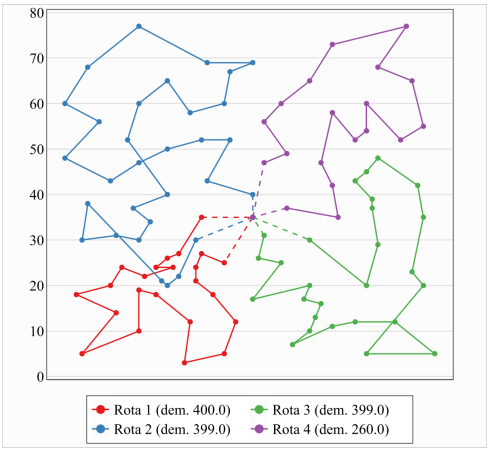
O conjunto de subfiguras apresentado na Figura 15 apresenta, respectivamente em (a), (b) e (c), a distribuição dos clientes e do depósito, a melhor solução do AG e a melhor solução da OCF para a instância de grande porte.



(a) Clientes e depósito



(b) Melhor solução do AG



(c) Melhor solução da OCF

Figura 15 – Comparação das Melhores Soluções – Instância Grande

Na análise visual das melhores soluções, observa-se que ambos os algoritmos geraram quatro rotas respeitando os limites de capacidade. No entanto, a solução da OCF apresenta uma setorização espacial mais definida, com as rotas claramente delimitadas e minimizando o cruza-

mento ou compartilhamento de áreas (como pode ser visto pela ausência de linhas sobrepostas em múltiplas regiões). Em contrapartida, a solução do AG exibe traçados mais longos e com diversas sobreposições, especialmente na área central onde múltiplos veículos parecem cruzar os mesmos pontos, resultando em uma organização menos coesa. Esta diferença reflete a capacidade superior da OCF em lidar com a complexidade do problema em instâncias de grande porte, resultando em uma organização logística mais eficiente.

Em síntese, na instância P-n101-k4, a OCF demonstrou superioridade expressiva sobre o AG em quase todos os aspectos analisados, exceto no tempo computacional. A qualidade das soluções, estabilidade entre execuções e setorização espacial das rotas foram significativamente melhores na OCF, o que compensa o maior custo computacional em aplicações onde a qualidade da solução é prioritária. Estes resultados sugerem que, para instâncias complexas do PRVC, a OCF apresenta maior escalabilidade e robustez, tornando-se a alternativa preferencial, especialmente quando o tempo de processamento não é um fator limitante crítico.

5 CONCLUSÃO

Este capítulo apresenta as principais considerações finais obtidas a partir da análise comparativa entre o AG e a OCF aplicadas ao PRVC. As avaliações consideraram métricas de qualidade das soluções, tempo de execução e estabilidade, utilizando diferentes instâncias *benchmark*.

5.1 Contribuições

A principal contribuição deste trabalho foi a implementação e análise comparativa detalhada de duas meta-heurísticas amplamente utilizadas para problemas combinatórios complexos, no contexto específico do PRVC.

Os testes demonstraram que o AG apresentou melhor desempenho na obtenção de soluções de alta qualidade em instâncias pequenas e médias. A OCF, por sua vez, destacou-se pela estabilidade e robustez, especialmente em instâncias de maior complexidade, cenário em que o AG demonstrou dificuldade para atingir a convergência completa dentro do limite de iterações estabelecido, evidenciando sua sensibilidade aos parâmetros fixos em problemas de grande escala.

Além disso, o estudo utilizou uma metodologia sistemática, com 30 execuções por instância, garantindo confiabilidade estatística às análises. Os resultados reforçam a ideia de que a escolha do algoritmo mais adequado depende do porte do problema e dos critérios priorizados (qualidade, tempo ou estabilidade).

5.2 Limitações

O estudo concentrou-se na análise de três instâncias *benchmark* representativas de diferentes portes, o que foi suficiente para evidenciar padrões de comportamento dos algoritmos. No entanto, a inclusão de um número maior de instâncias ou de cenários mais variados poderia fortalecer ainda mais a generalização dos resultados.

Adicionalmente, optou-se por utilizar implementações puras dos algoritmos, sem recorrer a técnicas de hibridização ou paralelismo. Essa escolha foi motivada pelo objetivo de realizar uma comparação direta e justa, mas reconhece-se que melhorias de desempenho podem ser obtidas com versões mais sofisticadas.

Por fim, uma limitação inerente à metodologia de comparação adotada é a utilização de uma parametrização fixa. A calibração, que buscou um tempo de execução similar para uma comparação justa, pode não representar a configuração ótima para cada algoritmo em cada tipo

de instância. A dificuldade de convergência do AG no problema de grande porte é um indicativo dessa limitação.

5.3 Trabalhos Futuros

Com base nas lições aprendidas neste trabalho, especialmente a observação sobre a convergência do AG, a principal sugestão para investigações futuras é a exploração de estratégias de ajuste de parâmetros por instância. Em vez de uma configuração única, poder-se-ia desenvolver uma abordagem que calibra os parâmetros mais sensíveis — como a taxa de mutação ou o tamanho da população — de acordo com as características do problema a ser resolvido. Essa abordagem tem o potencial de extrair o máximo desempenho de cada heurística em diferentes cenários.

Além dessa linha principal de investigação, outras oportunidades incluem:

- Investigar abordagens híbridas entre AG e OCF, aproveitando as vantagens de cada método;
- Aplicar técnicas de paralelismo para reduzir o tempo de execução em instâncias grandes;
- Explorar métricas multiobjetivo, considerando simultaneamente distância, tempo e balanceamento de carga;
- Desenvolver mecanismos de replanejamento dinâmico para lidar com imprevistos, como a interdição de rotas previamente selecionadas.

Essas propostas visam não apenas aprofundar o conhecimento sobre os algoritmos metaheurísticos, mas também aproximar os estudos acadêmicos das demandas reais da logística moderna.

REFERÊNCIAS

ABOUSAEIDI, M.; FAUZI, R.; MUHAMAD, R. Geographic information system (gis) modeling approach to determine the fastest delivery routes. *Saudi Journal of Biological Sciences*, v. 23, n. 5, p. 555–564, 2016. ISSN 1319-562X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1319562X15001370>. Citado na página 17.

AHMED, Z. H. et al. Genetic crossover operators for the capacitated vehicle routing problem. *Computers, Materials & Continua*, v. 75, n. 1, 2023. Citado 4 vezes nas páginas 14, 24, 25 e 30.

AKPINAR, S. Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem. *Expert Systems with Applications*, v. 61, p. 28–38, 2016. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417416302482>. Citado 2 vezes nas páginas 17 e 19.

AUGERAT, P. *Approche polyèdrale du problème de tournées de véhicules*. Tese (Doutorado) — Institut National Polytechnique de Grenoble-INPG, 1995. Citado na página 35.

AZAD, T.; HASIN, M. A. A. Capacitated vehicle routing problem using genetic algorithm: a case of cement distribution. *International Journal of Logistics Systems and Management*, Inderscience Publishers (IEL), v. 32, n. 1, p. 132–146, 2019. Citado na página 23.

CAI, J. et al. A dynamic space reduction ant colony optimization for capacitated vehicle routing problem. *Soft Computing*, Springer, v. 26, n. 17, p. 8745–8756, 2022. Citado 2 vezes nas páginas 27 e 28.

COSTA, J. *Repositório do projeto de comparação entre AG e OCF para o PRVC*. 2025. <https://github.com/jardsonn/ag-ocf-analise/>. Acesso em: 11 maio 2025. Citado 2 vezes nas páginas 32 e 53.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. *Management science*, Informs, v. 6, n. 1, p. 80–91, 1959. Citado na página 17.

DESHMUKH, A. R.; DORLE, S. S. Bio-inspired optimization algorithms for improvement of vehicle routing problems. In: *2015 7th International Conference on Emerging Trends in Engineering & Technology (ICETET)*. [S.l.: s.n.], 2015. p. 14–18. Citado na página 14.

DORIGO, M. Positive feedback as a search strategy. *Technical report*, p. 91–16, 1991. Citado na página 26.

DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine*, IEEE, v. 1, n. 4, p. 28–39, 2006. Citado 3 vezes nas páginas 26, 27 e 28.

DORIGO, M.; GAMBARDELLA, L. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 53–66, 1997. Citado na página 26.

ELSHAER, R.; AWAD, H. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, v. 140, p. 106242, 2020. ISSN 0360-8352. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360835219307119>. Citado na página 18.

GOMES, A. C. et al. Logistics management in e-commerce: challenges and opportunities. *Revista de Gestão e Secretariado*, v. 14, n. 5, p. 7252–7272, 2023. Citado na página 14.

HAROUN, S. A.; JAMAL, B.; HICHAM, E. H. A performance comparison of ga and aco applied to tsp. *International Journal of Computer Applications*, Foundation of Computer Science, v. 117, n. 19, p. 28–35, 2015. ISSN 0975-8887. Citado na página 30.

HOLLAND, J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. [S.l.]: University of Michigan Press, 1975. ISBN 9780472084609. Citado na página 20.

HUANG, Y.-H. et al. Solving the feeder vehicle routing problem using ant colony optimization. *Computers & Industrial Engineering*, v. 127, 10 2018. Citado na página 15.

KUMARI, M. et al. Utilizing a hybrid metaheuristic algorithm to solve capacitated vehicle routing problem. *Results in Control and Optimization*, Elsevier, v. 13, p. 100292, 2023. Citado na página 14.

LAPORTE, G. The traveling salesman problem, the vehicle routing problem, and their impact on combinatorial optimization. *International Journal of Strategic Decision Sciences*, v. 1, n. 2, p. 82–92, apr–jun 2010. Citado 2 vezes nas páginas 14 e 18.

LIN, S.-W. et al. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, Elsevier, v. 36, n. 2, p. 1505–1512, 2009. Citado na página 19.

MARSH, L.; ONOF, C. Stigmergic epistemology, stigmergic cognition. *Cognitive Systems Research*, Elsevier, v. 9, n. 1-2, p. 136–149, 2007. Citado na página 26.

MATIJEVIć, L. Metaheuristic approaches for the green vehicle routing problem. *Yugoslav Journal of Operations Research*, v. 33, p. 16–16, 01 2022. Citado na página 19.

MAZZEO, S.; LOISEAU, I. An ant colony algorithm for the capacitated vehicle routing. *Electronic Notes in Discrete Mathematics*, Elsevier, v. 18, p. 181–186, 2004. Citado 2 vezes nas páginas 27 e 28.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998. Citado 2 vezes nas páginas 20 e 21.

MOHAMMED, M. A.; AHMAD, M. S.; MOSTAFA, S. A. Using genetic algorithm in implementing capacitated vehicle routing problem. In: IEEE. *2012 International conference on computer & information science (ICCIS)*. [S.l.], 2012. v. 1, p. 257–262. Citado 3 vezes nas páginas 21, 23 e 25.

OCHELSKA-MIERZEJEWSKA, J.; PONISZEWSKA-MARAńDA, A.; MARAńDA, W. Selected genetic algorithms for vehicle routing problem solving. *Electronics*, v. 10, n. 24, 2021. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/10/24/3147>. Citado 8 vezes nas páginas 15, 20, 22, 23, 24, 25, 29 e 35.

OTHMAN, W. A. F. W. et al. Solving vehicle routing problem using ant colony optimisation (aco) algorithm. *International Journal of Research and Engineering*, International Journal of Research and Engineering, v. 5, n. 9, p. 500–507, 2018. ISSN 2348-7860. Citado na página 30.

Pontifícia Universidade Católica do Rio de Janeiro. *CVRPLIB: Capacitated Vehicle Routing Problem Library*. 2025. Acesso em: 8 mar. 2025. Disponível em: <http://vrp.galgos.inf.puc-rio.br/>. Citado na página 36.

RAJ, R. et al. Assessing the e-commerce last-mile logistics' hidden risk hurdles. *Cleaner Logistics and Supply Chain*, Elsevier, v. 10, p. 100131, 2024. Citado na página 14.

SHARMA, S. K.; ROUTROY, S.; YADAV, U. Vehicle routing problem: recent literature review of its variants. *International Journal of Operational Research*, Inderscience Publishers (IEL), v. 33, n. 1, p. 1–31, 2018. Citado na página 18.

SIVANANDAM, S.; DEEPA, S. Genetic algorithms. In: ____. *Introduction to Genetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 15–37. ISBN 978-3-540-73190-0. Disponível em: <https://doi.org/10.1007/978-3-540-73190-0_2>. Citado na página 20.

SOUZA, A. M. de et al. Better safe than sorry: a vehicular traffic re-routing based on traffic conditions and public safety issues. *Journal of Internet Services and Applications*, Springer, v. 10, p. 1–18, 2019. Citado na página 17.

Statista. *Retail e-commerce sales worldwide from 2014 to 2027*. 2023. Acesso em : 19 out. 2024. Disponível em: <https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/>. Citado na página 15.

STODOLA, P. et al. Using the ant colony optimization algorithm for the capacitated vehicle routing problem. In: IEEE. *Proceedings of the 16th international conference on mechatronics-mechatronika 2014*. [S.l.], 2014. p. 503–510. Citado na página 27.

TOTH, P.; VIGO, D. (Ed.). *Vehicle Routing: Problems, Methods, and Applications*. 2. ed. Philadelphia, USA: Society for Industrial and Applied Mathematics (SIAM), 2014. ISBN 978-1-61197-358-7. Citado na página 15.

WANG, Y. et al. Cross-border e-commerce firms as supply chain integrators: The management of three flows. *Industrial Marketing Management*, Elsevier, v. 89, p. 72–88, 2020. Citado na página 14.

WORONIUK, C.; MARINOV, M. Simulation modelling to analyse the current level of utilisation of sections along a rail route. *Journal of Transport Literature*, Sociedade Brasileira de Planejamento dos Transportes, v. 7, n. 2, p. 235–252, Apr 2013. ISSN 2238-1031. Disponível em: <https://www.scielo.br/j/jtl/a/4LX3pXX9zTHd75gfWTx6SMr/>. Citado na página 17.

ZHANG, J. et al. Vehicle routing problem with fuel consumption and carbon emission. *International Journal of Production Economics*, Elsevier, v. 170, 2015. Citado na página 15.

APÊNDICE A – FRAGMENTOS DO CÓDIGO-FONTE

Segue abaixo a apresentação de fragmentos selecionados do código-fonte desenvolvido para este trabalho. Os trechos a seguir representam as partes centrais da implementação do Algoritmo Genético e da Otimização por Colônia de Formigas, incluindo suas respectivas classes principais e operadores essenciais.

O código-fonte completo, incluindo os demais componentes auxiliares, ferramentas de análise e geração de gráficos, encontra-se disponível publicamente no repositório GitHub associado a este trabalho (COSTA, 2025).

A.1 Algoritmo Genético

A.1.1 Classe principal

```
class GeneticAlgorithm private constructor(
    private val selectionOperator: SelectionOperator,
    private val crossoverOperator: CrossoverOperator,
    private val mutationOperator: MutationOperator,
    private val problem: VehicleRoutingProblem,
    private val populationSize: Int,
    private val maxGenerations: Int,
    private val crossoverRate: Double,
    private val mutationRate: Double,
    private val elitism: Boolean,
) {
    private lateinit var population: Population
    private lateinit var bestIndividual: Individual
    private val bestFitnessHistory = mutableListOf<Double>()
    private val averageFitnessHistory = mutableListOf<Double>()

    fun run(onProgress: (iteration: Int, bestDistance: Double, avgDistance:
    Double) -> Unit): AlgorithmExecutionData {
        val startTime = System.currentTimeMillis()
        val iterationDataList = mutableListOf<IterationData>()

        initializePopulation()

        var generation = 0
        while (generation < maxGenerations) {
```

```
            population = evolvePopulation()
            evaluatePopulation()
            updateBestIndividual()
            updateStatistics()

            val bestRoutes = problem.decodeRoutes(bestIndividual)
            val totalDistance = bestRoutes.sumOf { it.getTotalDistance() }
            val averageFitness = population.getAverageFitness()

            onProgress(generation, totalDistance, averageFitness)

            iterationDataList.add(
                IterationData(
                    iteration = generation,
                    totalDistance = totalDistance,
                    averageDistance = averageFitness,
                    routes = bestRoutes,
                    executionTimeMs = System.currentTimeMillis() -
startTime
                )
            )

            generation++
        }

        val totalExecutionTime = System.currentTimeMillis() - startTime
        val finalBestRoutes = problem.decodeRoutes(bestIndividual)
        val bestRoute = finalBestRoutes.minByOrNull { it.getTotalDistance()
 } ?: finalBestRoutes.firstOrNull() ?: throw IllegalStateException("
Nenhuma rota encontrada")

        return AlgorithmExecutionData(
            algorithmName = "Algoritmo Genético",
            iterations = iterationDataList,
            totalExecutionTime = totalExecutionTime,
            routes = finalBestRoutes,
            bestRoute = bestRoute
        )
    }

    private fun initializePopulation() {
        population = Population(populationSize, problem).apply {
            initializeRandomly()
        }
        evaluatePopulation()
        updateBestIndividual()
    }
```

```
69
70      private fun evolvePopulation(): Population {
71          val newPopulation = Population(populationSize, problem)
72
73          if (elitism) {
74              newPopulation.addIndividual(Individual(bestIndividual))
75          }
76
77          while (newPopulation.size() < populationSize) {
78              val parent1 = selectionOperator.select(population)
79              val parent2 = selectionOperator.select(population)
80
81              if (parent1 != null && parent2 != null) {
82                  var offspring1 = Individual(parent1)
83                  var offspring2 = Individual(parent2)
84
85                  if (Math.random() < crossoverRate) {
86                      val offspring = crossoverOperator.crossover(parent1,
    parent2)
87                      offspring1 = offspring[0]
88                      offspring2 = offspring[1]
89                  }
90
91                  if (Math.random() < mutationRate) {
92                      mutationOperator.mutate(offspring1)
93                      mutationOperator.mutate(offspring2)
94                  }
95
96                  newPopulation.addIndividual(offspring1)
97                  if (newPopulation.size() < populationSize) {
98                      newPopulation.addIndividual(offspring2)
99                  }
100             }
101         }
102
103         return newPopulation
104     }
105
106     private fun evaluatePopulation() {
107         population.individuals.forEach { it.calculateFitness() }
108     }
109
110     private fun updateBestIndividual() {
111         val currentBest = population.getFittest()
112         if (!::bestIndividual.isInitialized || currentBest.getFitness() >
    bestIndividual.getFitness()) {
113             bestIndividual = Individual(currentBest)
```

```
114         }
115     }
116
117     private fun updateStatistics() {
118         bestFitnessHistory.add(bestIndividual.getFitness())
119         averageFitnessHistory.add(population.getAverageFitness())
120     }
121
122     companion object {
123         inline fun build(block: Builder.() -> Unit): GeneticAlgorithm =
    Builder().apply(block).build()
124     }
125
126     class Builder {
127         var selectionOperator: SelectionOperator? = null
128         var crossoverOperator: CrossoverOperator? = null
129         var mutationOperator: MutationOperator? = null
130         var problem: VehicleRoutingProblem? = null
131         var populationSize: Int = GeneticConfig.POPULATION_SIZE
132         var maxGenerations: Int = GeneticConfig.MAX_GENERATIONS
133         var crossoverRate: Double = GeneticConfig.CROSSOVER_RATE
134         var mutationRate: Double = GeneticConfig.MUTATION_RATE
135         var elitism: Boolean = true
136
137         fun build(): GeneticAlgorithm {
138             requireNotNull(selectionOperator) { "Operador de seleção não
    especificado" }
139             requireNotNull(crossoverOperator) { "Operador de crossover não
    especificado" }
140             requireNotNull(mutationOperator) { "Operador de mutação não
    especificado" }
141             requireNotNull(problem) { "Problema não especificado" }
142
143             return GeneticAlgorithm(
144                 selectionOperator = selectionOperator!!,
145                 crossoverOperator = crossoverOperator!!,
146                 mutationOperator = mutationOperator!!,
147                 problem = problem!!,
148                 populationSize = populationSize,
149                 maxGenerations = maxGenerations,
150                 crossoverRate = crossoverRate,
151                 mutationRate = mutationRate,
152                 elitism = elitism,
153             )
154         }
155     }
156 }
```

Código-Fonte A.1 – Classe principal do Algoritmo Genético

A.1.2 Classe `Individual`

```
1
2  class Individual(
3      val problem: VehicleRoutingProblem,
4  ) {
5      val chromosome: IntArray = IntArray(problem.dimension * 2)
6      private var fitness: Double = 0.0
7      private var fitnessCalculated: Boolean = false
8
9      constructor(other: Individual) : this(other.problem) {
10         other.chromosome.copyInto(chromosome)
11         fitness = other.fitness
12         fitnessCalculated = other.fitnessCalculated
13     }
14
15     fun generateRandomChromosome() {
16         val dimension = problem.dimension
17         val trucks = problem.numberOfTrucks
18
19         for (i in 0 until dimension) {
20             chromosome[i] = (1..trucks).random()
21         }
22
23         for (i in dimension until dimension * 2) {
24             chromosome[i] = (0 until dimension).random()
25         }
26     }
27
28     fun getGene(index: Int) = chromosome[index]
29
30     fun setGene(index: Int, value: Int) {
31         chromosome[index] = value
32         fitnessCalculated = false
33     }
34
35     fun calculateFitness() {
36         if (!fitnessCalculated) {
37             fitness = problem.evaluateFitness(this)
38             fitnessCalculated = true
39         }
40     }
41
```

```
42      fun getFitness(): Double {
43          if (!fitnessCalculated) calculateFitness()
44          return fitness
45      }
46  }
```

Código-Fonte A.2 – Classe `Individual` utilizada pelo AG

A.1.3 Classe `Population`

```
1   class Population(
2       private val maxSize: Int,
3       private val problem: VehicleRoutingProblem
4   ) {
5       private val _individuals = ArrayList<Individual>(maxSize)
6       val individuals: List<Individual> get() = _individuals.toList()
7
8       fun initializeRandomly() {
9           repeat(maxSize) {
10              Individual(problem).also { individual ->
11                  individual.generateRandomChromosome()
12                  _individuals.add(individual)
13              }
14          }
15      }
16
17      fun addIndividual(individual: Individual) {
18          if (_individuals.size < maxSize) {
19              _individuals.add(individual)
20          }
21      }
22
23      fun getFittest(): Individual = _individuals.maxByOrNull { it.getFitness
        () }
24          ?: throw IllegalStateException("Population is empty")
25
26      fun getAverageFitness(): Double = _individuals
27          .sumOf { it.getFitness() } / _individuals.size
28
29      operator fun get(index: Int): Individual = _individuals[index]
30
31      fun size(): Int = _individuals.size
32  }
```

Código-Fonte A.3 – Classe `Population` utilizada pelo AG

A.1.4 Operadores Genéticos

A.1.4.1 Seleção

```
class TournamentSelection(private val tournamentSize: Int) :
    SelectionOperator {

    override fun select(population: Population): Individual? {
        var best: Individual? = null
        for (i in 0 until tournamentSize) {
            val competitor: Individual = population.individuals[RandomUtils
    .nextInt(0, population.size())]
            if (best == null || competitor.getFitness() > best.getFitness()
    ) {
                best = competitor
            }
        }
        return best
    }
}
```

Código-Fonte A.4 – Operador de Seleção por Torneio

A.1.4.2 Crossover

```
class SequentialConstructiveCrossover: CrossoverOperator {
    companion object {
        const val CROSSOVER_PROBABILITY = 0.8
    }

    private val distanceCache = ConcurrentHashMap<String, Double>()

    override fun crossover(parent1: Individual, parent2: Individual): Array
    <Individual> {
        if (Math.random() >= CROSSOVER_PROBABILITY) {
            return arrayOf(Individual(parent1), Individual(parent2))
        }

        val problem = parent1.problem
        val length: Int = parent1.chromosome.size
        val dimension = length / 2
        val vehicleCapacity: Double = problem.vehicleCapacity
        val numCustomers = problem.customers.size

        val offspring = arrayOf(
            Individual(parent1),
            Individual(parent1)
        )
```

```
23
24          val child = offspring[0]
25          val routes: MutableList<List<Int>> = ArrayList()
26          var currentRoute: MutableList<Int> = ArrayList()
27          routes.add(currentRoute)
28          currentRoute.add(0) // Depósito
29
30          val used = BooleanArray(numCustomers)
31          var currentDemand = 0.0
32
33          while (hasUnvisitedCustomers(used)) {
34              val currentCustomer = currentRoute.last()
35
36              val customerFromParent1: Int = findNextLegitimate(parent1,
    currentCustomer, used, dimension)
37              val customerFromParent2: Int = findNextLegitimate(parent2,
    currentCustomer, used, dimension)
38
39              var selectedCustomer = -1
40              var bestCost = Double.MAX_VALUE
41
42              if (customerFromParent1 > 0 && customerFromParent1 <=
    numCustomers) {
43                  val customerIndex = customerFromParent1 - 1
44                  val demand: Double = problem.customers[customerIndex].
    demand
45                  if (currentDemand + demand <= vehicleCapacity) {
46                      val cost: Double = getCachedDistance(problem,
    currentCustomer, customerFromParent1)
47                      if (cost < bestCost) {
48                          bestCost = cost
49                          selectedCustomer = customerFromParent1
50                      }
51                  }
52              }
53
54              if (customerFromParent2 > 0 && customerFromParent2 <=
    numCustomers) {
55                  val customerIndex = customerFromParent2 - 1
56                  val demand: Double = problem.customers[customerIndex].
    demand
57                  if (currentDemand + demand <= vehicleCapacity) {
58                      val cost: Double = getCachedDistance(problem,
    currentCustomer, customerFromParent2)
59                      if (cost < bestCost) {
60                          bestCost = cost
61                          selectedCustomer = customerFromParent2
```

```
                    }
                }
            }

            // Procura primeiro cliente viável
            if (selectedCustomer == -1) {
                for (i in 0 until numCustomers) {
                    if (!used[i]) {
                        val demand: Double = problem.customers[i].demand
                        if (currentDemand + demand <= vehicleCapacity) {
                            selectedCustomer = i + 1  // Converte índice do array para ID do cliente
                            break
                        }
                    }
                }
            }

            // Se não encontrou cliente viável
            if (selectedCustomer == -1) {
                currentRoute.add(0)
                currentRoute = ArrayList()
                currentRoute.add(0)
                routes.add(currentRoute)
                currentDemand = 0.0
                continue
            }

            // Adiciona cliente selecionado
            currentRoute.add(selectedCustomer)
            used[selectedCustomer - 1] = true  // Converte ID do cliente para índice do array
            currentDemand += problem.customers[selectedCustomer - 1].demand
        }

        // Fecha última rota
        if (currentRoute.last() != 0) {
            currentRoute.add(0)
        }

        convertRoutesToChromosome(child, routes, dimension)
        offspring[1] = Individual(offspring[0])

        return offspring
    }

    private fun hasUnvisitedCustomers(used: BooleanArray): Boolean {
```

```
107         return used.any { !it }
108     }
109
110     private fun findNextLegitimate(parent: Individual, currentCustomer:
    Int, used: BooleanArray, dimension: Int): Int {
111         var foundCurrent = false
112
113         for (i in dimension until parent.chromosome.size) {
114             val  customer= parent.getGene(i)
115             if (customer == 0) continue
116
117             val customerIndex = customer - 1
118
119             if (!foundCurrent && customer == currentCustomer) {
120                 foundCurrent = true
121             } else if (foundCurrent && customerIndex < used.size && !used[
    customerIndex]) {
122                 return customer
123             }
124         }
125         return -1
126     }
127
128
129     private fun convertRoutesToChromosome(child: Individual, routes: List<
    List<Int>>, dimension: Int) {
130         for (i in 0 until dimension) {
131             child.setGene(i, 1) // Inicializa com caminhão 1
132         }
133
134         for (routeIndex in routes.indices) {
135             val route = routes[routeIndex]
136             for (customer in route) {
137                 if (customer != 0) { // Ignora depósito
138                     child.setGene(customer, routeIndex + 1)
139                 }
140             }
141         }
142
143         var pos = dimension
144         for (route in routes) {
145             for (i in 1 until route.size - 1) { // Ignora depósitos
146                 child.setGene(pos++, route[i])
147             }
148         }
149     }
150
```

```
151     private fun getCachedDistance(problem: VehicleRoutingProblem, from:
    Int, to: Int): Double {
152         val key = "${minOf(from, to)}-${maxOf(from, to)}"
153 
154         return distanceCache.getOrPut(key) {
155             val fromLocation = if (from == 0) problem.depot else problem.
    customers[from - 1]
156             val toLocation = if (to == 0) problem.depot else problem.
    customers[to - 1]
157             fromLocation.distanceTo(toLocation)
158         }
159     }
160 
161 }
```

Código-Fonte A.5 – Operador de Crossover Sequencial Construtivo

A.1.4.3 Mutação

```
1  class SwapMutation : MutationOperator {
2      override fun mutate(individual: Individual) {
3          val length: Int = individual.chromosome.size
4          val pos1: Int = RandomUtils.nextInt(0, length)
5          val pos2: Int = RandomUtils.nextInt(0, length)
6  
7          val temp = individual.getGene(pos1)
8          individual.setGene(pos1, individual.getGene(pos2))
9          individual.setGene(pos2, temp)
10     }
11 }
```

Código-Fonte A.6 – Operador de Mutação por Troca

A.2 Otimização por Colônia de Formigas (OCF)

A.2.1 Classe principal

```
1  class AntColonyOptimization private constructor(
2      private val problem: VehicleRoutingProblem,
3      private val numAnts: Int,
4      private val maxIterations: Int,
5      private val alpha: Double,
6      private val beta: Double,
7      private val rho: Double,
8      private val q0: Double
9  ) {
```

```
10      private lateinit var pheromoneMatrix: Array<Array<Double>>
11      private lateinit var distanceMatrix: Array<Array<Double>>
12      private val random = kotlin.random.Random
13
14      private val customers = problem.customers
15      private val depot = problem.depot
16      private val vehicleCapacity = problem.vehicleCapacity
17
18      private val tau0: Double by lazy { initializeTau0() }
19
20      init {
21          initializeMatrices()
22      }
23
24      private fun initializeTau0(): Double {
25          val nnDistance = nearestNeighborHeuristic()
26          return 1.0 / (customers.size * nnDistance)
27      }
28
29      private fun initializeMatrices() {
30          val size = customers.size + 1
31          pheromoneMatrix = Array(size) { Array(size) { 0.0 } }
32          distanceMatrix = Array(size) { Array(size) { 0.0 } }
33
34          for (i in 1 until size) {
35              val customer = customers[i - 1]
36              distanceMatrix[0][i] = depot.distanceTo(customer)
37              distanceMatrix[i][0] = depot.distanceTo(customer)
38          }
39
40          for (i in 1 until size) {
41              for (j in i + 1 until size) {
42                  val from = customers[i - 1]
43                  val to = customers[j - 1]
44                  val distance = from.distanceTo(to)
45                  distanceMatrix[i][j] = distance
46                  distanceMatrix[j][i] = distance
47              }
48          }
49
50          val initialPheromone = 1.0 / (customers.size *
    nearestNeighborHeuristic())
51          for (i in pheromoneMatrix.indices) {
52              for (j in pheromoneMatrix[i].indices) {
53                  if (i != j) {
54                      pheromoneMatrix[i][j] = initialPheromone
55                  }
```

```
56              }
57          }
58      }
59
60      fun run(onProgress: (iteration: Int, bestDistance: Double, avgDistance:
        Double) -> Unit): AlgorithmExecutionData {
61          var bestSolution: List<Route>? = null
62          var bestDistance = Double.MAX_VALUE
63          val iterationDataList = mutableListOf<IterationData>()
64          val startTime = System.currentTimeMillis()
65
66          var iteration = 0
67          while (iteration < maxIterations) {
68              val antSolutions = buildList {
69                  repeat(numAnts) {
70                      val solution = constructSolution()
71                      add(solution)
72
73                      val totalDistance = calculateTotalDistance(solution)
74                      if (totalDistance < bestDistance) {
75                          bestDistance = totalDistance
76                          bestSolution = solution.map { route ->
77                              Route(depot).apply {
78                                  var current = depot
79                                  route.locations.forEach { location ->
80                                      val distance = current.distanceTo(
    location)
81                                      addNode(location, distance)
82                                      current = location
83                                  }
84                              }
85                          }
86                      }
87                  }
88              }
89
90              val averageDistance = antSolutions.map { calculateTotalDistance
    (it) }.average()
91              onProgress.invoke(iteration, bestDistance, averageDistance)
92
93              iterationDataList.add(
94                  IterationData(
95                      iteration = iteration,
96                      totalDistance = bestDistance,
97                      averageDistance = averageDistance,
98                      routes = bestSolution ?: emptyList(),
```

```
                    executionTimeMs = System.currentTimeMillis() - startTime
                )
            )

            updatePheromones(antSolutions, bestSolution)
            iteration++
        }

        val finalBestSolution = bestSolution ?: emptyList()
        val bestRoute = finalBestSolution.minByOrNull { it.getTotalDistance() }
            ?: finalBestSolution.firstOrNull()
            ?: throw IllegalStateException("Nenhuma rota encontrada")

        return AlgorithmExecutionData(
            algorithmName = "ACO",
            iterations = iterationDataList,
            totalExecutionTime = System.currentTimeMillis() - startTime,
            routes = finalBestSolution,
            bestRoute = bestRoute
        )
    }

    private fun constructSolution(): List<Route> {
        val solution = mutableListOf<Route>()
        val unvisited = customers.toMutableSet()

        while (unvisited.isNotEmpty()) {
            val route = Route(depot)
            var current = depot

            while (unvisited.isNotEmpty()) {
                val feasibleCustomers = unvisited.filter {
                    route.getTotalDemand() + it.demand <= vehicleCapacity
                }
                if (feasibleCustomers.isEmpty()) break

                val next = selectNextLocation(current, feasibleCustomers.toSet(), route) ?: break

                val distance = current.distanceTo(next)
                route.addNode(next, distance)

                val currentIndex = if (current == depot) 0 else customers.indexOf(current) + 1
                val nextIndex = customers.indexOf(next) + 1
```

```
142                 localPheromoneUpdate(currentIndex, nextIndex)
143
144                 unvisited.remove(next)
145                 current = next
146             }
147
148             if (route.locations.isNotEmpty()) {
149                 solution.add(route)
150             }
151         }
152
153         return solution
154     }
155
156     private fun selectNextLocation(current: Location, feasibleCustomers:
    Set<Location>, route: Route): Location? {
157         if (feasibleCustomers.isEmpty()) return null
158         val currentIndex = if (current == depot) 0 else customers.indexOf(
    current) + 1
159         return if (random.nextDouble() < q0) {
160             getBestNextLocation(currentIndex, feasibleCustomers)
161         } else {
162             getRandomNextLocation(currentIndex, feasibleCustomers)
163         }
164     }
165
166     private fun getBestNextLocation(currentIndex: Int, feasibleCustomers:
    Set<Location>): Location? {
167         var bestValue = -1.0
168         var bestLocation: Location? = null
169         for (location in feasibleCustomers) {
170             val locationIndex = customers.indexOf(location) + 1
171             val value = calculateTransitionValue(currentIndex,
    locationIndex)
172             if (value > bestValue) {
173                 bestValue = value
174                 bestLocation = location
175             }
176         }
177         return bestLocation
178     }
179
180     private fun getRandomNextLocation(currentIndex: Int, feasibleCustomers:
     Set<Location>): Location? {
181         if (feasibleCustomers.isEmpty()) return null
182         val probabilities = DoubleArray(feasibleCustomers.size)
183         var total = 0.0
```

```
184         feasibleCustomers.forEachIndexed { index, location ->
185             val locationIndex = customers.indexOf(location) + 1
186             val probability = calculateTransitionValue(currentIndex,
    locationIndex)
187             probabilities[index] = probability
188             total += probability
189         }
190         if (total == 0.0) return feasibleCustomers.first()
191         val randomValue = random.nextDouble() * total
192         var sum = 0.0
193         feasibleCustomers.forEachIndexed { index, location ->
194             sum += probabilities[index]
195             if (sum >= randomValue) return location
196         }
197         return feasibleCustomers.last()
198     }
199 
200     private fun calculateTransitionValue(currentIndex: Int, nextIndex: Int)
    : Double {
201         val distance = distanceMatrix[currentIndex][nextIndex]
202         if (distance == 0.0) return 0.0
203         return Math.pow(pheromoneMatrix[currentIndex][nextIndex], alpha) *
204                 Math.pow(1.0 / distance, beta)
205     }
206 
207     private fun updatePheromones(allSolutions: List<List<Route>>,
    bestSolution: List<Route>?) {
208         val minPheromone = tau0 * 0.1
209         val maxPheromone = if (bestSolution != null) {
210             1.0 / (rho * calculateTotalDistance(bestSolution))
211         } else {
212             1.0 / rho
213         }
214 
215         pheromoneMatrix.indices.forEach { i ->
216             pheromoneMatrix[i].indices.forEach { j ->
217                 pheromoneMatrix[i][j] = maxOf(minPheromone, pheromoneMatrix
    [i][j] * (1.0 - rho))
218             }
219         }
220 
221         val sortedSolutions = allSolutions.sortedBy {
    calculateTotalDistance(it) }
222         sortedSolutions.take(ACOConfig.NUM_ELITE_ANTS).forEachIndexed {
    index, solution ->
223             val solutionQuality = 1.0 / calculateTotalDistance(solution)
```

```
224             val contribution = solutionQuality * (ACOConfig.NUM_ELITE_ANTS
    - index)
225             updateSolutionPheromone(solution, contribution)
226         }
227
228         bestSolution?.let {
229             val bestQuality = 1.0 / calculateTotalDistance(it)
230             updateSolutionPheromone(it, bestQuality * ACOConfig.NUM_ELITE_
    ANTS)
231         }
232
233         pheromoneMatrix.indices.forEach { i ->
234             pheromoneMatrix[i].indices.forEach { j ->
235                 pheromoneMatrix[i][j] = pheromoneMatrix[i][j].coerceIn(
    minPheromone, maxPheromone)
236             }
237         }
238     }
239
240     private fun updateSolutionPheromone(solution: List<Route>, quality:
    Double) {
241         solution.forEach { route ->
242             var fromIndex = 0
243             route.locations.forEach { location ->
244                 val toIndex = customers.indexOf(location) + 1
245                 pheromoneMatrix[fromIndex][toIndex] += quality
246                 pheromoneMatrix[toIndex][fromIndex] += quality
247                 fromIndex = toIndex
248             }
249             pheromoneMatrix[fromIndex][0] += quality
250             pheromoneMatrix[0][fromIndex] += quality
251         }
252     }
253
254     private fun calculateTotalDistance(solution: List<Route>): Double {
255         var total = 0.0
256         solution.forEach { route ->
257             if (route.locations.isNotEmpty()) {
258                 total += depot.distanceTo(route.locations.first())
259                 for (i in 0 until route.locations.size - 1) {
260                     total += route.locations[i].distanceTo(route.locations[
    i + 1])
261                 }
262                 total += route.locations.last().distanceTo(depot)
263             }
264         }
265         return total
```

```
266     }
267
268     private fun localPheromoneUpdate(from: Int, to: Int) {
269         val phi = ACOConfig.LOCAL_UPDATE_RATE
270         val newPheromone = (1 - phi) * pheromoneMatrix[from][to] + phi *
    tau0
271         pheromoneMatrix[from][to] = newPheromone
272         pheromoneMatrix[to][from] = newPheromone
273     }
274
275     private fun nearestNeighborHeuristic(): Double {
276         var totalDistance = 0.0
277         val unvisited = customers.toMutableSet()
278         var current = depot
279
280         while (unvisited.isNotEmpty()) {
281             var currentCapacity = 0.0
282             var routeStart = true
283             while (unvisited.isNotEmpty() && currentCapacity <
    vehicleCapacity) {
284                 val feasibleCustomers = unvisited.filter {
285                     currentCapacity + it.demand <= vehicleCapacity
286                 }
287                 if (feasibleCustomers.isEmpty()) break
288
289                 var nearest: Location? = null
290                 var minDistance = Double.MAX_VALUE
291                 for (customer in feasibleCustomers) {
292                     val distance = current.distanceTo(customer)
293                     if (distance < minDistance) {
294                         minDistance = distance
295                         nearest = customer
296                     }
297                 }
298                 if (nearest != null) {
299                     totalDistance += minDistance
300                     current = nearest
301                     currentCapacity += nearest.demand
302                     unvisited.remove(nearest)
303                     routeStart = false
304                 }
305             }
306             if (!routeStart) {
307                 totalDistance += current.distanceTo(depot)
308                 current = depot
309             }
310         }
```

```
311
312         return totalDistance
313     }
314
315     companion object {
316         inline fun build(block: Builder.() -> Unit): AntColonyOptimization
    = Builder().apply(block).build()
317     }
318
319     class Builder {
320         var problem: VehicleRoutingProblem? = null
321         var numAnts: Int = ACOConfig.NUM_ANTS
322         var maxIterations: Int = ACOConfig.MAX_ITERATIONS
323         var alpha: Double = ACOConfig.ALPHA
324         var beta: Double = ACOConfig.BETA
325         var rho: Double = ACOConfig.RHO
326         var q0: Double = ACOConfig.Q0
327
328         fun build(): AntColonyOptimization {
329             requireNotNull(problem) { "Problem must be specified" }
330             return AntColonyOptimization(
331                 problem = problem!!,
332                 numAnts = numAnts,
333                 maxIterations = maxIterations,
334                 alpha = alpha,
335                 beta = beta,
336                 rho = rho,
337                 q0 = q0,
338             )
339         }
340     }
341 }
```

Código-Fonte A.7 – Classe principal da OCF