UNIVERSIDADE FEDERAL DO PAMPA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

ÉRIC DIAS DA SILVA ROSSO

ALGORITMO DE MINIMIZAÇÃO DA COANCESTRALIDADE PARA SISTEMAS DE RECOMENDAÇÃO DE ACASALAMENTOS BASEADO EM OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

Bagé
2026

ÉRIC DIAS DA SILVA ROSSO

ALGORITMO DE MINIMIZAÇÃO DA COANCESTRALIDADE PARA SISTEMAS DE RECOMENDAÇÃO DE ACASALAMENTOS BASEADO EM OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientadora: Ana Paula Lüdtke Ferreira

Coorientador: Fernando Flores Cardoso

Bagé
2026

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

R838a Rosso, Éric Dias da Silva

Algoritmo de minimização da coancestralidade para sistemas de recomendação de acasalamentos baseado em otimização por colônia de formigas / Éric Dias da Silva Rosso. – 2026.

212 f.: il.

Orientadora: Ana Paula Lüdtke Ferreira
Coorientador: Fernando Flores Cardoso
Dissertação (Mestrado) – Universidade Federal do Pampa, Campus Bagé, Programa de Pós-Graduação em Computação Aplicada, 2026.

1. Melhoramento animal. 2. Meta-heurísticas bioinspiradas. 3. Sistemas de acasalamento. 4. Banco de dados de grafos. I. Ana Paula Lüdtke Ferreira. II. Título.

Éric Dias da Silva Rosso

Algoritmo de minimização da coancestralidade para sistemas de recomendação de acasalamentos baseado em otimização por colônia de formigas

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Computação Aplicada.

Dissertação defendida e aprovada em: 19 de dezembro de 2025.

Banca examinadora:

_____

Prof.ª Dr.ª Ana Paula Lüdtke Ferreira
Orientadora
(Unipampa)

_____

Prof.ª Dr.ª Renata Hax Sander Reiser
(UFPel)

_____

Prof. Dr. Marcos Jun-Iti Yokoo
(Embrapa)

_____

Prof. Dr. Érico Marcelo Hoff do Amaral
(Unipampa)

Assinado eletronicamente por **ANA PAULA LUDTKE FERREIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/12/2025, às 09:54, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **ERICO MARCELO HOFF DO AMARAL**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/12/2025, às 09:59, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Marcos Jun-Iti Yokoo**, **Usuário Externo**, em 22/12/2025, às 15:50, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Renata Hax Sander Reiser**, **Usuário Externo**, em 26/02/2026, às 09:23, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1919069** e o código CRC **5782544E**.

RESUMO

O aprimoramento da produção animal envolve a seleção de indivíduos com características desejáveis e estratégias de acasalamento que considerem o bem-estar, a produtividade e a rentabilidade do rebanho. A consanguinidade e a coancestralidade são fatores críticos nesse processo, pois cruzamentos entre indivíduos aparentados podem comprometer a saúde e o desempenho da prole. Este trabalho propõe o uso de uma abordagem baseada no algoritmo Max-Min Ant System (MMAS) para definir estratégias de acasalamento entre animais previamente selecionados, com o objetivo de minimizar a coancestralidade entre as proles, sem desprezar a qualidade genética medida por índices de seleção.

As relações de parentesco foram armazenadas e manipuladas em um banco de dados orientado a grafos (Neo4j), facilitando a construção dinâmica das soluções. Foram definidos formalmente diferentes problemas de otimização genética e realizadas simulações com o algoritmo Max-Min Ant System (MMAS) para comparar estratégias. Os parâmetros do algoritmo, especificamente os fatores de influência do feromônio (α) e da heurística (β), foram testados em diferentes configurações. Os testes revelaram que configurações com equilíbrio entre esses fatores ($\alpha = 1, \beta = 1$), bem como aquelas que priorizam o feromônio ($\alpha = 2, \beta = 1$), proporcionam melhores resultados. A melhor configuração (MMAS com $\alpha = 2, \beta = 1$, 5 formigas e 30 iterações) alcançou uma consanguinidade média de 0,0045, desempenho superior ao método comparativo baseado na contagem de relações de parentesco, que obteve 0,0069.

Além disso, verificou-se que o número de iterações e o número de formigas afetam de forma distinta a qualidade da solução e o tempo de execução. O tempo aumentou exponencialmente com o crescimento do número de relações de parentesco — função do número de pares testados —, indicando limitação prática para cenários de larga escala. A versão do algoritmo baseada na coancestralidade média demonstrou melhor desempenho geral, tanto em qualidade genética quanto em eficiência computacional.

Palavras-chave: Melhoramento animal; Meta-heurísticas bioinspiradas; Sistemas de acasalamento; Banco de dados de grafos.

ABSTRACT

The improvement of livestock production involves the selection of individuals with desirable traits and the implementation of mating strategies that promote animal welfare, productivity, and herd profitability. Coancestry and inbreeding are critical factors in this process, as mating closely related animals can compromise offspring health and performance. This study proposes a strategy based on the Max-Min Ant System (MMAS) algorithm to define mating plans among preselected animals, aiming to minimize coancestry within the resulting offspring while maintaining genetic quality as measured by selection indices. Kinship relationships were stored and manipulated using a graph-oriented database (Neo4j), which facilitated the dynamic construction and evaluation of mating solutions. Several breeding optimization problems were formally defined, and simulations using the MMAS algorithm were conducted to compare different configurations. The experiments showed that configurations balancing pheromone and heuristic influence ($\alpha = 1, \beta = 1$), as well as those favoring pheromone influence ($\alpha = 2, \beta = 1$), led to superior outcomes in terms of solution cost and mean inbreeding levels. Additionally, the number of iterations and ants used had distinct effects on solution quality and computational time. Execution time increased exponentially with the number of kinship relations—proportional to the number of mating pairs—highlighting practical limitations in large-scale scenarios. The MMAS variant using mean coancestry as the cost function outperformed the version based on counting positive coancestry relations, achieving better genetic outcomes and greater computational efficiency.

Keywords: Animal breedng. Bio-inspired metaheuristics. Mating systems. Graph database.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE DEFINIÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
APR	*Angus Performance Register*
BLUP	*Best Linear Unbiased Prediction*
EBV	*Estimated Breeding Values*
EMBRAPA	Empresa Brasileira de Pesquisa Agropecuária
IECC	Índice Econômico de Ciclo Completo
UNIPAMPA	Universidade Federal do Pampa
USDA	*United States Department of Agriculture*

SUMÁRIO

1 INTRODUÇÃO

O presente capítulo introduz o contexto do melhoramento genético animal, destacando a importância dos índices de seleção e das estratégias de acasalamento. Aborda-se a problemática do aumento da consanguinidade decorrente da seleção intensiva e a necessidade de métodos de controle baseados na coancestralidade. Por fim, são apresentados os conceitos fundamentais que baseiam este trabalho, incluindo a utilização de grafos e algoritmos de otimização.

1.1 Melhoramento animal

Os sistemas de melhoramento animal englobam processos destinados a qualificar o desempenho das futuras gerações dentro de um rebanho, visando aprimorar as características desejáveis e reduzir as indesejáveis. Esse processo envolve a seleção dos melhores indivíduos do grupo, com base em métricas que caracterizam os objetivos de aprimoramento, para se tornarem os progenitores da próxima geração. A seleção de animais para reprodução pode ser baseada em critérios empíricos – baseados na experiência/conhecimento dos criadores – ou em índices de seleção (OES et al., 2020).

Um índice de seleção é composto por características que podem ser quantificadas e que refletem o conjunto dos objetivos de seleção. Esses objetivos podem incluir desempenho econômico, saúde, rusticidade, entre outros. Cada objetivo de seleção é composto por indicadores mensuráveis que avaliam a qualidade do animal em relação ao objetivo. A soma ponderada desses indicadores resulta no índice de seleção, que atribui um único valor a cada animal (FERREIRA; YOKOO; MOTTA, 2021). A obtenção desse valor singular permite a ordenação e classificação automatizada dos animais, auxiliando ou determinando o processo de seleção. Essa abordagem se concentra em classificar os animais com base nas características desejadas, geralmente aquelas que resultam em maior lucro para o produtor (OES et al., 2020).

A partir da definição dos animais selecionados para reprodução, podem ser aplicadas diferentes estratégias de acasalamento com o objetivo de maximizar o valor genético e econômico da descendência. Os sistemas de melhoramento buscam fixar características fenotípicas que são lucrativas para o produtor e benéficas para a saúde do rebanho (CARDOSO, 2009). Como exemplos de características fenotípicas que podem ser avaliadas, tem-se medidas como tamanho da área do lombo, idade da primeira

cria, peso ao nascer e resistência a carrapatos, entre outras métricas possíveis. Esses indicadores não apenas fornecem informações sobre a qualidade do animal, mas também sobre seu valor no contexto de seleção. O processo de melhoramento animal, assim, visa não só a maximização de características desejáveis, mas também a adaptação às necessidades e objetivos específicos do criador.

A Embrapa Pecuária Sul desenvolveu o chamado **índice econômico de ciclo completo** (IECC) com o propósito de avaliar a eficiência do sistema de criação de bovinos da raça Brangus (YOKOO et al., 2021). Esse índice considera uma variedade de critérios, abrangendo aspectos tanto zootécnicos quanto econômicos, os quais são derivados dos objetivos de aprimoramento genético animal. O IECC é um parâmetro econômico que emprega valores expressos em dólares americanos (US$) para cada exemplar, possibilitando a estimativa de seu valor individual. O propósito desse índice é classificar os animais de acordo com um único valor que representa sua contribuição para o retorno financeiro do sistema de reprodução. Em outras palavras, o IECC atribui a cada animal um número que reflete sua eficiência econômica dentro do sistema de criação de bovinos da raça Brangus.

A amplitude de um índice de seleção depende dos critérios considerados em seu cálculo. Outros índices, com diferentes objetivos de seleção para diferentes raças de animais (não somente bovinos), podem ser construídos usando esses mesmos princípios. Isso significa que pode-se levar em conta características como taxa de ganho de peso, eficiência alimentar, características reprodutivas, custos de produção, preço de venda do animal, entre outros fatores relevantes para a rentabilidade do sistema de criação. O cálculo exato do índice envolve a ponderação desses critérios com base na importância relativa de cada um deles para o sucesso econômico do sistema de reprodução.

Além do processo de seleção, sistemas de melhoramento animal também operam com sistemas de acasalamento, que determinam como os animais selecionados serão pareados para a reprodução. O esquema de acasalamento, da mesma forma que o sistema de seleção, pode ter diversos objetivos. O atendimento aos objetivos de reprodução define o tipo de pareamento que deverá ser realizado entre os animais.

Os tipos de acasalamento mais elementares que podem ser implementados são o acasalamento entre semelhantes e o acasalamento compensatório (ELER, 2017). O primeiro visa obter exemplares de excelente valor genético/fenotípico por meio do acasalamento dos melhores animais entre si (e, consequentemente, dos piores com os piores), gerando uma prole heterogênea em relação aos objetivos de seleção; o segundo

consiste em acasalar os melhores animais com os menos qualificados, e vice-versa, buscando alcançar homogeneidade no rebanho. Nesse processo, ocorre o acasalamento entre os machos de maior qualidade com as fêmeas menos qualificadas, e as melhores fêmeas com os machos menos qualificados, de forma que um dos animais compense a característica que é mais deficiente no outro.

As estratégias de acasalamento não impactam somente no valor individual de cada animal. O acasalamento entre os melhores indivíduos permite a obtenção de animais com maior valor genético/econômico, mas pode levar a um aumento da consanguinidade nos animais, visto que os melhores animais de um rebanho tendem a ter relações de parentesco. O mesmo princípio se aplica aos piores animais, que também tendem a ter parentesco entre si.

Acasalamentos endogâmicos, ou acasalamentos entre indivíduos aparentados, podem ser realizados de forma intencional pelos produtores, com o objetivo de consolidar características fenotípicas específicas no rebanho. Ainda que necessários pra fixação de fenótipos importantes, esse tipo de acasalamento resulta no aumento de pares de genes homozigotos, ou seja, com alelos idênticos, levando à diminuição de pares de genes heterozigotos, que possuem alelos diferentes. Esse fenômeno ocorre devido à repetição de genes semelhantes na prole (ELER, 2017).

O aumento da consanguinidade dos indivíduos que irão compor a próxima geração do rebanho, causado por acasalmentos endogâmicos, pode resultar em um fenômeno conhecido como depressão por consanguinidade (*inbreeding depression*), que acarreta problemas de saúde, infertilidade da prole e outros problemas relacionados à diminuição da variabilidade genética do rebanho, o que pode causar prejuízos ao produtor (MI *et al.*, 1965; CHARLESWORTH AND WILLIS, 2009 *apud* FERREIRA, 2021, p. 2). Por essa razão, o controle dos níveis de consanguinidade do rebanho deve fazer parte das estratégias de acasalamento dos sistemas de melhoramento animal. Os acasalamentos exogâmicos, ou acasalamentos entre indivíduos sem parentesco, devem ser promovidos com o intuito de aumentar a diversidade genética e o valor genotípico dos animais, aproveitando a complementaridade genética entre os parceiros e resultando em descendentes que incorporam uma variedade mais ampla de alelos (ELER, 2017).

As informações sobre a genealogia (ascendência e descendência) dos animais são registradas em seu *pedigree*. O *pedigree* fornece os dados que permitem a montagem de estratégias de acasalamento entre os indivíduos de um rebanho. Nesse contexto, dois conceitos relacionados são relevantes: a coancestralidade e a consanguinidade.

A coancestralidade, também conhecida como coeficiente de parentesco, é uma relação binária que reflete a quantidade de antepassados que dois indivíduos têm em comum. A consanguinidade é um atributo associado a um único indivíduo, que resulta da relação de parentesco entre seus pais (VLECK, 1993).

A definição de coancestralidade pode ser dada de diferentes maneiras, mas em sua essência ela remete à conexão entre diferentes linhagens ancestrais, reconhecendo a intersecção de heranças genéticas em um grupo específico (COCKERHAM, 1967). Isso ocorre inclusive quando se utilizam marcadores genéticos em situações em que as informações do *pedigree* estão ausentes (TORO et al., 2002). A ideia geral do cálculo da relação de coancestralidade é que, quanto mais indivíduos em comum dois animais tiverem em suas linhas ancestrais, maior será o valor de coancestralidade entre eles. De forma similar, quanto mais estreita for a relação de parentesco entre os pais, maior será o nível de consanguinidade do descendente.

A coancestralidade pode ser computada a partir das informações contidas no *pedigree*, que estabelece as conexões de parentesco entre os animais do rebanho. A informação sobre valores de coancestralidade entre os animais permite evitar cruzamentos entre indivíduos aparentados. No entanto, é notável que, em um rebanho fechado, do qual não há introdução de novos animais, em algum momento no futuro todos os animais compartilharão pelo menos um ancestral comum após várias gerações (ROSSO, 2023). Dessa forma, os esquemas de acasalamento devem levar em consideração não apenas os aspectos relacionados ao desempenho econômico, mas também as informações contidas no *pedigree*.

1.2 Objetivos e resultados esperados

O objetivo deste trabalho é desenvolver um algoritmo que implemente estratégias de acasalamento para controlar os coeficientes de parentesco entre os animais de um rebanho, sem comprometer o ganho na produção animal estabelecido pelo índice de seleção empírico da raça (Promebo/Angus).

São objetivos específicos deste trabalho:

1. Realizar uma revisão bibliográfica para identificar as principais técnicas utilizadas para calcular coeficiente de parentesco entre dois indivíduos.
2. Realizar uma revisão bibliográfica para identificar as principais técnicas utilizadas

para minimização da coancestralidade em rebanhos de produção animal.

3. Desenvolver um banco de dados orientado a grafos que armazene as informações de cada animal, incluindo seu coeficiente de parentesco com cada outro animal do rebanho, permitindo consultas eficientes sem necessidade de recálculo de valores.
4. Descrever formalmente o problema da minimização da coancestralidade em rebanhos e estudar técnicas de solução apropriadas para o problema.
5. Desenvolver um algoritmo para determinar os melhores pares de animais para a produção da próxima geração do rebanho, com o objetivo de minimizar a coancestralidade.
6. Avaliar o desempenho dos algoritmos propostos através da simulação de diferentes cenários de produção animal, considerando diferentes níveis de endogamia.

São resultados esperados deste trabalho:

- Apoiar o aprimoramento de programas de acasalamento em sistemas de produção animal, possibilitando a obtenção de resultados melhores e mais sustentáveis.
- Contribuir para o uso de abordagens formais na área de melhoramento animal.
- Contribuir para o fortalecimento do Programa de Pós-graduação em Computação Aplicada, pelo incremento da parceria entre os pesquisadores da UNIPAMPA e da Embrapa Pecuária Sul.

1.3 Organização do texto

O texto desta monografia está organizado conforme descrito nos parágrafos a seguir.

O Capítulo 2 trata da metodologia de pesquisa utilizada para encontrar os resultados mais relevantes relacionados ao problema discutido, o estado da arte no tema e os trabalhos correlatos. Na Seção 2.1 são definidas as fases da pesquisa e como ela é caracterizada. A Seção 2.2.1 apresenta o protocolo de revisão utilizado para investigar o estado da arte no cálculo do coeficiente de parentesco e os resultados obtidos a partir desse processo. Enquanto na Seção 2.2.2 é apresentado o protocolo de revisão utilizado para investigar o estado da arte nas técnicas para redução da consanguinidade. A Seção 2.2.3 apresenta o protocolo de revisão utilizado para investigar o estado da arte da utilização de heurísticas e meta-heurísticas para otimização de problemas combinatórios. Na Seção 2.3

está listado o ferramental tecnológico utilizado no desenvolvimento do banco de dados proposto por este trabalho. Na Seção 2.4 é discutida a fonte dos dados que estão sendo utilizados no desenvolvimento deste trabalho.

O Capítulo 3 traz os resultados apurados pelo método de pesquisa, apresentando o estado da arte sobre abordagens técnicas e demais temas das questões de pesquisa da revisão bibliográfica. Especificamente, na Seção 3.1 são descritas as principais abordagens e métodos utilizados para calcular a consanguinidade e coancestralidade. Enquanto na Seção 3.2 são descritas as principais abordagens e métodos utilizados para reduzir a consanguinidade. Na Seção 3.2.1 discute-se a formalidade do problema tratado por parte dos trabalhos analisados. A Seção 3.3 são descritas as principais abordagens e métodos que utilizam meta-heurísticas para lidar com problemas combinatórios.

O Capítulo 4 apresenta os fundamentos teóricos e conceituais necessários para o entendimento e desenvolvimento do modelo proposto. Na Seção 4.1 são introduzidas as principais definições que servem de base para a formulação formal do problema de acasalamento com controle de coancestralidade. A Seção 4.2 explora conceitos gerais sobre problemas de otimização, destacando suas características e desafios, em especial no contexto de problemas combinatórios. Na Seção 4.3 é discutido o algoritmo Otimização por Colônia de Formigas (ACO), abordando seus fundamentos, funcionamento, variantes e aplicações. Já a Seção 4.4 descreve os conceitos fundamentais relacionados a bancos de dados orientados a grafos, ressaltando sua utilidade para representar relações de parentesco entre indivíduos de forma eficiente e escalável.

O Capítulo 5 trata da formulação do modelo de otimização proposto para o problema de acasalamento seletivo. Na Seção 5.1, é descrita a modelagem e a implementação do banco de dados orientado a grafos desenvolvido especificamente para armazenar as relações de parentesco entre os indivíduos do rebanho. A Seção 5.2 detalha o cálculo dos coeficientes de coancestralidade e geração, enquanto a Seção 5.3 é apresentada a formulação matemática e computacional da solução, destacando os elementos do problema, variáveis, objetivos e restrições envolvidas. A Seção 5.3.1 apresenta o procedimento de execução do algoritmo MMAS desenvolvido, incluindo a lógica da construção de soluções, atualização de feromônio e estratégia de exploração.

O Capítulo 6 apresenta os principais resultados obtidos por meio da aplicação prática do algoritmo MMAS sobre os dados gerados a partir da base de dados do rebanho. Na Seção 6.1, descreve-se o comportamento do algoritmo durante a execução, destacando a evolução das soluções e os padrões observados. A Seção 6.2 demonstra como o modelo

proposto pode ser utilizado em aplicação prática para sugerir acasalamentos viáveis dentro dos critérios definidos. A seção Seção 6.2.1 detalha os resultados alcançados utilizando a função de custo baseada na coancestralidade média entre os pares selecionados, explorando a configuração de parâmetros do algoritmo para otimizar essa métrica. Em seguida, a Seção 6.2.2 apresenta os resultados da função de custo baseada na quantidade de relações de parentesco não nulas (arestas com peso), comparando os impactos sobre o tempo de execução e a qualidade da solução. Enquanto a Seção 6.2.3 apresenta os resultados da modificação do algoritmo com o objetivo de maximizar o índice de seleção simultaneamente à minimização da coancestralidade. Já a Seção 6.3 reúne os dados referentes ao desempenho computacional, analisando o tempo necessário para a execução de diferentes configurações e o custo computacional envolvido nas abordagens.

O Capítulo 7 apresenta a conclusão deste texto. A Seção 7.1.1 apresenta perspectivas para a continuidade do estudo, abordando lacunas identificadas e propondo novos aprimoramentos.

2 MATERIAL E MÉTODOS

2.1 Caracterização e fases da pesquisa

O trabalho pode ser caracterizado como de pesquisa bibliográfica, exploratória e aplicada (PRODANOV; FREITAS, 2012). As fases de execução do trabalho são as seguintes:

1. Revisão da literatura sobre estratégias e algoritmos utilizados para calcular o coeficiente de parentesco em rebanhos, com o objetivo de compreender melhor as técnicas existentes e as possibilidades de implementação em bancos de dados.
2. Revisão da literatura sobre estratégias de minimização da coancestralidade em rebanhos, com o propósito de aprofundar o entendimento do problema e conhecer as soluções já propostas ou desenvolvidas.
3. Construção de um banco de dados orientado a grafos capaz de armazenar os dados dos animais, incluindo o coeficiente de parentesco entre cada par de indivíduos do rebanho.
4. Formalização do problema da minimização da coancestralidade do rebanho, com apoio das definições já feitas sobre o problema da maximização do índice da prole.
5. Investigação empírica da complexidade dos problemas formalizados e de problemas computacionais similares, bem como das técnicas pertinentes à sua solução.
6. Desenvolvimento do algoritmo de seleção de acasalamentos com vistas à redução da coancestralidade.
7. Escrita do texto da dissertação, escrita de artigos e publicação dos resultados.

As seções seguintes apresentam a descrição detalhada de cada fase listada acima. Vale ressaltar que a pesquisa bibliográfica (fases 1 e 2) foi estruturada de forma a cobrir dois domínios distintos e complementares: o domínio do problema (zootecnia/genética), focado nos métodos de cálculo e minimização de parentesco, e o domínio da solução (computação/otimização), focado nas técnicas heurísticas. Essa distinção justifica a utilização de diferentes protocolos e bases de dados para cada revisão, conforme detalhado nas seções subsequentes.

2.2 Protocolos para revisão da literatura

A revisão da literatura foi feita por meio do método de revisão de escopo, também conhecido como mapeamento sistemático da literatura, que permite questões mais amplas e não é focado em levantamentos de evidências de intervenções com forte componente empírico, como é o caso das revisões sistemáticas (KITCHENHAM, 2004; ARKSEY; O'MALLEY, 2005). O objetivo da revisão de escopo é determinar a extensão e a cobertura da literatura em relação a um tópico, identificando características-chave relacionadas a um determinado conceito e evidenciando lacunas que podem ser preenchidas por atividades de pesquisa (MUNN et al., 2018). De forma similar às revisões sistemáticas da literatura, as revisões de escopo também são produzidas a partir de um protocolo que garante a reprodutibilidade do procedimento.

A definição do protocolo de revisão e os procedimentos de condução e análise dos resultados foram feitos usando a ferramenta on-line Parsifal[1].

2.2.1 Cálculo do coeficiente de parentesco

O protocolo de revisão delineado consta das seguintes fases: (*i*) definição das questões de pesquisa, (*ii*) definição das *strings* de busca, (*iii*) explicitação dos critérios de inclusão e exclusão e (*iv*) seleção das fontes de pesquisa. Os resultados da execução do protocolo são apresentados ao final desta seção e os achados são discutidos e sintetizados na Seção 3.1.

A partir dos objetivos do trabalho, foram definidas as seguintes questões de pesquisa para a revisão da literatura:

Q1 Quais são os principais métodos e algoritmos utilizados para calcular a consanguinidade/coeficiente de parentesco em rebanhos?

Q2 Como esses métodos e algoritmos podem ser implementados em um banco de dados[2] para facilitar o cálculo e a análise da consanguinidade/coeficiente de parentesco em rebanhos?

A Tabela 1 contém os principais termos derivados das questões de pesquisa e

[1] <https://parsif.al/>

[2] Neste trabalho, considera-se um banco de dados como sendo um conjunto organizado de informações, estruturado de forma a possibilitar a recuperação, inserção, atualização e gerenciamento dos dados armazenados.

Tabela 1 – Termos de pesquisa para a revisão da literatura sobre cálculo do coeficiente de parentesco

Termo	Termos relacionados (inglês)	Termos relacionados
Inbreeding coefficient	Coancestry, Pedigree	Coeficiente de endogamia, Coancestralidade
Algorithm	Recursive	Algoritmo, Recursivo
Kinship coefficient	Relatedness coefficient	Coeficiente de parentesco
Population of animals	Animal population	População de animais
Relationship matrix	Coancestry matrix, Genetic relationship matrix	Matriz de coancestralidade, Matriz de parentesco genético

Fonte: Autor (2025)

seus respectivos sinônimos. Os sinônimos são necessários para que as buscas – restritas ao título e ao resumo dos trabalhos – consigam recuperar o maior número possível de trabalhos relacionados. Foram criadas *strings* de busca com base nos termos listados na tabela, e aquelas que apresentaram maior efetividade foram as seguintes:

***String* 1 -** Inbreeding coefficient AND Pedigree AND Algorithm AND NOT Genome AND NOT Clonal AND NOT Effects.

***String* 2 -** Inbreeding coefficient AND Pedigree AND Algorithm.

***String* 3 -** (Kinship coefficient OR Pedigree database) AND Algorithm.

***String* 4 -** Population of animals AND Relationship matrix AND Method AND Recursive AND NOT Genomic AND NOT Genotype.

***String* 5 -** Recursive algorithm AND Kinship coefficient OR Relationship matrix AND Method OR Procedure OR Algorithm AND NOT Quantitative AND NOT Diagnosing AND NOT Gravimetrically.

Essas *strings* foram desenvolvidas com o objetivo de obter o maior número possível de resultados capazes de responder pelo menos uma das questões de pesquisa. Observa-se que as *strings* 1, 4 e 5 incluem o termo *"NOT"* para delimitar resultados indesejados que foram observados em pesquisas anteriores. Não foram realizadas buscas em Português, uma vez que mesmo os trabalhos publicados em Português possuem metadados em Inglês nas bases que foram usadas. Os filtros utilizados nas buscas estão relacionados à restrição da pesquisa a artigos publicados em revistas científicas e à busca restrita a termos presentes apenas nos títulos, resumos ou palavras-chave especificadas

pelos autores, como pode ser visto na Tabela 2.

O critério estabelecido para inclusão dos trabalhos está relacionado à sua capacidade de responder a pelo menos uma das questões de pesquisa definidas. Em relação aos critérios de exclusão, qualquer trabalho que não atenda ao critério de inclusão deve ser descartado. Além disso, os trabalhos que não estão em Língua Portuguesa ou Inglesa também são excluídos. No caso de trabalhos de um mesmo autor abordando o mesmo assunto, foi priorizada a inclusão do trabalho mais recente. Da mesma forma, para trabalhos clássicos que serviram como base para a maioria dos estudos atuais e são frequentemente referenciados, é preferido incluir as publicações mais recentes. A análise dos resultados da busca foi iniciada pela revisão dos títulos dos trabalhos, avaliando sua relevância em relação às questões de pesquisa. Em seguida, foi feita a leitura dos resumos e, se necessário, dos trabalhos completos, para determinar se atendiam aos critérios de inclusão estabelecidos.

A seleção das fontes de pesquisa foi realizada visando garantir a abrangência e relevância dos dados obtidos, considerando a natureza específica de cada questão de pesquisa. Para o cálculo do coeficiente de parentesco, de cunho zootécnico e biológico, foram priorizados repositórios com forte ênfase nessas áreas, como ScienceDirect e Wiley, não sendo utilizada a base IEEE Xplore, reservada para a revisão de heurísticas computacionais (Seção 2.2.3). Resumidamente, a seleção das fontes de pesquisa foi orientada pela busca de qualidade e especificidade temática, a fim de fornecer uma base sólida para a análise e discussão dos resultados obtidos.

Dentre os resultados obtidos, encontram-se estudos que propõem abordagens distintas para o cálculo do coeficiente de coancestralidade ou de valores de consanguinidade, considerando fatores como a extensão do *pedigree*, a presença de pais desconhecidos, a eficiência computacional e a modelagem de estruturas demográficas complexas. Na Seção 3.1 são apresentados os trabalhos selecionados que foram considerados relevantes de acordo com os critérios de inclusão, totalizando 9 (nove) trabalhos encontrados por meio das *strings* de busca. No entanto, também foram encontrados trabalhos por outras fontes que estão relacionados a outros temas e não abordam diretamente o cálculo do coeficiente de parentesco ou consanguinidade. Esses trabalhos trazem conceitos sobre coancestralidade, consanguinidade, índice econômico e outros assuntos discutidos neste estudo.

A Tabela 2 apresenta o resultados da condução do protocolo acima descrito.

Tabela 2 – Resultados da aplicação do protocolo

Chave Utilizada	**Fonte**	**Filtro**	**Resultados**	**Trabalhos Selecionados**
String 1	Oxford Academic	–	7	4
String 2	ScienceDirect	No título, resumo ou palavras-chave especificadas pelo autor	11	1
String 3	ScienceDirect	No título, resumo ou palavras-chave especificadas pelo autor	16	1
String 4	ScienceDirect	No título, resumo ou palavras-chave especificadas pelo autor	2	1
String 5	Wiley Online Library	Revistas e palavra-chave 'Kinship'	13	2
Total	–	–	49	9

Fonte: Autor (2025)

2.2.2 Minimização da coancestralidade/consanguinidade

A revisão da literatura sobre estratégias para minimização da coancestralidade/consanguinidade em sistemas de acasalamento foi feita com o mesmo método apresentado na Seção 2.2.1. Apesar do interesse maior deste trabalho estar relacionado à minimização da coancestralidade em rebanhos, diversos trabalhos nesse sentido usam métrica de consanguinidade pra suas análises. Dessa forma, a busca envolve os dois conceitos.

A seguir é apresentado o protocolo de revisão delineado, que igualmente consta das fases: (*i*) definição das questões de pesquisa, (*ii*) definição das *strings* de busca, (*iii*) explicitação dos critérios de inclusão e exclusão e (*iv*) seleção das fontes de pesquisa. Os resultados da execução do protocolo são apresentados ao final desta seção.

As questões de pesquisa para essa parte da revisão da literatura são as seguintes:

Q1 Quais técnicas são utilizadas para minimizar a coancestralidade ou a consanguinidade

em sistemas de acasalamento?

Q2 Existem descrições formais do problema da minimização da coancestralidade em sistemas de acasalamento?

Tabela 3 – Termos de pesquisa para a revisão da literatura sobre minimização da coancestralidade/consanguinidade

Termo	Termos relacionados (inglês)	Termos relacionados
Endogamia	inbreeding coefficient, consanguinity, inbreeding depression, coancestry	endogamia, consanguinidade, coancestralidade
Reprodução	breeding system, mating system	reprodução, sistema de acasalamento
Minimizar	minimization, minimize, reduction, reduce, attenuate, decrease	minimizar, reduzir, atenuar, diminuir
Técnica	technique, method, strategy, scheme	técnica, método, estratégia, esquema

Fonte: Autor (2025)

A Tabela 3 apresenta os principais termos derivados das questões de pesquisa e seus respectivos sinônimos. Utilizando os termos que constam nessa tabela, foram desenvolvidas as seguintes *strings* de busca:

***String* 1 -** (inbreeding OR consanguinity OR coancestry) AND (technique OR method OR strategy OR scheme) AND (minimize OR reduce OR attenuate OR decrease) AND (breeding OR mating system).

***String* 2 -** coancestry AND (technique OR method) AND (attenuate OR reduce) AND (breeding OR mating system).

***String* 3 -** (inbreeding OR consanguinity OR coancestry) AND (technique OR strategy) AND (minimize OR reduce) AND (breeding OR mating system).

***String* 4 -** (inbreeding coefficient OR inbreeding depression OR coancestry) AND (strategy OR scheme) AND (minimization OR reduction) AND (mating OR breeding system).

Estas *strings* foram desenvolvidas com o objetivo de obter o maior número de resultados capazes de responder a pelo menos uma das questões de pesquisa. Como pode ser observado, as *strings* 3 e 4 são versões reduzidas da *string* 1; no entanto, o uso de versões reduzidas justifica-se pela necessidade de lidar com a plataforma do *ScienceDirect*

[3], que opera com um número limitado de conectores lógicos do tipo *AND* ou *OR* em seus mecanismos de busca. A *string* 2 tem o propósito de restringir o foco da busca, visando uma maior obtenção de resultados relacionados. Não foram realizadas buscas em Português, uma vez que a maioria dos trabalhos acadêmicos é publicada em Língua Inglesa.

As áreas de pesquisa relacionadas ao melhoramento animal e aos procedimentos para seleção de animais para reprodução e escolha de acasalamentos são de natureza multidisciplinar. Publicações relacionadas a esses temas podem ser identificadas em diversos periódicos, que abrangem campos como Medicina Veterinária, Zootecnia, Genética, Estatística e Computação. Embora uma busca mais ampla em uma quantidade maior de fontes de pesquisa pudesse potencialmente aumentar o número de resultados, é importante destacar que essas publicações podem conter as palavras-chave desejadas, mas não necessariamente contribuir para o foco deste trabalho. Esta observação é baseada na análise preliminar de parte dos trabalhos encontrados, os quais, apesar de abordarem superficialmente os temas relacionados, não apresentavam a profundidade ou a relevância necessárias para enriquecer a discussão proposta. Ressalta-se que os filtros de busca utilizados estavam relacionados principalmente à Agricultura, Ciência Animal e Genética, conforme detalhado na Tabela 4, que apresenta a relação entre *strings* e repositórios usados para as buscas, juntamente com o número de resultados obtidos e selecionados.

Quanto aos critérios de inclusão, a seleção baseou-se na capacidade dos trabalhos responderem a pelo menos uma das questões de pesquisa previamente estabelecidas, ou seja, apresentarem soluções potenciais para os problemas investigados. Trabalhos que não atenderam a esse critério de inclusão foram excluídos da análise. Além disso, foram excluídas publicações que se concentravam exclusivamente em técnicas sem aplicabilidade direta em animais, mesmo que fossem aplicadas a outras espécies de seres vivos. Também foram eliminados trabalhos que não estavam disponíveis em Língua Portuguesa ou Inglesa. Quando trabalhos de um mesmo autor abordando o mesmo tema são encontrados, dá-se preferência à inclusão do mais recente. O mesmo critério de preferência por publicações mais recentes foi aplicado a trabalhos clássicos que serviram como base para grande parte das pesquisas atuais e são frequentemente referenciados. A análise dos resultados da busca começou com uma avaliação dos títulos dos trabalhos, avaliando sua relevância em relação ao tema em questão. Em seguida, à leitura dos resumos e, quando necessário, dos trabalhos completos, a fim de determinar se eles

[3]https://www.sciencedirect.com/

Tabela 4 – Resultados da aplicação do protocolo

Chave Utilizada	**Fonte**	**Filtro**	**Resultados**	**Trabalhos Selecionados**
String 1	Wiley Online Library	Published in: Animal Science Journal	51	1
String 2	Wiley Online Library	Subject: Animal Agriculture	64	1
String 4	Oxford Academic	Journals	19	3
String 3	ScienceDirect	Title, abstract, keywords; Open access & Open archive	28	3
String 4	ScienceDirect	Title, abstract, keywords; Open access & Open archive	16	2
Total	–	–	178	10

Fonte: Autor (2025)

satisfaziam os critérios de inclusão estabelecidos.

Dentre os resultados obtidos nas pesquisas, a grande maioria não traz soluções para a redução da coancestralidade, mas dados que estimam a depressão endogâmica de um grupo especifico, estudos de técnicas que trabalham com genotipagem, predição genômica e seleção por dados genômicos, além de trabalhos que apresentam técnicas para redução da coancestralidade em espécies de plantas. É possível perceber uma maior incidência de trabalhos relacionados ao assunto entre os anos de 1990 e 2005, após este período o enfoque se torna maior em relação a métodos e análises com foco na genotipagem.

Na Seção 3.2 são apresentados os trabalhos selecionados, que foram considerados relevantes de acordo com os critérios de inclusão. Sendo um total de 10 (dez) trabalhos encontrados a partir das *strings* de busca.

2.2.3 Heurísticas e meta-heurísticas

Uma revisão da literatura sobre heurísticas e meta-heurísticas foi realizada utilizando uma abordagem similar à descrita na Seção 2.2.1. O foco desta revisão foi identificar técnicas que lidam com problemas de otimização combinatória, área relacionada com o problema de minimização de coancestralidade em sistemas de acasalamento. Além disso, considerou-se a capacidade das técnicas de lidar com múltiplos objetivos, uma vez que o problema envolve não apenas a redução da coancestralidade, mas também o aumento do valor econômico.

O protocolo de revisão seguiu as seguintes fases: (*i*) definição das questões de pesquisa, (*ii*) definição das *strings* de busca, (*iii*) explicitação dos critérios de inclusão e exclusão e (*iv*) seleção das fontes de pesquisa. Os resultados da execução do protocolo são apresentados ao final desta seção.

A questão de pesquisa para essa revisão da literatura é a seguinte:

Q1 Quais heurísticas e meta-heurísticas são aplicadas para resolver problemas de otimização combinatória com múltiplos objetivos?

Tabela 5 – Termos de pesquisa para a revisão da literatura sobre heurísticas e meta-heurísticas

Termo	Termos relacionados (inglês)	Termos relacionados
Heurística	heuristic, approximate method, local search	heurística, método aproximado, busca local
Meta-heurística	metaheuristic, algorithm, optimization technique	meta-heurística, algoritmo, técnica de otimização
Múltiplos objetivos	multi-objective, multi-criteria, pareto optimization	múltiplos objetivos, multi-critério, otimização de pareto
Otimização combinatória	combinatorial optimization, discrete optimization, scheduling	otimização combinatória, otimização discreta, escalonamento

Fonte: Autor (2025)

A Tabela 5 apresenta os principais termos derivados da questão de pesquisa, juntamente com outros conceitos relacionados. A otimização de Pareto está intrinsecamente ligada ao conceito de otimização, abordando múltiplos objetivos e

critérios (NGATCHOU; ZAREI; EL-SHARKAWI, 2005). Com base nos termos listados nesta tabela, foram desenvolvidas as seguintes *strings* de busca:

***String* 1 -** (metaheuristic OR optimization technique) AND (pareto OR multi-criteria) AND (combinatorial optimization OR scheduling).

***String* 2 -** (heuristic OR approximate method) AND (multi-objective OR multi-criteria) AND (combinatorial optimization OR discrete optimization).

***String* 3 -** (heuristic OR local search) AND (multi-objective OR pareto) AND (discrete optimization OR scheduling).

Essas *strings* foram elaboradas com o objetivo de cobrir a maior parte das técnicas relevantes para a pesquisa, respondendo às questões formuladas anteriormente. As buscas foram realizadas nas bases de dados do *IEEE Xplore* e *ScienceDirect*. Não foram incluídas buscas em português, dado que a maioria das publicações sobre esses tópicos é em língua inglesa.

Para os critérios de inclusão, foram considerados artigos que apresentassem heurísticas ou meta-heurísticas aplicadas a problemas de otimização combinatória com múltiplos objetivos. Além disso, foram incluídos trabalhos que abordassem problemas de otimização combinatória. Quanto aos critérios de exclusão, foram descartados estudos que não tratassem de problemas multiobjetivo ou que não envolvessem otimização combinatória. Também foram excluídos artigos publicados antes de 2020, com o objetivo de priorizar estudos mais recentes.

A seleção das fontes de pesquisa foi realizada com o intuito de garantir tanto a abrangência quanto a relevância dos dados obtidos. Foram priorizados repositórios reconhecidos nas áreas de otimização combinatória, programação linear binária e heurísticas/meta-heurísticas, conhecidos por sua consistência e diversidade de materiais. Entre os repositórios utilizados, destaca-se a base de dados do *IEEE Xplore*, que oferece uma vasta gama de publicações voltadas para técnicas de otimização e pesquisa operacional.

Os resultados da execução deste protocolo são apresentados a seguir na Tabela 6.

2.3 Ferramental tecnológico

O trabalho fez uso de um modelo de dados organizado como um grafo, para armazenamento das informações de *pedigree* do rebanho. Grafos são usados para

Tabela 6 – Resultados da aplicação do protocolo

Chave Utilizada	**Fonte**	**Filtro**	**Resultados**	**Trabalhos Selecionados**
String 1	IEEE Xplore	Open Access Only, Year Range 2020 - 2024	50	1
String 2	IEEE Xplore	Open Access Only, Year Range 2020 - 2024	42	2
String 3	ScienceDirect	Title, abstract, keywords; Open access	58	3
–	Outras fontes	–	–	3
Total	–	–	86	9

Fonte: Autor (2025)

representar relações entre elementos de um conjunto base. Matematicamente, um grafo é um par ordenado $G = (V, E \subseteq V \times V)$, em que V é um conjunto de vértices e E é uma relação binária sobre o conjunto V, denominada conjunto de arestas (DIESTEL, 2005). Grafos podem ser representados graficamente por nodos (vértices) e setas que os conectam (arestas), facilitando a visualização e o entendimento do modelo por não especialistas em Computação ou bases de dados.

O fato de grafos expressarem relações entre elementos serviu de motivação para o desenvolvimento de bancos de dados baseados em grafos, com os métodos usuais *Create*, *Read*, *Update*, e *Delete* (usualmente denotados pela sigla CRUD). Nesse tipo de banco de dados, o foco principal recai sobre os relacionamentos entre valores, representados por nodos do grafo, o que simplifica a busca não apenas por elementos indexados por chaves, como é o caso dos bancos de dados relacionais, mas também pelos valores de dados. Sistemas de gerenciamento de banco de dados relacionais requerem a criação de conexões entre entidades usando chaves estrangeiras ou outros mecanismos, que não favorecem a busca por relacionamentos em si, mas apenas por tuplas individuais no banco de dados. Ao reunir os conceitos fundamentais de nodos e relacionamentos em estruturas interconectadas, os bancos de dados de grafos permitem construir modelos que mapeiam de forma mais próxima o domínio do problema trabalhado. Os modelos resultantes desse método costumam ser mais simples e expressivos em comparação aos

modelos produzidos por bancos de dados relacionais tradicionais e sistemas NoSQL (Not Only SQL) (ROBINSON; WEBBER; EIFREM, 2015). Os sistemas de gerenciamento de bancos de dados de grafo além de seguirem os princípios do CRUD empregam também o conceito de "*index-free adjacency*". Isso significa que os nodos mantêm referências diretas aos nodos adjacentes, tornando as consultas independentes do tamanho total do grafo e proporcionais apenas ao tamanho do grafo pesquisado, o que contribui para o desempenho das consultas (RODRIGUEZ; NEUBAUER, 2012).

O banco de dados usado neste trabalho é o *Neo4j*[4], também conhecido como *Neo Technology*, que oferece suporte a várias linguagens de programação, incluindo Python, Java e R, para operações CRUD. O *Neo4j* é escalável, podendo suportar centenas de milhões de nodos e relacionamentos em uma rede; possui suporte para transações ACID (*atomicidade*, *consistência*, *isolamento* e *durabilidade*), com registros de transação em memória e gerenciamento de bloqueio, permitindo ainda a inclusão de propriedades tanto em nodos como em relacionamentos (KALIYAR, 2015).

O sistema de gerenciamento de banco de dados *Neo4j* foi utilizado na versão gratuita v. 5.11.0 para o desenvolvimento de um modelo de banco de dados para atender às necessidades de armazenamento de informações relacionadas a animais. Para isso, está sendo utilizada a linguagem de consulta *Cypher*[5], juntamente com a linguagem de programação *Python*[6], na versão 3.11.0.

Para integrar o banco de dados *Neo4j* ao ambiente *Python*, foi utilizada a biblioteca *neo4j*[7], a qual proporciona uma interface para interação com o banco de dados orientado a grafos *Neo4j* no ambiente *Python*. A classe *GraphDatabase* proveniente dessa biblioteca permitiu a execução de consultas e a manipulação de dados nesse contexto.

O ambiente computacional onde foi desenvolvido o banco de dados, além dos testes e validações, é um notebook com processador AMD Ryzen™ 7 3700U de 2.3 GHz; 8 GB de memória RAM; armazenamento SSD de 250 GB. Com o sistema operacional Windows 11 Home de 64 bits.

[4]<https://neo4j.com/>
[5]<https://neo4j.com/developer/cypher/>
[6]<https://www.python.org/>
[7]<https://neo4j.com/developer/python/>

2.4 Fontes de dados

A base de dados utilizada para execução do algoritmo proposto neste trabalho foi fornecida pela EMBRAPA Pecuária Sul. Inicialmente, foram utilizados dados referentes a bovinos da raça Brangus, porém optou-se posteriormente pela utilização da base da raça Angus por esta conter um volume significativamente maior de informações, o que favorece análises mais robustas e a validação dos resultados obtidos. A base contém informações sobre 1.215.892 bovinos da raça Angus, dos quais 443.826 são machos e 772.066 são fêmeas. No entanto, há 128.497 fêmeas e 10 machos listados apenas nas colunas ID_Mae e ID_Pai, respectivamente, sem informações adicionais registradas. A Figura 1 apresenta um recorte da planilha fornecida.

Figura 1 – Extrato da planilha dos dados

ID	Safra	Sex	ID_Pai	ID_Ma	Ind_Sel
268627	1995-01-28	M	268495	297397	7.32697416028067
268650	1994-09-03	M	268564	282969	0.708628276182183
268768	1997-02-19	M	268564	284203	7.51009467431537
268779	1996-09-09	M	268629	277816	-3.62724951392155
268784	1998-02-21	M	297714	297814	5.7847013397425
268788	1989-06-19	M	274000	269958	-11.3103064239521
268832	1998-02-08	M	268627	297732	12.2392446169834
268834	1998-08-21	M	296952	297805	12.556146663653
268839	1997-02-02	M	296618	297265	10.3188254037858
268860	1999-10-16	M	273313	278282	4.53736755765768
268867	2002-01-29	M	286049	285467	10.3848939503371
268897	2002-04-05	M	1663593	297805	5.93375123478162
268901	2001-06-21	M	268779	278318	2.06653019421546
268905	1998-01-02	M	295497	297493	10.0945218361288

Fonte: Autor (2025)

Os dados fornecidos estão organizados em planilhas eletrônicas no formato *comma-separated-values* (*.csv*). O formato *.csv* é comumente empregado para representar dados tabulares, onde os valores são separados por vírgulas e organizados em várias linhas. Cada linha contém informações relacionadas a um animal específico. Para importar os dados, foram realizadas consultas em *Cypher*. Após a importação, os dados são armazenados de forma que cada animal seja representado como um nodo e suas relações como arestas. No banco de dados Neo4j, os dados são estruturados e armazenados de acordo com o modelo de grafos.

Os dados consistem em identificações únicas de cada animal, incluindo seus pais e mães, o que forma o seu *pedigree*. Além disso, os dados trazem informações como a

safra de nascimento de cada animal e o valor do seu índice de seleção.

O valor do índice de seleção pode variar no tempo, visto que alguns indicadores usados podem ter seus valores alterados. Isso não influi no funcionamento dos algoritmos que trabalham com esses índices e, como os dados de entrada dos algoritmos serão coletados do banco de dados, basta que os valores sejam atualizados quando necessário, usando a identificação do animal que já foi usada na criação do nodo correspondente. O modelo de dados usado para registro dos animais será apresentado e discutido na Seção 5.1.

Para reduzir o tempo de preparação da base de dados e minimizar o espaço de armazenamento necessário, foi realizada uma filtragem dos animais presentes no pedigree. Inicialmente, foram selecionados os indivíduos nascidos em 2024, pois representam a população ativa mais recente e que em breve estará apta à reprodução, sendo o foco principal das decisões de acasalamento. Em seguida, foram identificados seus ancestrais até a terceira geração (pais, avós e bisavós). Essa profundidade genealógica foi definida como suficiente para capturar a maior parte da contribuição para a coancestralidade recente, equilibrando a precisão do cálculo com a viabilidade computacional do processamento em grafos. Essa abordagem permite reduzir o volume de dados sem comprometer a integridade das informações essenciais para o estudo. Como resultado dessa seleção, restaram 8.049 animais, dos quais 5.795 são fêmeas e 2.254 são machos.

3 TRABALHOS RELACIONADOS

3.1 Abordagens para o cálculo do coeficiente de parentesco

As questões de pesquisa da primeira revisão bibliográfica (Seção 2.2.1, pág. 24) têm como objetivo identificar os principais métodos e algoritmos utilizados no cálculo do coeficiente de parentesco em rebanhos. Buscou-se também investigar a viabilidade de implementação dessas técnicas em bancos de dados para facilitar a análise da consanguinidade, visto que a relação de parentesco entre dois animais mantém-se idêntica por toda a vida, necessitando ser calculada uma única vez. Os estudos encontrados que abordam esses temas estão detalhados abaixo e são sintetizados na Tabela 7.

Boyce (1983) em seu trabalho analisa e compara dois métodos diferentes de cálculo de endogamia e coeficientes de parentesco a partir de informações de *pedigree*. O objetivo principal é determinar os coeficientes de endogamia e parentesco em *pedigrees* extensos, envolvendo milhares de indivíduos e mais de 10 gerações. As duas principais abordagens discutidas são: um algoritmo iterativo proposto por Quaas (1976 *apud* BOYCE, 1983, p. 403) e Henderson (1976 *apud* BOYCE, 1983, p. 403) e uma modificação de um algoritmo de busca de caminho proposto por Stevens (1975 *apud* BOYCE, 1983, p. 401). O estudo destaca a importância da endogamia remota na contribuição para o parentesco atual e mostra que o uso de métodos que exploram *pedigrees* apenas superficialmente pode levar a uma subestimação significativa do grau de consanguinidade entre os indivíduos. Dentre os métodos computacionais apresentados, estão a detecção de ancestrais comuns por comparação de linhagens ancestrais e métodos de busca de caminho. O autor discute as vantagens e limitações de cada método, levando em consideração a necessidade de armazenamento e a complexidade computacional. O trabalho propõe uma modificação do algoritmo de busca de caminho de Stevens, que se mostra mais eficiente em *pedigrees* extensos quando comparado ao método de Quaas e Henderson. Essa abordagem permite explorar *pedigrees* com maior profundidade do que os métodos anteriores, revelando informações adicionais sobre os coeficientes de endogamia. Uma análise de *pedigrees* de 10 garanhões *Standardbred* é apresentada como exemplo, mostrando os desafios computacionais envolvidos e ilustrando a evolução dos coeficientes de endogamia com o aumento da profundidade do *pedigree*. Os desafios computacionais envolvidos na evolução dos coeficientes de endogamia com o aumento da profundidade do pedigree incluem requisitos de armazenamento, à medida que a

profundidade do pedigree aumenta, o número de ancestrais comuns e caminhos a serem considerados cresce exponencialmente, resultando em um aumento nos requisitos de armazenamento para registrar ancestrais comuns e suas contribuições para os coeficientes de endogamia. Além disso, a subestimação do endogamia é um risco ao limitar a exploração a profundidades rasas em pedigrees, pois a endogamia remota, que pode não ser imediatamente aparente em análises rasas, pode contribuir significativamente para a relação presente. Por fim, a ordem de classificação dos indivíduos em termos de níveis de endogamia pode não ser estabelecida até que pedigrees com pelo menos 11 gerações de profundidade sejam explorados, indicando a necessidade de uma análise minuciosa para evitar conclusões enganosas. O funcionamento do algoritmo de Stevens ocorre da seguinte maneira: o algoritmo começa com dois indivíduos, chamados X e Y, para os quais o coeficiente de parentesco está sendo calculado. Para o indivíduo X, são geradas as linhagens ascendentes no *pedigree*, enquanto para o indivíduo Y, são geradas as linhagens descendentes. Essas linhagens representam a ancestralidade de cada indivíduo. Os links ascendentes para X e os links descendentes para Y são classificados e catalogados. A enumeração sistemática dos caminhos começa com o indivíduo X e progride pelo *pedigree* até que um ancestral comum seja encontrado. O caminho então desce pelo *pedigree* até que o indivíduo Y seja localizado. Durante o percurso do caminho, são aplicadas algumas restrições para evitar repetições e evitar que o caminho volte sobre si mesmo. No entanto, Boyce (1983) destaca algumas limitações do algoritmo de Stevens em sua formulação original. Por exemplo, ele pode não calcular corretamente a endogamia ou o coeficiente de parentesco quando o indivíduo com o qual o caminho termina é um ancestral do indivíduo com o qual o caminho começa. Além disso, o trabalho detalha o algoritmo iterativo de Quaas e Henderson, que utiliza uma estratégia similar à programação dinâmica, realizando o cálculo incremental dos valores dos animais mais velhos para os mais novos.

O estudo de Khang (1989) apresenta uma sub-rotina em *FORTRAN IV* que calcula coeficientes de endogamia e parentesco com base em informações genealógicas de uma população. A sub-rotina, implementada por meio de um algoritmo recursivo, gera caminhos entre dois indivíduos, X e Y, considerando restrições, como a não repetição de links pai-filho. A fórmula utilizada para o cálculo dos coeficientes de parentesco leva em consideração todos os níveis de ancestralidade até o número de níveis de ancestralidade dos indivíduos. Essa fórmula permite quantificar o grau de parentesco entre X e Y, considerando todas as conexões possíveis por meio de seus ancestrais comuns. O artigo

também aborda considerações relacionadas ao tamanho dos dados genealógicos e ao método de armazenamento, destacando a importância da sub-rotina em comparação com as existentes. Além disso, apresenta um exemplo prático de como a sub-rotina pode ser aplicada e detalha sua implementação em *FORTRAN IV*, incluindo os parâmetros de entrada e saída, visando facilitar seu uso em contextos de pesquisa.

Gholami e Thomas (1994) apresentam em seu trabalho um algoritmo que calcula os coeficientes de parentesco entre pares de indivíduos em um conjunto, levando em consideração um *pedigree* conhecido. É dito que o algoritmo tem a vantagem de ter tempo de execução proporcional ao tamanho do conjunto, com a constante de proporcionalidade determinada pelo número de gerações abrangidas pelo *pedigree*. Para quantificar o parentesco de um conjunto de indivíduos, é utilizado o índice genético de familiaridade (GIF), que é o parentesco médio entre todos os pares de indivíduos no conjunto. O trabalho de Gholami e Thomas (1994) foi motivado por um estudo sobre a familiaridade de cânceres em uma base de dados populacional. O GIF foi usado para detectar casos em que uma doença tende a se agrupar em famílias, indicando uma possível causa genética ou ambiental comum. O cálculo dos GIF e dos coeficientes de parentesco *pairwise* constitui um problema computacional, sendo que o tempo necessário para esses cálculos depende tanto do tamanho do conjunto quanto do número de gerações no *pedigree*. Ele utiliza o método de contagem de caminhos, no qual são rastreados todos os caminhos para os ancestrais fundadores dos indivíduos e são realizadas comparações. Para calcular o GIF, é utilizado um algoritmo chamado TOTKIN. Ele inicia marcando as arestas e vértices, define o conjunto de casos, aloca recursos e executa o algoritmo COLINES a partir de cada vértice ativo, culminando no cálculo do GIF. Por outro lado, o algoritmo COLINES é usado para encontrar caminhos independentes de arestas (colineagens) que conectam um indivíduo a todos os outros, permitindo a determinação dos coeficientes de parentesco. Ele opera explorando as arestas ascendentes e descendentes de um vértice, atualizando o comprimento do caminho e marcando as arestas conforme necessário, garantindo a identificação precisa das relações de parentesco entre os indivíduos no pedigree. Segundo os autores, esses algoritmos tem complexidade $N \log(N)$, sendo N o numero de indivíduos que compõem um *pedigree*. Além disso, o artigo destaca a relevância desses cálculos para diferentes áreas, como a genética, a demografia e o manejo de populações pequenas. O algoritmo proposto por Gholami e Thomas (1994) tem implicações práticas e pode auxiliar na detecção de doenças hereditárias.

Backus e Gilpin (2002) apresentam um método para calcular coeficientes de

parentesco em *pedigrees* considerados de grande escala, tanto em número de indivíduos quanto em gerações. Os autores propõem uma abordagem chamada "matriz de parentesco comprimida", que utiliza programação orientada a objetos para implementar o algoritmo. Backus e Gilpin (2002) afirmam que o cálculo desses coeficientes de parentesco pode ser realizado de duas maneiras principais: algoritmos de análise de caminho e algoritmos recursivos. Os algoritmos de análise de caminho são mais adequados para *pedigrees* estendidos, que podem envolver milhares de indivíduos, devido ao grande número de caminhos a serem gerados e armazenados. Já os algoritmos recursivos são considerados mais simples de implementar para *pedigrees* de profundidade substancial. O algoritmo proposto pelos autores baseia-se na recursão e aproveita o fato de que o parentesco entre dois indivíduos pode ser expresso em termos do parentesco entre um deles, o indivíduo mais velho, e os pais do indivíduo mais novo. O algoritmo recursivo permite o cálculo dos coeficientes de parentesco, assumindo que os parentescos de animais mais velhos são armazenados. Uma questão prática relacionada ao uso de coeficientes de parentesco é o consumo de recursos computacionais. O algoritmo proposto pelos autores evita armazenar todos os coeficientes em uma matriz aditiva tradicional: o algoritmo comprime a matriz, armazenando apenas os coeficientes relativos aos animais vivos. Isso resulta em uma economia de memória, já que apenas os parentescos entre animais vivos são necessários para a maioria das análises. A implementação do algoritmo utiliza conceitos da programação orientada a objetos e a linguagem de programação Java. Os autores criam classes e objetos para representar os animais e seus coeficientes de parentesco. O uso de uma estrutura de dados dinâmica *Vector* da linguagem Java foi utilizada com objetivo de manipular as listas de animais vivos. O algoritmo também lida com adições e remoções de animais vivos, atualizando a "matriz de parentesco comprimida" conforme necessário.

O estudo realizado por Aguilar e Misztal (2008) apresenta uma abordagem para calcular os coeficientes de endogamia levando em consideração a presença de pais desconhecidos e a existência de endogamia não nula. Os autores modificaram um algoritmo recursivo já existente, introduzindo alterações em uma parte específica da fórmula recursiva. Essa modificação substitui a endogamia de um animal com pelo menos um pai desconhecido por uma média da endogamia de todos os animais nascidos no mesmo ano. O algoritmo proposto é iterativo e foi testado em uma população de 17 milhões de animais da raça *Holstein* americana. A cada rodada, o algoritmo recalculava os coeficientes de endogamia com base nas informações disponíveis nos *pedigrees* dos animais. Após seis rodadas, a convergência dos coeficientes de

endogamia foi alcançada. Além disso, o tempo de computação por rodada foi de 4 minutos, tornando-o duas vezes mais rápido do que o algoritmo *VanRaden* (1992 *apud* AGUILAR; MISZTAL, 2008, p. 1670), baseado no método tabular. Uma característica destacada do algoritmo recursivo é a necessidade de levar em conta a ordem dos animais. No entanto, os pesquisadores conseguiram fazer uma modificação para lidar com *pedigrees* desordenados, obtendo resultados de endogamia próximos da realidade. Comparando os resultados com o algoritmo de *VanRaden*, utilizado para calcular coeficientes de endogamia, observou-se que o algoritmo proposto forneceu estimativas de parentesco semelhantes após a convergência. Além disso, o tempo de computação foi significativamente menor, demonstrando a eficiência do algoritmo recursivo modificado. Segundo os autores, a complexidade aproximada do algoritmo é $n \times 2^{2p}$, onde n é o tamanho da população e p é o número médio de gerações do *pedigree* por animal.

Abney (2009) propõe um algoritmo que utiliza uma abordagem voltada a grafos, transformando o problema de múltiplas recursões no *pedigree* em uma única travessia por um grafo de parentesco. O algoritmo começa construindo um grafo de identidade (IG) com base nos genes que foram considerados. Em seguida, é definido o grafo de parentesco (KG), que é construído de forma recursiva, adicionando nodos filhos com base nas restrições dos coeficientes de parentesco generalizados. O algoritmo gráfico proposto pelo autor é dito vantajoso em relação aos métodos recursivos tradicionais, pois, enquanto nos métodos tradicionais é necessário realizar uma recursão para cada coeficiente desejado, o algoritmo gráfico calcula todos os coeficientes de uma vez durante a construção do KG. No entanto, o tamanho do KG pode se tornar um problema em *pedigrees* com um grande número de genes, exigindo estratégias eficientes de busca e armazenamento de dados. O autor implementou o algoritmo no software *IdCoefs*, escrito em *ISO C*, e testou nos sistemas operacionais *macOS* e *Linux*. O software calcula os coeficientes de identidade condensados para pares de indivíduos, considerando quatro genes autossômicos. O trabalho menciona a possibilidade de extensões para um maior número de genes ou genes ligados ao sexo, embora ainda não implementadas.

Elliott et al. (2009) abordam o cálculo do coeficiente de endogamia em bancos de dados de *pedigrees*. Os autores propõem o uso de um esquema de codificação chamado *NodeCodes* para agilizar a avaliação de consultas em estruturas de *pedigree*. O esquema de codificação *NodeCodes* é baseado em um grafo acíclico direcionado que representa o *pedigree*. O artigo discute a avaliação do coeficiente de endogamia

de um indivíduo específico usando *NodeCodes* e apresenta um novo esquema de codificação chamado *Family NodeCodes*, otimizado para gráficos de linhagem. Os autores implementaram e testaram essas abordagens em dados de *pedigree* sintéticos e reais, avaliando seu desempenho e escalabilidade. Os resultados experimentais mostraram que o uso de *NodeCodes* oferece uma alternativa para consultas envolvendo o coeficiente de endogamia, com melhorias em relação aos métodos tradicionais de avaliação iterativa. O trabalho ainda destaca a importância do cálculo eficiente da endogamia em *pedigrees* de grande porte, especialmente para aplicações em tempo real, como aconselhamento genético. Segundo os autores, o algoritmo *NodeCodes* para o cálculo do coeficiente de endogamia envolve: operações do indivíduo ($O(k)$); encontrar ancestrais comuns ($O(k \log k)$ e $O(s \cdot k)$); identificar caminhos sobrepostos e calcular comprimento ($O(c \cdot s \cdot k)$ e $O(s)$). A complexidade é estimada como $O(c \cdot s \cdot k + k \log k)$, onde c é o número de ancestrais comuns, s é a quantidade máxima de caracteres que compõem um único código de nodo atribuído a um indivíduo na árvore genealógica e k é o número de relações associadas ao indivíduo no qual o coeficiente de consanguinidade está sendo calculado. Essa complexidade é comparada a um método alternativo chamado *PedHunter*, que usa consultas SQL e tem uma complexidade de tempo $O(N^2)$, onde N é o número total de indivíduos na árvore genealógica. O método proposto fornece uma alternativa mais eficiente em termos de tempo de execução, sendo até 10,1 vezes mais rápido em comparação com as consultas iterativas do *PedHunter*.

Sitzenstock, Ytournel e Simianer (2013) apresentam um método para prever a taxa de endogamia em programas de melhoramento animal. A abordagem proposta neste artigo baseia-se no método de fluxo gênico (processo pelo qual genes ou alelos são transferidos de uma população para outra dentro de uma mesma espécie) e é aplicável a programas de melhoramento dinâmicos e complexos. O programa de melhoramento é estruturado em coortes, que são grupos homogêneos de idade e sexo com uma origem definida de genes. A partir de uma configuração inicial, são definidas regras de transição para calcular o parentesco dentro e entre as coortes, considerando processos de reprodução e envelhecimento. O uso recursivo dessa abordagem permite prever o desenvolvimento esperado do parentesco ao longo do tempo, tanto dentro quanto entre as coortes. Com base nos parentescos calculados, é possível derivar parâmetros relevantes, como a taxa de endogamia e o tamanho efetivo da população. A taxa de endogamia se destaca, pois está relacionada a efeitos negativos na performance e características genéticas dos animais. O artigo ilustra o método proposto com exemplos estáticos e

dinâmicos de programas de melhoramento em ovinos. Os exemplos mostraram como o método pode ser aplicado em situações de crescimento populacional exponencial e gargalo genético. No entanto, o estudo reconhece que a abordagem não considera o efeito da seleção no desenvolvimento da endogamia, mas discute ideias para superar essa limitação.

Coste et al. (2021) tem por objetivo em seu estudo desenvolver um método geral para inferir a estrutura de parentesco de uma população com base em sua demografia. Neste artigo, os autores propõem uma fórmula geral para calcular o número esperado de parentes de um indivíduo focal, levando em consideração a classe desse indivíduo e de seus parentes. Eles utilizam ferramentas conhecidas como matrizes genealógicas, para inferir o parentesco com base em modelos populacionais matriciais. O método proposto pode ser aplicado para obter métricas de parentesco individual e populacional, além de analisar a sensibilidade da estrutura de parentesco em relação aos traços implementados no modelo. O método proposto pelos autores baseia-se na decomposição da matriz de projeção da população em suas componentes de sobrevivência e reprodução. Eles explicam como calcular as matrizes de parentesco diretamente da matriz de projeção e ilustram o método usando o ciclo de vida de um esquilo terrestre como exemplo. Ao usar matrizes genealógicas e modelagem populacional matricial, o método proposto pelos autores oferece uma maneira de inferir o parentesco entre os indivíduos, levando em consideração sua classe demográfica. Essa abordagem pode fornecer informações sobre a estrutura familiar, incluindo laços parentais diretos e indiretos, como pais, filhos, irmãos, primos, entre outros.

A Tabela 7 apresenta uma síntese comparativa dos principais estudos que propõem algoritmos ou métodos para o cálculo do coeficiente de parentesco, sendo que cada linha da tabela corresponde a um trabalho analisado na Seção 3.1 e destaca as características mais relevantes para sua aplicação prática e teórica. O campo “Tipo de abordagem” descreve o modelo lógico adotado no cálculo dos coeficientes, como algoritmos recursivos, análise de caminhos no *pedigree*, estruturas baseadas em grafos ou modelagens matriciais, permitindo observar a evolução conceitual das soluções ao longo do tempo. A coluna “Tipo de dado” indica se foram utilizadas informações ancestrais por meio de *pedigrees* ou informações genômicas para estabelecer o parentesco entre os indivíduos. A coluna “Tecnologia / Linguagem” refere-se à linguagem de programação ou plataforma utilizada na implementação do método, aspecto utilizado para avaliar sua viabilidade prática, especialmente no contexto de bancos de dados modernos,

como SQL, Java ou C. A “Complexidade” apresenta, quando disponível, a estimativa do custo computacional em função do número de indivíduos (N), gerações (p) ou relações envolvidas, sendo que métodos com menor complexidade tendem a ser mais eficientes para aplicação em rebanhos extensos. O campo “Escalável”, por sua vez, indica se o método é adequado para bases de dados com crescentes números de animais, fator essencial em programas de melhoramento genético e estudos populacionais. A análise conjunta desses elementos permite concluir que houve uma evolução das abordagens propostas, partindo de algoritmos iterativos simples até métodos mais sofisticados. Nesse contexto, o trabalho de Coste et al. (2021) destaca-se pela generalização do método de inferência de parentesco baseado em demografia, sendo uma referência importante para a compreensão teórica das estruturas populacionais. Por outro lado, para a aplicação computacional em bancos de dados de larga escala, as abordagens baseadas em grafos, como a proposta por Abney (2009), e em codificação, como Elliott et al. (2009), mostram-se mais relevantes para os objetivos deste trabalho, fundamentando a escolha por um banco de dados orientado a grafos. Observa-se ainda que os métodos mais recentes apresentam maior escalabilidade e integração com sistemas computacionais.

Tabela 7 – Comparação entre abordagens para cálculo do coeficiente de parentesco

Autor (Ano)	**Tipo de abordagem**	**Tipo de dado**	**Tecnologia / Linguagem**	**Complexidade**	**Escalável**
Boyce (1983)	Caminhos e algoritmo iterativo	Ancestral	Não especificado	Não estimada	Sim
Khang (1989)	Recursivo com restrições	Ancestral	FORTRAN IV	Não estimada	Parcial
Gholami e Thomas (1994)	Caminhos com índice genético (GIF)	Ancestral	TOTKIN / COLINES	$O(N \log N)$	Sim
Backus e Gilpin (2002)	Recursivo com compressão de matriz	Ancestral	Java (POO)	Não estimada	Sim
Aguilar e Misztal (2008)	Iterativo com modificação recursiva	Ancestral	Não especificado	$n \times 2^{2p}$	Sim
Abney (2009)	Grafo de parentesco (KG)	Genômico	ISO C	Não estimada	Sim
Elliott et al. (2009)	Codificação em DAG (NodeCodes)	Ancestral	SQL / DAG	$O(c \cdot s \cdot k + k \log k)$	Sim
Sitzenstock, Ytournel e Simianer (2013)	Recursivo por coortes (fluxo gênico)	Genômico	Não especificado	Não estimada	Sim
Coste et al. (2021)	Matricial com modelos demográficos	Ancestral	Matrizes genealógicas	Não estimada	Sim

Fonte: Autor (2025)

3.2 Abordagens para redução da coancestralidade/consanguinidade

As questões de pesquisa da revisão bibliográfica (Seção 2.2.2) têm como objetivo identificar os principais métodos utilizados para estabelecer estratégias de acasalamento com o objetivo de reduzir a coancestralidade ou a consanguinidade em rebanhos. Os trabalhos selecionados no processo de revisão são discutidos a seguir.

Fernández et al. (2011) explora estratégias de manejo genético para populações de animais de fazenda de pequeno porte, particularmente em contextos de programas de conservação e criação. As estratégias discutidas visam a manutenção da diversidade genética e a minimização da consanguinidade. Uma das abordagens mencionadas é o uso da coancestralidade, que mede o grau de parentesco entre indivíduos na população. Quando dados de pedigree estão disponíveis, a coancestralidade pode ser calculada com base nesses dados. No entanto, em situações em que os dados de pedigree são escassos ou ausentes, o artigo sugere atribuir a coancestralidade média dos contemporâneos aos indivíduos sem pais conhecidos. Essa abordagem pressupõe que a coancestralidade de um indivíduo aleatório da mesma população no momento do nascimento pode ser usada como substituto. Além disso, o artigo destaca o papel da informação molecular, como marcadores genéticos, na estimação da coancestralidade e dos relacionamentos entre os animais. Os marcadores genéticos podem ser usados para calcular a coancestralidade, mas a precisão dessas estimativas pode depender da quantidade e do tipo de marcadores disponíveis. A seleção genômica é apresentada como uma estratégia que permite a seleção de características específicas com base em marcadores genéticos distribuídos por todo o genoma. Isso possibilita previsões do potencial genético dos animais, bem como a seleção de animais jovens, o que reduz a necessidade de testes tradicionais de progênie. A seleção genômica é apontada como uma maneira de aumentar o ganho genético sem aumentar a consanguinidade de forma significativa. O artigo também considera cenários em que não há dados de pedigree nem informações moleculares. Nesses casos, o gerenciamento genético precisa se basear em informações demográficas sobre a população, mas esse tipo de gerenciamento pode ser menos detalhado e intensivo. Outras estratégias discutidas incluem seleção retroativa (*Walk-Back Selection*), na qual indivíduos são selecionados com base no desempenho, e acasalamento rotativo (*Rotational Mating*), que envolve a rotação de animais reprodutores entre subpopulações para minimizar a consanguinidade. Em resumo, o artigo destaca a importância do manejo genético adequado para preservar a diversidade genética em pequenas populações de animais de fazenda. O uso de

informações de pedigree, marcadores genéticos e seleção genômica pode desempenhar um papel na conservação e no aprimoramento dessas populações, garantindo sua viabilidade a longo prazo. Também são mencionados programas de software que podem ser úteis para otimizar programas de conservação e seleção genética, como *KINSHIP*, *ML-RELATE*, *SPAGEDI* e *COLONY*.

Liu, Henryon e Sørensen (2017) investigaram estratégias de acasalamento com informações genômicas para reduzir as taxas de consanguinidade (ΔF) sem comprometer as taxas de ganho genético (ΔG). Eles realizaram simulações estocásticas comparando ΔF e ΔG obtidos por duas estratégias de acasalamento com informações de pedigree e genômicas em cinco esquemas de seleção. As duas estratégias de acasalamento eram o acasalamento mínimo de coancestralidade (MC) e a minimização da covariância entre as contribuições genéticas ancestrais (MCAC). Eles também incluíram o acasalamento aleatório (RAND) como ponto de referência. Os resultados mostraram que MC e MCAC com informações genômicas reduziram as taxas de consanguinidade em 6% a 22% em comparação com o uso de informações de pedigree, sem comprometer o ΔG em todos os esquemas de seleção. MC e MCAC com informações genômicas e de *pedigree* tiveram resultados semelhantes em ΔG, mas superaram RAND em até 14% de ΔG. A principal conclusão do estudo é que o uso de informações genômicas é mais eficaz do que as informações de pedigree na redução das taxas de consanguinidade. Isso implica que as informações genômicas podem ser aplicadas em esquemas de seleção além da simples previsão de valores genéticos, melhorando o controle da consanguinidade. O estudo também destacou a importância das informações genômicas na dispersão das contribuições genéticas dos ancestrais, tornando a seleção mais eficaz na manutenção de uma relação linear entre as contribuições genéticas de longo prazo. Além disso, as informações genômicas aumentaram o número de ancestrais que contribuíram para a descendência. Em resumo, o estudo mostra que o uso de informações genômicas em estratégias de acasalamento é eficaz na redução das taxas de consanguinidade, contribuindo na seleção dos programas de melhoramento genético.

O trabalho de Pryce, Hayes e Goddard (2012) aborda a criação de gado, com foco na redução da endogamia e no ganho genético. A pesquisa envolve a comparação de três estratégias diferentes para reduzir as taxas de endogamia da progênie em planos de acasalamento. Essas estratégias utilizam informações de coeficientes de consanguinidade de *pedigree*, relações genômicas e segmentos homozigóticos compartilhados. Os resultados destacam a eficácia da Matriz de Relações Genômicas (GRM) em comparação

com dados de pedigree para mitigar a endogamia, mantendo o ganho genético. A utilização de dados genômicos possibilita uma redução de 1% na consanguinidade da progênie, sem comprometer o objetivo global de aprimoramento genético. O estudo utiliza dados reais de pedigree, genótipo (43.115 marcadores de polimorfismo de nucleotídeo único) e valor genético estimado de bovinos da raça *Holstein*. Planos de acasalamento foram derivados para rebanhos de 300 vacas com 20 reprodutores disponíveis para acasalamento. Uma das descobertas é que o uso de informações de GRM resultou em menos marcadores de polimorfismo de nucleotídeo único (SNP) de baixa frequência, em comparação com a estratégia que utilizava informações de *pedigree*. Reduzir esses marcadores pode contribuir para uma estabilidade genética, pois as variações genéticas raras podem ser mais suscetíveis a flutuações e perda ao longo do tempo. Além disso, o estudo enfatiza a importância da escala de medição ao avaliar estratégias de controle de endogamia. A pesquisa sugere que essa metodologia pode ser eficaz na indústria de gado leiteiro, especialmente ao escolher touros e considerar as relações entre touros e vacas. Em última análise, o estudo destaca a importância do controle de endogamia à medida que a genética avança, beneficiando particularmente a pecuária leiteira.

A pesquisa de Henryon, Sørensen e Berg (2009) explora estratégias de acasalamento em programas de reprodução seletiva de animais com o propósito de reduzir a endogamia sem comprometer os ganhos genéticos. O estudo apresenta um critério de acasalamento, chamado *Mating by Minimising the Covariance of Ancestral Contributions* (MCAC) e o compara com o critério de acasalamento de coancestralidade miníma, que é dito ser a atual preferência na área. A seleção determina quais animais serão usados como pais e qual será a contribuição genética de cada um deles para a próxima geração, com o objetivo de maximizar os ganhos genéticos e manter a endogamia dentro de limites predefinidos. O acasalamento emparelha os pais selecionados para produzir a prole da próxima geração. Neste trabalho é argumentado que a redução da endogamia pode ser alcançada sem comprometer os ganhos genéticos, propondo o MCAC como uma alternativa que reduz a endogamia e mantém os ganhos genéticos. Para investigar essa proposta, a pesquisa se baseou em simulações estocásticas em dois tipos de programas de reprodução: um hierárquico com 20 machos e 120 fêmeas, e um fatorial com 60 machos e 60 fêmeas. O MCAC foi comparado com o método de acasalamento de coancestralidade miníma e com o acasalamento aleatório em ambos os cenários. Os resultados revelaram que o MCAC reduziu a endogamia em comparação

com o de coancestralidade miníma, mantendo ganhos genéticos comparáveis. Quando comparado ao acasalamento aleatório, o MCAC também reduziu significativamente a endogamia e produziu mais ganhos genéticos. Os pesquisadores concluíram que o MCAC é uma alternativa viável ao acasalamento de coancestralidade miníma, especialmente em programas de seleção truncada. Seleção truncada é uma estratégia de melhoramento genético que envolve a escolha seletiva apenas de indivíduos que ultrapassam ou ficam abaixo de um limiar predefinido para uma característica específica, visando otimizar o progresso genético em relação a essa característica-alvo Hallauer, Carena e Filho (2010). Além disso, os autores sugeriram que o MCAC pode ser utilizado de forma benéfica em conjunto com sistemas de seleção de contribuição ideal. Seleção de contribuição ideal refere-se a estratégias de melhoramento genético que buscam maximizar a contribuição genética de indivíduos selecionados para a próxima geração, considerando múltiplas características simultaneamente Charmantier, Garant e Kruuk (2014). Em resumo, o artigo demonstra que o MCAC é uma estratégia de acasalamento para reduzir a endogamia em programas de reprodução seletiva de animais, ao mesmo tempo em que mantém ou até mesmo melhora os ganhos genéticos.

No trabalho de Weigel (2001), discute-se estratégias para gerir a endogamia em programas de reprodução de animais, com foco nas populações de gado comercial. Abordando o impacto negativo da endogamia na saúde e no desempenho do gado. A principal mensagem destacada é que a intensidade da seleção é mais determinante da endogamia do que o tamanho da população, exigindo um equilíbrio entre a seleção de animais superiores e a manutenção da diversidade genética. A "Teoria da Contribuição Ótima" é introduzida como uma abordagem baseada na seleção de candidatos com base em valores genéticos estimados e variâncias de erro de previsão. O artigo também explora estratégias propostas para controlar a endogamia, tais como maximizar o lucro com restrições à endogamia, controle da endogamia na seleção de touros jovens, seleção com restrições, ajuste do Valor Genético Estimado (EBV) para a relação genética e maximização da contribuição genética ótima. Os autores reconhecem desafios relacionados aos dados de genealogia, como a falta de informações precisas, e destacam a importância de manter a diversidade genética nas populações de gado. No entanto, eles reconhecem que a aplicação dessas estratégias pode enfrentar desafios econômicos, pois as empresas de reprodução podem priorizar ganhos a curto prazo. Em resumo, o artigo oferece uma visão das estratégias e desafios na gestão da endogamia em programas de reprodução de gado comercial, enfatizando a necessidade de equilibrar o progresso

genético com a prevenção da endogamia, mas reconhecendo os obstáculos práticos e econômicos à sua implementação.

Fernández, Toro e Caballero (2003) tem como objetivo principal em seu trabalho comparar estratégias para controlar a coancestralidade, relevantes tanto para a conservação de espécies ameaçadas como para o melhoramento genético. Essas estratégias são avaliadas por meio de simulações computacionais que consideram uma gama de cenários, inclusive situações que envolvem falhas na implementação ideal das estratégias e os efeitos da coancestralidade preexistente nas populações. Além disso, o artigo investiga a aplicabilidade prática dessas técnicas, levando em consideração fatores como restrições fisiológicas, econômicas e de manejo, bem como os efeitos estocásticos que afetam a reprodução. Isso proporciona uma visão holística das estratégias, considerando não apenas seus aspectos teóricos, mas também os desafios reais que podem surgir durante a implementação. O estudo se aprofunda nas estratégias de redução da coancestralidade, incluindo Modelos Hierárquicos, Minimização de Coancestralidade Global (MGC) e Estratégias de Acasalamento, destacando suas características e aplicabilidade em diferentes contextos. Por exemplo, Modelos Hierárquicos envolvem estruturas hierárquicas de acasalamento para otimizar a prole e minimizar a coancestralidade global entre os pais. Por outro lado, o MGC é uma abordagem que busca minimizar a coancestralidade global, considerando as contribuições ponderadas dos pais, permitindo acasalamentos mais adaptáveis. Estratégias de Acasalamento, como *"maximum avoidance of inbreeding"*, *"compensatory mating"* e *"minimum coancestry mating"*, são exploradas, ressaltando como elas podem ser aplicadas em diferentes cenários. Os resultados apontaram para um desempenho superior do modelo MGC quanto a curto e médio prazo, sendo superado por modelos hierárquicos quando se trata de longo prazo. Fernández, Toro e Caballero (2003) considera 30-50 gerações valores distantes e raramente requisitados, o que torna o modelo MGC melhor em termos de aplicabilidade real.

O trabalho de Sánchez, Bijma e Woolliams (2003) discute a importância da gestão da diversidade genética em populações, com ênfase na necessidade de reduzir a coancestralidade em populações com números desiguais de machos e fêmeas reprodutoras. O artigo propõe um sistema de reprodução no qual um filho substitui seu pai e uma filha substitui sua mãe, visando reduzir a variância do tamanho da família e otimizar as decisões de seleção ao longo de várias gerações. O estudo estabelece um limite inferior para o coeficiente de endogamia (F) e apresenta uma solução para atingir esse

limite. Ele compara essa abordagem com sistemas anteriores e valida a teoria por meio de simulações. A técnica proposta demonstrou ser eficaz na minimização do coeficiente de endogamia em populações com números desiguais de machos e fêmeas reprodutores, e os resultados foram respaldados por simulações de Monte Carlo (ROBERT; CASELLA; CASELLA, 1999), reforçando a validade da abordagem proposta.

O trabalho de Sanchez, Toro e García (1999) aborda estratégias para mitigar a coancestralidade em esquemas de seleção artificial, visando aprimorar a eficiência da seleção genética. Três métodos são propostos: ajustar o número de indivíduos selecionados ou sua contribuição à próxima geração, utilizar critérios de seleção sub-ótima com menor ênfase em informações familiares e implementar sistemas de acasalamento não aleatórios, como coancestralidade mínima, acasalamento fatorial ou acasalamento compensatório. O estudo baseia-se no modelo experimental de *Drosophila melanogaster*, o que significa que a pesquisa utiliza a espécie de mosca-das-frutas como um organismo modelo central para suas investigações. É descrito um método de seleção de acasalamento que busca otimizar a resposta à seleção enquanto limita o aumento da coancestralidade. Os resultados revelaram que linhagens otimizadas apresentam maior seleção diferencial fenotípica e avaliações genéticas em comparação com as linhagens de referência, com uma redução na coancestralidade. A pesquisa também examina a manutenção da variabilidade genética e a precisão das avaliações genéticas, concluindo que a abordagem proposta reduz a coancestralidade e melhora a resposta à seleção, sem comprometer a variabilidade genética de longo prazo.

No trabalho de NOMURA et al. (2002) os autores exploraram estratégias de seleção e acasalamento em linhas de criação de frangos, comparando o método de acasalamento de coancestralidade mínima (MC) com o acasalamento aleatório (RM). As estratégias de seleção incluíram o uso de um índice de seleção (DIS) e uma combinação de índice de família com técnicas de programação linear (LPS). As características selecionadas foram o peso corporal às 6 semanas de idade e a idade da maturidade sexual das fêmeas. Os resultados demonstraram que as quatro combinações de métodos de seleção e acasalamento produziram ganhos genéticos semelhantes. No entanto, os esquemas com MC resultaram em menor variação nos ganhos genéticos e uma redução nos coeficientes médios de endogamia em comparação com os esquemas com RM. Como resultado, os autores recomendaram a estratégia LPS + MC para linhas de criação de frangos fechadas, enfatizando sua eficácia na redução da endogamia e na manutenção de ganhos genéticos.

A pesquisa conduzida por Nomura (1999) aborda a problemática da endogamia em programas de seleção, apresentando métodos para reduzi-la. Entre esses métodos, destaca-se o sistema de acasalamento compensatório, que se baseia em considerações teóricas sobre o tamanho efetivo da população sob seleção. O tamanho efetivo da população de gado refere-se a uma medida que leva em consideração a variabilidade genética de uma população, levando em conta não apenas o número total de indivíduos, mas também a distribuição de parentesco e a contribuição relativa de cada indivíduo para a variabilidade genética total Templeton (2021). O autor propõe uma modificação desse sistema, incorporando a troca de parceiros com base nos valores genéticos estimados (EBVs). Através de simulações estocásticas, é evidenciado que essa modificação aprimora o efeito do acasalamento compensatório, principalmente quando EBVs precisos estão disponíveis na seleção do índice e BLUP. Além disso, são comparados o sistema de acasalamento compensatório modificado (CM + ASEBV) com outros métodos, como acasalamento aleatório (RM), acasalamento compensatório (CM), acasalamento compensatório combinado ao acasalamento de mínima coancestralidade (CM + MC) e acasalamento de mínima coancestralidade (MC). Os resultados mostram que o acasalamento de mínima coancestralidade tende a proporcionar maiores tamanhos efetivos em populações, especialmente em cenários de seleção BLUP, onde a precisão dos EBVs é fundamental. No entanto, a eficiência do CM + ASEBV é prejudicada na seleção fenotípica, devido à falta de precisão dos dados EBV, mas seu desempenho se destaca quando EBVs precisos estão disponíveis, oferecendo a possibilidade de potencializar os efeitos do sistema de acasalamento compensatório.

Como pode ser observado, a grande maioria dos trabalhos encontrados busca encontrar meios para redução da consanguinidade e da coancestralidade, trazendo comparações entre métodos já existentes e comumente usados. A Tabela 8 apresenta uma síntese das abordagens discutidas.

Dentre os trabalhos revisados, as contribuições de Nomura (1999) e NOMURA et al. (2002) mostram-se particularmente influentes para esta pesquisa. A proposta de combinar índices de seleção com a minimização da coancestralidade (LPS+MC) estabelece uma base metodológica sólida para o equilíbrio entre ganho genético e controle da endogamia. Embora Fernández, Toro e Caballero (2003) traga comparações abrangentes que validam o uso de heurísticas como o *Simulated Annealing*, a abordagem de Nomura de integrar a otimização econômica (índice) com a restrição biológica (coancestralidade) alinha-se diretamente com os objetivos deste trabalho.

Tabela 8 – Resumo das abordagens para redução da coancestralidade

Autor (Ano)	**Estratégia**	**Método/Ferramenta**
Sanchez, Toro e García (1999)	Ajuste de contribuições e acasalamento não aleatório	Seleção otimizada em *Drosophila*
Nomura (1999)	Acasalamento Compensatório Modificado	Simulação estocástica (CM + ASEBV)
Weigel (2001)	Teoria da Contribuição Ótima	Seleção restrita (programação quadrática)
NOMURA et al. (2002)	Índice de Seleção + Mínima Coancestralidade	Programação Linear (LPS) + MC
Fernández, Toro e Caballero (2003)	Comparação (MGC vs Hierárquico)	Simulação computacional (Algoritmos genéticos/SA)
Sánchez, Bijma e Woolliams (2003)	Sistema de substituição (pai-filho)	Simulação de Monte Carlo
Henryon, Sørensen e Berg (2009)	Minimização da Covariância de Contribuições (MCAC)	Simulação estocástica
Fernández et al. (2011)	Gestão de pequenas populações	Marcadores moleculares e *software* (KINSHIP, etc.)
Pryce, Hayes e Goddard (2012)	Informação Genômica (GRM)	Comparação de planos de acasalamento
Liu, Henryon e Sørensen (2017)	Estratégias genômicas (MC e MCAC)	Simulação com dados genômicos vs pedigree

Fonte: Autor (2025)

3.2.1 Grau de formalização do problema de minimização

Esta subseção trata da resposta à segunda questão de pesquisa apresentada na Seção 2.1 – [Q2] Existem descrições formais do problema da minimização da coancestralidade em sistemas de acasalamento?

Nenhum dos estudos analisados fornece uma descrição formal do problema em questão, o que dificulta a aplicação de abordagens analíticas e a demonstração de uma solução ótima para o cenário em discussão. Um exemplo de tratamento formal para problemas de otimização pode ser observado no trabalho de Motta (2021), que aborda questões financeiras, embora não esteja diretamente relacionado ao tema em discussão. Outro exemplo é o trabalho de Rosso (2023), que procura resolver o problema da redução da coancestralidade, simultaneamente maximizando o índice econômico e oferecendo definições formais para os problemas abordados. A busca por uma formalização do problema em estudo, seguida pela definição de sua complexidade, proporciona a oportunidade de determinar os limites de otimização e, em casos favoráveis, até mesmo identificar soluções ótimas com complexidade polinomial.

A principal diferença entre o presente trabalho e o de Rosso (2023) reside na abordagem metodológica: enquanto Rosso (2023) utiliza um modelo de programação linear binária, este trabalho identifica a necessidade de meta-heurística para sua resolução. Além disso, este estudo também propõe a complementação de um banco de dados orientado a grafos, visando facilitar a manipulação e análise dos dados relacionados aos pedigrees, proporcionando uma ferramenta para lidar com a complexidade e a escala do problema.

3.3 Abordagens heurísticas e meta-heurísticas

A questão de pesquisa da revisão bibliográfica (Seção 2.2.3) tem como objetivo identificar os principais métodos utilizados para estabelecer estratégias de otimização aplicadas ao problema de acasalamento em contextos com múltiplos objetivos e restrições combinatórias. Essa investigação busca mapear abordagens heurísticas e meta-heurísticas que possam ser adaptadas à redução da coancestralidade, ao mesmo tempo em que respeitam limites operacionais, como o número máximo de acasalamentos por macho e a obrigatoriedade de uso de todas as fêmeas. Os trabalhos selecionados no processo de revisão são discutidos a seguir, com foco nas técnicas empregadas, nos métodos adotados e na aplicabilidade potencial ao problema em questão.

A técnica descrita no trabalho de Ahmed, Sadjadpour e Yousefi (2022) é um algoritmo de Programação Dinâmica (DP) assistida por informação para resolver problemas de otimização discreta com restrições. A programação dinâmica é uma técnica de otimização que pode ser usada para resolver problemas complexos, quebrando-os em subproblemas menores e resolvendo cada um deles apenas uma vez, armazenando seus resultados. A essência da programação dinâmica é evitar a computação repetida, aproveitando os resultados já calculados (subproblemas) para construir a solução do problema original (SNIEDOVICH, 1991). O trabalho de (AHMED; SADJADPOUR; YOUSEFI, 2022) mostra a aplicação desta técnica para problemas de alocação de bits em conversores analógicos-digitais (ADC) com restrições de energia e seleção de recursos em sistemas de comunicação massivos MaMIMO.

O trabalho de Li et al. (2021) trata do problema de rastreamento em sistemas não lineares de tempo discreto, cujo objetivo é controlar o sistema para que siga uma trajetória de referência, minimizando os erros de rastreamento. Para isso, os autores utilizam a técnica de Programação Dinâmica Adaptativa (ADP), na qual uma função de

valor incorpora diretamente o erro de rastreamento e é otimizada por meio de iterações de valor (VI) e de política (PI), refinando sucessivamente a política de controle. A abordagem emprega uma arquitetura baseada em redes neurais, permitindo atualizações das políticas até a convergência para uma solução ótima. Embora originalmente aplicada ao controle de sistemas dinâmicos, a estratégia pode ser adaptada ao problema de minimização da coancestralidade em sistemas de acasalamento, considerando-se a escolha de cada par como um passo de controle em uma trajetória, e o objetivo como a redução cumulativa da sobreposição genética — análogo à minimização do erro em trajetórias de referência.

O método Rider Optimization Algorithm (ROA) foi criado em 2018 por Binu e Kariyappa (2019). Ele é descrito como um algoritmo composto por várias etapas que simula o comportamento de motociclistas durante uma corrida, na qual cada piloto ajusta sua estratégia para alcançar o objetivo de forma eficiente. O processo inclui a otimização dos parâmetros do ciclista, a avaliação da taxa de sucesso, a identificação do ciclista líder e a atualização da posição dos ciclistas (segundo os papéis de desvio, seguidor, ultrapassador e atacante). Além disso, o algoritmo envolve a atualização dos parâmetros do ciclista, como o contador de atividades, o ângulo de direção, a marcha, o acelerador e o freio (BINU; KARIYAPPA, 2019). Este método é aplicado em ambientes de produção, visando minimizar o tempo total de produção (*makespan*), ruído e poluição por poeira, com o objetivo de otimizar a eficiência da produção e minimizar os impactos ambientais (FU et al., 2020). O trabalho de Fu et al. (2020) apresenta o Multi-Objective Rider Optimization Algorithm (MOROA), uma extensão do ROA que busca soluções que atendam a vários objetivos conflitantes simultaneamente, utilizando arquivamento de Pareto e técnicas de classificação de vizinhança. Em seguida, o MOROA é "discretizado" para trabalhar com números discretos, dando origem ao *Discrete Multi-Objective Rider Optimization Algorithm* (DMOROA), utilizado para resolver o problema de programação de fluxo híbrido (HFSP). O HFSP é um problema de otimização combinatória que integra a programação de fluxo tradicional e a programação de máquinas paralelas (FU et al., 2020). A adaptação de um algoritmo multiobjetivo como o MROA poderia ajudar a balancear as diferentes metas no problema de redução da coancestralidade em sistemas de acasalamento: minimizar a coancestralidade ao mesmo tempo que maximiza o índice econômico e respeita os limites de acasalamento para machos e fêmeas.

O algoritmo de colônia de formigas (*Ant Colony Optimization*) é inspirado no comportamento coletivo das formigas em busca de alimentos. Ele constrói soluções incrementais, onde cada formiga representa uma possível solução para o problema

desejado. As formigas depositam feromônios nas soluções mais promissoras, orientando a busca das próximas gerações. Este método é utilizado para encontrar soluções para problemas complexos e combinatórios (DORIGO; SOCHA, 2018). Entre as aplicabilidades deste método está o algoritmo de otimização baseado em colônia de formigas (ACO), adaptado para resolver problemas de otimização com múltiplos objetivos discretos, conforme descrito por Zhao et al. (2022). Sua aplicação é direcionada para problemas práticos que envolvem múltiplos objetivos conflitantes, como o problema do caixeiro-viajante com múltiplos objetivos (MaOP), em que é necessário encontrar um conjunto de soluções Pareto-ótimas. Na versão modificada por Zhou et al. (2023), o algoritmo, denominado *Random Following Ant Colony Optimization* (RFACO), foi desenvolvido para lidar tanto com problemas de otimização global quanto com a seleção de características (*feature selection*). O RFACO incorpora uma estratégia chamada *Random Follower* (RF), que aprimora a capacidade do algoritmo de equilibrar as fases de exploração e exploração durante a busca por soluções. O método também é utilizado em problemas de otimização considerados de alta dimensionalidade, como no conjunto de testes do Congress on Evolutionary Computation 2017 (CEC 2017), (BREST; MAUCEC; BOSKOVIC, 2017 *apud* ZHAO et al., 2022, p. 11). A técnica de ACO também é aplicada à composição de serviços web orientados à qualidade de serviço (QoS), um problema que envolve a escolha de serviços da web com base em atributos como custo, tempo de resposta, disponibilidade e confiabilidade (DAHAN et al., 2021). A ideia de usar colônias de formigas, que exploram o espaço de soluções depositando "feromônios" nas melhores escolhas, pode ser adaptada para explorar combinações de pares de acasalamento, premiando combinações que levam à menor coancestralidade.

Um algoritmo evolutivo (EA) é uma técnica de otimização inspirada nos princípios da evolução natural, como seleção natural, mutação, recombinação e sobrevivência do mais apto. Rechenberg (1973) foi um dos pioneiros no campo dos EAs, especificamente nas estratégias evolutivas (Evolution Strategies, ES). Essencialmente, os EAs simulam um processo evolutivo onde, dada uma população de indivíduos em um ambiente com recursos limitados, a competição por esses recursos leva à seleção natural, resultando em um aumento da aptidão média da população ao longo do tempo. O processo evolutivo nos EAs começa com a inicialização de uma população de soluções candidatas aleatórias, seguida pela avaliação de cada candidato. Em cada iteração, os pais são selecionados com base em suas aptidões, recombinados para produzir novos filhos, e submetidos a mutações para introduzir diversidade. Os novos candidatos são então avaliados e os

indivíduos são selecionados para formar a próxima geração. Este ciclo se repete até que uma solução de qualidade suficiente seja encontrada ou que um limite computacional seja atingido. A aplicação combinada de variação (recombinação e mutação) e seleção tende a melhorar os valores de aptidão nas populações consecutivas, aproximando as soluções ótimas ao longo do tempo (EIBEN et al., 2015). Um exemplo de aplicação de técnicas de algoritmos evolutivos, como o NSGA-II e NSGA-III, pode ser encontrado no trabalho de Chakravarthi e Kumar (2020). Nesse estudo, o objetivo é otimizar a cobertura e a vida útil de uma rede de sensores sem fio (WSN), maximizando a área coberta pelos sensores enquanto se minimiza o consumo de energia. Ao mesmo tempo, busca-se limitar a conectividade da rede para garantir que o menor número de sensores seja utilizado de forma eficiente. Os algoritmos genéticos de classificação não-dominada NSGA-II e NSGA-III são aplicados para resolver esse problema de múltiplos objetivos. NSGA-III, em particular, é adequado para lidar com problemas de otimização não lineares e complexos. Essas técnicas são utilizadas para melhorar o desempenho de redes de monitoramento e vigilância, em que os sensores, ao se moverem, podem impactar a conectividade (CHAKRAVARTHI; KUMAR, 2020). O problema da redução da coancestralidade pode se beneficiar do uso de NSGA-III, que pode gerar soluções que equilibram múltiplos objetivos. A capacidade de manter a diversidade de soluções pode ser útil para explorar diferentes pareamentos possíveis. A técnica de otimização usada para maximizar a cobertura e minimizar o consumo de energia poderia ser adaptada para maximizar a diversidade genética e minimizar a coancestralidade, garantindo que cada fêmea seja pareada apenas uma vez e respeitando o limite de pareamentos por macho.

A técnica apresentada no trabalho de Singh et al. (2020) é uma otimização baseada no comportamento migratório das borboletas-monarca, conhecida como Monarch Butterfly Optimization (MBO). Este método meta-heurístico foi proposto por Wang, Deb e Cui (2019). Nesse estudo de Singh et al. (2020), foi utilizado o MBO, em conjunto com a técnica TOPSIS (Ordem de Preferência por Similaridade com a Solução Ideal), para resolver problemas de múltiplos objetivos relacionados à integração ótima de recursos energéticos distribuídos (DERs). O objetivo da técnica MBO-TOPSIS é encontrar os melhores locais, tamanhos e combinações de DERs (despacháveis e não-despacháveis), equilibrando múltiplos critérios de desempenho, como a redução das perdas anuais de energia, melhoria na estabilidade da tensão dos nodos e aumento da capacidade de carregamento dos ramos da rede elétrica. O problema da redução da coancestralidade em sistemas de acasalamento é semelhante à otimização de recursos energéticos distribuídos

(DERs) em sistemas de distribuição de energia; ambos envolvem múltiplos objetivos a serem equilibrados. No caso dos acasalamentos, o objetivo é minimizar o parentesco, maximizar o índice econômico e respeitar restrições de acasalamento, enquanto no problema de DERs, o foco é minimizar perdas de energia e melhorar a estabilidade da rede.

Algoritmos meméticos são uma combinação de um método de busca global baseado em população (como algoritmos genéticos) com uma busca local heurística feita por cada indivíduo. Essa abordagem combina as capacidades de explorar o espaço de soluções, através da interação entre diferentes indivíduos da população, com a capacidade de refinar localmente as soluções, buscando o ótimo em regiões específicas. Em essência, o algoritmo memético utiliza ideias de evolução biológica (como reprodução e cruzamento) e de evolução cultural (como a transmissão e refinamento de ideias) para otimizar problemas complexos (MOSCATO, 2000). Entre suas aplicabilidades, destaca-se a resolução do problema de agendamento em máquinas paralelas heterogêneas não relacionadas (DHUPMSP), visando minimizar tanto o atraso total quanto o tempo de conclusão em sistemas de manufatura distribuída. Esse método pode ser aplicado em diversas indústrias, como a de chicotes elétricos, sistemas de frete e agendamento de satélites de observação da Terra, entre outros (WANG; LI; GONG, 2023). A abordagem do Algoritmo Memético Baseado em Conhecimento e Pareto (KPMA) de Wang, Li e Gong (2023), que integra regras heurísticas e estruturas de vizinhança para melhorar a eficiência da busca, poderia ser adaptada para selecionar combinações ótimas de pares de acasalamento. As regras heurísticas usadas no agendamento (como a de menor tempo de processamento ou menor carga de trabalho) poderiam ser substituídas por regras que considerem o parentesco mínimo, a diversidade genética e as restrições de acasalamento. Além disso, as estruturas de vizinhança poderiam melhorar a taxa de sucesso da busca por melhores combinações de pares, assim como otimizam a alocação de tarefas no agendamento.

A Tabela 9 apresenta um panorama das abordagens utilizadas em diferentes estudos de otimização. Dentre as meta-heurísticas analisadas, o *Ant Colony Optimization* (ACO) destaca-se como a abordagem mais promissora para o problema deste trabalho. Sua natureza construtiva permite a geração passo a passo das soluções de acasalamento, facilitando a verificação e satisfação de restrições complexas (como limites de uso de reprodutores) durante o próprio processo de construção, algo que é mais custoso em abordagens evolutivas tradicionais que requerem operadores de reparo. Além disso, a

representação do problema em grafo alinha-se naturalmente com a estrutura dos dados genealógicos armazenados no Neo4j. Essa escolha se fundamenta na capacidade do ACO de balancear múltiplos objetivos (coancestralidade e índices econômicos) através da informação heurística e do feromônio, guiando a busca para regiões de alta qualidade no espaço de soluções.

Tabela 9 – Técnicas e aplicações

Trabalho	**Técnica**	**Método**
Singh et al. (2020)	Otimização por borboletas-monarca (MBO)	MCDM com MBO-TOPSIS
Wang, Li e Gong (2023)	Algoritmo Memético	Pareto-KPMA (baseado em heurísticas)
Chakravarthi e Kumar (2020)	Algoritmo Genético (NSGA-III)	Pareto-NSGA-III
Ahmed, Sadjadpour e Yousefi (2022)	Programação Dinâmica	Otimização Multiobjetiva com Kullback-Leibler
Fu et al. (2020)	Discrete Multi-Objetive Rider Optimization Algorithm	Rider Optimization Algorithm com Arquivamento Pareto
Dahan et al. (2021)	Ant Colony Optimization	Algoritmo Aprimorado com Seleção de Vizinhança e Pheromônios
Li et al. (2021)	Adaptive Dynamic Programming (ADP)	Iteração de Valor (VI) e Iteração de Política (PI)
Zhao et al. (2022)	Ant Colony Optimization	Decomposição adaptativa com atualização de feromônios e seleção adaptativa
Zhou et al. (2023)	Ant Colony Optimization com Random Following	Troca de informações e estratégias adaptativas

Fonte: Autor (2025)

4 FUNDAMENTOS

4.1 Definições e formalização do problema

A abordagem formal para resolução de problemas requer definições precisas dos termos sobre os quais se deseja trabalhar. Nesta seção são definidos formalmente os conceitos relacionados aos objetivos do trabalho. As Definições 1 a 4 foram adaptadas de Ferreira, Yokoo e Motta (2021); as Definições 5 e 7 foram primeiro apresentadas em Rosso, Rosa e Ferreira (2023). As demais definições são resultados deste trabalho.

Definição 1 (Índice de seleção) *Seja T um conjunto de características mensuráveis de animais. Um índice de seleção é um par $\mathscr{I} = (T, w)$ onde $w : T \to \mathbb{R}$ é a função de ponderação do índice.*

A função de ponderação determina o peso de cada característica no índice, sem restrições de valores. A principal vantagem dos índices de seleção é classificar os animais por um valor único. O índice adotado baseia-se em (YOKOO et al., 2021) e (MOTTA, 2021), abrangendo objetivos de seleção e seus respectivos ponderadores.

A Tabela 10 apresenta as características avaliadas pelo índice, juntamente com seus respectivos ponderadores. Os indicadores avaliados são a contagem de carrapatos (TICK), a contagem de ovos por grama de fezes (OPG), e o Peso de carcaça quente (PCQ), que se refere ao peso da carcaça após a conclusão de todos os procedimentos de abate. Além disso, alguns critérios de seleção associados ao rendimento e à qualidade da carcaça também são considerados, incluindo a área de olho de lombo (AOL), a espessura de gordura subcutânea (EG), e a espessura de gordura subcutânea na garupa (EGP8). Outros fatores como o peso da vaca adulta (PVA), a taxa de prenhez (TP), e o temperamento (T), o qual se refere à resposta comportamental do animal durante o manejo do rebanho, também são levados em consideração. Os ponderadores indicam o quanto (valor absoluto) uma característica é desejada (valor positivo) ou indesejada (valor negativo).

O índice de seleção permite usar um único valor para classificar um animal. Esse valor também é um indicativo do quanto esse animal pode contribuir para sua descendência, caso seja escolhido para reprodução. A Definição 2 define essa ideia e mostra como o cálculo do valor do animal é realizado, a partir do índice de seleção.

Definição 2 (Contribuição individual) *Seja A um conjunto de animais, $\mathscr{I} = (T, w)$ um índice de seleção e $v : T \times A \to \mathbb{R}$ uma função que retorna o valor de uma característica*

Tabela 10 – Índice econômico de ciclo completo (IEEC)

Objetivo de seleção	**Critério de seleção**	**Ponderador**
TICK	Contagem de carrapatos	116.0879975
OPG	Contagem de ovos por grama de fezes	-9.1900268
PCQ	AOL	0.1866388
	EG	131.3764082
	EGP8	65.8745087
PVA	Peso adulto da vaca	-0.5495934
TP	Peso ao nascer	6.0030468
T	Temperamento	23.8195584

Fonte: (MOTTA, 2021)

$t \in T$ para cada animal no conjunto A. A função $\iota^{\mathscr{I}} : A \to \mathbb{R}$ retorna a contribuição individual de cada animal $a \in A$, de acordo com o índice de seleção $\mathscr{I}$, calculado como

$$\iota^{\mathscr{I}}(a) = \sum_{t \in T} v(t,a) \cdot w(t) \tag{1}$$

Quando o índice de seleção é entendido a partir do contexto, escrevemos $\iota(a)$ em vez de $\iota^{\mathscr{I}}(a)$.

A contribuição do acasalamento entre dois animais, formalmente apresentada na Definição 3, representa a contribuição individual esperada da prole resultante do acasalamento entre um macho e uma fêmea. Essa contribuição é calculada como a média das contribuições individuais dos progenitores. Novamente, o índice de seleção é adotado como a unidade de medida para quantificar a contribuição individual de cada indivíduo.

Definição 3 (Contribuição de acasalamento) *Seja $A = M \cup F$ um conjunto de animais, onde M é o conjunto de reprodutores e F é o conjunto de matrizes, com $M \cap F = \emptyset$ e seja $\mathscr{I}$ um índice de seleção. A função de contribuição de acasalamento $\pi_{\mathscr{I}} : M \times F \to \mathbb{R}$ de um reprodutor $m \in M$ e de uma matriz $f \in F$ é calculada da seguinte forma:*

$$\pi_{\mathscr{I}}(m,f) = \frac{\iota^{\mathscr{I}}(m) + \iota^{\mathscr{I}}(f)}{2} \tag{2}$$

onde $\iota^{\mathscr{I}}$ é a contribuição individual de cada animal, dado o índice de seleção $\mathscr{I}$.

A ideia por trás da contribuição de acasalamento é estimar onde cada $\pi(m,f)^{\mathscr{I}}$ representa o índice esperado do animal nascido do par $(m,f) \in M \times F$, ou a contribuição esperada para a prole do cruzamento entre esses dois animais.

A Definição 3 é ligeiramente diferente da apresentada em Ferreira, Yokoo e Motta (2021), em que dois animais com relação de parentesco acima de um certo limiar (estabelecido de forma *ad hoc*) tinham como resultados da contribuição de acasalamento $-\infty$ (na prática, o menor valor inteiro representado pela máquina). Dado que o algoritmo proposto não lidava com a consanguinidade da prole, essa era uma forma de eliminar o par da solução do problema.

Ferreira, Yokoo e Motta (2021) também demonstram que o acasalamento que maximiza o valor esperado do índice da prole ocorreria se o melhor macho fosse acasalado com todas as fêmeas. No entanto, essa abordagem não é fisicamente factível, além do que limitaria as opções de acasalamento para a próxima geração, já que todos os descendentes seriam meio-irmãos. Note-se que os descendentes obtidos de matrizes diferentes a partir do mesmo reprodutor não teriam um índice de consanguinidade alto, dado que seus pais não teriam parentesco direto. No entanto, os valores de coancestralidade seriam altos, devido ao progenitor comum. Isso sugere que a coancestralidade é uma métrica melhor para frear o aumento da consanguinidade em rebanhos. Portanto, a ênfase deve ser colocada na redução da coancestralidade, isto é, no grau de parentesco entre os indivíduos de um rebanho, para otimizar os resultados genéticos/fenotípicos sem comprometer a diversidade e a saúde da população.

A partir das definições acima, o problema da maximização do índice da prole pode ser definido, de acordo com a Definição 4.

Definição 4 (Problema da maximização do índice de seleção)
***Entrada**: um índice de seleção $\mathscr{I}$, um conjunto de reprodutores M, um conjunto de matrizes F, $\pi^{\mathscr{I}} : M \times F \to \mathbb{R}$ a função de contribuição de acasalamento (Definição 3) e uma função* $\max : M \to \mathbb{N}$ *que informa o número máximo de vezes que um reprodutor pode ser usado no processo de acasalamento.*
***Saída**: uma função $b^* : F \to M$ que obedece à restrição de acasalamentos, i.e., para cada $m \in M$, $|\{f \in F \mid b^*(f) = m\}| \leq \max(m)$ e que para qualquer outra função $b : F \to M$ que também respeite a restrição de acasalamento* max*, tem-se que*

$$\sum_{k \in F} \pi(b(k), k) \leq \sum_{k \in F} \pi(b^*(k), k)$$

Ferreira, Yokoo e Motta (2021) apresentaram um algoritmo polinomial ótimo para determinar uma estratégia de acasalamento visando maximizar o valor esperado da prole resultante, em relação ao índice usado na seleção dos animais, resolvendo o problema

apresentado na Definição 4. Posteriormente, o mesmo algoritmo foi usado como solução inicial para resolver o problema do acasalamento compensatório de forma polinomial, mantendo o valor da solução ótima (FERREIRA, 2021). Contudo, esses algoritmos não tratam da questão da coancestralidade entre os animais do rebanho nem entre a prole produzida. O controle da consanguinidade é feito por um parâmetro do algoritmo, que informa o grau de coancestralidade aceitável para que dois animais possam reproduzir (que pode ser zero). A estratégia para que o melhor reprodutor não seja escolhido para todas as matrizes (melhor valor possível do índice da prole) é limitar o número de vezes que cada touro pode ser usado, que também são parâmetros do algoritmo. A origem desse valor não é calculado de forma explícita, mas seria interessante que fosse obtido por métodos algorítmicos, garantindo a possibilidade de acasalamentos dentro do rebanho no maior prazo possível.

Com base no contexto apresentado, existe a demanda por um sistema de acasalamento que controle a coancestralidade e a consanguinidade média do rebanho sem comprometer futuros acasalamentos no mesmo rebanho, ao mesmo tempo que mantenha o maior valor possível para a prole produzida. Isso implica em acasalamentos que não prejudiquem o processo de melhoramento, ao mesmo tempo em que evitem perdas devido à depressão por consanguinidade. Em ambos os casos, o impacto negativo na lucratividade do produtor deve ser minimizado, exigindo um equilíbrio entre as restrições do acasalamento. Reduzir as taxas médias de coancestralidade do rebanho implica em controlar a presença de ancestrais comuns nas gerações futuras.

A heurística ótima apresentada em Ferreira, Yokoo e Motta (2021), contudo, não funciona para o problema da minimização da coancestralidade. Outras estratégias precisam ser delineadas para que esse objetivo seja atingido. Quando somente o índice de seleção é levado em consideração, o valor da contribuição de acasalamento apresentada na Definição 3 é suficiente para que o algoritmo de maximização do valor da prole seja usado. Contudo, neste trabalho planeja-se também considerar a coancestralidade do rebanho. As definições a seguir estabelecem formalmente os conceitos usados neste trabalho.

Definição 5 (Rebanho) *Um* rebanho *é uma tupla* $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$ *onde* A *é um conjunto finito de animais particionado em um conjunto de reprodutores (machos)* S *e um conjunto de matrizes (fêmeas)* D*, em que cada animal* $a \in A$ *tem um pai* $s(a)$ *e uma mãe* $d(a)$*. Se o progenitor do animal* a *não pertence ao rebanho, o resultado da função é* $\perp$*.*

A Definição 5 expressa formalmente o significado do termo “rebanho”: uma

coleção de animais e duas funções que indicam, para cada animal, quem são os seus progenitores. Para tornar as funções totais, foi criado o símbolo $\perp$, que indica que o progenitor do animal não se encontra entre os animais do rebanho. Essa situação pode ocorrer tanto com animais adquiridos de outros produtores ou produzidos por meio de inseminação artificial com sêmen de origem externa.

Definição 6 (Prole) *Seja* $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$ *um rebanho e sejam* $S^* \subseteq S$ *e* $D^* \subseteq D$ *os conjuntos de machos e fêmeas selecionados para reprodução, respectivamente. Uma* prole *é qualquer função* $p : D^* \to S^*$.

A Definição 6 considera a existência de machos e fêmeas selecionados para reprodução. O processo de seleção, que determina que são todas as fêmeas que serão mães da próxima geração e os machos que poderão cruzar com elas, é assumido como pronto neste trabalho. A definição assume que todas as fêmeas selecionadas deverão cruzar com algum macho selecionado. A função p faz a atribuição dos machos escolhidos para cada uma das fêmeas.

O cálculo da contribuição individual (Definição 2) dos indivíduos da prole é derivado da contribuição de acasalamento (Definição 3) de seus genitores. Dado que os indivíduos da prole ainda não existem e são meramente uma projeção, seu sexo não é definido como macho ou fêmea, mas sim concebido como um produto em potencial.

O primeiro passo para o cálculo do índice de coancestralidade é a atribuição de um valor de *geração* a cada animal. Aqui, a ideia de geração é determinar uma partição no conjunto total de animais, permitindo que eles possam ser estruturados de acordo com o número de cruzamentos existentes dentro do rebanho que foi necessário para o animal ser concebido. A Definição 7 expressa essa ideia formalmente.

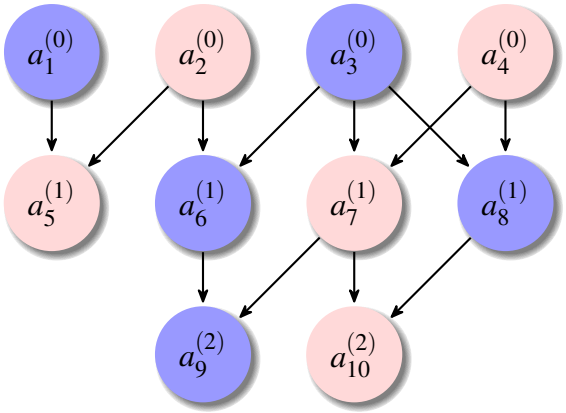
Definição 7 (Geração) *Seja* $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$ *um rebanho. A* geração *de cada animal* $a \in A$ *é dada pela função* $g : A \to \mathbb{N}$ *definida como se segue:*

$$g(a) = \begin{cases} 0 & s(a) = \perp \wedge d(a) = \perp \\ g(s(a)) + 1 & s(a) \neq \perp \wedge d(a) = \perp \\ g(d(a)) + 1 & s(a) = \perp \wedge d(a) \neq \perp \\ \max\{g(s(a)), g(d(a))\} + 1 & s(a) \neq \perp \wedge d(a) \neq \perp \end{cases} \tag{3}$$

A geração de um animal é o valor seguinte ao maior valor de geração dos pais do animal. A função g determina uma partição no conjunto dos animais, em que cada classe de equivalência é um valor geracional. A Figura 2 mostra um exemplo de rebanho

com 10 animais, referenciados por $a_1, a_2, \ldots, a_{10}$. Para facilitar a visualização, nodos azuis e rosa representam machos e fêmeas, respectivamente. As relações de parentesco diretas induzem um grafo dirigido acíclico (*directed acyclic graph* ou DAG) (CORMEN et al., 2002) e o fecho transitivo e reflexivo da relação de ancestralidade é uma relação antissimétrica. O valor entre parênteses denota a geração de cada animal, conforme a Definição 7. As arestas representam a relações diretas de parentesco, dadas pelas funções s e d da Definição 5.

Figura 2 – DAG de estrutura do rebanho – exemplo



Fonte: (ROSSO; ROSA; FERREIRA, 2023)

O índice de coancestralidade indica o grau com que dois animais são aparentados e é uma estimativa do grau de similaridade genética entre dois animais e da quantidades de cruzamentos consanguíneos em sua linha ancestral. Assim, um animal tem 100% de similaridade genética consigo mesmo e 50% com cada um de seus pais, se eles não forem aparentados; se eles forem, há um acréscimo relacionado ao índice de coancestralidade entre o pai e a mãe do animal. A ideia é mantida para qualquer outro animal do rebanho, com os índices dos ancestrais aparentados somando, mas diminuindo à medida que a distância geracional aumenta. A Definição 8 formaliza essa ideia.

Definição 8 (Coeficiente de Parentesco) *(adaptado de (MRODE; THOMPSON, 2005)pag. 44) Seja A um conjunto de animais, e $s(a)$ e $d(a)$ as funções que mapeiam um animal $a \in A$ para seu pai (sire) e mãe (dam), respectivamente. O coeficiente de parentesco entre dois animais $i, j \in A$, denotado por $c(i, j)$, é definido como:*

$$c(i,j) = \begin{cases} 1 & i = j \\ 0{,}5 \cdot (c(i,s(j)) + c(i,d(j))) & s(j) \neq \perp \wedge d(j) \neq \perp \\ 0{,}5 \cdot c(i,s(j)) & s(j) \neq \perp \wedge d(j) = \perp \\ 0{,}5 \cdot c(i,d(j)) & s(j) = \perp \wedge d(j) \neq \perp \\ 0 & s(j) = \perp \wedge d(j) = \perp \end{cases} \quad (4)$$

onde $c(i,s(j))$ é a coancestralidade entre i e o pai de j, $s(j)$, e $c(i,d(j))$ é a coancestralidade entre i e a mãe de j, $d(j)$.

Outras definições do coeficiente de parentesco podem ser construídas de forma diferente, dependendo da intenção dos autores. Essa é a definição que será usada para a construção do algoritmo dinâmico de cálculo do coeficiente de parentesco entre os animais para registro no banco de dados. A flexibilidade dos modelos de dados em bancos orientados a grafos permite que outras definições sejam registradas no banco concomitantemente e identificadas pela rotulação das arestas. Maiores detalhes sobre a implementação são dados na Seção 5.1.

Definição 9 (Aptidão) *Seja $H = (A,s,d)$ um rebanho. A função de* aptidão $v : A \to \{0,1\}$ *atribui* 1 *a um animal cuja carga genética seja relevante para o planejamento das proles futuras e* 0 *caso contrário.*

O estado de aptidão de um animal dentro do rebanho, formalizado na Definição 9, indica se ele ainda contribui ou tem potencial de contribuir com material genético para as próximas gerações. Caso o animal esteja nessas condições, é desejável que os demais animais utilizados para reprodução apresentem baixa coancestralidade em relação a ele. O contrário também é válido; caso o animal não faça mais parte do rebanho, não é necessário atentar-se à coancestralidade das futuras gerações em relação a ele. Note-se que um animal pode ter valor 1 sem ser selecionado para a reprodução, por ser jovem demais para reproduzir, por exemplo. Essa definição será usada nos algoritmos construídos, de forma a não aumentar desnecessariamente o tempo de execução dos algoritmos: animais que não vão mais participar do processo não precisam ser levados em consideração. Caso seus descendentes ainda estejam no rebanho, os valores de coancestralidade deles com os restantes animais são suficientes.

Definição 10 (Coancestralidade da prole com o rebanho) *Sejam um rebanho $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$, a função de estado de aptidão $v : A \to \{0,1\}$, os*

conjuntos de animais selecionados $S^* \in S$ e $D^* \in D$. A coancestralidade da prole com o rebanho *para uma prole $p : D^* \to S^*$ é o valor $C(p)$ calculado como:*

$$C(p) = \sum_{x \in D^*} \sum_{a \in A, v(a)=1} c([x, p(x)], a) \tag{5}$$

em que c é a função que retorna o valor de coancestralidade entre dois animais (Definição 8); para o cálculo do valor de coancestralidade entre um animal $[x, p(x)]$ da prole e um animal a do rebanho, assume-se que $d([x, p(x)]) = x$ e que $s([x, p(x)]) = p(x)$.

O valor médio da coancestralidade da prole p com o rebanho H, $\overline{C}(p)$, é dado por

$$\overline{C}(p) = \frac{C(p)}{|S^* \times \{a \in A \mid v(a) = 1\}|} \tag{6}$$

Definição 11 (Coancestralidade entre os animais da prole) *Sejam um rebanho $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$, uma função de aptidão $v : A \to \{0, 1\}$, os conjuntos de animais selecionados $S^* \in S$ e $D^* \in D$ e uma prole $p : D^* \to S^*$. A* coancestralidade entre os animais da prole *p é o valor $C(\overline{p})$ calculado como:*

$$C(\overline{p}) = \sum_{x \in D^*} \sum_{y \in D^*} c([x, p(x)], [y, p(y)]) \tag{7}$$

em que c é a função que retorna o valor de coancestralidade entre dois animais (Definição 8); novamente, assume-se que para um animal $[x, p(x)]$ da prole tem-se que $d([x, p(x)]) = x$ e que $s([x, p(x)]) = p(x)$.

O valor médio da coancestralidade entre os animais da prole p, $\overline{C}(\overline{p})$, é dado por

$$\overline{C}(\overline{p}) = \frac{C(\overline{p})}{|S^*|^2} \tag{8}$$

A definição do problema de minimização da coancestralidade pode ser feita de diversas formas. Concretamente, pode-se tentar minimizar o valor da coancestralidade (ou da média da coancestralidade) somente entre os animais da prole (Definição 11), dos animais da prole com todos os outros animais do rebanho (Definição 10) ou da soma dos dois valores. Outra possibilidade é buscar minimizar, para essas mesmas situações (i.e., prole-prole, prole-rebanho, prole-prole + prole-rebanho) não o valor em si, mas o número de arestas entre os animais (no grafo que representa o coeficiente de parentesco) com valor maior que zero. Essa estratégia pode eliminar um rebanho com uma média baixa mas muito conectado, o que causaria impedimentos de cruzamentos mais cedo do que

um rebanho com vários *clusters* familiares mas com mais possibilidades de cruzamentos. Outras possibilidades visam combinar o problema da minimização da coancestralidade com a maximização do valor do índice de seleção. As definições abaixo correspondem a essas ideias e serão desenvolvidas por meio dos algoritmos desenvolvidos no restante deste trabalho.

Definição 12 (Problema da minimização da coancestralidade)
***Entrada**: um rebanho (A,s,d), um conjunto de fêmeas D^* e de machos S^* selecionados para reprodução, duas funções* $\max,\min : S^* \to \mathbb{N}$*, que informam o maior e o menor número de vezes, respectivamente, que um macho deve ser usado no processo de reprodução*
***Saída**: uma função $p : D^* \to S^*$ que obedece às restrições de acasalamentos dos machos, i.e., para cada $y \in S^*$,* $\min(y) \leq |\{x \in D^* \mid p(x) = y\}| \leq \max(y)$*, e que para qualquer outra função $f : D^* \to S^*$ que também respeite as restrições de acasalamento* $\min,\max$*, tem-se que*

$$\sum_{x \in D^*} \mathscr{C}(p) \leq \sum_{x \in D^*} \mathscr{C}(f)$$

em que $\mathscr{C}(p)$ pode ser $C(p)$, $\overline{C}(p)$ (Definição 10), $C(\overline{p})$, $\overline{C}(\overline{p})$ (Definição 11), $C(p)+C(\overline{p})$ ou qualquer outra combinação de cálculos de coancestralidade.

Os problemas relacionados à minimização das arestas no grafo de coancestralidade buscam minimizar a soma de arestas com valor maior que zero. Ou seja, minimizar a quantidade de animais que têm relações de parentesco (i.e., ancestrais comuns) entre si. Da mesma forma que para o problema da minimização do valor (total, médio) de coancestralidade, a minimização do número de relações pode ser feita entre a prole com o resto do rebanho, da prole com a própria prole ou se um soma das duas.

Definição 13 (Densidade do grafo de coancestralidade da prole com o rebanho)
Sejam um rebanho $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}), v : A \to \{0,1\})$ e os conjuntos de animais selecionados para reprodução $S^* \in S$ e $D^* \in D$. A densidade do grafo de coancestralidade *para uma prole $p : D^* \to S^*$ é o valor $A(p)$ calculado como:*

$$A(p) = \sum_{x \in D^*} \sum_{y \in A, v(y)=1} a([x, p(x)], y) \tag{9}$$

em que $a(n,m) = 1$ se $c(n,m) > 0$ e $a(n,m) = 0$ se $c(n,m) = 0$, onde c é a função que retorna o valor de coancestralidade entre dois animais (Definição 8); para o cálculo do

valor de coancestralidade entre um animal $[x, p(x)]$ *da prole e um animal a do rebanho, assume-se que* $d([x, p(x)]) = x$ *e que* $s([x, p(x)]) = p(x)$.

Definição 14 (Densidade do grafo de coancestralidade da prole) *Sejam um rebanho* $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}), v : A \to \{0,1\})$ *e os conjuntos de animais selecionados para reprodução* $S^* \in S$ *e* $D^* \in D$. *A* densidade do grafo de coancestralidade *entre os animais de uma prole* $p : D^* \to S^*$ *é o valor* $A(\overline{p})$ *calculado como:*

$$A(\overline{p}) = \sum_{x \in D^*} \sum_{y \in D^*} a([x, p(x)], [y, p(y)]) \tag{10}$$

em que $a(n,m) = 1$ *se* $c(n,m) > 0$ *e* $a(n,m) = 0$ *se* $c(n,m) = 0$, *onde* $c(n,m)$ *é o valor de coancestralidade entre os animais n e m (Definição 8); entende-se que, para um animal* $[x, p(x)]$ *da prole tem-se que* $d([x, p(x)]) = x$ *e que* $s([x, p(x)]) = p(x)$.

A partir das definições acima, que contam o número de arestas com coancestralidade estritamente positiva entre os animais, pode-se definir o problema da minimização de arestas do grafo de coancestralidade.

Definição 15 (Problema da minimização de arestas de coancestralidade)
Entrada*: um rebanho* (A, s, d), *um conjunto de fêmeas* $D^* \subseteq A$ *e de machos* $S^* \subseteq A$ *selecionados para reprodução, duas funções* $\max, \min : S^* \to \mathbb{N}$, *que informam o maior e o menor número de vezes, respectivamente, que um macho deve ser usado no processo de reprodução*
Saída*: uma função* $p : D^* \to S^*$ *que obedece às restrições de acasalamentos dos machos, i.e., para cada* $y \in S^*$, $\min(y) \leq |\{x \in D^* \mid p(x) = y\}| \leq \max(y)$, *e que para qualquer outra função* $f : D^* \to S^*$ *que também respeite as restrições de acasalamento* $\min, \max$, *tem-se que*

$$\sum_{x \in D^*} \mathscr{A}(p) \leq \sum_{x \in D^*} \mathscr{A}(f)$$

em que $\mathscr{A}(p)$ *pode ser* $A(p)$ *(Definição 13),* $A(\overline{p})$ *(Definição 14) ou* $A(p) + A(\overline{p})$.

O problema de combinar a minimização da coancestralidade com a maximização do índice de seleção depende de definirmos a contribuição de cada par de animais em relação a essas duas medidas. As definições a seguir são necessárias para a definição do problema, apresentado na Definição 17.

Definição 16 (Contribuição mista) *Seja* $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$ *um rebanho,* $D^* \subseteq D$ *e* $S^* \subseteq S$ *os conjuntos de matrizes e reprodutores selecionados para*

reprodução, e seja $\mathscr{I}$ um índice de seleção associado a cada candidato $c \in A$. A contribuição mista *de um candidato $c \in A$ é definida como:*

$$\pi_M^{\mathscr{I}}(c) = \frac{\mathscr{I}(c)}{1 + \sum_{z \in A} c(z,c)} \tag{11}$$

onde $\mathscr{I}(c)$ representa o valor do índice de seleção do candidato c, e $\sum_{z \in A} c(z,c)$ corresponde à soma dos coeficientes de parentesco entre c e todos os demais animais do rebanho A.

Definição 17 (Problema da maximização de melhoramento com contribuição mista)

***Entrada**: um rebanho $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$, uma função de aptidão $v : A \to \{0,1\}$, um subconjunto de animais selecionados $D^* \subseteq D$ e $S^* \subseteq S$, e duas funções* $\max, \min : S^* \to \mathbb{N}$*, respectivamente, as funções que indicam os limites máximo e mínimo de utilização de reprodutor no processo de acasalamento.*
***Saída**: uma função $p : D^* \to S^*$ tal que* $\min(y) \leq |\{x \in D^* \mid p(x) = y\}| \leq \max(y)$ *e que maximize*

$$\sum_{x \in D^*} \pi_M^{\mathscr{I}}(p(x))$$

sendo $\pi_M^{\mathscr{I}}(\cdot)$ a contribuição mista definida na Definição 16.

O problema da maximização de melhoramento ponderado, apresentado na Definição 17, teve uma solução desenvolvida e apresentada em (ROSSO, 2023). A solução representou o problema com um modelo de programação inteira, relaxado para programação linear e resolvido por uma combinação dos métodos simplex e branch-and-bound. A experiência mostrou que várias instâncias maiores do problema faziam o algoritmo demorar muito tempo para apresentar uma solução, o que foi posteriormente analisado e apresentado em Rosso e Ferreira (2023). Nesse trabalho, foi verificado que à medida que a ponderação da coancestralidade crescia o algoritmo demorava mais tempo para executar. Isso, além do fato de que a troca dos pares acasalados requer um recálculo do valor da coancestralidade da solução (o que não é o caso quando se considera somente o índice de seleção, vide (FERREIRA; YOKOO; MOTTA, 2021)), é um indicativo importante de que, mesmo que o problema não seja computacionalmente intratável, a complexidade da solução deve requerer estratégias heurísticas para sua solução para rebanho de tamanho grandes.

Problemas que buscam uma solução ótima ou aproximadamente ótima em um espaço de soluções possíveis são denominados *problemas de otimização*. A Seção 4.2 apresenta os fundamentos desse tipo de problema e as estratégias possíveis para sua solução.

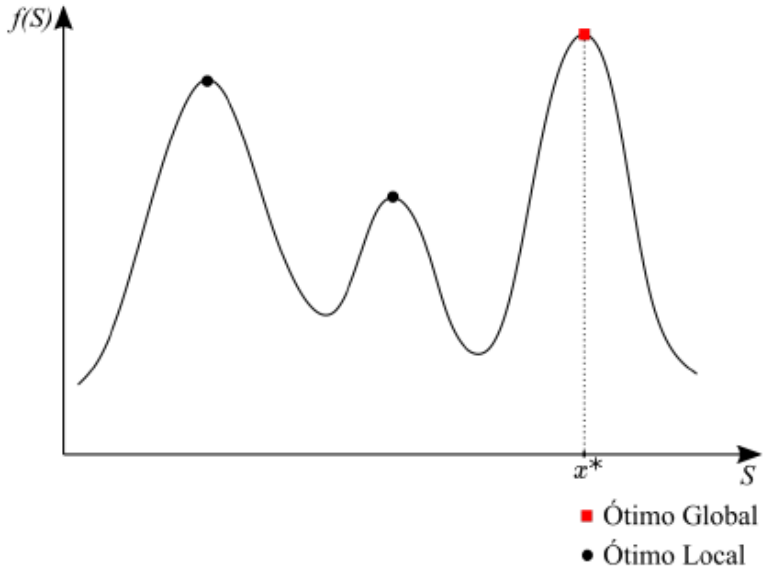
4.2 Problemas de otimização

A solução de um problema de otimização envolve encontrar a melhor solução possível dentro de um conjunto de soluções viáveis. Isso geralmente implica maximizar ou minimizar uma função objetivo, sujeita a um conjunto de restrições. A função objetivo é determinada por uma expressão matemática que quantifica o valor de uma solução do problema, em função das variáveis que o descrevem. O valor expresso pela função objetivo usualmente representa lucro ou eficiência – e nesse caso se deseja maximizar o valor, ou pode representar valores de custo ou tempo de produção e, nesse caso, o objetivo envolve minimizar o valor.

As restrições do problema determinam quais são as suas soluções viáveis. Restrições comuns incluem limitações de recursos, capacidades máximas ou mínimas de realização de trabalho, restrições nos valores que as variáveis do problema podem tomar, restrições de tempo, entre outras. O objetivo final é identificar a configuração ou conjunto de variáveis que otimiza essa função – para mais ou para menos – garantindo simultaneamente a aderência às restrições, resultando na solução que melhor atende aos requisitos específicos do problema em questão (BAGHEL; AGRAWAL; SILAKARI, 2012).

De maneira formal, um problema de otimização é caracterizado por um conjunto S, conhecido como domínio ou espaço de busca, uma função objetivo ou função de ganho/custo $f : S \rightarrow \mathbb{R}$, em que $\mathbb{R}$ denota o conjunto dos números reais. O termo "ganho" é empregado quando o objetivo é maximizar a função objetivo, enquanto "custo" refere-se à minimização da mesma. A solução de uma instância de um problema de otimização $P = (S, f)$ envolve a identificação de um elemento $x^* \in S$ do espaço de soluções tal que $f(x^*) \geq f(x)$, para qualquer $x \in S$, no caso de maximização de f; ou que $f(x^*) \leq f(x)$, para todo $x \in S$, se a intenção for minimizar f (PAPADIMITRIOU; STEIGLITZ, 1998). O conjunto S abrange todas as soluções viáveis para o problema P, sendo que o elemento x representa uma solução factível. O termo x^* é designado como ótimo global ou solução ótima (PAPADIMITRIOU; STEIGLITZ, 1998).

Figura 3 – Ótimo global e ótimo local



Fonte: (MOTTA, 2021)

Soluções para problemas de otimização podem ser locais ou globais. Um mínimo (ou máximo) local é um ponto em que a função objetivo f atinge um valor menor (ou maior) do que em seus pontos vizinhos, mas pode não ser o menor (ou maior) valor possível globalmente. Um mínimo (ou máximo) global é o menor (ou maior) valor possível da função objetivo f em todo o conjunto factível. A Figura 3 ilustra os conceitos de ótimo global e ótimo local, para um problema de maximização.

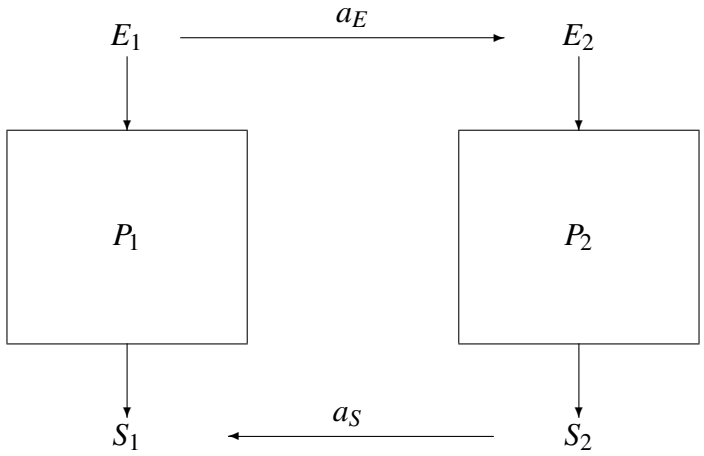
Quando o problema de otimização envolve encontrar a melhor solução dentro de um conjunto finito de possíveis soluções com ênfase na combinação, arranjo e seleção de elementos discretos, o problema é referido como de otimização combinatória (BLUM; ROLI, 2003). O número de combinações, arranjos ou permutações é expresso em termo da função fatorial, que cresce rapidamente com os seus valores de entrada. Dessa forma, ainda que o espaço de soluções dos problemas seja finito, o seu tamanho torna a busca sequencial do melhor resultado possível inviável. Áreas como logística, projeto de redes, escalonamento de tarefas, corte e empacotamento, entre outras áreas relevantes do ponto de vista econômico, podem ser modelados como problemas de otimização combinatória (GOLDBARG; LUNA, 2005).

A investigação da classe de complexidade dos problemas de otimização combinatória usualmente é realizada a partir da formulação de problemas teóricos, os quais geralmente apresentam formulações simplificadas e, consequentemente, mais

acessíveis para tratamento matemático. Problemas teóricos são abstrações que permitem aos pesquisadores analisar suas propriedades fundamentais, desenvolver algoritmos exatos ou heurísticos e estabelecer os seus limites de complexidade (GAREY; JOHNSON, 1979). O estudo de um problema real, realizado a partir de problemas teóricos, é feito por meio de um processo chamado *redução* (GAREY; JOHNSON, 1979; ARORA; BARAK, 2009; CORMEN et al., 2002).

A ideia de uma redução é que um problema pode ser resolvido por meio da solução de um outro problema. Formalmente, se temos um problema P_1 com entrada E_1 e saída S_1 e um P_2 com entrada E_2 e saída S_2, uma redução é um par de algoritmos (a_E, a_S) que transformam, respectivamente, qualquer instância de E_1 numa instância de E_2 e qualquer instância de S_2 em uma instância de S_1 e forma que para qualquer instância x de E_1 tem-se que $P_1(x) = s_S(P_2(a_E(x)))$. Ou seja, resolver o problema diretamente para a entrada x por meio de algum algoritmo que resolva o problema P_1 deve dar o mesmo resultado que transformar a entrada x na entrada de P_2, resolver o problema pra essa entrada com algum algoritmo que resolva P_2 e transformando a saída encontrada na saída do problema P_1 com o algoritmo a_S. A Figura 4 mostra esse processo graficamente.

Figura 4 – Representação visual da redução de um problema P_1 a um problema P_2



A existência de uma redução de um problema P_1 para um problema P_2 pode ser usada para inferir a dificuldade computacional de resolver um problema, visto que se a complexidade de resolver o problema P_1 via P_2 (ou seja, o caminho $x - a_E - P_2 - a_S - y$) tiver uma complexidade menor do que o caminho direto (ou seja o caminho $x - P_1 - y$) significa que temos um algoritmo mais eficiente do que o algoritmo direto, o que faz pouco sentido. Dessa forma, o caminho da redução estabelece um limite superior para a complexidade dos algoritmos que são capazes de resolver P_1. Correspondentemente, se conhecemos a complexidade mínima do problema P_1 então a solução via P_2 não pode ser

melhor do que a solução direta de P_1. Ou seja, a complexidade de P_1 é um limite mínimo para a complexidade de P_2.

O Problema do Caixeiro Viajante é um clássico problema de otimização combinatória (CORMEN et al., 2002), que representa a situação hipotética de um viajante que deve visitar um conjunto de cidades exatamente uma vez e retornar à cidade de origem, buscando minimizar a distância total percorrida. A dificuldade reside na necessidade de explorar todas as possíveis permutações de rotas para determinar a mais curta. Apesar da formulação simples, encontrar uma solução eficiente para o Caixeiro Viajante torna-se cada vez mais desafiador à medida que o número de cidades aumenta (CUNHA; BONASSER; ABRAHÃO, 2002), pois o número de caminhos possíveis é o fatorial do número de cidades. O Problema da Mochila é outro desafio clássico de otimização combinatória (CORMEN et al., 2002). Nesse problema, um agente deve decidir quais itens incluir em sua mochila, considerando seus pesos e valores, de modo a maximizar o valor total, sem exceder a capacidade de peso da mochila. A complexidade surge da necessidade de explorar diferentes combinações de itens para encontrar a configuração ideal que maximize o valor total. Apesar da aparente simplicidade da formulação, a busca pela solução ótima se torna mais desafiadora à medida que a quantidade de itens a serem considerados aumenta (CORMEN et al., 2002; CUNHA; BONASSER; ABRAHÃO, 2002), visto que o número de combinações aumenta exponencialmente com o número de itens.

Os problemas do Caixeiro Viajante e da Mochila são problemas **NP**-difíceis[1] e não se conhece algoritmo polinomial para resolvê-los. Para provar que outro problema é **NP**-difícil, basta pegar um problema com essa característica e fazer uma redução polinomial (ou seja, uma redução em que os algoritmos a_E e a_S são polinomiais) para o problema de interesse. Se isso for possível, então a complexidade mínima do problema que estamos tentando resolver é pelo menos a complexidade de um problema **NP**-difícil. Encontrar uma redução, ainda que ela exista, não é necessariamente simples. Dessa forma, mesmo que às vezes se suspeite que o problema seja intratável, nem sempre é possível construir uma prova formal de que de fato ele é.

Existem duas abordagens distintas para resolver problemas computacionais, cada uma com suas características e aplicações específicas: algoritmos exatos ou algoritmos aproximados (BLUM; ROLI, 2003). Algoritmos exatos são métodos de solução que garantem encontrar a solução ótima para um problema, ou seja a solução encontrada é

[1] Um problema é dito **NP**-difícil, ou **NP**-hard, quando o seu problema de decisão equivalente é **NP**-completo.

a melhor possível de acordo com a definição do problema ou ainda, seu ótimo global. Já algoritmos aproximados buscam soluções que estão próximas da ótima, ou seja, ótimos locais. Ainda que não garantam a solução ideal, podem fornecer soluções aceitáveis em um tempo aceitável, mesmo para problemas intratáveis computacionalmente (BLUM; ROLI, 2003).

Determinar a eficiência e analisar a complexidade de algoritmos envolve avaliar como o tempo de execução (ou o uso de recursos em geral) crescem em relação ao tamanho da entrada do problema. A disciplina de Complexidade Computacional busca estabelecer uma classificação para problemas computacionais, especialmente aqueles relacionados à otimização combinatória. Ela se concentra em compreender e categorizar a dificuldade intrínseca desses problemas, desenvolvendo medidas de complexidade que ajudam a classificar a eficiência dos algoritmos utilizados para resolvê-los (AGUIAR, 1998).

A complexidade de um problema, na teoria da complexidade computacional, é muitas vezes associada à complexidade do algoritmo mais eficiente conhecido para resolvê-lo. A complexidade de um problema refere-se à quantidade de recursos computacionais necessários para resolver o problema em função do tamanho da entrada (BROOKSHEAR, 2013).

Os problemas na teoria da complexidade computacional são frequentemente agrupados em classes de complexidade para refletir o grau de dificuldade computacional associado a cada problema. As classes de complexidade são categorias que incluem problemas com características computacionais semelhantes, e essas categorias são definidas com base na eficiência dos algoritmos capazes de resolvê-los (SIPSER, 1996). Entre as classes de complexidade existem a classe **P** (polinomial determinístico), que inclui problemas para os quais existe um algoritmo eficiente, capaz de resolver o problema em tempo polinomial em relação ao tamanho da entrada. Ou seja, o tempo de execução do algoritmo cresce de maneira polinomial à medida que o tamanho da entrada aumenta (CORMEN et al., 2002; AGUIAR, 1998).

A classe **NP** (polinomial não-determinístico) inclui problemas para os quais, se uma solução candidata for dada, ela pode ser verificada em tempo polinomial (AGUIAR, 1998). Um exemplo de problema da classe **NP** é o problema de decisão equivalente do Caixeiro Viajante, onde se verifica em tempo polinomial se uma determinada rota passando por todas as cidades tem um comprimento total inferior a um certo valor dado. Contudo, não se conhece algoritmo eficiente para encontrar uma rota mais curta que

um valor específico. A classe **NP**-completa inclui problemas em **NP** que são, de certa forma, "tão difíceis quanto os mais difíceis" em NP (GAREY; JOHNSON, 1979). Se um problema **NP**-completo possuir uma solução eficiente, ou seja, um algoritmo que execute em tempo polinomial, então todos os problemas em **NP** também teriam uma solução eficiente (a classe **P** seria igual à classe **NP**). A classe **NP**-difícil inclui problemas de otimização cujo problema de decisão equivalente seja **NP**-completo. Ou seja, que são, no mínimo, tão difíceis quanto os mais difíceis em **NP**, mas não necessariamente pertencem à classe **NP**(AGUIAR, 1998). A distinção fundamental entre **NP**-difícil e **NP**-completo reside no fato de que um problema **NP**-completo pode ser verificado em tempo polinomial por uma Máquina de Turing determinística, ao contrário dos problemas **NP**-difíceis (GAREY; JOHNSON, 1979).

Problemas **NP**-completos ou **NP**-difíceis não possuem nenhum algoritmo exato conhecido que possa encontrar a solução ótima em tempo polinomial para todas suas instâncias (ARORA; BARAK, 2009). Neste caso, quando os algoritmos exatos conhecidos demonstram ineficiência devido à sua complexidade exponencial ou superior, torna-se apropriado recorrer ao emprego de algoritmos aproximados que façam uso de heurísticas. Heurísticas são estratégias práticas baseadas em regras prontas, experiência passada ou intuição para simplificar a resolução de problemas complexos (YANG, 2010).

Algoritmos aproximados ou heurísticos são comumente aplicados em problemas para os quais algoritmos polinomiais não são conhecidos ou quando a busca pela solução ótima pode ser impraticável para instâncias de tamanho significativo (YANG, 2010). No caso dos problemas associados ao melhoramento animal, mesmo os algoritmos polinomiais existentes possuem limitações na quantidade de elementos que podem tratar. A relação de coancestralidade, por exemplo, é uma relação binária no número de animais. Mesmo considerando somente os animais selecionados para reprodução, temos uma relação com o tamanho do quadrado do número de animais. Algoritmos quadráticos são polinomiais, mas com entradas grandes são bem mais ineficientes que algoritmos lineares. Como rebanhos com alguns milhares de animais não são incomuns, não é suficiente um algoritmo ser polinomial para ser eficiente: o grau do polinômio também deve ser baixo.

Encontrar boas heurísticas para resolver um problema diretamente pode não ser uma tarefa simples, como também não é verificar a optimalidade (ou não) da heurística criada. No caso dos problemas relacionados à coancestralidade, o algoritmo para acasalamento compensatório proposto em (FERREIRA, 2021) pode ser um bom ponto de partida, visto que garante a solução ótima relacionada ao índice de seleção esperado

da prole e esse caminho será investigado ao longo do restante deste trabalho.

Outra abordagem no trato de problemas de otimização são as meta-heurísticas. Uma meta-heurística é um arcabouço conceitual para encontrar sucessivas soluções melhores que as anteriores, partindo de um ponto aleatório no espaço de soluções do problema. No contexto deste trabalho, que se concentra na otimização da produção animal, a aplicação de técnicas de otimização combinatória se torna viável devido à natureza complexa do problema em questão. A seleção cuidadosa de acasalamentos para minimizar a coancestralidade em rebanhos, enquanto atende-se a diversas restrições, implica em uma busca por soluções eficientes em um espaço de possíveis combinações. Nesse sentido, a complexidade computacional do problema se destaca, sugerindo que encontrar a configuração ótima pode ser um desafio, e há indicações, conforme discutido por Rosso, Rosa e Ferreira (2023), de que o problema talvez possa ser classificado como **NP**-difícil.

Diante dessa perspectiva, a escolha de recorrer a meta-heurísticas neste trabalho é fundamentada na necessidade de lidar eficientemente com problemas nos quais os métodos exatos se tornam impraticáveis para instâncias de tamanho significativo (ARORA; BARAK, 2009). A aplicação de meta-heurísticas é respaldada pela capacidade dessas técnicas em encontrar soluções próximas da ótima (YANG, 2010). A flexibilidade dessas estratégias tornam-se relevantes quando a busca pela solução ideal é computacionalmente desafiadora, conforme sugerido pelas evidências de NP-completude em estudos anteriores. Portanto, a decisão de incorporar meta-heurísticas na resolução do problema de otimização na produção animal é guiada pela necessidade de contornar a complexidade computacional inerente, buscando soluções eficazes que atendam às demandas práticas do setor.

O estudo de meta-heurísticas apropriadas para uso nos problemas de melhoramento foi conduzido com apoio da literatura (HERTZ; WIDMER, 2003; BLUM; ROLI, 2003) e está descrito na Seção 3.3.

4.3 Otimização por Colônia de Formigas

A Otimização por Colônia de Formigas (ACO) é uma metaheurística desenvolvida a partir da observação do comportamento natural de formigas durante a busca por alimento. Proposta por Colorni et al. (1991), essa abordagem representa um caso clássico de inteligência coletiva aplicada à resolução de problemas complexos de natureza

combinatória. A ideia central é simular o modo como formigas reais interagem indiretamente, marcando caminhos por onde passaram com feromônio, substância química que influencia o comportamento de outras formigas.

Na prática, quando uma formiga encontra uma fonte de alimento, ela retorna ao ninho liberando uma trilha de feromônio ao longo do caminho. Com o tempo, os caminhos mais eficientes acumulam maior concentração dessa substância, uma vez que são mais frequentemente percorridos. Esse reforço positivo tende a direcionar as formigas subsequentes para as melhores rotas, geralmente aquelas com menor distância ou custo (GRASSE, 1959; DENEUBOURG et al., 1990).

Inspirado por esse fenômeno natural, a ACO foi desenvolvida com o objetivo de abordar problemas de otimização em espaços de busca discretos, onde o desafio é encontrar a melhor combinação possível entre diversas opções. A seguir, são apresentados os componentes fundamentais desse algoritmo, com base em Dorigo, Birattari e Stutzle (2006).

O funcionamento da ACO exige a definição de parâmetros iniciais, dentre os quais se destacam:

- A quantidade de agentes (formigas) empregados no processo (m);
- A taxa de dissipação do feromônio ao longo do tempo (ρ);
- O valor inicial atribuído às trilhas de feromônio (τ_0).

A evaporação do feromônio impede o acúmulo excessivo em caminhos explorados precocemente, favorecendo a diversidade de soluções durante a busca. O valor inicial da trilha, por sua vez, influencia o nível de exploração do espaço de busca nas primeiras iterações.

Durante cada iteração, as formigas constroem soluções movendo-se através de um grafo, onde os nodos representam partes da solução. A escolha do próximo nodo é feita de forma probabilística, levando em conta a quantidade de feromônio (τ_{ij}) presente na aresta (i, j) e uma heurística local (η_{ij}), que mede a atratividade daquela escolha. A probabilidade de uma formiga k selecionar o nodo j a partir do nodo i é dada pela equação:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \tag{12}$$

onde:

- τ_{ij} é a quantidade de feromônio na aresta (i, j);

- η_{ij} é a visibilidade ou heurística local, normalmente representada pelo inverso da distância (ou custo) da aresta (i, j);
- α e β controlam a influência do feromônio e da heurística, respectivamente;
- N_i é o conjunto de nodos adjacentes ao nodo i.

Esse processo estabelece um equilíbrio entre a exploração de novas possibilidades (favorecida por maiores valores de η_{ij}) e a intensificação sobre caminhos já identificados como promissores (por meio de maiores valores de τ_{ij}).

Após cada ciclo de construção de soluções, o algoritmo realiza a atualização das trilhas de feromônio, composta por duas etapas principais:

1. **Evaporação:** A concentração de feromônio em cada conexão é reduzida proporcionalmente à taxa ρ, simulando o desgaste natural da trilha com o passar do tempo:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{13}$$

2. **Deposição de Feromônio:** As formigas depositam feromônio nas arestas das soluções que construíram. A quantidade de feromônio depositada por cada formiga é proporcional à qualidade da solução, conforme a equação:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{14}$$

onde $\Delta\tau_{ij}^{k}$, o feromônio depositado pela formiga k na aresta (i, j), é calculado por:

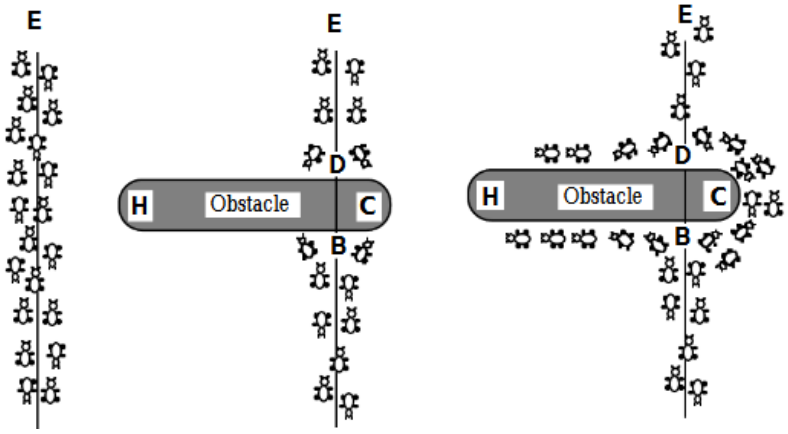
$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{se a formiga } k \text{ usou a aresta } (i, j) \\ 0 & \text{caso contrário} \end{cases} \tag{15}$$

onde Q é uma constante associada ao depósito de feromônio e L_k representa o custo da solução encontrada pela formiga k.

Existem várias abordagens para atualizar as trilhas de feromônio, que vão desde reforçar globalmente as trilhas das melhores soluções até atualizações locais, que aumentam a diversidade durante o processo de busca. Algumas variantes populares da ACO são:

- **Ant System (AS):** A primeira versão da ACO (DORIGO; MANIEZZO; COLORNI, 1996).

Figura 5 – Exemplo com formigas reais



Fonte: (DORIGO; MANIEZZO; COLORNI, 1996)

- **Ant Colony System (ACS):** Introduz a atualização local do feromônio, priorizando a exploração de novos caminhos (DORIGO; GAMBARDELLA, 1997).
- **MAX-MIN Ant System (MMAS):** Limita os valores de feromônio para evitar convergência prematura (STÜTZLE; HOOS, 2000).

Essas variantes procuram otimizar o equilíbrio entre a exploração de novas soluções e a intensificação sobre soluções promissoras já encontradas, tornando a ACO uma abordagem versátil para diferentes problemas de otimização. As principais diferenças entre elas estão destacadas na Tabela 11.

Tabela 11 – Comparação entre variações de algoritmos de colônia de formigas

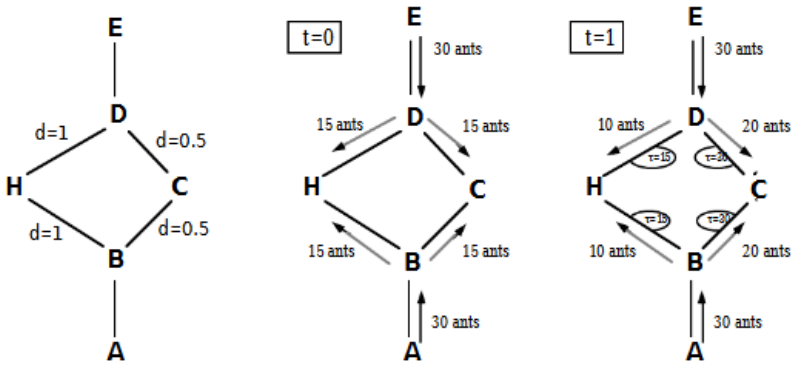
Característica	**AS**	**ACS**	**MMAS**
Todas as formigas atualizam feromônio	✓	×	×
Apenas melhor formiga atualiza	×	✓ (melhor parcial)	✓ (melhor global ou iteração)
Atualização local durante construção	×	✓	×
Evaporação global	✓	✓	✓
Limites explícitos τ_{min} e τ_{max}	×	×	✓

Fonte: Autor (2025)

As imagens apresentadas em (DORIGO; MANIEZZO; COLORNI, 1996) ilustram o funcionamento do algoritmo de colônia de formigas, contribuindo para uma melhor compreensão de seus princípios fundamentais.

Na Figura 5, observa-se que, inicialmente, as formigas seguem uma trilha direta entre os pontos A e E. Com a introdução de um obstáculo, elas passam a contornar

Figura 6 – Exemplo com formigas artificiais



Fonte: (DORIGO; MANIEZZO; COLORNI, 1996)

o impedimento, distribuindo-se aleatoriamente entre dois caminhos possíveis. Com o tempo, o percurso mais curto tende a ser preferido, uma vez que acumula mais feromônio devido ao retorno mais rápido das formigas, fortalecendo a trilha e atraindo mais indivíduos.

Na Figura 6, é possível visualizar a lógica aplicada às formigas artificiais. No instante inicial, todas as arestas do grafo apresentam chances iguais de serem escolhidas, pois não há feromônio acumulado. Com o passar do tempo e o acúmulo de interações, as arestas mais curtas começam a receber mais feromônio, tornando-se, em média, as mais escolhidas pelas formigas nas iterações subsequentes.

Dentre as diferentes variações dos algoritmos de colônia de formigas, optou-se pela utilização do Max-Min Ant System (MMAS), devido às suas características que proporcionam maior controle sobre o equilíbrio entre exploração e intensificação da busca. O MMAS impõe limites explícitos ao valor do feromônio nas arestas, restringindo-o entre $\tau_{\min}$ e $\tau_{\max}$, o que evita a convergência prematura e impede que todas as formigas passem a seguir rapidamente o mesmo caminho, mantendo a diversidade das soluções [(STÜTZLE; HOOS, 2000)].

Segundo (STÜTZLE; HOOS, 2000), o algoritmo *MAX–MIN Ant System (MMAS)* utiliza estes limites superior e inferior para os valores de feromônio, definidos por:

$$\tau_{\max} = \frac{1}{(1-\rho)\cdot f(s^*)} \tag{16}$$

$$\tau_{\min} = \frac{\tau_{\max}\cdot\left(1-\sqrt[n]{p_{\text{best}}}\right)}{(\text{avg}-1)\cdot\sqrt[n]{p_{\text{best}}}} \tag{17}$$

Esses limites controlam a intensidade da exploração e previnem a convergência

prematura do algoritmo. O parâmetro p_{best} representa a probabilidade desejada de reconstrução da melhor solução encontrada após a convergência.

A equação de $\tau_{\max}$ determina o valor máximo de feromônio que pode ser depositado em um componente de solução, sendo diretamente influenciado pela taxa de evaporação e pela qualidade da melhor solução s^* encontrada até o momento. Já a equação de $\tau_{\min}$ é derivada com o objetivo de garantir que, mesmo após a convergência do algoritmo, a melhor solução ainda possa ser reconstruída com uma probabilidade controlada p_{best}.

Essa derivação assume que cada solução é composta por n decisões consecutivas e que a probabilidade de fazer a escolha correta em cada ponto de decisão deve ser $p_{\text{dec}} = \sqrt[n]{p_{\text{best}}}$. Considerando que, em média, cada decisão oferece avg possibilidades, a probabilidade de selecionar a melhor escolha é:

$$p_{\text{dec}} = \frac{\tau_{\max}}{\tau_{\max} + (\text{avg} - 1) \cdot \tau_{\min}}$$

Resolvendo essa equação para $\tau_{\min}$, obtém-se a expressão usada acima. Isso garante que mesmo os componentes com menos feromônio ainda tenham chance não nula de serem escolhidos, permitindo a exploração local ao redor da melhor solução conhecida.

Onde:

- $\tau_{\max}$ – valor máximo do feromônio;
- $\tau_{\min}$ – valor mínimo do feromônio;
- ρ – taxa de evaporação do feromônio ($0 < \rho < 1$);
- $f(s^*)$ – valor da função objetivo da melhor solução conhecida até o momento;
- p_{best} – probabilidade desejada de reconstruir a melhor solução após a convergência;
- n – número total de decisões consecutivas necessárias para construir uma solução completa;
- avg – número médio de escolhas disponíveis por ponto de decisão.

4.4 Banco de dados baseado em grafos

Grafos são estruturas de dados usadas para representar relações entre elementos. Grafos são compostos por nodos (ou vértices) que representam entidades ou elementos e arestas (ou arcos) que representam as conexões existentes entre as entidades. As

conexões entre as entidades de um grafo são, às vezes, chamadas de relacionamentos, em similaridade aos diagramas de entidades-relacionamentos (ER) usados para modelagem de dados em bancos de dados relacionais.

Grafos, do ponto de vista matemático, são formalmente definidos como um par ordenado $G = (V, E \subseteq V \times V)$, em que V é um conjunto de vértices e E é uma relação binária sobre o conjunto V, denominada conjunto de arestas (DIESTEL, 2005). Quando há a necessidade de termos arestas paralelas entre os mesmos dois nodos, a definição de grafo pode ser alterada para uma tupla $G = (V, E, src : E \to V, tar : E \to V)$, em que V é o conjunto de nodos, E é o conjunto de arestas e src e tar são as funções que determinam, respectivamente, os nodos de origem e destino de cada aresta. A definição formal de grafo pode ser enriquecida para diferenciar tipos diferentes de nodos ou arestas. Um grafo rotulado é uma tupla $G = (V, E, L_V, L_E, src, tar : E \to V, r_V : V \to L_V, r_E : E \to L_E)$ em que $G = (V, E, src, tar : E \to V)$ é um grafo, L_V é o conjunto de rótulos de vértices, L_E é o conjunto de rótulos de arestas e as funções r_V e r_E atribuem nodos e arestas aos seus respectivos rótulos (FERREIRA; RIBEIRO, 2006).

Grafos podem ser representados graficamente por círculos que representam os nodos (vértices) e por setas que os conectam (arestas). Formas ou cores podem ser usadas para rotular nodos ou arestas, tornando o diagrama mais expressivo, e formalizado por meio de grafos rotulados. A representação visual ajuda no entendimento do domínio que o grafo representa e pode ser entendida por qualquer pessoa, mesmo as que não têm formação em Matemática ou Computação. A maior parte dos diagramas usados para descrever sistemas computacionais é baseada, ou pode ser representada, como grafos (eventualmente rotulados). Exemplos são diagramas de circuitos lógicos, diagramas UML, modelos ER, entre outros.

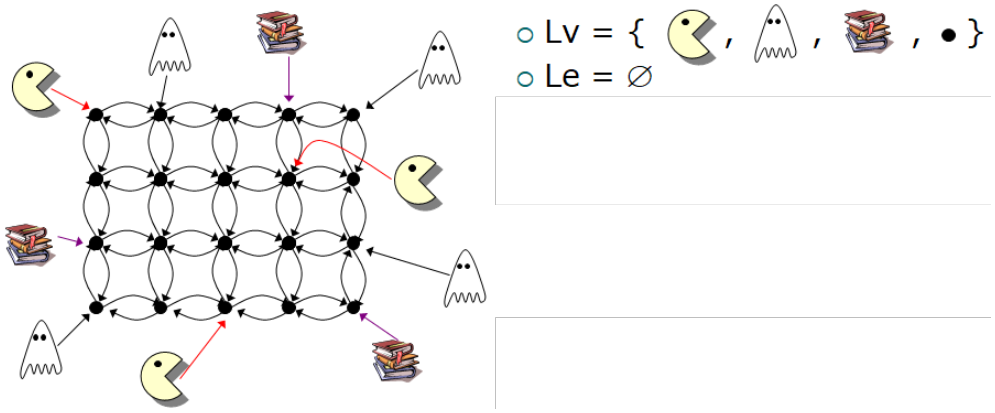
A Figura 7 exemplifica como um jogo PACMAN pode ser formalizado como um grafo rotulado. Os nodos podem ser do tipo tabuleiro, PAC MAN, Fantasma ou pilha de livros. A representação visual do tipo de cada nodo é o seu rótulo. Nesse caso, os arcos não têm rótulos.

O fato de grafos expressarem relações entre elementos[2] serviu de motivação para o desenvolvimento de bancos de dados baseados em grafos, com os métodos usuais *Create*, *Read*, *Update*, e *Delete* (usualmente denotadas pela sigla CRUD).

Sistemas de gerenciamento de banco de dados relacionais requerem a criação de conexões entre entidades usando chaves estrangeiras ou outros mecanismos, que não

[2]Relações entre múltiplos elementos também podem ser representados por hipergrafos.

Figura 7 – Exemplo do jogo PAC MAN modelado como um grafo



Fonte: Ferreira (2006)

favorecem a busca por relacionamentos em si, mas apenas por elementos individuais (tuplas) no banco de dados. Ao reunir os conceitos fundamentais de entidades e relacionamentos em estruturas interconectadas, os bancos de dados orientados a grafos permitem construir modelos que mapeiam de forma mais próxima o domínio do problema em questão. Os modelos resultantes desse método costumam ser mais simples e expressivos em comparação aos modelos produzidos por bancos de dados relacionais tradicionais e sistemas NoSQL (Not Only SQL) (ROBINSON; WEBBER; EIFREM, 2015).

Os sistemas de gerenciamento de bancos de dados de grafos além de seguirem os princípios do CRUD empregam também o conceito de *index-free adjacency*. Isso significa que os nodos mantêm referências diretas aos nodos adjacentes, tornando as consultas independentes do tamanho total do grafo e proporcionais apenas ao tamanho do grafo pesquisado, o que contribui para o desempenho das consultas (RODRIGUEZ; NEUBAUER, 2012). Os nodos do banco são indexados via um sistema de *hashing*, que também produz uma busca eficiente por nodos no grafo (ZHANG et al., 2018).

Entre os atuais bancos de dados baseados em grafos, destaca-se o *Neo4j*[3] (DEMIR; ERKAN, 2017). Abaixo estão alguns detalhes sobre esse banco de dados:

- O *Neo4j*, também conhecido como *Neo Technology*, é um banco de dados transacional baseado em grafos com armazenamento em disco.
- O *Neo4j* oferece suporte a várias linguagens de programação, incluindo Python, R e Java, para operações relacionadas a grafos que podem ser implementadas e compiladas direto no código dos programas.

[3] <https://neo4j.com/>

- O *Neo4j* é altamente escalável, suportando bilhões de nodos e relacionamentos em uma rede.
- O banco faz o gerenciamento das operações de modificação de dados em transações.
- O sistema permite a inclusão de rótulos e propriedades tanto nos nodos como nas arestas do grafo. Isso permite a criação de atributos nas entidades e também nos relacionamentos.
- É otimizado para estruturas de grafos em vez de tabelas e utiliza o modelo de grafos de propriedade para representar dados.
- Oferece suporte completo para transações ACID (*atomicidade*, *consistência*, *isolamento* e *durabilidade*), com registros de transação em memória e gerenciamento de bloqueio.
- O banco possui uma versão gratuita e uma versão paga. A versão gratuita não suporta garantia de unicidade dos nodos e arestas nas operações CREATE, que precisam ser garantida via software. Há uma perda de desempenho em relação à versão paga.

Essas características tornam o *Neo4j* uma escolha proeminente para aqueles que buscam os benefícios dos bancos de dados baseado em grafos para seus projetos e aplicações (KALIYAR, 2015). Atualmente, o banco é usado em diversas plataformas de software que trabalham com dados fortemente conectados e/ou sistemas de recomendação. Exemplos são eBay, JPMorgan, UBS, entre outros[4].

[4]<https://neo4j.com/who-uses-neo4j/>

5 MODELO DE OTIMIZAÇÃO

5.1 Armazenamento de dados

O sistema de gerenciamento de banco de dados *Neo4j* foi utilizado para implementar os dados da genealogia do rebanho. Embora existam outras tecnologias de banco de dados orientados a grafos ou multimodais no mercado, como *JanusGraph*[1], *TigerGraph*[2] e *ArangoDB*[3], a escolha pelo *Neo4j* se deu pela sua maturidade, ampla documentação e suporte nativo a algoritmos de grafos. Inicialmente, foi empregada a linguagem de consulta *Cypher*, própria do *Neo4j*, seguida pela utilização da linguagem de programação *Python* para operações mais complexas. As consultas *Cypher* utilizadas estão disponíveis no Apêndice C. A base de dados foi criada a partir dos dados fornecidos pela EMBRAPA Pecuária Sul, conforme discutido na Seção 2.4.

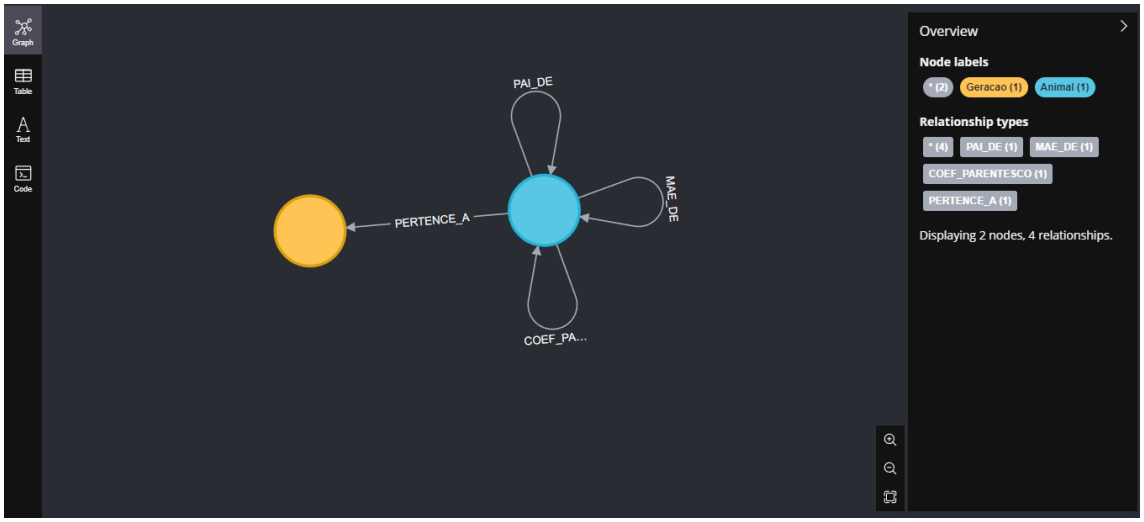
A disposição dos nodos e as conexões (limitadas ao problema em questão) do banco podem ser visualizadas na Figura 8. Os elementos representados no grafo consistem nos animais, nos valores das gerações (Definição 7) e as conexões entre eles são indicadas pelas arestas, representando as relações de coancestralidade (com rótulos correspondentes aos valores) e a geração dos animais. O modelo apresentado opera como um grafo-tipo (HECKEL et al., 1996), categorizando os nodos e as relações associadas aos valores incluídos no banco de dados.

Na Figura 8 é possível visualizar as entidades e os relacionamentos existentes no modelo de banco de dados que foi desenvolvido. Dois tipos de nodos são identificados: “Animal” (apresentado na figura em azul) e “Geracao” (apresentado na figura em amarelo). Os nodos do grafo podem ser enriquecidos com rótulos, assim como as arestas, que expressam as relações no grafo. O nodo “Animal” possui as propriedades ID, IECC (valor do índice econômico) e Sexo, equivalentes a atributos no modelo relacional. O nodo “Geracao” possui a única propriedade “Numero”, que define o número da geração à qual o animal pertence. Os nodos “Animal” estabelecem relações (arestas) entre si, indicando paternidade, maternidade e coeficiente de parentesco.

A Figura 9 ilustra uma seleção das relações dentro do banco de dados Neo4j, exemplificando a organização das principais conexões. Os nodos identificados como “Animal” são interconectados por meio das relações “PAI_DE” e “MAE_DE”,

[1] <https://janusgraph.org/>
[2] <https://www.tigergraph.com/>
[3] <https://arango.ai/>

Figura 8 – Modelo de dados e relações no banco Neo4j



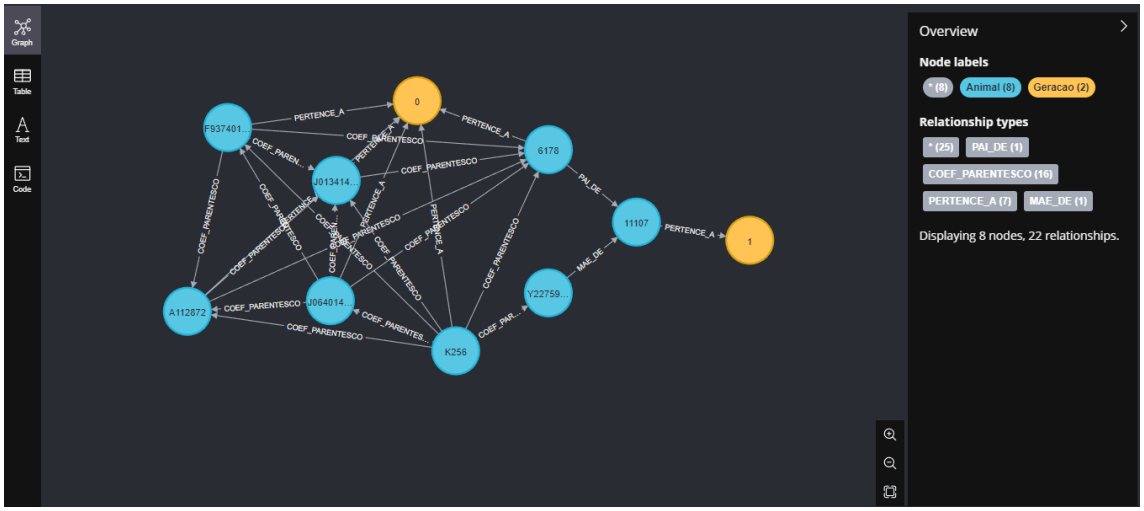
Fonte: Autor, (2025)

que denotam as relações de parentesco entre os animais. Além disso, a relação "PERTENCE_A" associa os nodos dos animais às suas respetivas gerações. Adicionalmente, a relação "COEF_PARENTESCO" é usada para registrar informações sobre os coeficientes de coancestralidade calculados entre os animais. Essa representação visual oferece uma visão da estrutura do banco de dados e esclarece como as informações genealógicas são armazenadas e recuperadas.

A Figura 10 exibe o cálculo do relacionamento de parentesco entre os animais no banco de dados, onde cada nodo do tipo "Animal" está interligado com todos os outros nodos por meio da relação COEF_PARENTESCO. A representação visual enfatiza a intricada natureza das conexões entre os animais e como as informações genealógicas foram organizadas para viabilizar o cálculo dos coeficientes.

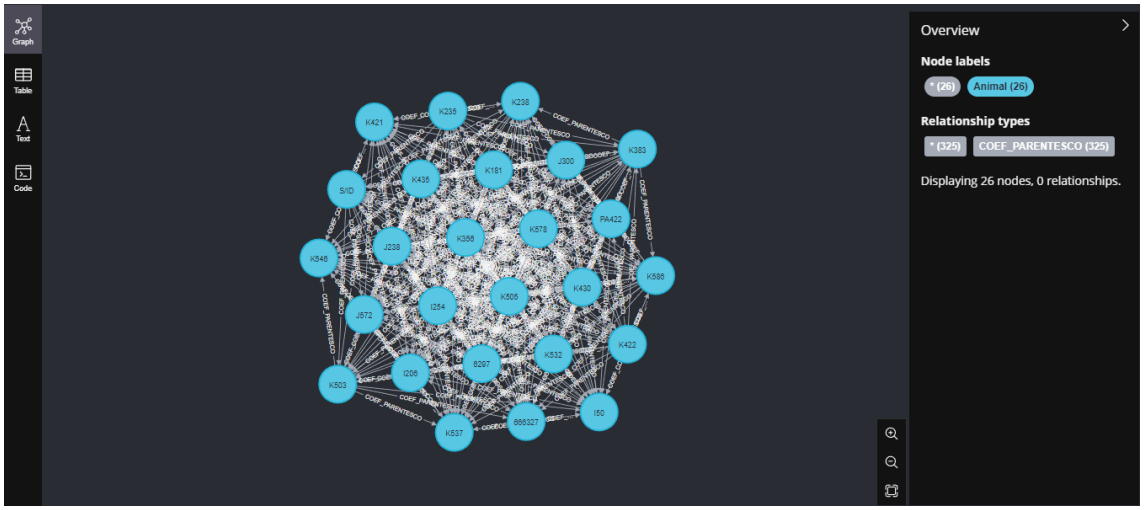
A representação de gerações por meio de nodos tem como finalidade otimizar a busca por animais de uma determinada geração. Essa abordagem é motivada pelo fato de que a implementação de bancos de dados baseados em grafos oferece uma eficiência superior na pesquisa de conexões quando comparada à busca por atributos de nodos (NEO4J, ano). A flexibilidade para adicionar ou remover tipos de nodos e arestas possibilita a inclusão ou reestruturação de novas informações sem afetar os dados previamente armazenados no banco. Esta característica representa a principal vantagem de um banco de dados baseado em grafos em relação a um banco de dados relacional. A incorporação de novos nodos pode ser realizada conforme a necessidade de construir consultas específicas, sem necessidade de alteração dos dados já inseridos.

Figura 9 – Amostra de relações existentes no banco Neo4J



Fonte: Autor, (2025)

Figura 10 – Relacionamento de parentesco entre os animais



Fonte: Autor, (2025)

Além disso, a escolha do Neo4j alinha-se diretamente com a natureza do algoritmo de Otimização por Colônia de Formigas (ACO). Como o ACO é fundamentado na exploração de caminhos em grafos, a persistência dos dados em uma estrutura de grafo nativa elimina a distorção de impedância entre o modelo de dados e o algoritmo. As relações `COEF_PARENTESCO` atuam efetivamente como as arestas ponderadas que as formigas virtuais devem avaliar, permitindo que a heurística de coancestralidade seja recuperada em tempo constante ($O(1)$) durante a construção da solução, o que é crucial para o desempenho dado o volume quadrático de relações.

5.2 Cálculo dos coeficientes de coancestralidade

Com o objetivo de organizar os animais de forma a facilitar o subsequente cálculo do coeficiente de parentesco, torna-se crucial que as relações de coancestralidade entre os pais dos indivíduos já estejam calculadas antes das relações de seus descendentes. Para atender a essa necessidade, foi desenvolvido um algoritmo que estabelece um sistema de "gerações", cujo conceito é formalizado na Definição 7 (pág. 65). A determinação da geração de cada animal é realizada por meio de um algoritmo que implementa a estratégia de Programação Dinâmica, assumindo que a geração dos pais já foi calculada. Os animais da geração 0 têm seus valores inicializados diretamente para garantir a base da recursão, identificando quais animais não possuem nenhum dos seus progenitores na base.

Para ilustrar, a consulta abaixo, escrita na linguagem *Cypher*, verifica quais animais não têm progenitores registrados e os associa à geração 0:

```
MATCH (a:Animal)
WHERE NOT (a)<-[:PAI_DE]-() AND NOT (a)<-[:MAE_DE]-()
MERGE (g:Geracao {Numero: 0})
CREATE (a)-[:PERTENCE_A]->(g);
```

A consulta busca todos os nodos do tipo animal (MATCH (a:Animal)) em que não existam arcos do tipo PAI_DE ou MAE_DE de outro animal qualquer (denotado pelos parênteses que abrem e fecham sem argumentos) para ele. Nesse caso, é criado um arco do animal para o nodo que representa a geração 0. O comando MERGE é usado em vez do comando CREATE para que não haja duplicação de nodos, visto que a versão gratuita do *Neo4j* não permite definir elementos que sejam únicos. O arco entre o animal e a geração 0 então é criado (com o comando CREATE) na base, rotulado com o seu tipo PERTENCE_A.

Figura 11 – Algoritmo dinâmico para o cálculo das gerações

CALCULOGERAÇÃO($a \in A, A, \{s,d\} : A \to A, g : A \to \mathbb{N}$) retorna k
(1) se ($g(a) = null$)
(2) então
(3) se ($g(s(a)) = null$)
(4) então $g_p \leftarrow$ CalculoGeração($s(a), A, s, d, g$)
(5) senão $g_p \leftarrow g(s(a))$
(6) se ($g(d(a)) = null$)
(7) então $g_m \leftarrow$ CalculoGeração($d(a), A, s, d, g$)
(8) senão $g_m \leftarrow g(s(a))$
(9) criar aresta $g(a) = \max\{g_p, g_m\} + 1$
(10) retornar $g(a)$

Fonte: Rosso, Rosa e Ferreira (2023)

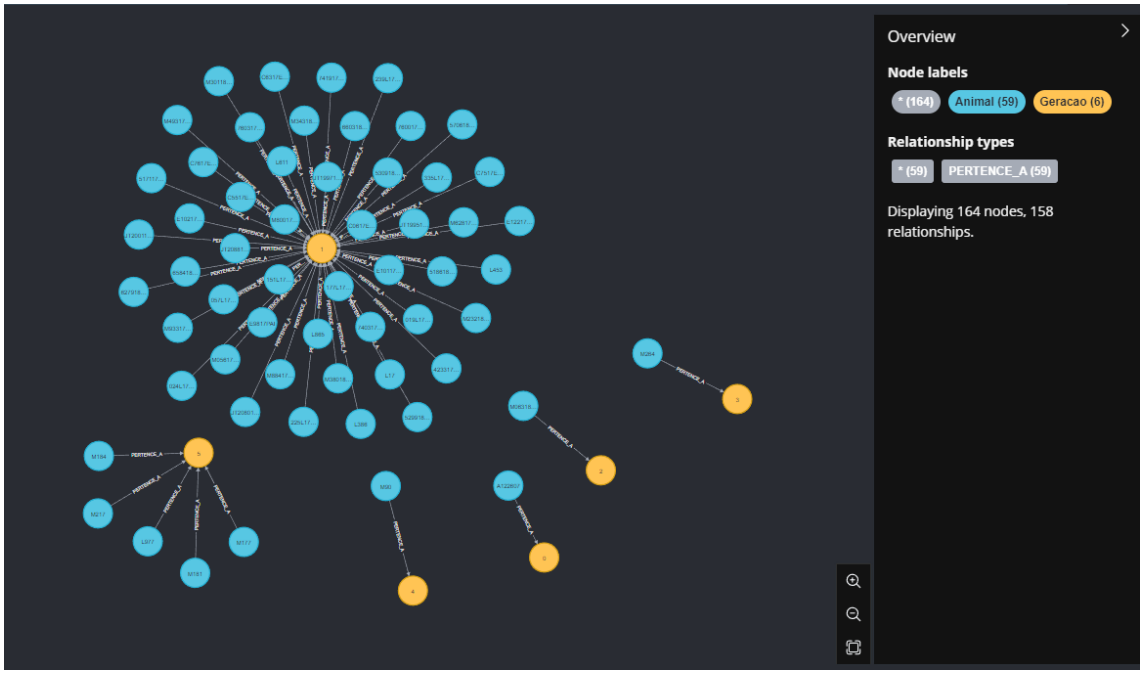
O algoritmo recursivo emprega uma estratégia dinâmica, calculando o valor geracional de cada animal apenas uma vez. A implementação, em *Python*, com chamadas para busca e inserção diretas no banco, está disponível no Apêndice D. Inicialmente, os animais da geração 0 são inicializados. Posteriormente, o algoritmo é aplicado aos demais animais do banco, sem considerar uma ordem específica, pois calcula as gerações dos ancestrais, se não estiverem disponíveis. O pseudo-código do trecho do algoritmo chamado para cada $a \in A$ é apresentado na Figura 11. O algoritmo verifica a existência da aresta do animal para sua geração. Se não existe, é então verificado se os arcos geracionais dos pais existem. Se existirem, o valor é retornado; caso contrário, a função é chamada recursivamente. O teste de existência garante o cálculo único dos valores.

Na Figura 12 é visualizada a representação do relacionamento dentro do banco de dados, onde os animais estão conectados às suas gerações correspondentes. Observa-se a presença das seis gerações (nodos amarelos) que foram identificadas na base de dados usada para teste do algoritmo.

O cálculo dos coeficientes de coancestralidade (relação de parentesco entre dois animais) requer a prévia determinação das gerações. No decorrer deste texto, será apresentada uma abordagem que utiliza matrizes, embora todas as operações sejam executadas diretamente no banco de dados com a assistência de programação em *Python* para a implementação de ciclos iterativos. O código desenvolvido pode ser encontrado no Apêndice E.

Considere um rebanho $H = (A = S \cup D, s, d : A \to (A \cup \{\perp\}))$, onde $K \in \mathbb{N}$ é o número de gerações de animais nascidos no rebanho, e $\overline{A}$ é uma permutação dos elementos de A ordenados por geração. Seja C a matriz de dimensões $n \times n$, em que $n = |A|$ é o

Figura 12 – Relacionamento entre os animais e suas respectivas gerações



número de elementos do conjunto A, e seja n_i, $0 \leq i \leq K$ o número de animais pertencentes à geração i, ou o tamanho do conjunto $\{a \in A \mid g(a) = i\}$.

Os índices dos elementos da matriz de coancestralidade C são construídos com a ordenação $\overline{A}$, de forma que para quaisquer dois índices $i, j \in \{1, \ldots, n\}$, se $i < j$ então $g(i) \leq g(j)$. Se o animal i é o l-ésimo animal pertencente à geração k, $k = 0, \ldots, K$, de acordo com a ordenação $\overline{A}$, então $i = l + \sum_{s=1}^{k} n_{s-1}$. Ou seja, os animais da geração 0 começam no índice 1, os animais da geração 1 começam no índice $n_0 + 1$, os animais da geração 2 começam no índice $n_0 + n_1 + 1$, e assim sucessivamente.

O cálculo do coeficiente de coancestralidade entre os animais do rebanho da Figura 2 é apresentado na Figura 13. Animais fortemente aparentados exibem um índice de coancestralidade alto, enquanto animais sem antepassados comuns têm coancestralidade igual a zero. O método de cálculo da coancestralidade pode variar, sendo o algoritmo dinâmico proposto neste trabalho independente do método específico.

O método usado para o cálculo da coancestralidade entre todos os animais do rebanho usa a Eq. (4), em uma estratégia de Programação Dinâmica (CORMEN et al., 2002). O preenchimento da matriz é feito por geração e, dentro de cada geração, por etapas. As cores usadas na matriz da Figura 13 ajudam a exemplificar a ordem de preenchimento. Como os animais estão ordenados por gerações, os pais sempre aparecem antes dos filhos na matriz. A ordem entre animais da mesma geração é arbitrária, e o preenchimento é feito de forma a que sempre que um valor de coancestralidade seja

Figura 13 – Matriz de coancestralidade – exemplo

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_1	1	0	0	0	0,5	0	0	0	0	0
a_2	0	1	0	0	0,5	0,5	0	0	0,25	0
a_3	0	0	1	0	0	0,5	0,5	0,5	0,5	0,5
a_4	0	0	0	1	0	0	0,5	0,5	0,25	0,5
a_5	0,5	0,5	0	0	1	0,25	0	0	0,125	0
a_6	0	0,5	0,5	0	0,25	1	0,25	0,25	0,75	0,25
a_7	0	0	0,5	0,5	0	0,25	1	0,5	0,75	1
a_8	0	0	0,5	0,5	0	0,25	0,5	1	0,375	1
a_9	0	0,25	0,5	0,25	0,125	0,75	0,75	0,375	1	1,125
a_{10}	0	0	0,5	0,5	0	0,25	1	1	1,125	1

Fonte: Rosso, Rosa e Ferreira (2023)

necessário, ele já tenha sido calculado.

A coancestralidade entre animais diferentes pertencentes à primeira geração do rebanho é sempre igual a zero, obedecendo à suposição de que eles não são aparentados. Aqui, assume-se que a geração 0 representa os animais que não têm ascendentes (pai e mãe) dentro do rebanho. Assim, a matriz superior esquerda da matriz C é uma matriz identidade, em que todos os seus elementos são iguais a zero, com exceção dos elementos da diagonal principal, que são iguais a 1 (visto que a coancestralidade de um animal consigo mesmo geralmente é 1. Essa suposição não apresenta perda de generalidade pois, se os animais forem aparentados, basta inicializar a matriz inicial com os valores corretos correspondentes. Na Figura 13, os índices de coancestralidade da geração 0 estão apresentados nesta cor, os da geração 1 nesta e os da geração 2 nesta.

Os animais pertencentes à geração 1 são, necessariamente, filhos de dois animais pertencentes à geração 0 (ver Eq. (3)). Os animais da geração 2 têm pelo menos um dos pais na geração 1, com o outro podendo ser da geração 1 ou de gerações anteriores, e assim sucessivamente. O preenchimento da matriz é descrito no pseudo-código apresentado na Figura 14, assumindo C a matriz em preenchimento, o número total de gerações K e os valores $n_0, n_1, \ldots, n_K$ para o número total de animais em cada geração e que os elementos da submatriz $C_{1,\ldots,n_0 \times 1,\ldots,n_0}$ (i.e., os valores da geração 0) já estão preenchidos.

Note que a matriz é simétrica em relação à diagonal principal, visto que coancestralidade é uma relação simétrica.

Figura 14 – Algoritmo dinâmico para o cálculo da coancestralidade

```
CALCULOCOANCESTRALIDADE(Ā, s, d : Ā → Ā, K, C)
(1)  para cada k = 1, ..., K fazer
(2)     para cada i ← (1 + Σ_{l=1}^{k-1} n_l), ..., (n_k + Σ_{l=1}^{k-1} n_l) fazer
(3)        para cada j = 1, ..., i fazer
(4)           se (i = j) então
(5)              C_ij ← 1
(6)           senão
(7)              se s(j) e d(j) são conhecidos então
(8)                 C_ij ← 0,5 · (C_{i,s(j)} + C_{i,d(j)})
(9)              senão se apenas s(j) é conhecido então
(10)                C_ij ← 0,5 · C_{i,s(j)}
(11)             senão se apenas d(j) é conhecido então
(12)                C_ij ← 0,5 · C_{i,d(j)}
(13)             senão
(14)                C_ij ← 0
(15)          C_ji ← C_ij
```

Fonte: Autor, (2025)

5.3 Modelagem do problema de minimização da coancestralidade

O problema de minimização da coancestralidade é um problema de otimização combinatória, resolvido por uma abordagem baseada em Otimização por Colônia de Formigas (ACO). O objetivo é formar um conjunto de pares de acasalamento (macho×fêmea), denominado *solução*, de forma que a coancestralidade média entre os produtos resultantes desses acasalamentos seja minimizada, respeitando restrições genéticas e reprodutivas.

Cada solução no problema de acasalamento pode ser modelada em um grafo, onde cada nodo representa um produto resultante de um possível emparelhamento macho×fêmea. As formigas percorrem esse grafo, selecionando sucessivamente pares até formar uma solução completa, isto é, um conjunto de acasalamentos que respeita as restrições impostas. A composição do grafo se dá por:

- **Nodos**: um nodo representa um par macho×fêmea disponível para acasalamento.
- **Arestas**: as arestas representam transições entre pares ao longo da construção da solução por uma formiga. A heurística (baseada na coancestralidade entre os produtos de dois acasalamentos) está associada a essas arestas.

O objetivo é que as formigas percorram esse grafo, escolhendo, em cada nodo, um produto que representa um par que deve acasalar, de modo que todas as fêmeas tenham

sido utilizadas uma única vez.

A função objetivo do problema é minimizar a coancestralidade média entre os produtos que compõem a solução baseada na Eq. (8) da Definição 11.

$$\overline{C}(\overline{p}) = \frac{C(\overline{p})}{|S^*|}$$

Onde:

- $\overline{C}(\overline{p})$ é a coancestralidade média da solução.
- $C(\overline{p})$ é o somatório da coancestralidade entre os produtos selecionados.
- $|S^*|$ representa o total de conjuntos de pares macho×fêmea selecionados para acasalar.

O objetivo é minimizar $\overline{C}(\overline{p})$, ou seja, buscar garantir que as soluções finais tenham o menor grau possível de parentesco.

O processo de construção de soluções no ACO envolve formigas que constroem sequencialmente uma solução completa, neste caso, um conjunto de pares de acasalamento, respeitando as restrições impostas pelo problema.

Cada formiga começa com um produto e precisa escolher o próximo par de acasalamento, levando em conta a relação deste produto com o próximo por meio da heurística, que leva em conta a coancestralidade. A construção segue até que todas as fêmeas tenham sido alocadas em acasalamentos únicos, garantindo que cada uma contribua com exatamente um produto. Durante essa caminhada no grafo, as formigas escolhem o próximo produto/par (macho×fêmea) com base em uma **probabilidade** que depende de dois fatores:

1. **Feromônio** depositado no nodo — indica a atratividade histórica de usar aquele par no passado;
2. **Heurística** na aresta entre pares — reflete a afinidade genética (coancestralidade) entre os produtos gerados por diferentes acasalamentos, sendo inversamente proporcional à coancestralidade esperada. A heurística é calculada como $1/(1+C(\overline{p})$, onde $C(\overline{p}$ é o somatório da coancestralidade envolvida entre os nodos que compõe a solução. Ou seja, a heurística entre dois pares considera a relação genética dos produtos esperados do par atual com os produtos esperados dos pares já escolhidos.

A probabilidade de uma formiga k escolher o próximo par macho×fêmea (i, j) a

partir de um outro par é dada por:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \tag{18}$$

Onde:

- τ_{ij} é o feromônio do nodo (a_{ij}).
- η_{ij} é a heurística da aresta (ligação entre dois pares).
- α e β são parâmetros que controlam o peso relativo de feromônio e heurística.
- N_i é o conjunto de pares viáveis (fêmeas não acasaladas e machos com acasalamentos disponíveis).

As formigas constroem soluções completas de maneira iterativa, produto a produto, par a par. Cada formiga lembra as decisões anteriores (pares já formados) e, ao final de sua caminhada, terá gerado uma solução completa, ou seja, um conjunto de pares macho×fêmea que atendem às restrições do problema.

- **Limite de acasalamento dos machos**: Ao selecionar o próximo macho para uma fêmea, a formiga só pode escolher machos que ainda não atingiram o limite de acasalamentos.
- **Acasalamento único para fêmeas**: Cada fêmea só pode ser acasalada uma vez, então cada formiga precisa assegurar que selecione machos para fêmeas que ainda não foram acasaladas.

Ao final de cada iteração, ocorre a evaporação do feromônio:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} \tag{19}$$

Onde ρ é a taxa de evaporação ($0 < \rho < 1$). Isso evita que o sistema convirja prematuramente para uma única solução.

Em seguida, realiza-se a deposição de feromônio nas trilhas (nodos) utilizadas pelas melhores soluções. No modelo adotado, utiliza-se a estratégia Max-Min Ant System (MMAS), que reforça apenas a melhor solução da iteração ou a melhor solução global, alternadamente. O feromônio é atualizado conforme:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij} \tag{20}$$

$$\Delta\tau_{ij} = \begin{cases} \dfrac{1}{\overline{C}(\overline{p})} & \text{se o nodo } (a_{ij}) \text{ está na melhor solução } S_k \\ \\ 0 & \text{caso contrário} \end{cases} \tag{21}$$

Além disso, os valores de feromônio são limitados a um intervalo fixo $[\tau_{\min}, \tau_{\max}]$, com o objetivo de controlar a influência da memória histórica das soluções anteriores. No algoritmo implementado, esses limites são definidos conforme as Eq. (16) e Eq. (17). Essa estratégia visa garantir que nenhum caminho se torne permanentemente dominante (superexploração), nem completamente negligenciado (subexploração), promovendo um equilíbrio entre intensificação e diversificação da busca ao longo das iterações. Os feromônios são inicializados em $\tau_{\max}$. Embora inicializar os trilhos de feromônio com $\tau_{\min}$ pareça uma estratégia exploratória, Stützle e Hoos (2000) afirmam que isso tende a concentrar rapidamente o reforço em poucas opções, reduzindo a diversidade de soluções logo nas primeiras iterações. Por isso, a inicialização com $\tau_{\max}$ é preferida, pois promove uma fase inicial de exploração equilibrada entre todos os componentes da solução, antes da intensificação gradual conduzida pelo histórico de desempenho.

Vale ressaltar que o feromônio é lido diretamente durante a decisão da próxima escolha da formiga, e não apenas depositado. A heurística está associada às arestas, representando a coancestralidade entre os produtos esperados de dois acasalamentos. O feromônio está nos nodos, representando a atratividade de emparelhar a fêmea e o macho que aquele produto representa. A solução final é extraída diretamente a partir da caminhada completa da formiga no grafo, representando a sequência de pares formados.

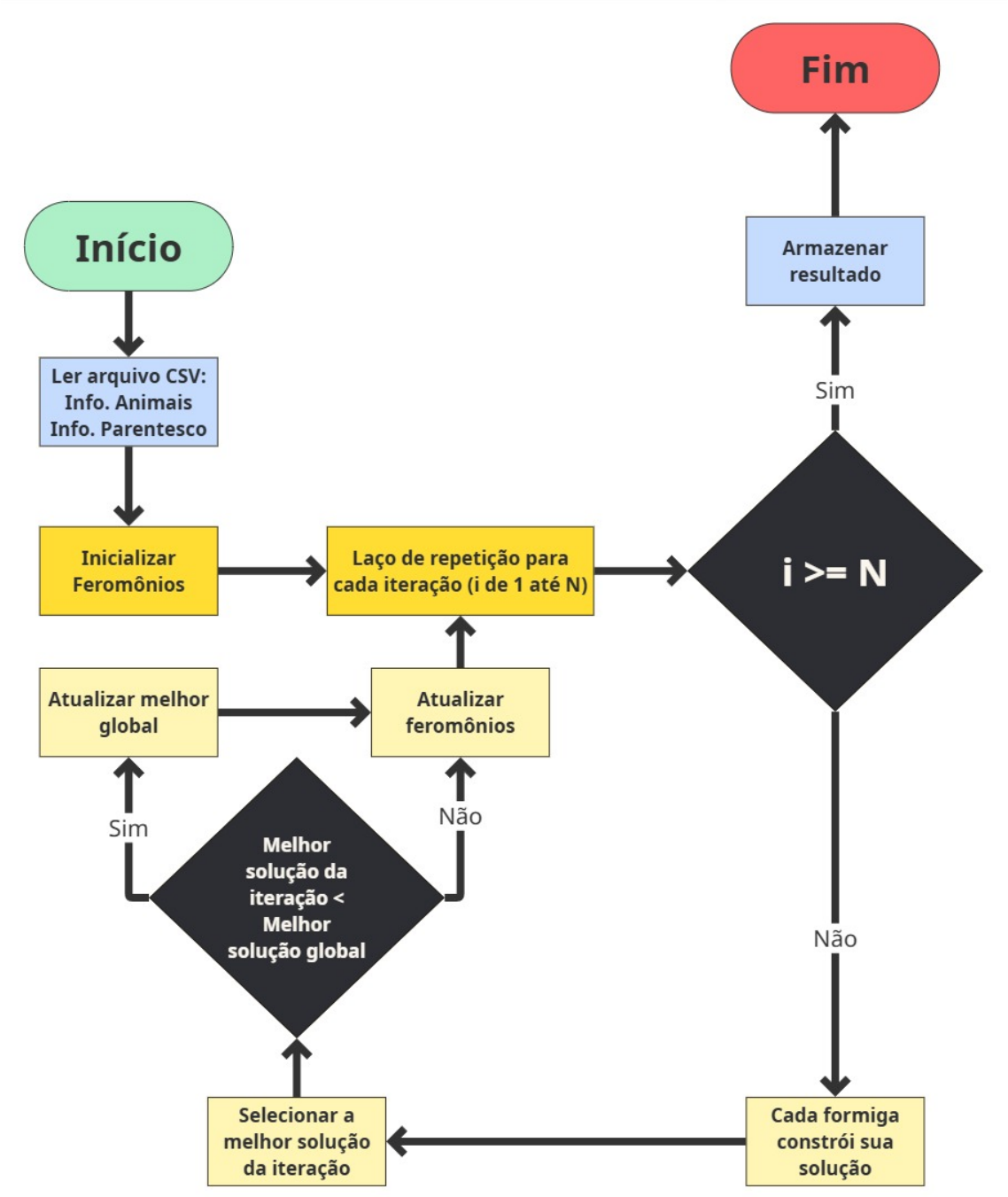
5.3.1 Exemplo de execução do algoritmo MMAS

A Figura 15 apresenta um fluxograma que ilustra o funcionamento do algoritmo proposto, com o objetivo de facilitar a compreensão do leitor. Os códigos correspondentes estão disponíveis no Apêndice H, para consulta.

Para ilustrar o funcionamento do algoritmo MMAS aplicado ao problema de minimização da coancestralidade entre os descendentes, apresentamos um exemplo fictício com os seguintes parâmetros:

- Número de fêmeas: 6

Figura 15 – Fluxograma



Fonte: Autor, (2025)

- Número de machos disponíveis: 2
- Máximo de cruzamentos por macho: 3 (O valor definido deve ser suficiente para garantir que todas as fêmeas sejam alocadas, ou seja, não pode ser menor do que a razão entre o número total de fêmeas e o número de machos disponíveis.)
- Número total de emparelhamentos possíveis: $2 \times 6 = 12$

Consideramos os seguintes animais:

- Machos: M1, M2
- Fêmeas: F1, F2, F3, F4, F5, F6

A matriz de feromônio armazena um valor τ_{ij} associado a cada produto (nodo), onde cada nodo representa um par reprodutivo macho×fêmea. Esse valor indica a atratividade da escolha daquele cruzamento para a construção das soluções pelas formigas. Conforme a abordagem do MMAS, os valores de feromônio são restritos a um intervalo definido, com um limite inferior τ_{min} e um limite superior τ_{max}, de modo a evitar que o algoritmo convirja prematuramente ou perca diversidade. Na primeira execução, todos os nodos recebem o valor inicial máximo 1.0, garantindo igualdade de atratividade antes da aplicação de qualquer decaimento. Essa escolha decorre do fato de que o valor ótimo de τ_{max}, geralmente definido como $\tau_{max} = \frac{1}{\rho \cdot f_{best}}$, depende de uma avaliação anterior do melhor valor objetivo, o que ainda não está disponível antes da geração da primeira solução. Assim, a atribuição uniforme de valor 1.0 assegura uma inicialização neutra e evita a introdução de viés prematuro na busca. A matriz de feromônio pode ser visualizada em formato de tabela se considerarmos o nodo/produto ao invés do par macho×fêmea como vemos na Tabela 12.

Ao longo da execução do algoritmo, a heurística (η) entra em ação, sendo calculada de forma progressiva durante a formação da solução. A heurística está associada às arestas do grafo, representando a coancestralidade esperada entre os descendentes dos pares de cruzamento. Para cada solução gerada, são registradas a coancestralidade total; a cada novo indivíduo selecionado, o valor de coancestralidade entre o segundo indivíduo selecionado e o primeiro, entre o terceiro e o primeiro e segundo, e assim por diante, é atualizado. Valores menores de coancestralidade média indicam melhor diversidade genética, sendo favoráveis à construção da solução. Para o primeiro nodo da solução, como ainda não há arestas disponíveis para o cálculo, adota-se inicialmente um valor heurístico de $\eta = 1$ e as seguintes restrições:

- Cada macho deve ser emparelhado com no máximo 3 fêmeas

Tabela 12 – Tabela de feromônio inicial τ

Produto	τ (feromônio)
M1_F1	1.0
M1_F2	1.0
M1_F3	1.0
M1_F4	1.0
M1_F5	1.0
M1_F6	1.0
M2_F1	1.0
M2_F2	1.0
M2_F3	1.0
M2_F4	1.0
M2_F5	1.0
M2_F6	1.0

Fonte: Autor, (2025)

- Cada fêmea deve ser emparelhada com exatamente 1 macho

A seguir, a Figura 16 apresenta a matriz de coancestralidade fictícia entre os produtos. Os valores indicados com $\times$ não são considerados no cálculo, pois correspondem à coancestralidade entre indivíduos que possuem a mesma mãe — e, portanto, não serão escolhidos mutuamente — ou à coancestralidade de um indivíduo consigo mesmo.

Calcula-se a coancestralidade média considerando apenas os pares que compõem uma solução:

$$\overline{C}(\overline{p}) = \frac{1}{\text{número de produtos selecionados}} C(\overline{p}) \tag{22}$$

Simulação: construção de solução por uma formiga

A formiga constrói uma solução **passo a passo**, escolhendo um cruzamento por vez com base na **probabilidade proporcional** à:

$$P_{ij} \propto \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} \tag{23}$$

Vamos usar:

Figura 16 – Matriz de coancestralidade

	M1_F1	M1_F2	M1_F3	M1_F4	M1_F5	M1_F6	M2_F1	M2_F2	M2_F3	M2_F4	M2_F5	M2_F6
M1_F1	×	0.20	0.40	0.50	0.20	0.30	×	0.25	0.40	0.55	0.20	0.30
M1_F2	0.20	×	0.30	0.20	0.40	0.50	0.20	×	0.30	0.20	0.40	0.50
M1_F3	0.40	0.30	×	0.60	0.30	0.40	0.40	0.30	×	0.60	0.30	0.40
M1_F4	0.50	0.20	0.60	×	0.20	0.30	0.50	0.20	0.60	×	0.20	0.30
M1_F5	0.20	0.40	0.30	0.20	×	0.50	0.20	0.40	0.30	0.20	×	0.50
M1_F6	0.30	0.50	0.40	0.30	0.50	×	0.30	0.50	0.40	0.30	0.50	×
M2_F1	×	0.20	0.40	0.50	0.20	0.30	×	0.20	0.40	0.50	0.20	0.30
M2_F2	0.25	×	0.30	0.20	0.40	0.50	0.20	×	0.30	0.80	0.40	0.50
M2_F3	0.40	0.30	×	0.60	0.30	0.40	0.40	0.30	×	0.60	0.30	0.40
M2_F4	0.55	0.20	0.60	×	0.20	0.30	0.50	0.80	0.60	×	0.20	0.30
M2_F5	0.20	0.40	0.30	0.20	×	0.50	0.20	0.40	0.30	0.20	×	0.50
M2_F6	0.30	0.50	0.40	0.30	0.50	×	0.30	0.50	0.40	0.30	0.50	×

Fonte: Autor, (2025)

- $\alpha = 1$
- $\beta = 2$ (peso maior para a heurística)

Passo 1: Escolha para F1

Para emparelhar a fêmea F1, a formiga pode escolher entre os produtos M1_F1 ou M2_F1. Vale destacar que, neste início, a formiga pode iniciar sua construção por qualquer nodo e, consequentemente, definir qualquer par. Isso significa que qualquer fêmea pode ter seu acasalamento decidido primeiro. As Tabelas 13 a 18 apresentam os cálculos realizados para determinar a probabilidade de escolha da formiga em cada passo, considerando sempre as decisões tomadas anteriormente.

Tabela 13 – Probabilidade da 1ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M1_F1	1	1	$1 \cdot 1 = 1$	$\frac{1}{1+1} = 0{,}5 = 50\%$
M2_F1	1	1	$1 \cdot 1 = 1$	$\frac{1}{1+1} = 0{,}5 = 50\%$

Fonte: Autor, (2025)

A formiga sorteia e escolhe **M1_F1**.

Passo 2: Escolha para F2

M1 foi usado uma vez, pode ser usado mais duas vezes.

Tabela 14 – Probabilidade da 2ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M1_F2	1	$\frac{1}{1+0,2}$	$1 \cdot 0,69 = 0,69$	$\frac{0,69}{0,69+0,64} = 0,5188 = 51,88\%$
M2_F2	1	$\frac{1}{1+0,25}$	$1 \cdot 0,64 = 0,64$	$\frac{0,64}{0,69+0,64} = 0,4812 = 48,12\%$

Fonte: Autor, (2025)

A formiga escolhe **M2_F2** e armazena o valor de coancestralidade entre a primeira escolha **M1_F1** e a segunda escolha **M2_F2**. Nesta fase, a solução tem coancestralidade total $= 0,25$.

Passo 3: Escolha para F3

M1 e M2 foram usados uma vez, podem ser usados mais duas vezes.

Tabela 15 – Probabilidade da 3ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M1_F3	1	$\frac{1}{1+(0,3+0,4)}$	$1 \cdot 0,346 = 0,346$	$\frac{0,346}{0,346+0,346} = 0,5 = 50\%$
M2_F3	1	$\frac{1}{1+(0,3+0,4)}$	$1 \cdot 0,346 = 0,346$	$\frac{0,346}{0,346+0,346} = 0,5 = 50\%$

Fonte: Autor, (2025)

A formiga sorteia e escolhe **M1_F3** e armazena a soma do valor de coancestralidade entre a terceira escolha e as duas anteriores. Nesta fase, a solução tem coancestralidade total $= 0,25 + 0,4 + 0,3 = 0,95$.

Passo 4: Escolha para F4

M1 foi usado duas vezes, pode ser usado mais uma vez.
M2 foi usado uma vez, pode ser usado mais duas vezes.

Tabela 16 – Probabilidade da 4ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M1_F4	1	$\frac{1}{1+(0,5+0,2+0,6)}$	$1 \cdot 0,189 = 0,945$	$\frac{0,189}{0,189+0,114} = 0,62 = 62\%$
M2_F4	1	$\frac{1}{1+(0,55+0,8+0,6)}$	$1 \cdot 0,114 = 0,574$	$\frac{0,114}{0,189+0,114} = 0,38 = 38\%$

Fonte: Autor, (2025)

A formiga escolhe **M1_F4** e armazena a soma do valor de coancestralidade entre a quarta escolha e as três anteriores. Nesta fase, a solução tem coancestralidade total $= 0,95 + 0,5 + 0,2 + 0,6 = 2,25$.

Passo 5: Escolha para F5

Resta apenas M2, que ainda pode ser utilizado mais duas vezes.

Tabela 17 – Probabilidade da 5ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M2_F5	1	$\frac{1}{1+(0,2+0,4+0,3+0,2)}$	$1 \cdot 0,189 = 0,189$	$= 1,0 = 100\%$

Fonte: Autor, (2025)

A formiga escolhe **M2_F5** e armazena a soma do valor de coancestralidade entre a quinta escolha e as quatro anteriores. Nesta fase, a solução tem coancestralidade total $= 2,25 + 0,2 + 0,4 + 0,3 + 0,2 = 3,35$.

Passo 6: Escolha para F6

Resta apenas M2, que ainda pode ser utilizado mais uma vez.

Tabela 18 – Probabilidade da 6ª escolha

Produto	τ	η	$\tau \cdot \eta^2$	Probabilidade
M2_F6	1	$\frac{1}{1+(0,3+0,5+0,4+0,3+0,5)}$	$1 \cdot 0,111 = 0,111$	$= 1,0 = 100\%$

Fonte: Autor, (2025)

A formiga escolhe **M2_F6** e armazena a soma do valor de coancestralidade entre a sexta escolha e as cinco anteriores. Nesta fase, a solução tem coancestralidade total $= 3,35 + 0,3 + 0,5 + 0,4 + 0,3 + 0,5 = 5,35$.

A Tabela 19 apresenta a solução construída com base nos produtos selecionados. Por exemplo, ao selecionar o produto M1_F1, isso indica que a fêmea F1 é a possível mãe e o macho M1 é o possível pai, formando assim um par reprodutivo. Essa solução é considerada válida por satisfazer ambas as restrições do problema: cada fêmea foi acasalada exatamente uma vez, e nenhum macho excedeu o limite de três cruzamentos.

Tabela 19 – Solução

Fêmea	Macho escolhido
F1	M1
F2	M2
F3	M1
F4	M1
F5	M2
F6	M2

Fonte: Autor, (2025)

A Figura 17 apresenta a matriz de coancestralidade entre os prováveis descendentes, que nada mais é do que um recorte da matriz previamente mostrada na Figura 16. A partir dessa matriz, calcula-se a qualidade da solução com base na coancestralidade média entre os produtos selecionados.

Calcula-se a coancestralidade média:

$$\overline{C}(\overline{p}) = \frac{1}{6} C(\overline{p})_{ij} = \frac{5,35}{6} \approx 0,891 \tag{24}$$

A qualidade da solução construída por esta formiga, representada pela

Figura 17 – Coancestralidade entre os produtos escolhidos

	M1_F1	M2_F2	M1_F3	M1_F4	M2_F5	M2_F6
M1_F1	1.00	0.25	0.40	0.50	0.20	0.30
M2_F2	0.25	1.00	0.30	0.20	0.40	0.50
M1_F3	0.40	0.30	1.00	0.60	0.30	0.40
M1_F4	0.50	0.20	0.60	1.00	0.20	0.30
M2_F5	0.20	0.40	0.30	0.20	1.00	0.50
M2_F6	0.30	0.50	0.40	0.30	0.50	1.00

Fonte: Autor, (2025)

coancestralidade média dos descendentes, é $\overline{C}(\overline{p}) = 0{,}891$. Como o objetivo do algoritmo é minimizar essa medida, soluções com menor coancestralidade média são preferidas ao longo das iterações. Esta implementação está descrita no Apêndice H. No entanto, existem outras formas de avaliar a qualidade da solução gerada. Entre elas, destaca-se o número de relações de parentesco igual a zero — isto é, o número de arestas com peso zero na matriz de coancestralidade. Esse critério de avaliação é formalmente definido na Definição 14, enquanto o problema associado está descrito na Definição 15. Esse método alternativo foi implementado algoritmicamente e encontra-se detalhado no Apêndice I.

Após a construção e avaliação de todas as soluções pelas formigas na iteração, o algoritmo realiza a atualização da matriz de feromônio. No MMAS, apenas a melhor solução (global ou da iteração) realiza depósito de feromônio.

Seja τ_{ij} o valor do feromônio associado ao acasalamento entre a fêmea i e o macho j. A atualização ocorre conforme:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \Delta\tau_{ij} \tag{25}$$

onde:

- $\rho \in (0,2)$ é a taxa de evaporação dos feromônios;
- $\Delta\tau_{ij}$ é o reforço de feromônio aplicado apenas aos acasalamentos presentes na melhor solução da iteração ou global.

Para os acasalamentos (i, j) da melhor solução:

$$\Delta\tau_{ij} = \frac{Q}{\overline{C}(\overline{p})} \tag{26}$$

- Q é uma constante de reforço;
- $\overline{C}(\overline{p})$ é a coancestralidade média da melhor solução.

Exemplo de aplicação da atualização

Suponha que os seguintes parâmetros tenham sido definidos:

- Feromônio inicial: $\tau_{ij}^{(t)} = 1.0$
- Taxa de evaporação: $\rho = 0{,}2$
- Melhor solução: $\{\text{M1\_F1}, \text{M2\_F2}, \text{M1\_F3}, \text{M1\_F4}, \text{M2\_F5}, \text{M2\_F6}\}$
- Coancestralidade média da solução/Custo: $\overline{C}(\overline{p}) = 0{,}891$
- Reforço: $Q = 1 \Rightarrow \Delta\tau = \frac{1}{0{,}891} \approx 1{,}122$

Cada nodo correspondente a um acasalamento da melhor solução receberá esse reforço. Após evaporação e reforço, o novo valor será:

$$\tau_{ij}^{(t+1)} = (1 - 20\%) \cdot 1 + 1{,}122 = 0{,}8 + 1{,}122 = 1{,}922 \tag{27}$$

Para os demais acasalamentos não utilizados na melhor solução, apenas a evaporação se aplica:

$$\tau_{ij}^{(t+1)} = (1 - 20\%) \cdot \tau_{ij}^{(t)} = 0{,}8 \cdot \tau_{ij}^{(t)} \tag{28}$$

Restrições dos valores de feromônio

No MMAS, os valores de feromônio são restringidos entre limites $[\tau_{\min}, \tau_{\max}]$ para evitar convergência prematura:

Então:

A partir da equação da melhor solução encontrada $f(s^*) = \overline{C}(\overline{p}) = 0{,}891$, com taxa de evaporação $\rho = 0{,}2$, número de decisões $n = 6$, valor médio de opções por decisão avg $= 2$, e probabilidade de construção da melhor solução $p_{\text{best}} = 0{,}05$, obtêm-se os limites de feromônio:

$$\tau_{\max} = \frac{1}{(1-\rho) \cdot f(s^*)} = \frac{1}{0{,}8 \cdot 0{,}891} \approx 1{,}402 \tag{29}$$

$$\tau_{\min} = \frac{\tau_{\max} \cdot (1 - \sqrt[6]{0{,}05})}{(2-1) \cdot \sqrt[6]{0{,}05}} = \frac{1{,}402 \cdot 0{,}3935}{0{,}6065} \approx 0{,}909 \tag{30}$$

Esses limites são aplicados após a evaporação e o reforço de feromônio para

garantir que os valores se mantenham dentro de uma faixa viável e que o algoritmo não perca diversidade ou convirja prematuramente.

- Para acasalamentos da melhor solução:

$$\tau_{ij}^{(t+1)} = \min(1{,}402, \max(0{,}909; 3{,}608)) = 1{,}402 \tag{31}$$

- Para os demais acasalamentos (não reforçados):

$$\tau_{ij}^{(t+1)} = \min(1{,}402, \max(0{,}909; 0{,}8)) = 0{,}909 \tag{32}$$

A tabela Tabela 20 demonstra o resumo dos valores de feromônio após as fases descritas e, com isso, a finalização de uma iteração.

Tabela 20 – Atualização de feromônios

Acasalamento	**Antes (t)**	**Após evaporação**	**Após reforço**	**Final ($\tau_{\min}/\tau_{\max}$)**
M1_F1 (melhor)	1	0,8	1,922	**1,402**
M2_F2 (melhor)	1	0,8	1,922	**1,402**
M1_F3 (melhor)	1	0,8	1,922	**1,402**
M1_F4 (melhor)	1	0,8	1,922	**1,402**
M2_F5 (melhor)	1	0,8	1,922	**1,402**
M2_F6 (melhor)	1	0,8	1,922	**1,402**
Outros	1	0,8	0,8	**0,909**

Fonte: Autor, (2025)

6 RESULTADOS E DISCUSSÃO

6.1 Execução do Algoritmo MMAS

Para validação do modelo proposto, foi construído um cenário experimental composto por dois machos e seis fêmeas da base da dados já mencionada anteriormente na Seção 2.4. Cada macho possui três irmãs no conjunto de fêmeas. As relações de parentesco entre os indivíduos são apresentadas a seguir:

O macho 541077 é irmão das fêmeas 541069, 541071 e 541075;

O macho 1631704 é irmão das fêmeas 565382, 1499060 e 1632210.

O exemplo é bastante restrito, mas ilustra a complexidade computacional do problema, cuja entrada é quadrática no número de pares de possíveis progenitores que, por sua vez, é a multiplicação do número de machos pelos número de fêmeas selecionadas para reprodução.

As relações de coancestralidade geradas com base em todos os possíveis indivíduos foram utilizadas como base heurística para a construção de soluções pelo algoritmo. A Tabela 21 apresenta os coeficientes de coancestralidade entre os pares de prole gerados a partir do cruzamento de cada macho com cada fêmea:

Tabela 21 – Relações de coancestralidade entre a prole

Animal 1	**Animal 2**	**Coef. de Coancestralidade**
541077_541069	541077_541071	0.375
541077_541069	541077_541075	0.375
541077_541069	541077_565382	0.375
541077_541069	541077_1499060	0.375
541077_541069	541077_1632210	0.375
541077_541069	1631704_541069	0.375
541077_541069	1631704_541071	0.0
541077_541069	1631704_541075	0.0
541077_541069	1631704_565382	0.0
541077_541069	1631704_1499060	0.0
541077_541069	1631704_1632210	0.0
541077_541071	541077_541075	0.375
541077_541071	541077_565382	0.375
541077_541071	541077_1499060	0.375
541077_541071	541077_1632210	0.375

Continua na próxima página

Animal 1	Animal 2	Coef. de Coancestralidade
541077_541071	1631704_541069	0.0
541077_541071	1631704_541071	0.375
541077_541071	1631704_541075	0.0
541077_541071	1631704_565382	0.0
541077_541071	1631704_1499060	0.0
541077_541071	1631704_1632210	0.0
541077_541075	541077_565382	0.375
541077_541075	541077_1499060	0.375
541077_541075	541077_1632210	0.375
541077_541075	1631704_541069	0.0
541077_541075	1631704_541071	0.0
541077_541075	1631704_541075	0.375
541077_541075	1631704_565382	0.0
541077_541075	1631704_1499060	0.0
541077_541075	1631704_1632210	0.0
541077_565382	541077_1499060	0.25
541077_565382	541077_1632210	0.25
541077_565382	1631704_541069	0.0
541077_565382	1631704_541071	0.0
541077_565382	1631704_541075	0.0
541077_565382	1631704_565382	0.2548828125
541077_565382	1631704_1499060	0.0
541077_565382	1631704_1632210	0.0
541077_1499060	541077_1632210	0.25
541077_1499060	1631704_541069	0.0
541077_1499060	1631704_541071	0.0
541077_1499060	1631704_541075	0.0
541077_1499060	1631704_565382	0.0
541077_1499060	1631704_1499060	0.2548828125
541077_1499060	1631704_1632210	0.0
541077_1632210	1631704_541069	0.0
541077_1632210	1631704_541071	0.0
541077_1632210	1631704_541075	0.0
541077_1632210	1631704_565382	0.0
541077_1632210	1631704_1499060	0.0
541077_1632210	1631704_1632210	0.2548828125
1631704_541069	1631704_541071	0.2548828125
1631704_541069	1631704_541075	0.2548828125

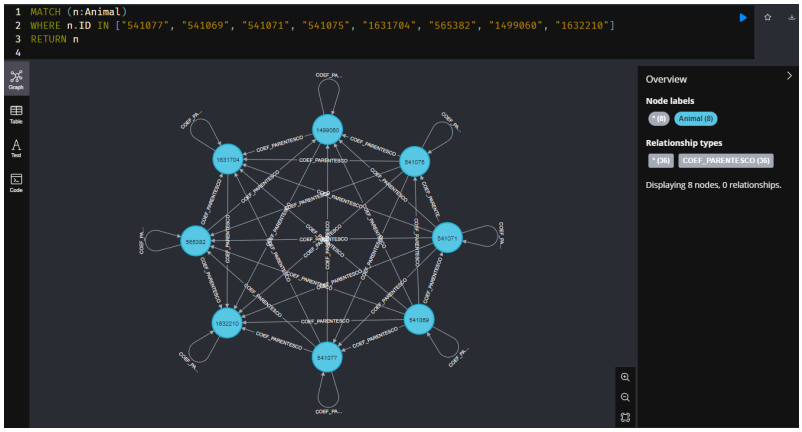
Continua na próxima página

Animal 1	Animal 2	Coef. de Coancestralidade
1631704_541069	1631704_565382	0.2548828125
1631704_541069	1631704_1499060	0.2548828125
1631704_541069	1631704_1632210	0.2548828125
1631704_541071	1631704_541075	0.2548828125
1631704_541071	1631704_565382	0.2548828125
1631704_541071	1631704_1499060	0.2548828125
1631704_541071	1631704_1632210	0.2548828125
1631704_541075	1631704_565382	0.2548828125
1631704_541075	1631704_1499060	0.2548828125
1631704_541075	1631704_1632210	0.2548828125
1631704_565382	1631704_1499060	0.3125
1631704_565382	1631704_1632210	0.3125
1631704_1499060	1631704_1632210	0.3125

Fonte: Autor, (2025)

É possível visualizar como esses dados estão armazenados na base a partir da Figura 18, que mostra as relações entre os animais e deles consigo mesmos. Como se tratam de 8 animais, a estrutura formada assume uma configuração próxima de um octógono.

Figura 18 – Animais selecionados no banco de dados



Fonte: Autor (2025)

Para a execução do algoritmo MMAS, foram definidos os parâmetros de teste apresentados nas Tabelas 22 e 23. Os experimentos foram executados no ambiente computacional descrito na Seção 2.3 (Processador AMD Ryzen 7, 8GB RAM). Os experimentos tiveram como objetivo analisar, inicialmente, a influência do número de

iterações em relação ao número de formigas e, em seguida, avaliar o impacto relativo entre os feromônios e a heurística. Embora seja possível configurar um limite máximo de consanguinidade — sendo 5% um valor comumente adotado na Embrapa —, tal restrição foi desconsiderada nesta etapa, a fim de validar o funcionamento do algoritmo de forma mais livre. A imposição desse limite poderia restringir artificialmente os resultados, mascarando o real desempenho do algoritmo. Os valores de taxa de evaporação do feromônio (ρ), constante de depósito (q) e limite máximo de uso por macho foram mantidos fixos em todos os testes, sendo, respectivamente, 0,1, 1,0 e 3.

Tabela 22 – Parâmetros utilizados no algoritmo MMAS - teste iteração e formigas

Parâmetro	**Valor 1**	**V. 2**	**V. 3**	**V. 4**	**V. 5**	**V. 6**	**V. 7**	**V. 8**	**V. 9**
Número de iterações	1	1	1	2	2	2	3	3	3
Número de formigas	1	2	3	1	2	3	1	2	3
Fator de influência do feromônio (α)	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
Fator de influência da heurística (β)	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0

Fonte: Autor (2025)

Tabela 23 – Parâmetros utilizados no algoritmo MMAS - teste α e β

Parâmetro	**Valor 1**	**V. 2**	**V. 3**	**V. 4**	**V. 5**	**V. 6**	**V. 7**	**V. 8**	**V. 9**
Número de iterações	1	1	1	1	1	1	1	1	1
Número de formigas	1	1	1	1	1	1	1	1	1
Fator de influência do feromônio (α)	1,0	1,0	1,0	2,0	2,0	2,0	3,0	3,0	3,0
Fator de influência da heurística (β)	1,0	2,0	3,0	1,0	2,0	3,0	1,0	2,0	3,0

Fonte: Autor (2025)

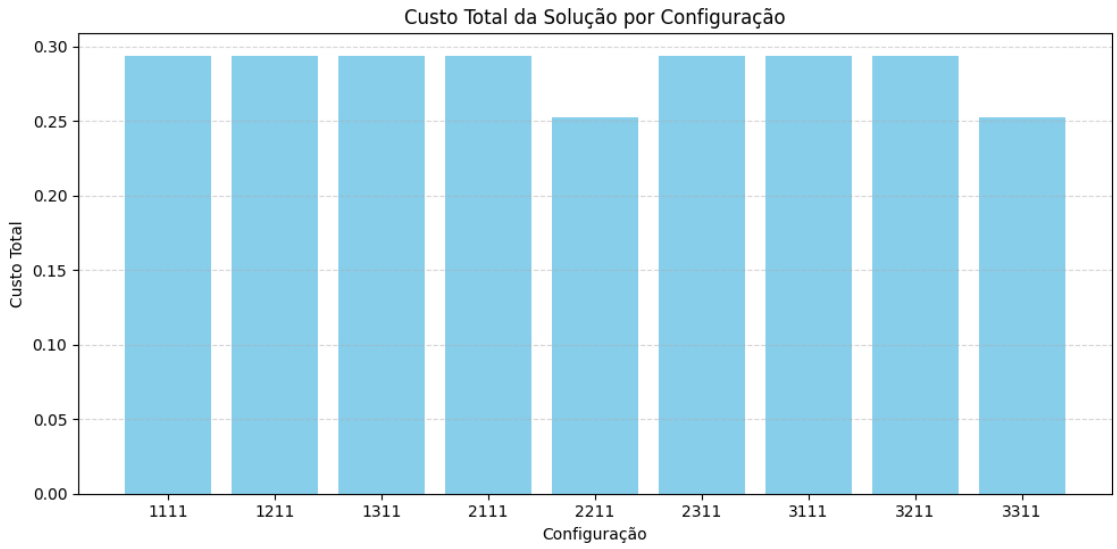
As combinações de configurações apresentadas na Tabela 22 resultaram nos gráficos apresentados nas Figura 19 e Figura 20, os quais ilustram, respectivamente, o custo total da solução e a média de consanguinidade observada para cada configuração testada.

Cada combinação é representada por um código de quatro dígitos, no formato *IIFB*, onde:

- **I** = número de iterações (1, 2 ou 3),
- **F** = número de formigas (1, 2 ou 3),
- **A** = valor de α, o peso do feromônio (1, 2 ou 3),
- **B** = valor de β, o peso da heurística (1, 2 ou 3).

Por exemplo, a configuração **1111** corresponde a 1 iteração, 1 formiga, $\alpha = 1{,}0$ e $\beta = 1{,}0$, enquanto **3311** representa 3 iterações, 3 formigas, $\alpha = 1{,}0$ e $\beta = 1{,}0$. As

Figura 19 – Custo



Fonte: Autor, (2025)

demais combinações seguem a mesma lógica, permitindo a análise isolada ou conjunta da influência de cada parâmetro sobre o desempenho do algoritmo MMAS.

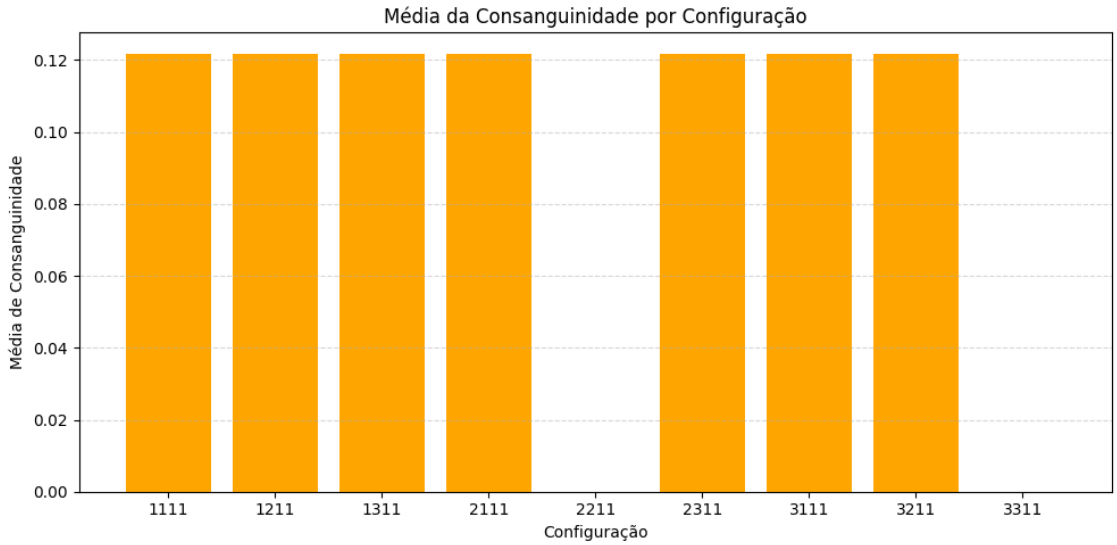
A Figura 19 mostra que as execuções que resultaram em menor custo total da solução — ou seja, menor parentesco entre os indivíduos selecionados — foram obtidas com as configurações **2211** e **3311**. Em ambas, observa-se que os valores de número de iterações e número de formigas são iguais, sugerindo que essa correspondência pode favorecer a convergência para soluções de menor consanguinidade no contexto analisado.

A Figura 20 confirma o desempenho observado na Figura 19, ao evidenciar que as configurações **2211** e **3311** resultaram em uma consanguinidade média igual a zero. Esse resultado reforça a eficácia dessas combinações de parâmetros na obtenção de soluções com menor grau de parentesco entre os indivíduos selecionados neste contexto.

Enquanto que para as combinações de configurações apresentadas na Tabela 23 resultaram nos gráficos apresentados nas Figura 21 e Figura 22, os quais ilustram, respectivamente, o custo total da solução e a média de consanguinidade observada para cada configuração testada. Nestes casos, a variação ocorreu entre os fatores de influência heurística e de feromônios.

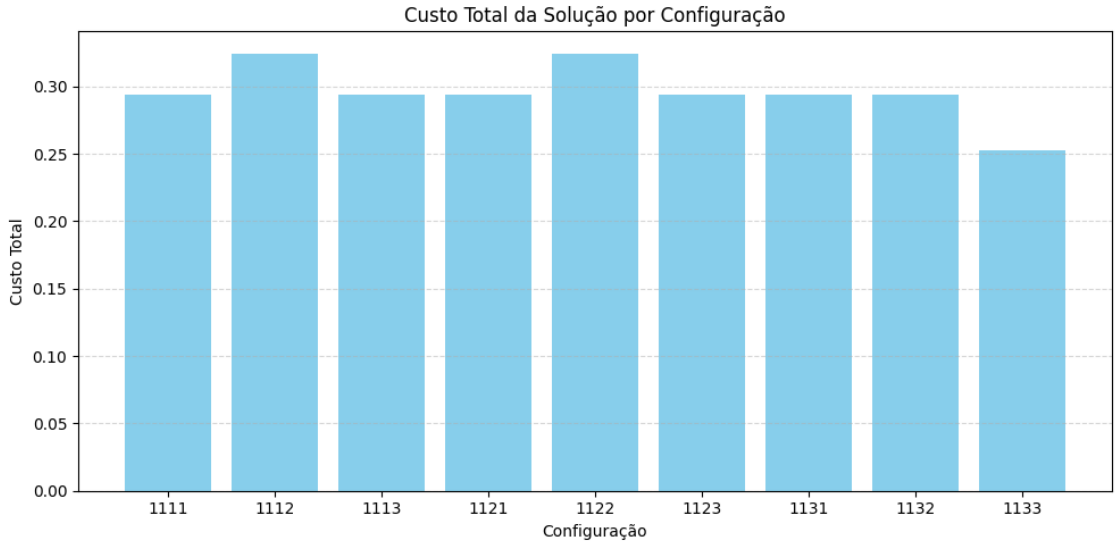
A Figura 21 apresenta resultados semelhantes ao longo das diferentes configurações testadas. O melhor desempenho foi obtido pela configuração **1133**, ainda que seus parâmetros apresentem pesos equivalentes aos das configurações **1122** e **1111**, nas quais α e β também possuem o mesmo valor.

Figura 20 – Consanguinidade média



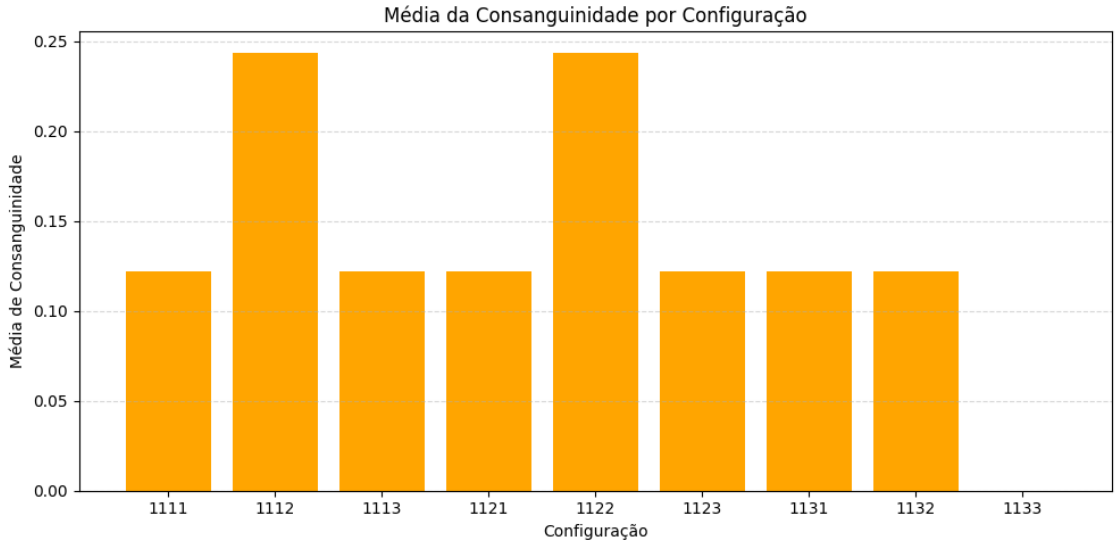
Fonte: Autor, (2025)

Figura 21 – Custo - α e β



Fonte: Autor, (2025)

Figura 22 – Consanguinidade média - α e β



Fonte: Autor, (2025)

A Figura 22 corrobora os resultados apresentados na Figura 21, confirmando que a configuração **1133** obteve o melhor desempenho. Essa configuração apresentou uma consanguinidade média igual a zero, indicando a formação de proles sem qualquer grau de parentesco aparente, o que reforça a qualidade da solução encontrada. Para este caso, em que havia apenas 2 touros e 6 vacas — um número relativamente pequeno de animais envolvidos no processo — foi suficiente utilizar apenas uma formiga e uma única iteração, com os parâmetros em equilíbrio, para encontrar a melhor solução.

6.2 Aplicação prática

Visando avaliar a aplicabilidade prática do algoritmo, foi obtido junto à Embrapa Pecuária Sul um cenário representativo para testes. Considera-se desejável que o algoritmo seja capaz de realizar, no mínimo, a predição de acasalamentos envolvendo 10 machos e 180 fêmeas — proporção equivalente a 1:18. Com base nesse cenário, foram realizados os testes descritos a seguir, cujos resultados são apresentados na sequência.

Os testes foram realizados utilizando o algoritmo descrito no Apêndice H, cuja função de custo se baseia na coancestralidade média entre os indivíduos da prole. Também foi avaliada uma versão alternativa do algoritmo, apresentada no Apêndice I, cuja função de custo considera o número de relações de coancestralidade com valor diferente de zero.

Foram testadas configurações com 30, 20 e 10 iterações, empregando 10, 5 e 1

formiga, combinadas com três diferentes configurações dos parâmetros α e β, a saber:

- Primeira configuração $\alpha = 1$ e $\beta = 2$: prioriza a influência da heurística na escolha das soluções;
- Segunda configuração $\alpha = 2$ e $\beta = 1$: confere maior peso à trilha de feromônio em detrimento da heurística;
- Terceira configuração $\alpha = 1$ e $\beta = 1$: atribui influência equilibrada (50% para cada fator) entre heurística e feromônio.

6.2.1 Custo: coancestralidade média

Utilizando o algoritmo descrito no Apêndice H, seguindo a **primeira** configuração, priorizando a heurística, obteve-se:

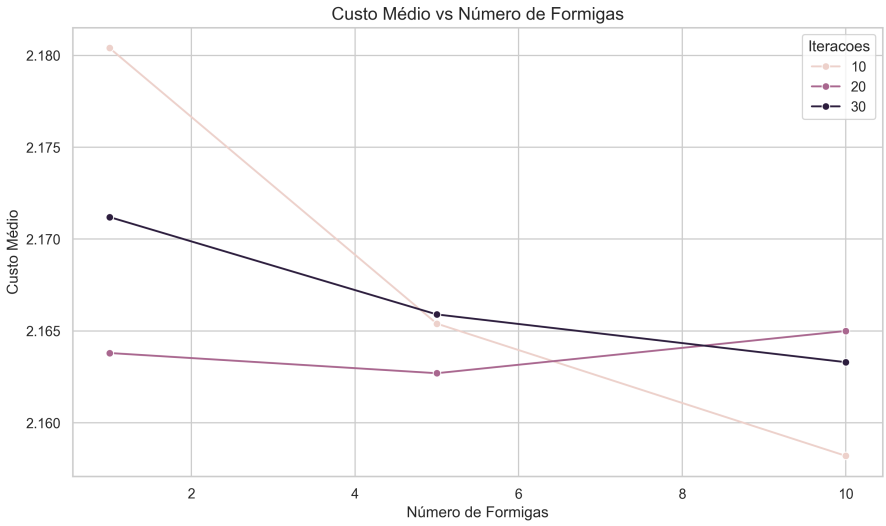


Figura 23 – Custo total em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

As Figuras 23, 24, 25 e 26 apresentam os gráficos que relacionam o custo (avaliado com base na coancestralidade média dos produtos resultantes) e a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 1$ e $\beta = 2$. Na Figura 23, observa-se que o custo tende a diminuir com o aumento do número de formigas utilizadas. No entanto, uma exceção ocorre no cenário com 20 iterações, em que o custo aumenta ao passar de 5 para 10 formigas. Na Figura 24, nota-se que a consanguinidade média das soluções permaneceu entre 0,004 e 0,006 — valores inferiores a 1%, o que é

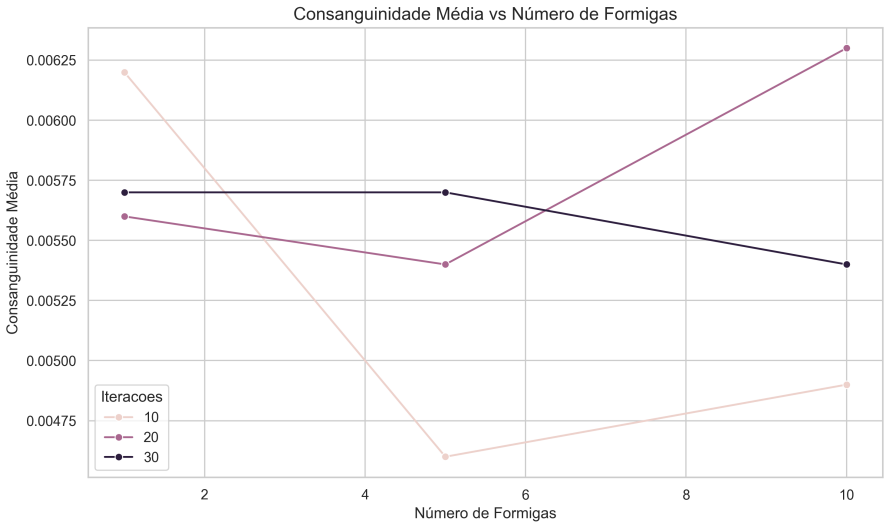


Figura 24 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

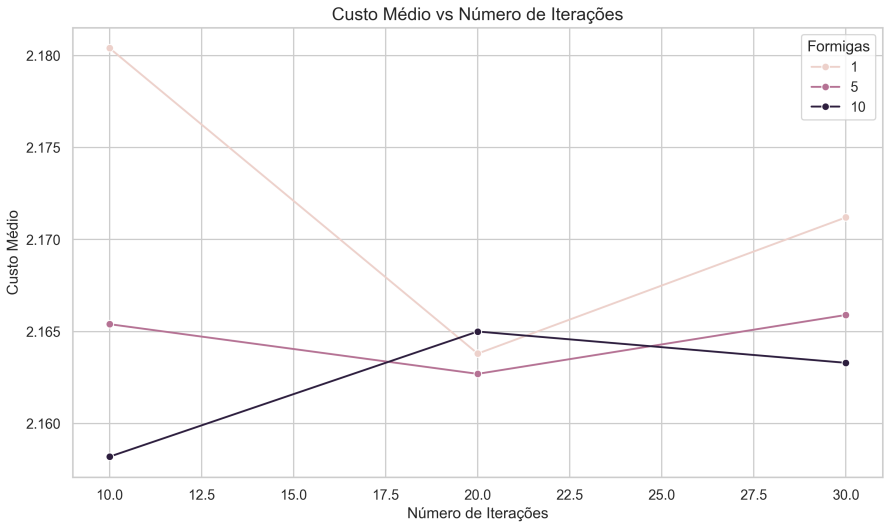


Figura 25 – Custo total em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

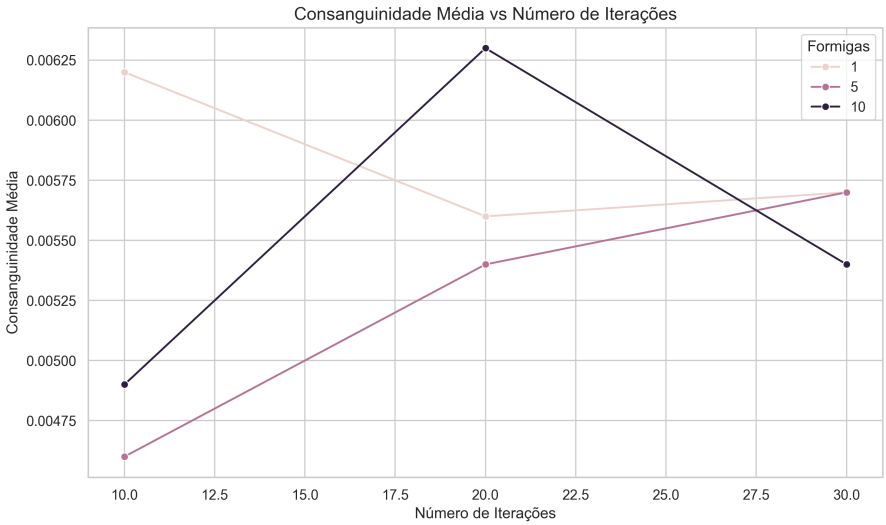


Figura 26 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

desejável. Houve uma tendência de queda ao aumentar de 1 para 5 formigas, porém, ao utilizar 10 formigas, os valores aumentaram novamente nos casos com 10 e 20 iterações. A Figura 25 mostra que o custo foi reduzido ao aumentar o número de iterações de 10 para 20, nos casos com 1 ou 5 formigas. Contudo, essa tendência não se manteve com 30 iterações, exceto no caso com 10 formigas, em que houve uma nova redução, embora menor do que o esperado. O melhor desempenho, em termos de custo, foi obtido com a configuração de 10 iterações e 10 formigas. Por fim, na Figura 26, verifica-se que a menor consanguinidade média registrada ocorreu na configuração com 10 iterações e 5 formigas.

Seguindo a **segunda** configuração, priorizando a trilha de feromônios, obteve-se:

As Figuras 27, 28, 29 e 30 apresentam os gráficos que relacionam o custo e a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 2$ e $\beta = 1$. Na Figura 27, observa-se que o custo tende a diminuir com o aumento do número de formigas até 5, mas volta a subir quando se utilizam 10 formigas. Na Figura 28, nota-se comportamento semelhante em relação à consanguinidade média, com exceção do cenário com 20 iterações, no qual o aumento do número de formigas resultou em uma leve melhora nesse parâmetro. A Figura 29 demonstra que o custo foi reduzido de forma consistente com o aumento do número de iterações em todos os casos analisados. O melhor desempenho, em termos de custo, foi obtido com a configuração de 30 iterações e 5 formigas. Por fim, na Figura 30, verifica-se que a menor consanguinidade média

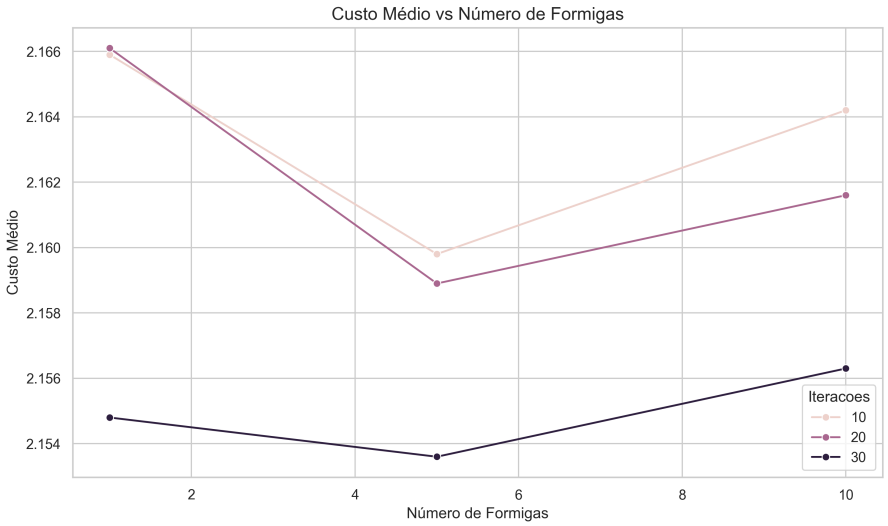


Figura 27 – Custo total em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

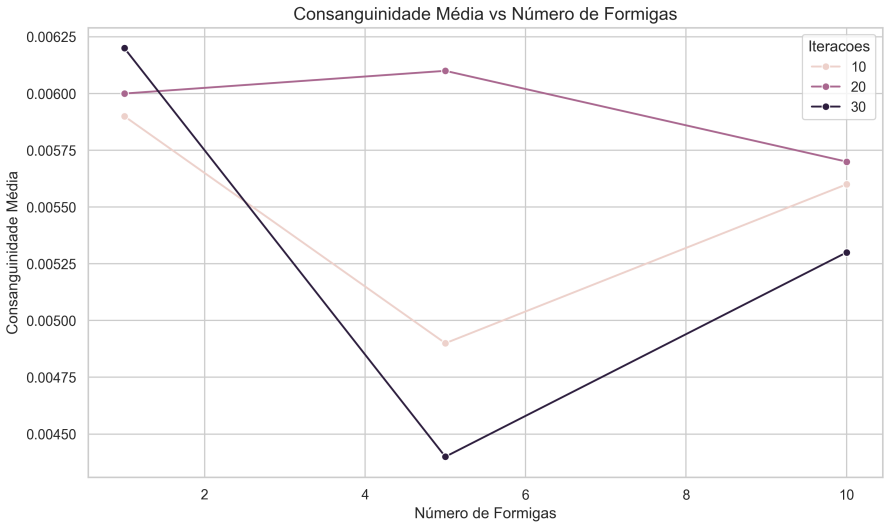


Figura 28 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

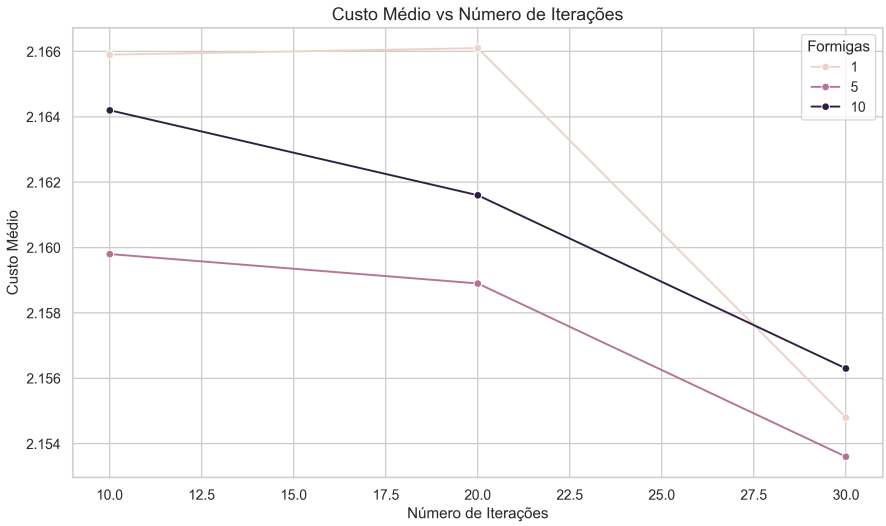


Figura 29 – Custo total em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

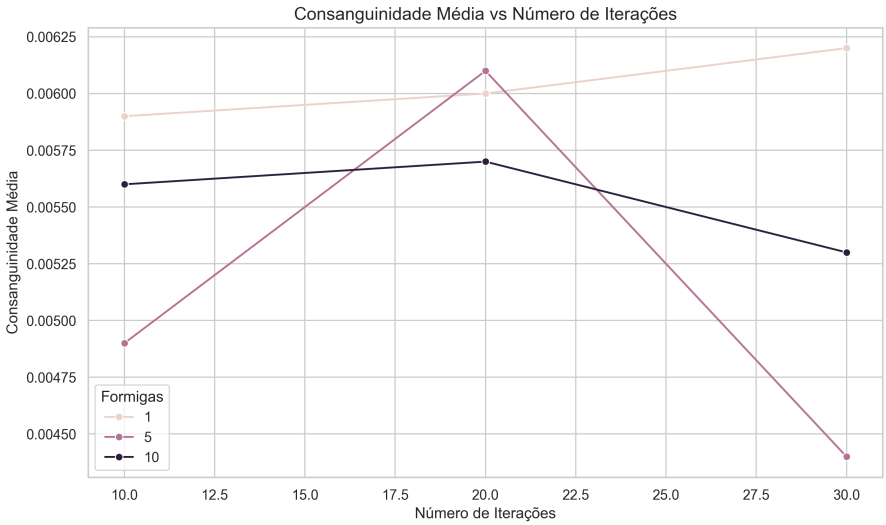


Figura 30 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

também foi registrada nessa mesma configuração, evidenciando uma solução eficiente tanto do ponto de vista de custo/coancestralidade quanto de consanguinidade.

Seguindo a **terceira** configuração, priorizando a equidade entre a influência dos feromônios e da heurística, obteve-se:

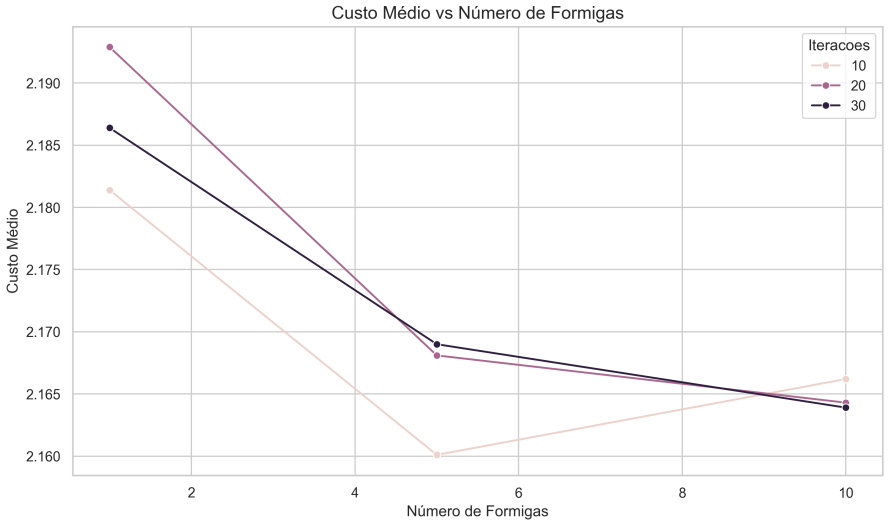


Figura 31 – Custo total em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

As Figuras 31, 32, 33 e 34 apresentam os gráficos que relacionam o custo e a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 1$ e $\beta = 1$. Na Figura 31, observa-se que o custo tende a diminuir com o aumento do número de formigas nos cenários com 20 e 30 iterações. No entanto, no caso com 10 iterações, o custo aumenta ao empregar 10 formigas, contrariando essa tendência. Na Figura 32, nota-se um aumento indesejado: o uso de 1 e 5 formigas é desfavorável nos casos com 10 e 20 iterações, enquanto o cenário com 30 iterações é o único em que o aumento para 10 formigas resulta em redução da consanguinidade média. A Figura 33 demonstra que a redução do custo com o aumento do número de iterações só é consistente no caso em que se utilizam 10 formigas. Nas demais configurações, o custo apresenta flutuações, ora aumentando, ora diminuindo. A melhor solução, em termos de custo, foi obtida com a configuração de 5 formigas e 10 iterações. Por fim, na Figura 34, observa-se que a menor consanguinidade média foi registrada com a configuração de 30 iterações e 10 formigas.

Ao fim é possível comparar o desempenho das diferentes configurações de α e β:

Na figura 35, observa-se que, com apenas uma formiga, a configuração de

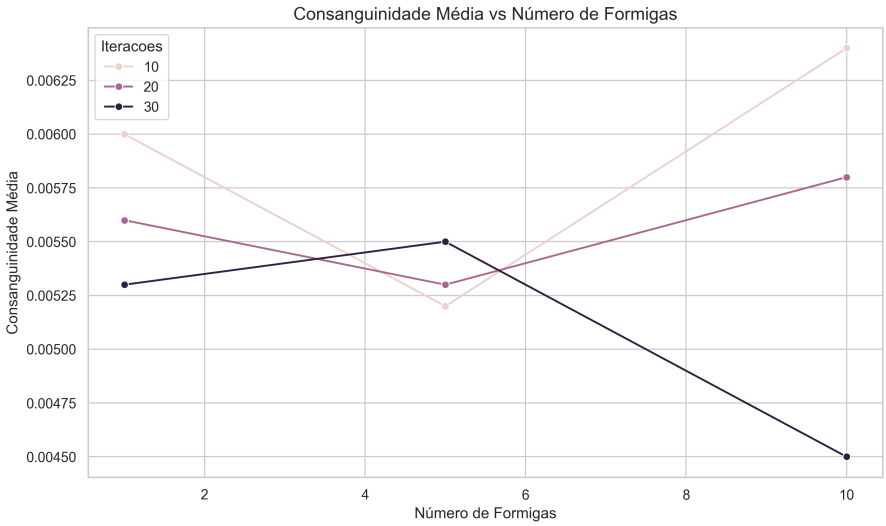


Figura 32 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

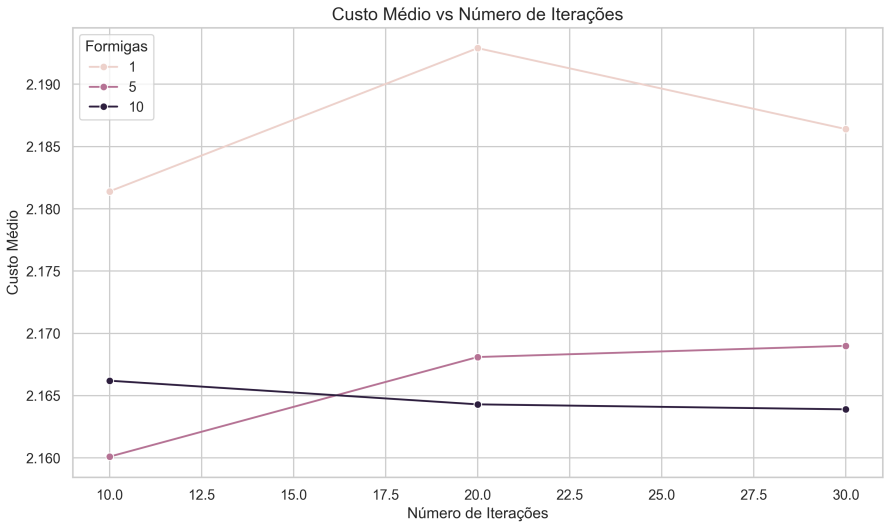


Figura 33 – Custo total em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

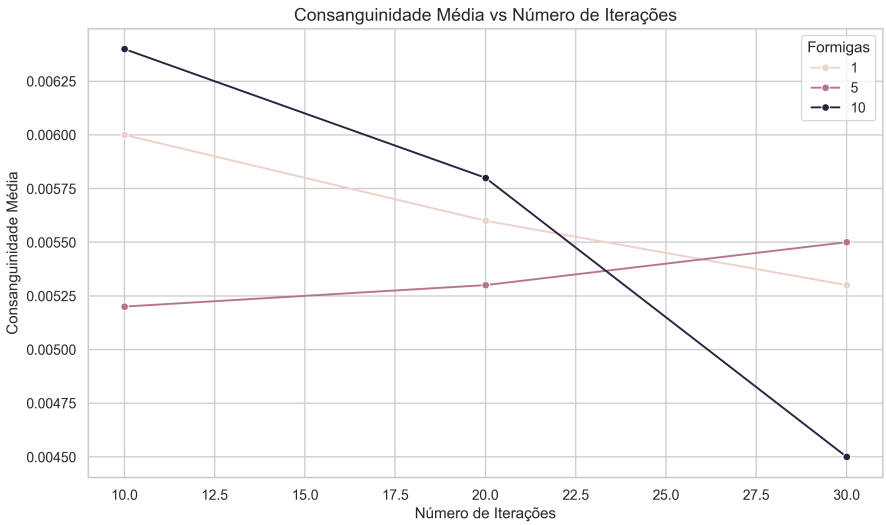
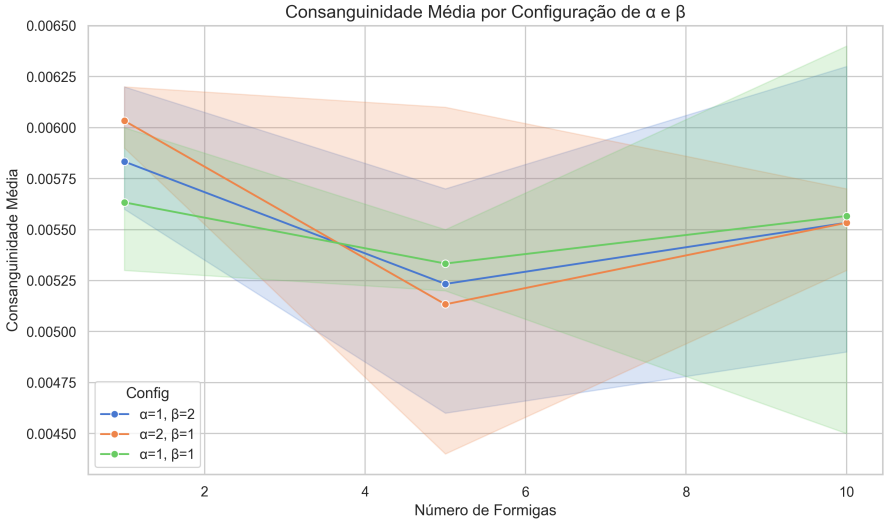


Figura 34 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

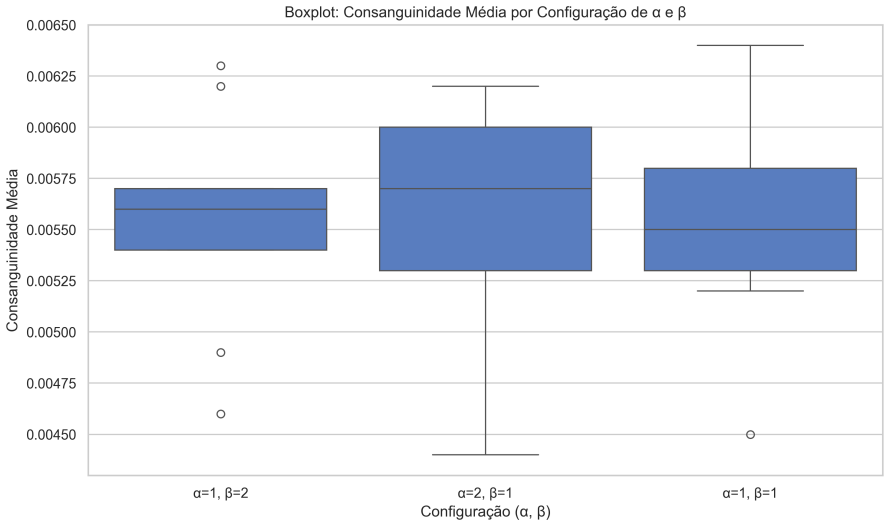
Figura 35 – Consanguinidade média por configuração (Comparação)



Fonte: Autor, (2025)

equilíbrio ($\alpha = 1, \beta = 1$) apresenta o melhor desempenho em termos de consanguinidade. No entanto, à medida que o número de formigas aumenta, a configuração com $\alpha = 2$ e $\beta = 1$, que atribui maior influência à trilha de feromônio em relação à heurística, passa a apresentar os melhores resultados.

Figura 36 – Boxplot de consanguinidade por configuração (Comparação)



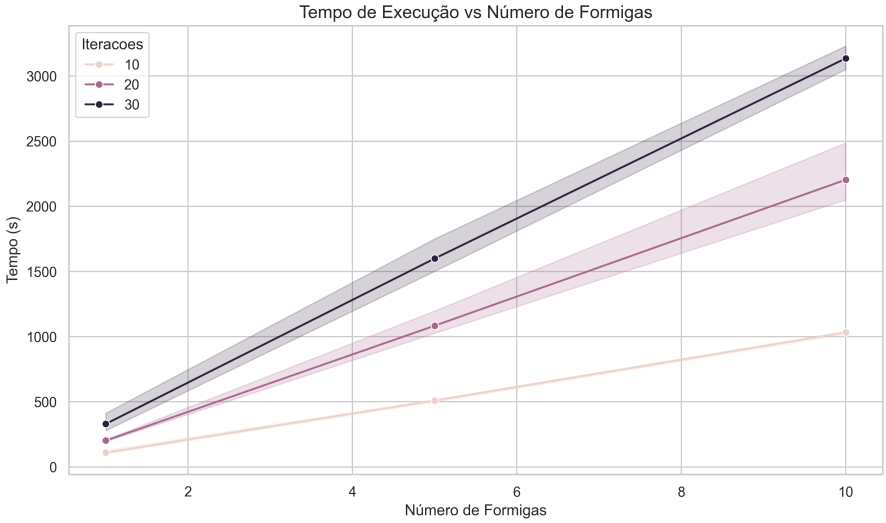
Fonte: Autor, (2025)

Já a figura 36 mostra o boxplot dos custos médios por configuração, evidenciando que a configuração com maior peso para o feromônio ($\alpha = 2$, $\beta = 1$) apresenta maior consistência nos resultados, com menor dispersão e tendência central inferior em relação às demais.

Na figura 37, observa-se que, com 10 iterações, o aumento do número de formigas tem impacto relativamente pequeno no tempo de execução. No entanto, à medida que o número de iterações cresce, esse impacto torna-se mais evidente. Por exemplo, no cenário com 30 iterações, o aumento de 1 para 10 formigas resulta em um tempo de execução aproximadamente seis vezes maior.

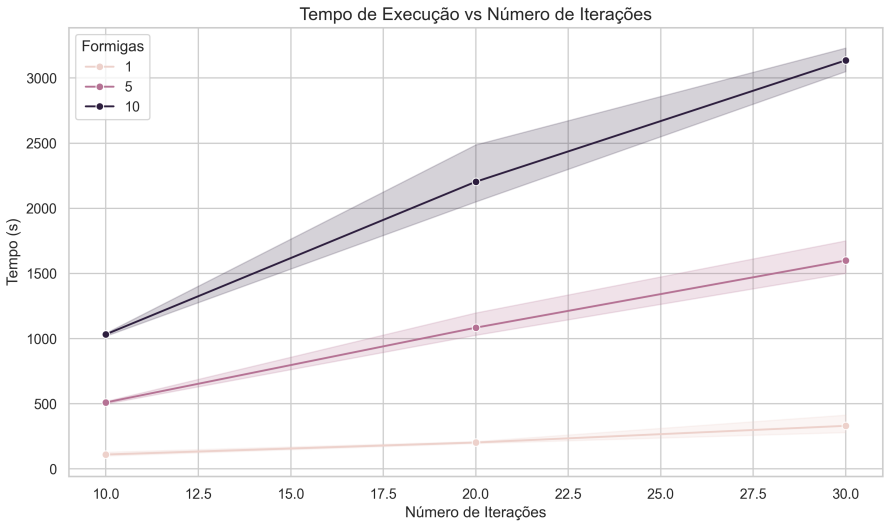
Já na figura 38, nota-se que o tempo de execução apresenta menor sensibilidade ao número de iterações do que ao número de formigas utilizadas. A maior variação é observada quando se utilizam 10 formigas: nesse caso, o tempo de execução triplica ao se passar de 10 para 30 iterações. Essa diferença de comportamento ocorre porque os valores considerados para o número de formigas (1, 5 e 10) e para o número de iterações (10, 20 e 30) têm escalas distintas de crescimento, o que influencia diretamente a forma como esses parâmetros afetam o tempo total de execução.

Figura 37 – Tempo de execução vs Formigas



Fonte: Autor, (2025)

Figura 38 – Tempo de execução vs Iterações



Fonte: Autor, (2025)

6.2.2 Custo: arestas diferentes de 0

Utilizando o algoritmo descrito no Apêndice I, seguindo a **primeira** configuração, priorizando a heurística, obteve-se:

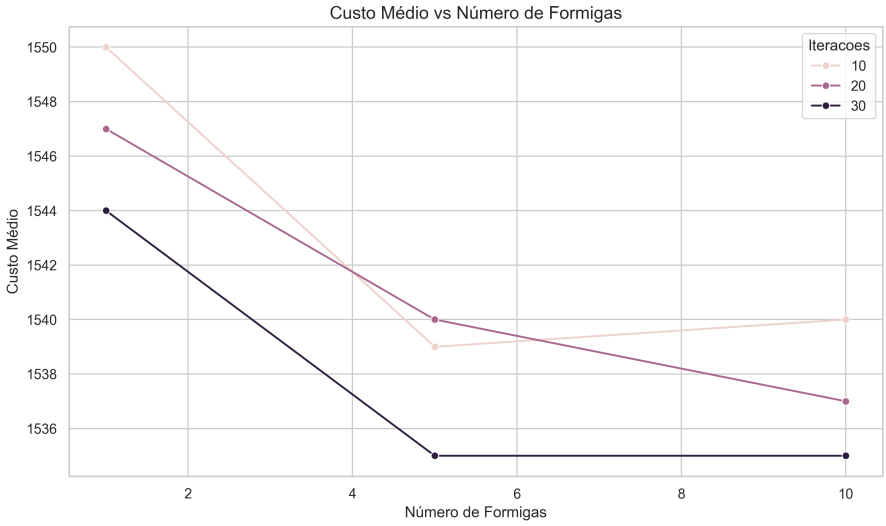


Figura 39 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

As Figuras 39, 40, 41 e 42 apresentam os gráficos que relacionam a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 1$ e $\beta = 2$. Na Figura 39, observa-se que a consanguinidade média tende a diminuir com o aumento do número de formigas utilizadas, sendo semelhante o desempenho dos casos com 10 e 20 iterações. No entanto, observa-se uma convergência do desempenho no caso com 30 iterações, a partir da utilização de 5 formigas. Na Figura 40, observa-se que a consanguinidade média das soluções aumentou nos casos em que a configuração passou de 1 para 5 formigas, especialmente quando se utilizam 10 e 30 iterações. Embora tenham sido observadas melhorias nos resultados com 20 iterações, o menor valor de consanguinidade média foi obtido na configuração com 30 iterações e apenas 1 formiga. A Figura 41 mostra que a consanguinidade média foi reduzida com o aumento do número de iterações. No caso com apenas uma formiga, observa-se uma tendência de queda contínua, enquanto os cenários com 20 e 30 iterações apresentaram o mesmo melhor desempenho ao se utilizar 10 formigas. Por fim, na Figura 42, verifica-se que a menor consanguinidade média registrada ocorreu na configuração com 30 iterações e apenas 1 formiga.

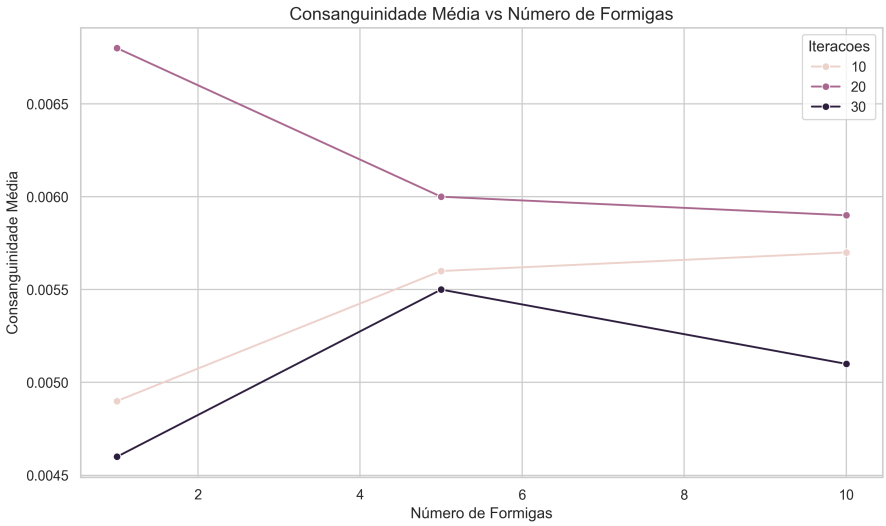


Figura 40 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

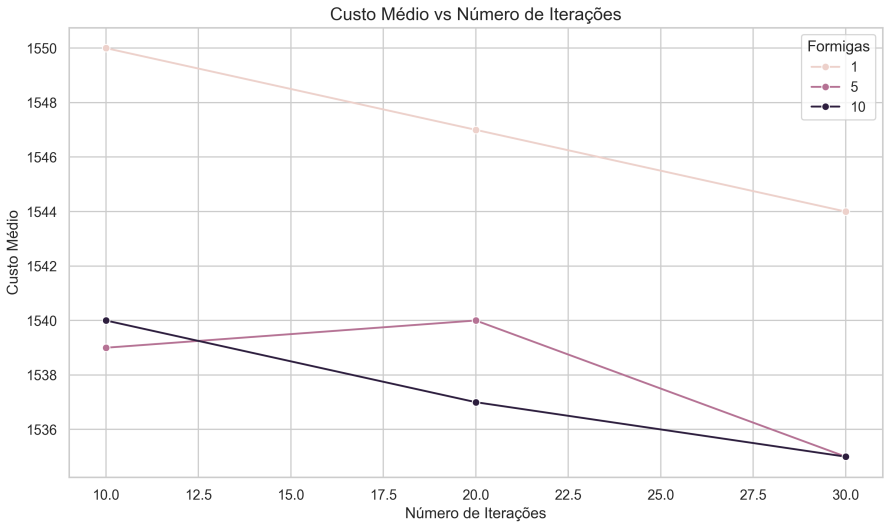


Figura 41 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

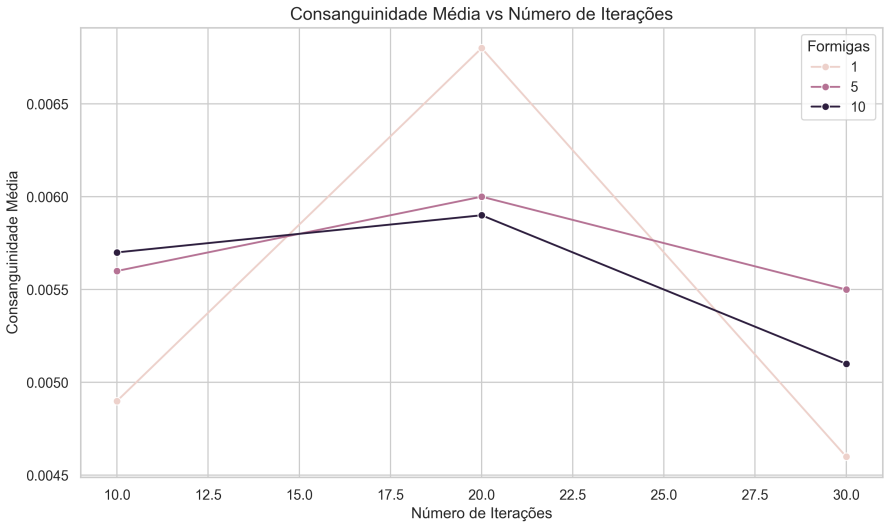


Figura 42 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.

Fonte: Autor, (2025)

Seguindo a **segunda** configuração que prioriza a trilha de feromônios, obteve-se:

As Figuras 43, 44, 45 e 46 apresentam os gráficos que relacionam a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 2$ e $\beta = 1$. Na Figura 43, observa-se que a consanguinidade média tende a diminuir com o aumento do número de formigas, com o melhor resultado obtido com 5 formigas e 10 iterações. Na Figura 44, nota-se um comportamento contrário à tendência observada nas demais configurações, com a consanguinidade média apresentando tendência de aumento à medida que se incrementa o número de formigas. A Figura 45 demonstra que a consanguinidade média manteve-se praticamente constante nos casos com 5 e 10 formigas, independentemente do número de iterações utilizadas. Por fim, na Figura 46, verifica-se que a menor consanguinidade média foi registrada na configuração com 20 iterações e apenas uma formiga.

Seguindo a **terceira** configuração que prioriza a equidade entre a influência dos feromônios e da heurística, obteve-se:

As Figuras 47, 48, 49 e 50 apresentam os gráficos que relacionam a consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 1$ e $\beta = 1$. Na Figura 47, observa-se que a consanguinidade média diminui de maneira consistente apenas no caso com 10 iterações, sendo o melhor resultado obtido com 10 formigas. Na Figura 48, nota-se um comportamento com oscilações na consanguinidade média (aumento e redução), mas com tendência geral de crescimento, o que contraria

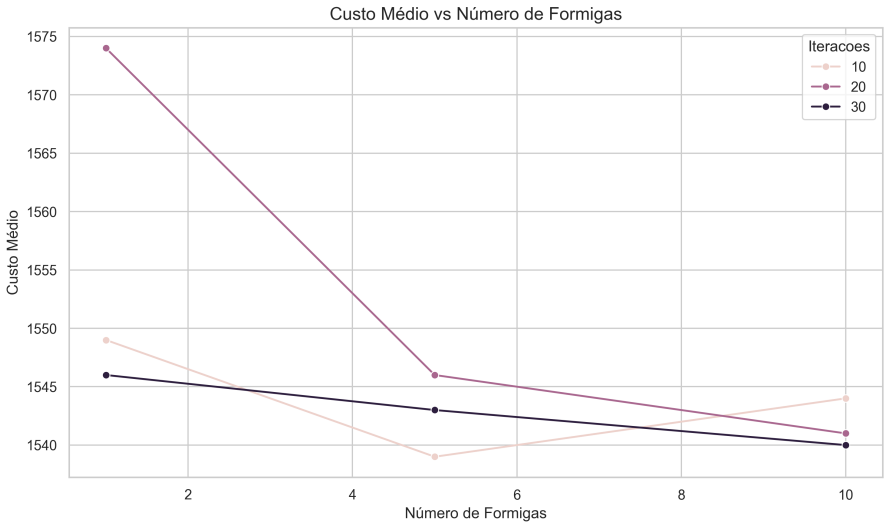


Figura 43 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

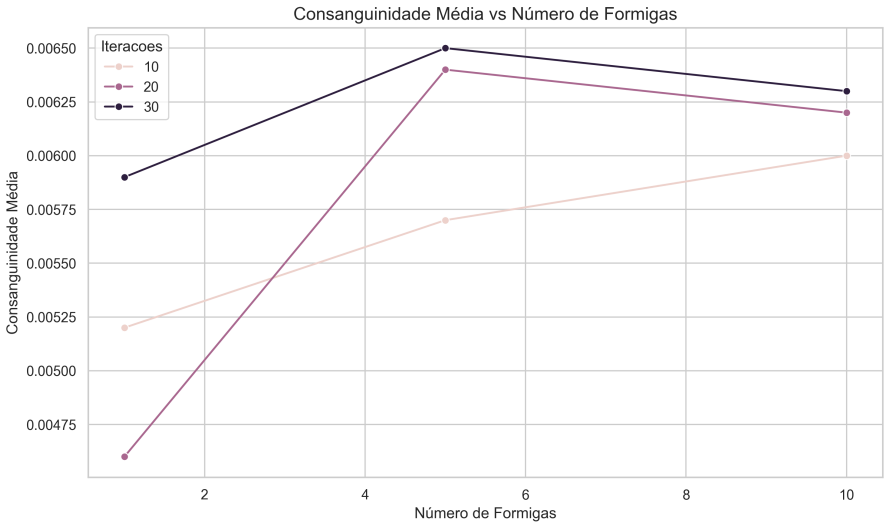


Figura 44 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

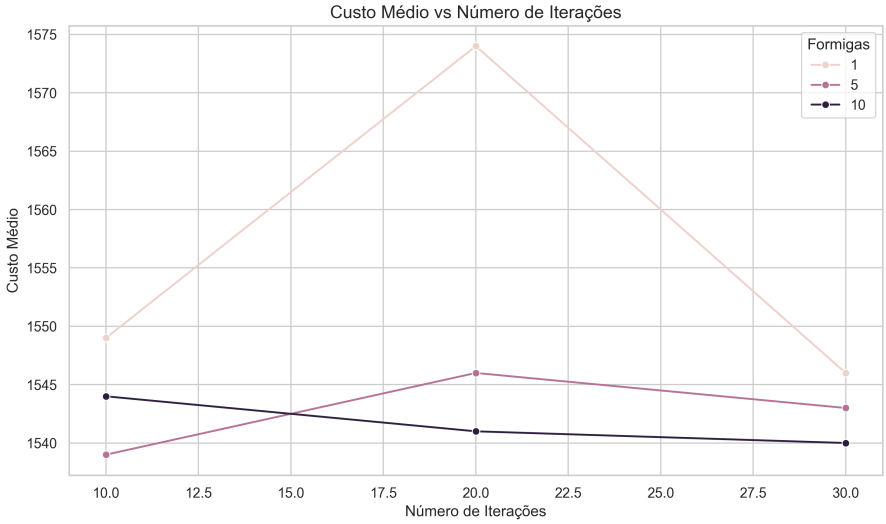


Figura 45 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

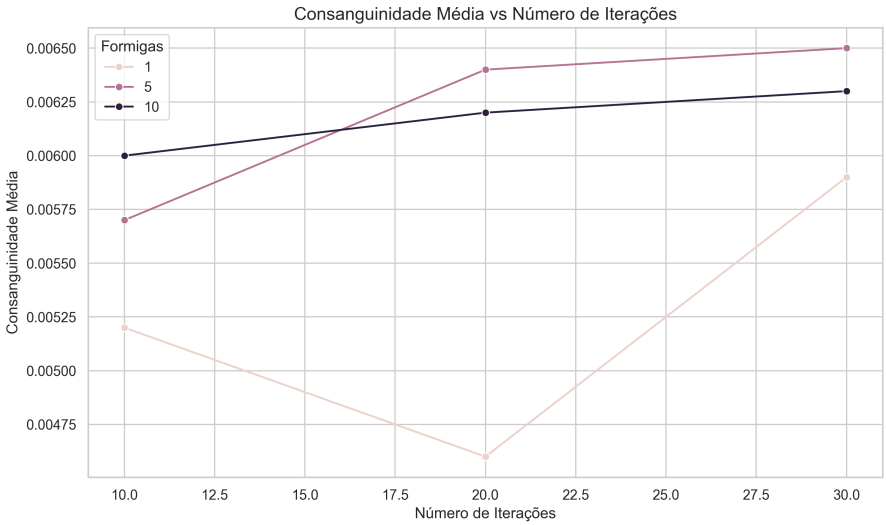


Figura 46 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.

Fonte: Autor, (2025)

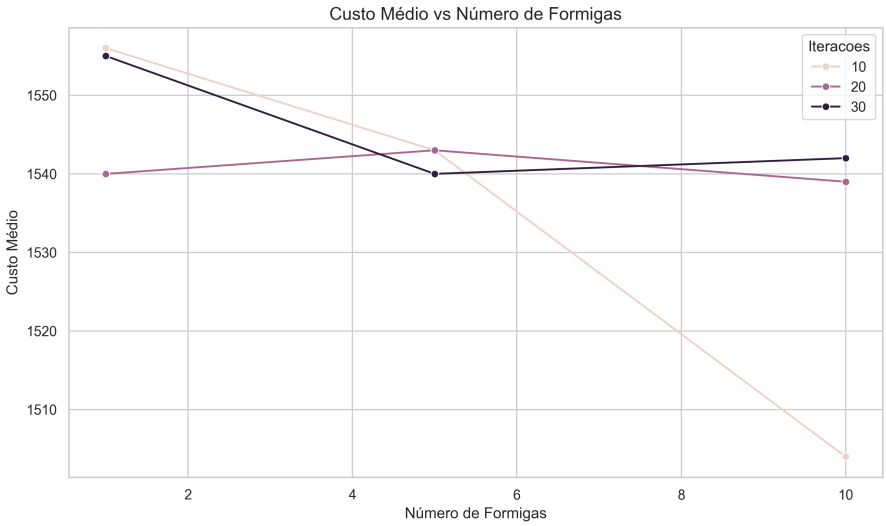


Figura 47 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

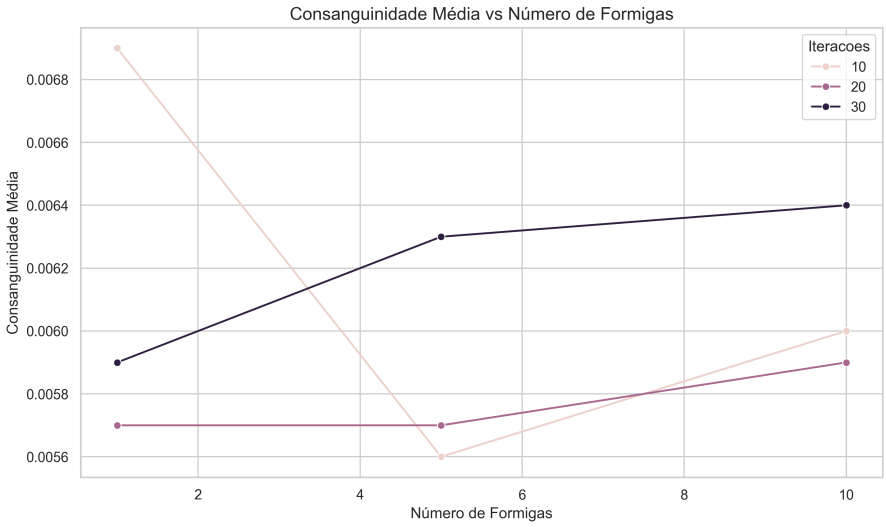


Figura 48 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

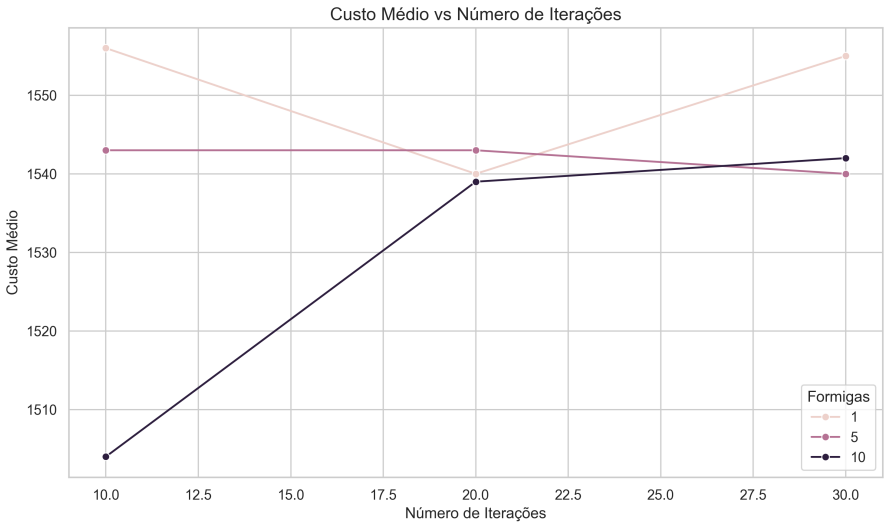


Figura 49 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.

Fonte: Autor, (2025)

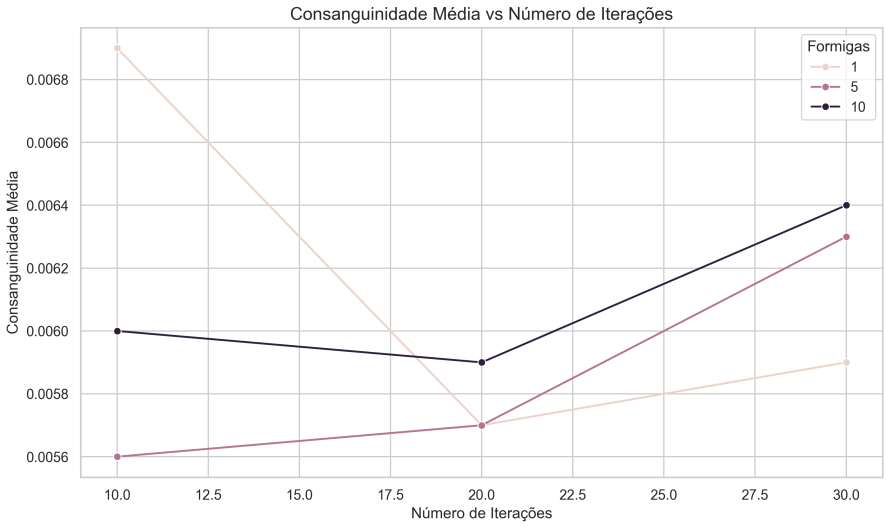


Figura 50 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.

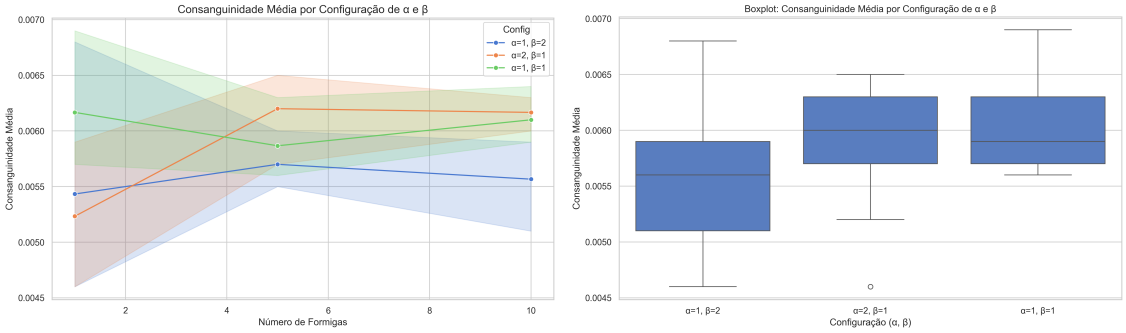
Fonte: Autor, (2025)

o objetivo de minimizar esse valor. A Figura 49 demonstra um baixo decaimento na consanguinidade média, com tendência de aumento conforme se eleva o número de iterações, o que indica menor efetividade da intensificação do processo iterativo nessa configuração. Por fim, na Figura 50, observa-se que a menor consanguinidade média foi registrada na configuração com 20 iterações e 1 ou 5 formigas, embora haja uma leve tendência de aumento ao se utilizar 30 iterações.

Ao fim é possível comparar o desempenho das diferentes configurações de α e β:

Figura 51 – Comparação entre configurações

(a) Consanguinidade média por configuração (b) Boxplot de consanguinidade por configuração



Fonte: Autor, (2025)

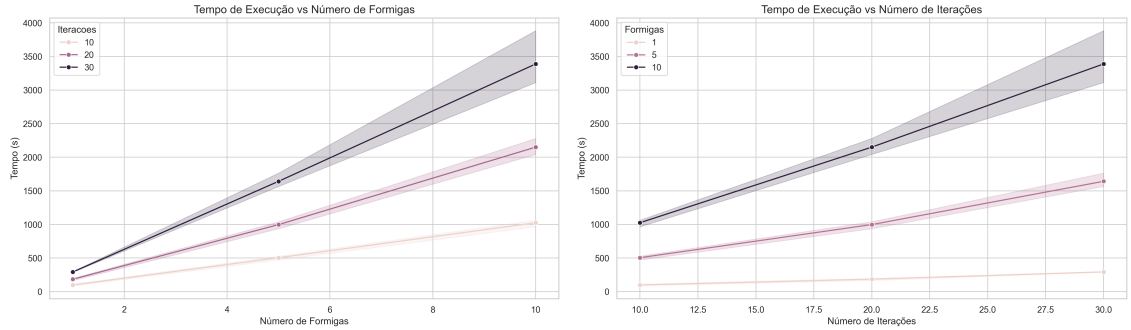
A Figura 51 apresenta os gráficos que comparam o desempenho das diferentes configurações dos parâmetros α e β. Na subfigura 51a, observa-se que, com apenas uma formiga, a configuração que favorece a heurística ($\alpha = 1$, $\beta = 2$) apresenta o melhor desempenho em termos de consanguinidade média. Já a subfigura 51b mostra o boxplot dos custos médios por configuração, evidenciando que a configuração com maior influência da heurística ($\alpha = 1, \beta = 2$) apresenta maior consistência nos resultados e valores médios de custo inferiores aos das demais configurações.

Quando comparado ao desempenho do algoritmo que considera a coancestralidade média como critério de avaliação, o método baseado no número de relações de coancestralidade estritamente positivas apresenta desempenho inferior. Enquanto o primeiro mantém a consanguinidade média dos indivíduos selecionados entre as faixas de 0,0045 e 0,00625, o segundo exibe resultados em faixas superiores, variando de 0,0045 até valores próximos de 0,0070.

A Figura 52 apresenta os gráficos que relacionam o tempo de execução com o número de formigas empregadas e com o número de iterações utilizadas. Na subfigura 52a, observa-se que, à medida que o número de iterações aumenta, o tempo de execução

Figura 52 – Comparações entre número de formigas/iterações e tempo de execução

(a) Tempo de execução vs Formigas (b) Tempo de execução vs Iterações



Fonte: Autor, (2025)

também cresce, aproximando-se dos 4 mil segundos. Esse desempenho é inferior ao observado no método baseado na coancestralidade média, que apresentou tempos próximos a 3 mil segundos. Já na subfigura 52b, nota-se que o tempo de execução é menos sensível ao número de iterações do que ao número de formigas utilizadas, indicando que o aumento do número de formigas tem maior impacto no tempo total do que o aumento do número de iterações.

6.2.3 Heurística: Maximização de índice de seleção + minimização de coancestralidade

Utilizando o algoritmo descrito no Apêndice H substituindo a linha de código `heuristica = 1 / (1 + soma_parentesco[c['id']])` pela linha `heuristica = c['iecc'] / (1 + soma_parentesco[c['id']])`, de modo a buscar a otimização conjunta dos aspectos como descrito na Definição 16.

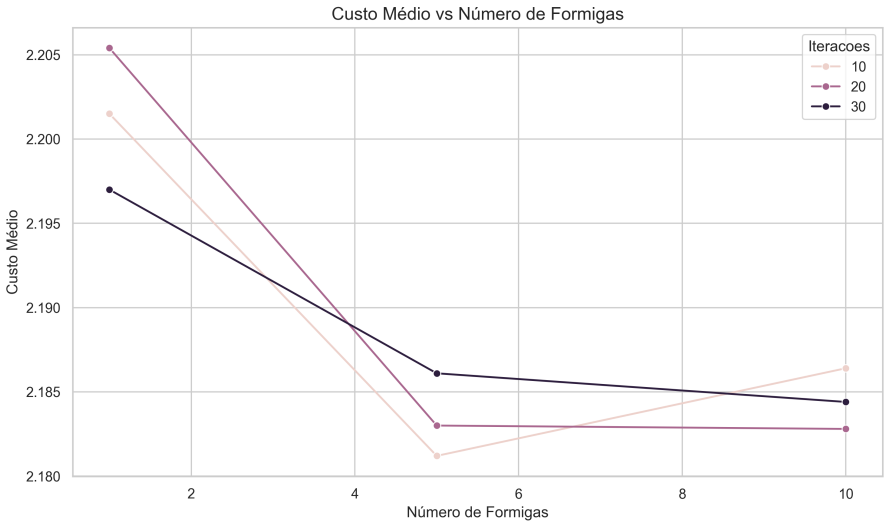
Seguindo a **primeira** configuração, priorizando a heurística, obteve-se:

As Figuras 53, 54, 55 e 56 apresentam os gráficos referentes ao custo e à consanguinidade média dos indivíduos que compõem a solução final na configuração com $\alpha = 1$ e $\beta = 2$.

Além disso, os resultados do índice de seleção estão representados nas Figuras 57, 58, 59 e 60:

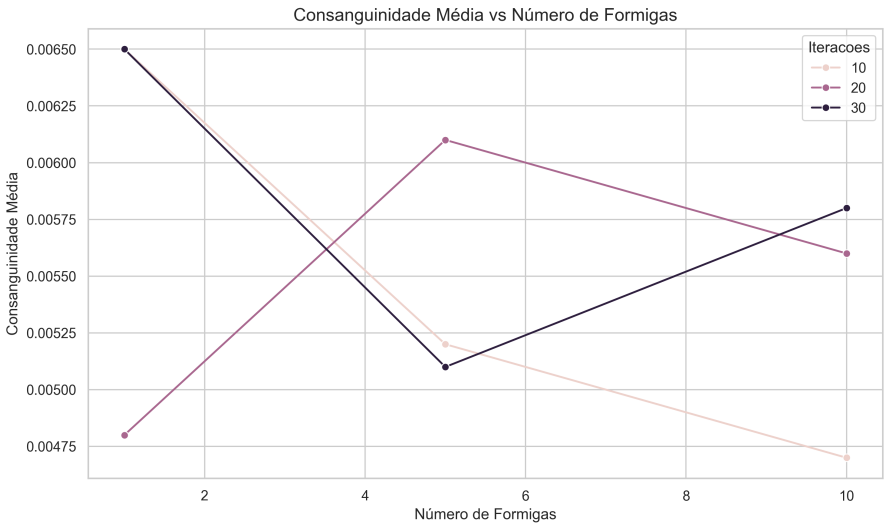
As Figuras 57 a 60 permitem avaliar como o índice de seleção médio variou ao longo das execuções, considerando tanto o número de formigas quanto o número de iterações.

Figura 53 – Custo médio em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.



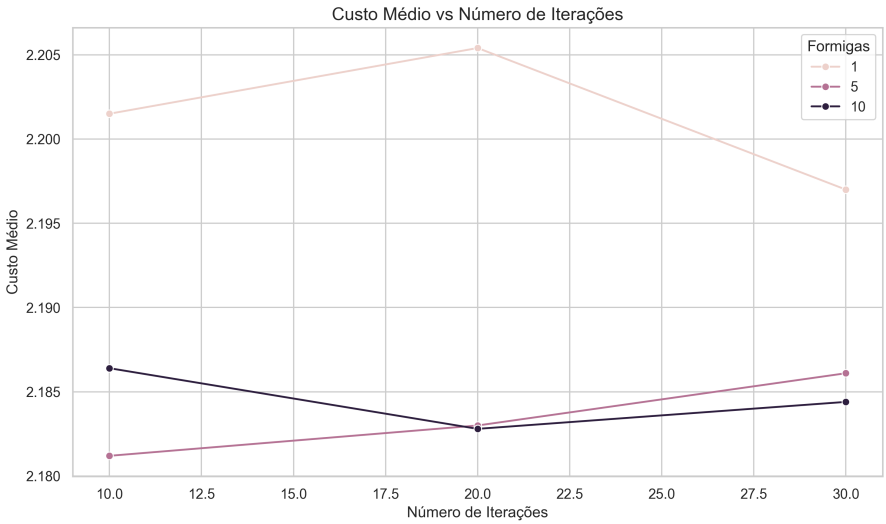
Fonte: Autor, (2025)

Figura 54 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.



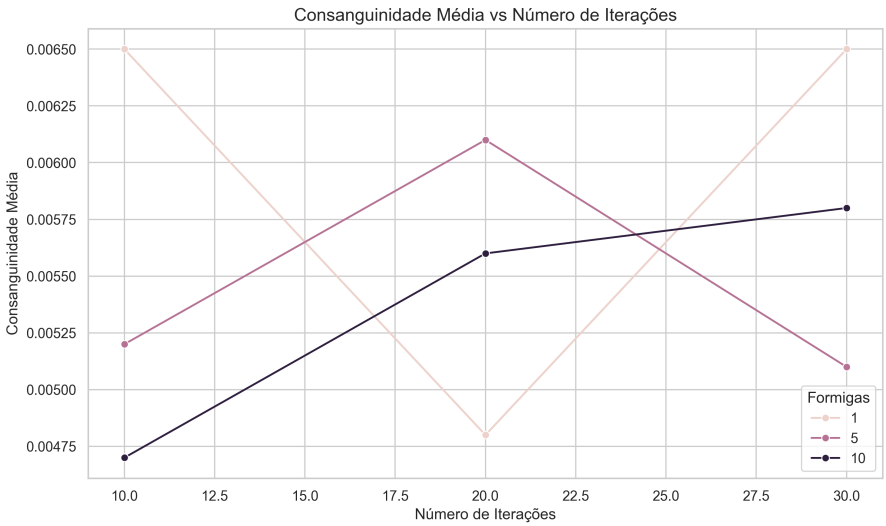
Fonte: Autor, (2025)

Figura 55 – Custo médio em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.



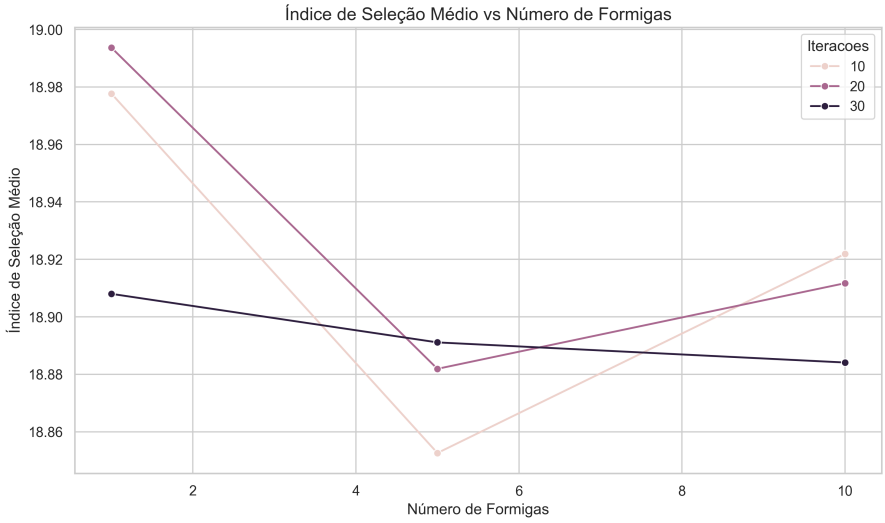
Fonte: Autor, (2025)

Figura 56 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.



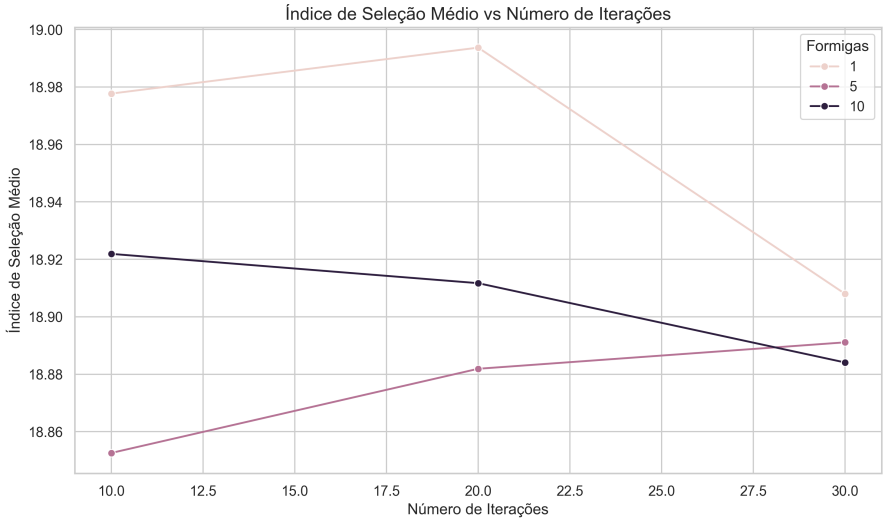
Fonte: Autor, (2025)

Figura 57 – Índice de seleção médio em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 2$.



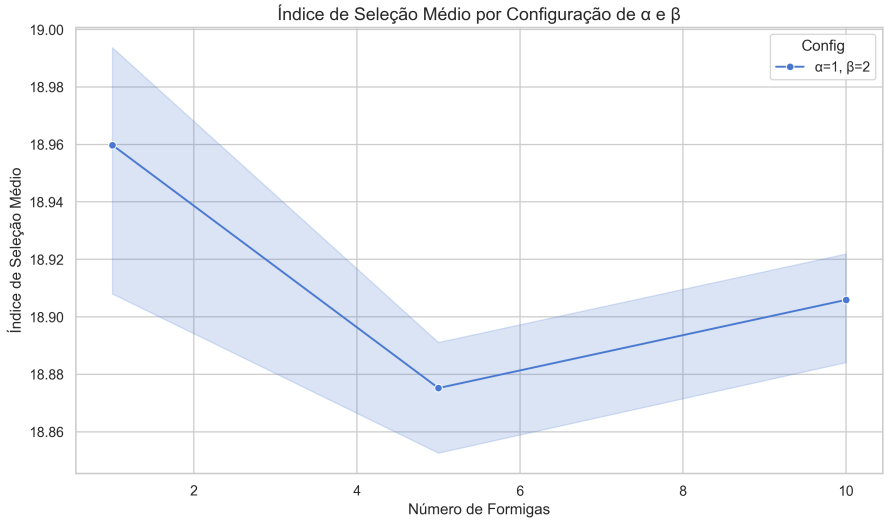
Fonte: Autor, (2025)

Figura 58 – Índice de seleção médio em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 2$.



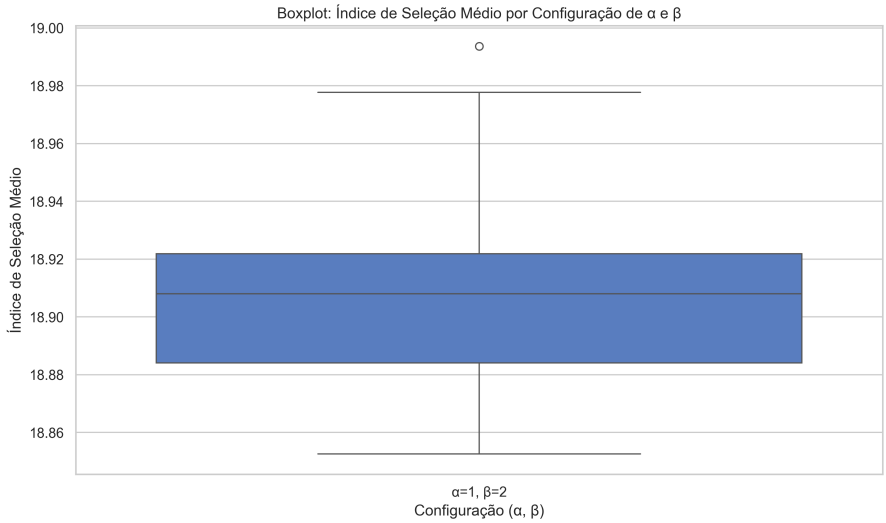
Fonte: Autor, (2025)

Figura 59 – Índice de seleção médio por configuração para $\alpha = 1$ e $\beta = 2$.



Fonte: Autor, (2025)

Figura 60 – Boxplot do índice de seleção por configuração para $\alpha = 1$ e $\beta = 2$.

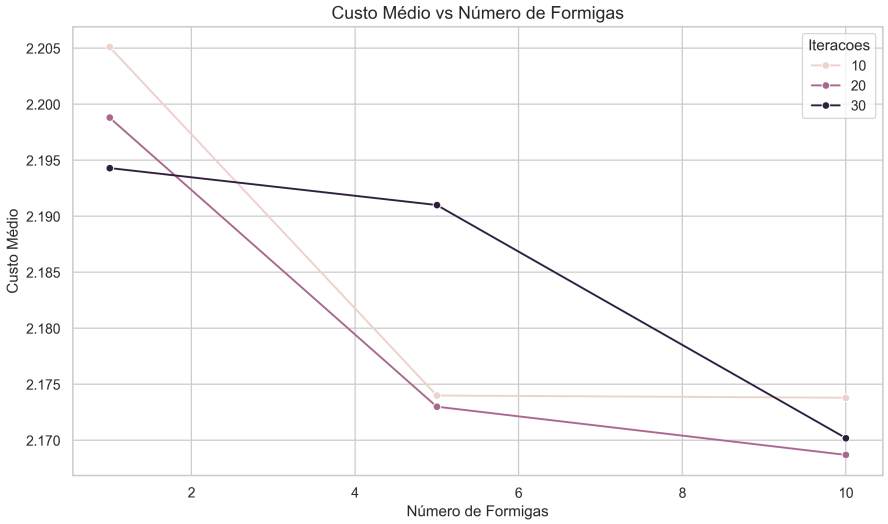


Fonte: Autor, (2025)

Na Figura 57, observa-se uma tendência geral de redução no índice de seleção médio à medida que o número de formigas aumenta, com os valores mais altos concentrados na configuração com 1 formiga. Os casos com 10 e 20 iterações mostram declínios mais suaves, enquanto com 30 iterações o declínio é mais acentuado, indicando que um excesso de agentes pode levar a soluções subótimas devido a uma exploração excessiva do espaço de busca. Já na Figura 58, o comportamento varia conforme o número de formigas: com 1 formiga, o índice sobe inicialmente, atinge um pico em torno de 20 iterações e depois declina levemente, sugerindo convergência prematura; com 5 formigas, há um aumento gradual e consistente, apontando para uma melhoria progressiva; com 10 formigas, ocorre um declínio contínuo, possivelmente devido a uma diversificação excessiva que dilui o foco na maximização do índice. A Figura 59 resume o índice médio por combinação de parâmetros, destacando que as melhores performances ocorrem com poucas formigas e iterações moderadas. Por fim, o boxplot na Figura 60 revela a variabilidade dos resultados, com algumas configurações exibindo maior dispersão e presença de outliers inferiores, o que indica instabilidade em execuções com mais formigas.

Seguindo a **segunda** configuração que prioriza a trilha de feromônios, obteve-se:

Figura 61 – Custo médio em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.

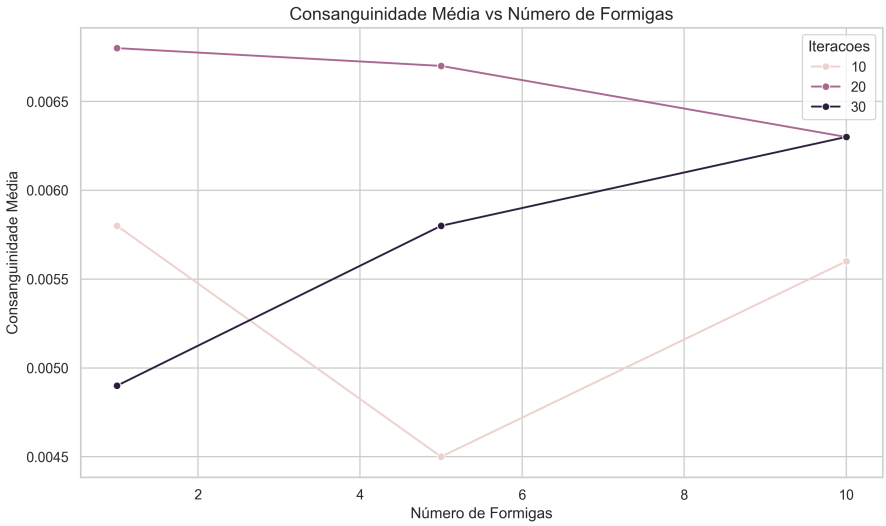


Fonte: Autor, (2025)

Na Figura 61 é possível notar uma redução no custo total conforme ocorre o aumento do número de iterações e formigas.

Na Figura 62, observa-se que o aumento do número de formigas, mantendo 20

Figura 62 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.



Fonte: Autor, (2025)

iterações, resulta em uma redução progressiva da consanguinidade média. Em contraste, nos demais métodos analisados ocorre um aumento desse valor.

Na Figura 63, nota-se que a redução do custo ocorre apenas no cenário em que é utilizada uma única formiga.

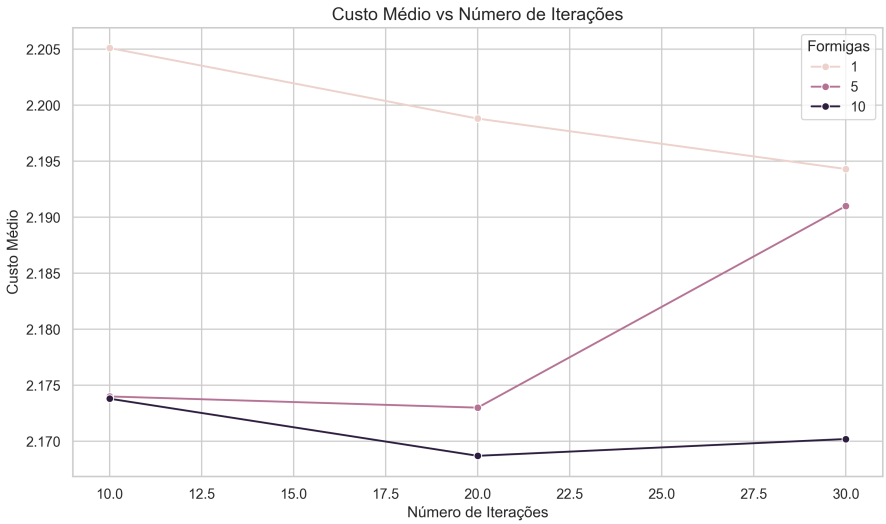
Na Figura 64, observa-se que, para 5 e 10 formigas, a consanguinidade cresce até aproximadamente 20 iterações e, em seguida, decresce de forma consistente. Já no caso de apenas 1 formiga, a consanguinidade aumenta até cerca de 20 iterações e se mantém praticamente estável nas iterações subsequentes. Esses resultados sugerem que o aumento do número de formigas favorece a redução da consanguinidade em iterações mais avançadas.

Na Figura 65, o índice de seleção apresenta um declínio geral com o aumento de formigas, mas com flutuações, como um vale em torno de 5 formigas para 10 iterações, indicando pontos de instabilidade. Para iterações mais altas, o declínio é mais linear.

Na Figura 66, com 1 formiga, há uma queda inicial seguida de recuperação; com 5 formigas, flutuações com tendência ascendente; com 10 formigas, declínio suave. Isso sugere que essa configuração, com maior ênfase em α, promove exploração inicial, mas pode sobrecarregar com muitos agentes.

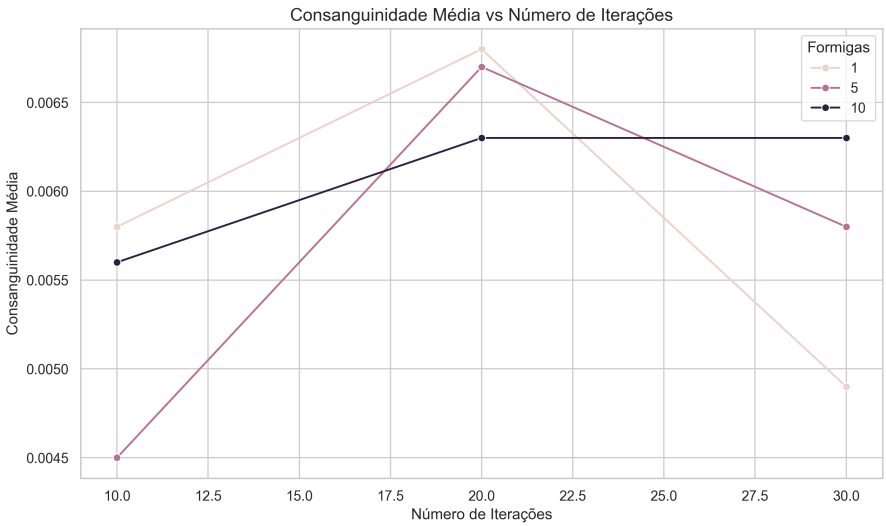
A Figura 67 destaca combinações ótimas, como poucas formigas e muitas iterações, enquanto o boxplot na Figura 68 mostra menor variabilidade em configurações equilibradas, com poucos outliers.

Figura 63 – Custo médio em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.



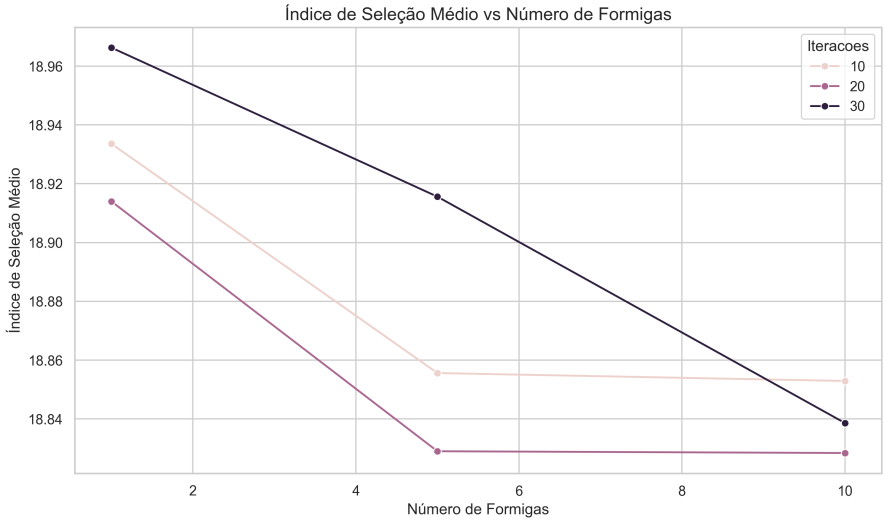
Fonte: Autor, (2025)

Figura 64 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.



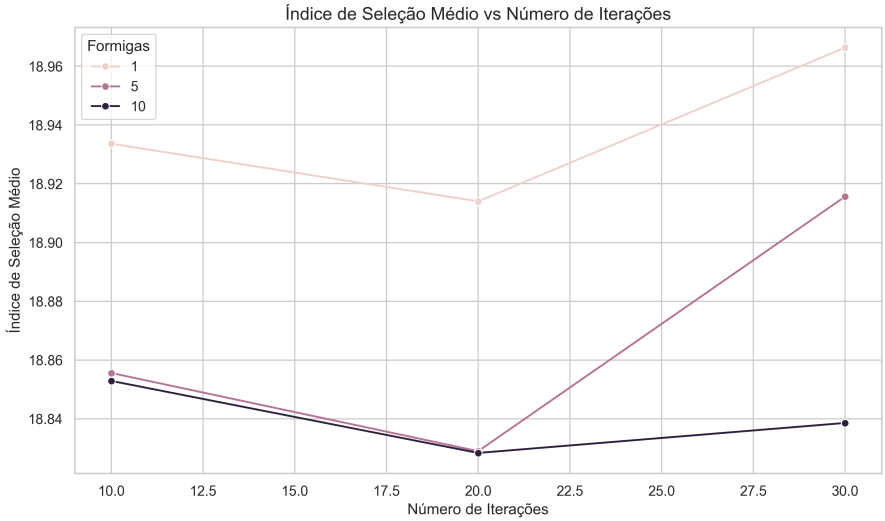
Fonte: Autor, (2025)

Figura 65 – Índice de seleção médio em função do número de formigas para a configuração com $\alpha = 2$ e $\beta = 1$.



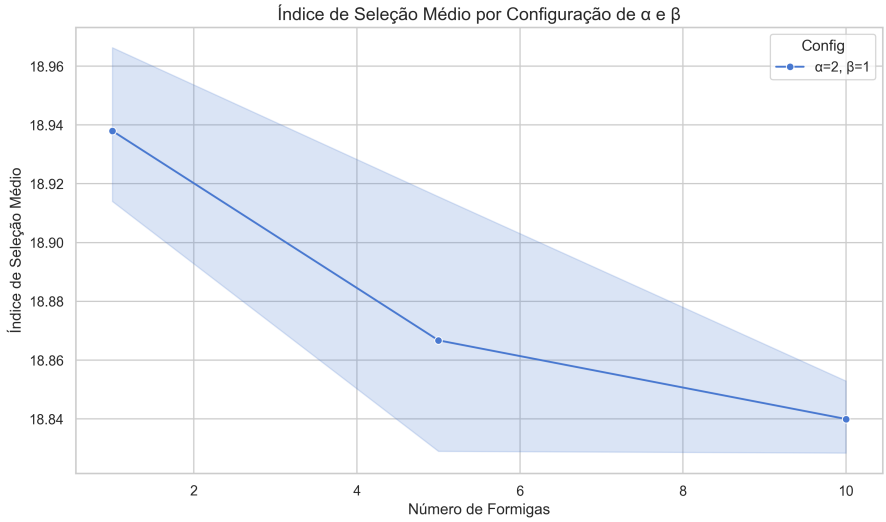
Fonte: Autor, (2025)

Figura 66 – Índice de seleção médio em função do número de iterações para a configuração com $\alpha = 2$ e $\beta = 1$.



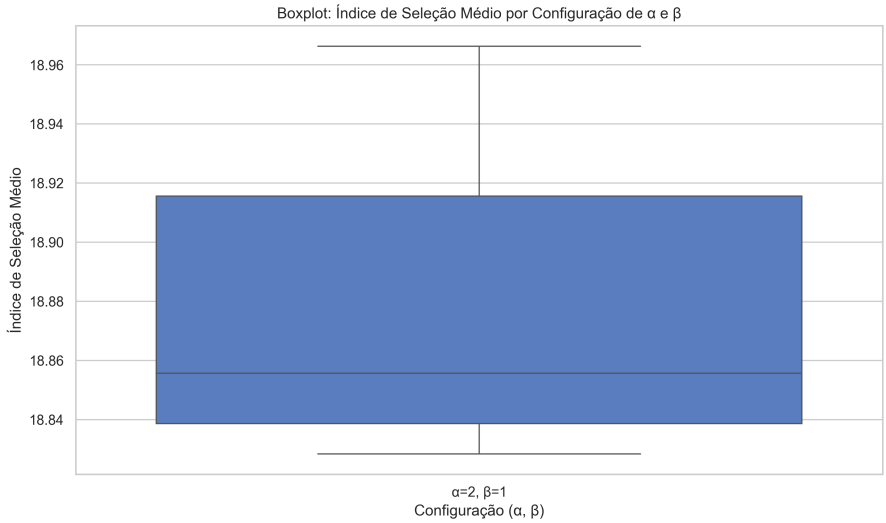
Fonte: Autor, (2025)

Figura 67 – Índice de seleção médio por configuração para $\alpha = 2$ e $\beta = 1$.



Fonte: Autor, (2025)

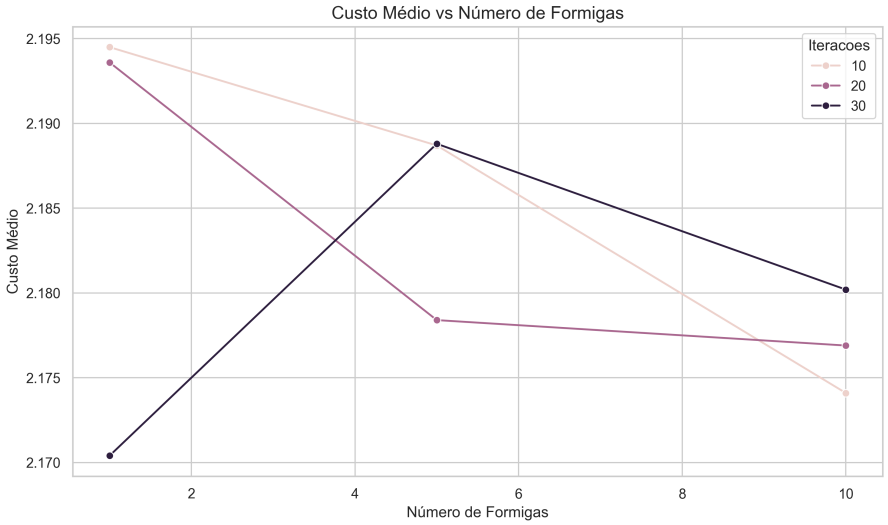
Figura 68 – Boxplot do índice de seleção por configuração para $\alpha = 2$ e $\beta = 1$.



Fonte: Autor, (2025)

Seguindo a **terceira** configuração que prioriza a equidade entre a influência dos feromônios e da heurística, obteve-se:

Figura 69 – Custo médio em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.



Fonte: Autor, (2025)

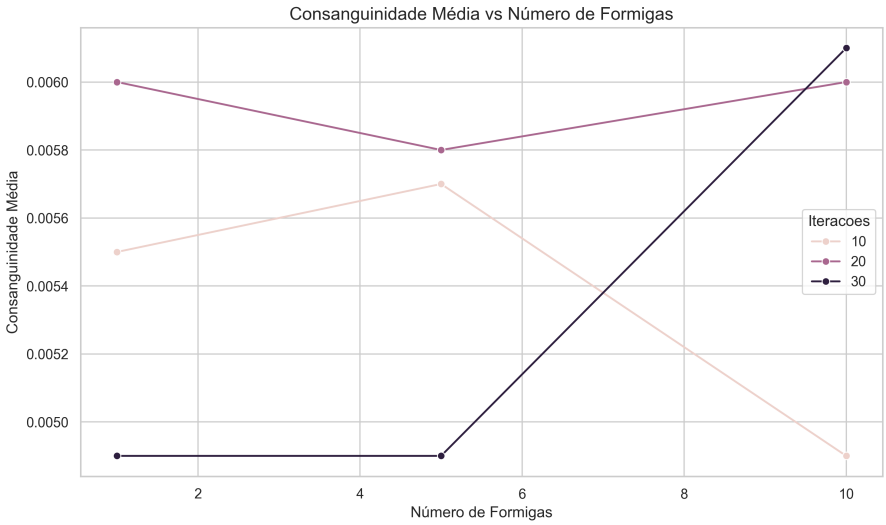
Na Figura 73, há um declínio inicial seguido de estabilização ou leve recuperação em alguns casos, com o melhor desempenho em iterações intermediárias. Na Figura 74, as linhas mostram tendências mistas: ascendente para poucas formigas, mas com picos e vales para mais formigas, indicando um equilíbrio sensível aos parâmetros. A Figura 75 identifica configurações ideais, como 5 formigas e 20 iterações, enquanto o boxplot na Figura 76 evidencia baixa variabilidade geral, com dispersão maior em extremos.

Comparação entre configurações

Ao fim é possível comparar o desempenho das diferentes configurações de α e β tanto para a consanguinidade quanto para o índice de seleção:

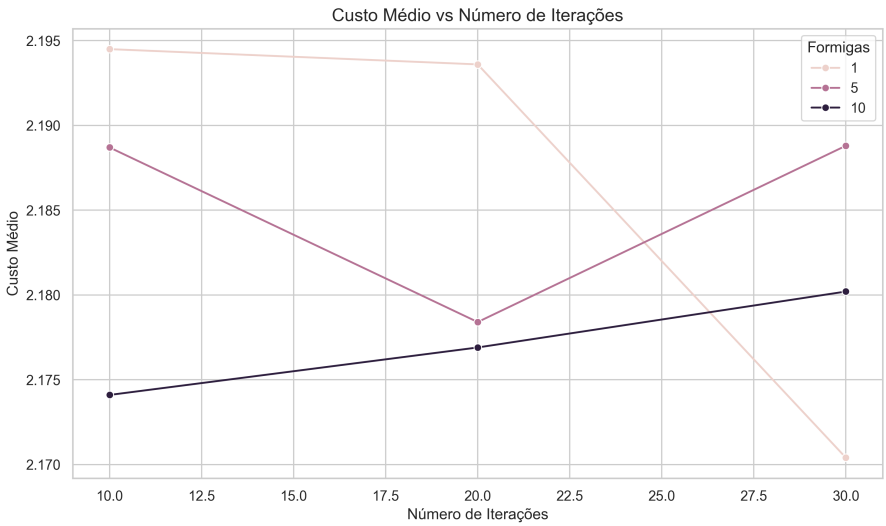
A Figura 77 apresenta os gráficos que comparam o desempenho das diferentes configurações dos parâmetros α e β em relação tanto à consanguinidade média quanto ao índice de seleção médio. Observa-se que a configuração com $\alpha = 1$ e $\beta = 2$ oferece o maior índice de seleção médio, mas com maior variabilidade na consanguinidade, enquanto $\alpha = 2$ e $\beta = 1$ minimizam a consanguinidade de forma mais consistente, e $\alpha = 1$ e $\beta = 1$ fornecem um equilíbrio intermediário com menor dispersão geral.

Figura 70 – Consanguinidade média em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.



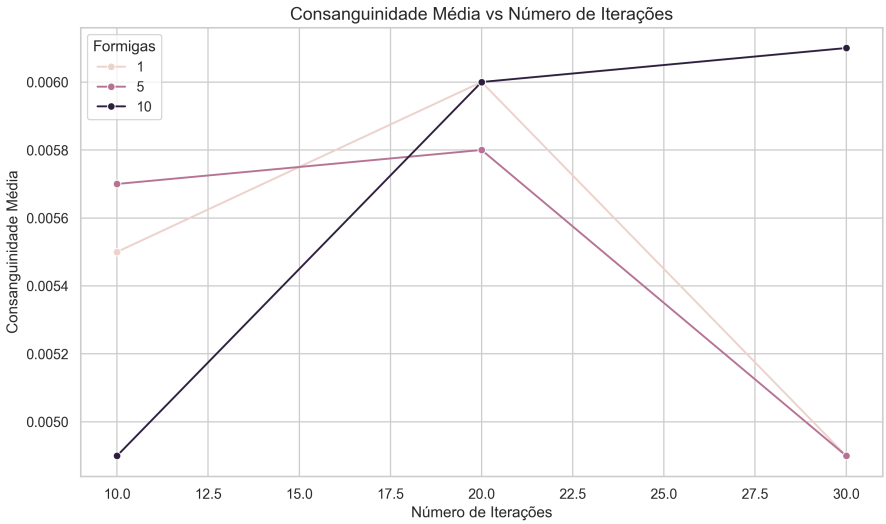
Fonte: Autor, (2025)

Figura 71 – Custo médio em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.



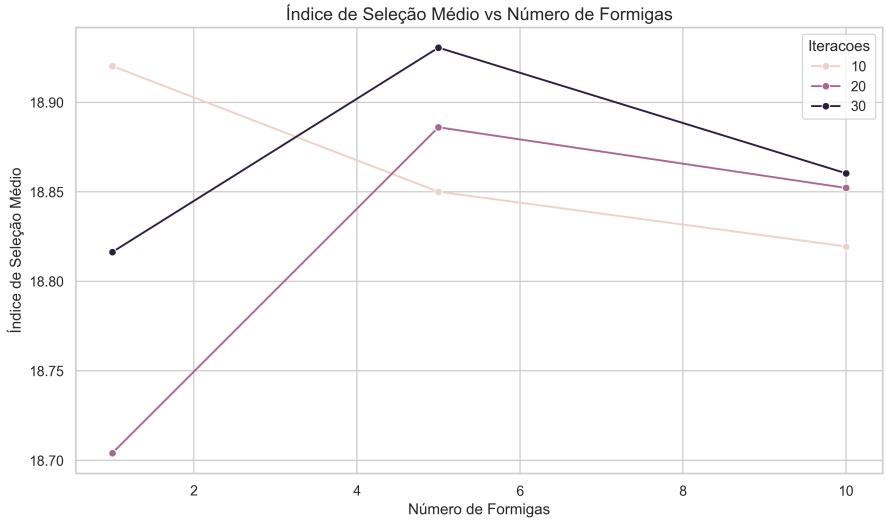
Fonte: Autor, (2025)

Figura 72 – Consanguinidade média em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.



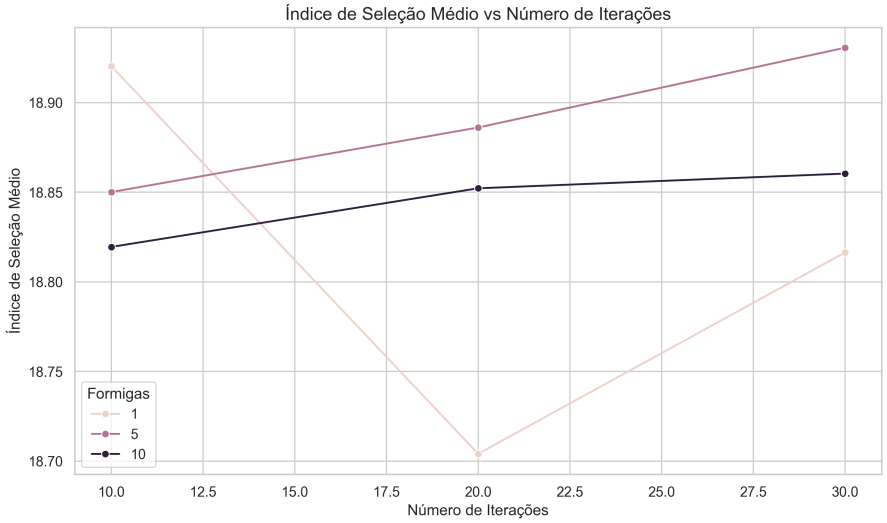
Fonte: Autor, (2025)

Figura 73 – Índice de seleção médio em função do número de formigas para a configuração com $\alpha = 1$ e $\beta = 1$.



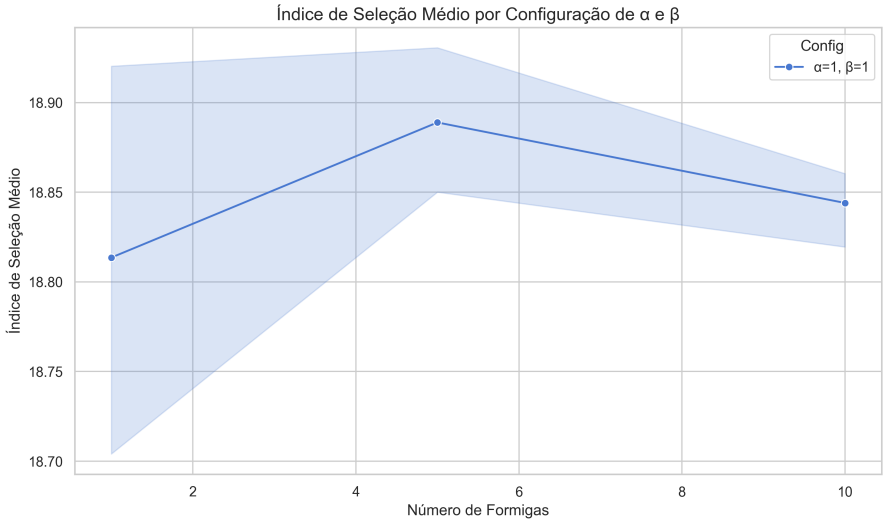
Fonte: Autor, (2025)

Figura 74 – Índice de seleção médio em função do número de iterações para a configuração com $\alpha = 1$ e $\beta = 1$.



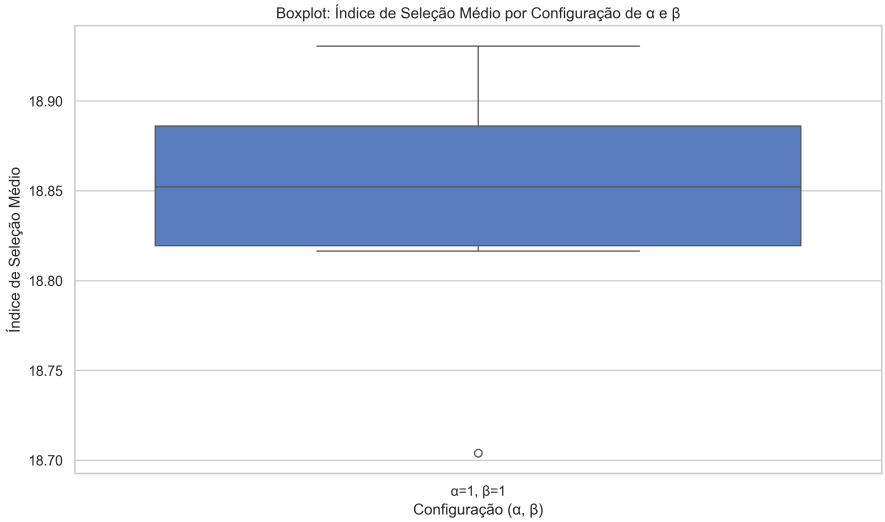
Fonte: Autor, (2025)

Figura 75 – Índice de seleção médio por configuração para $\alpha = 1$ e $\beta = 1$.



Fonte: Autor, (2025)

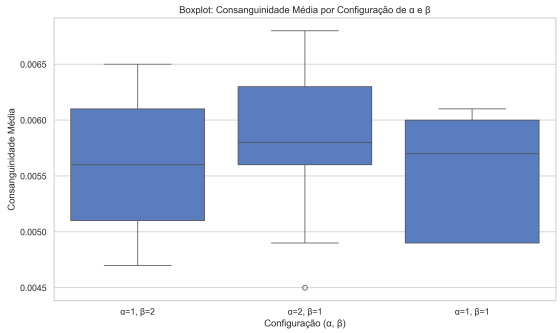
Figura 76 – Boxplot do índice de seleção por configuração para $\alpha = 1$ e $\beta = 1$.



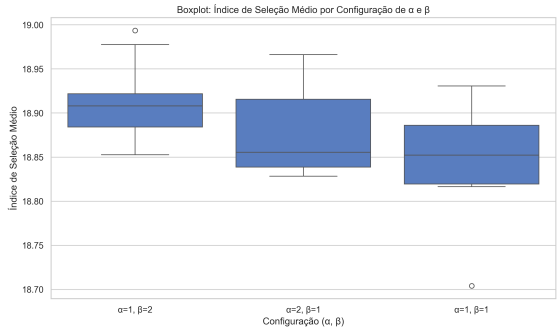
Fonte: Autor, (2025)

Figura 77 – Comparação entre configurações

(a) Boxplot de consanguinidade por configuração



(b) Boxplot do índice de seleção por configuração



Fonte: Autor, (2025)

6.3 Tempo de execução em função da quantidade de pares

Foram realizados testes utilizando o algoritmo que emprega o método de avaliação do custo por meio da coancestralidade média (Apêndice H) com o objetivo de avaliar o comportamento do algoritmo e o tempo de execução por iteração única, conforme o número de pares a serem organizados. Para isso, foram definidos alguns parâmetros fixos, como 1 iteração, 1 formiga, $\alpha = 2{,}0$ e $\beta = 1{,}0$, além de parâmetros adaptativos. Como exemplo de parâmetro adaptativo, destaca-se o limite máximo de uso dos machos, que varia de acordo com o número total de possíveis pares envolvidos em cada execução. Esses parâmetros estão detalhados na Tabela 24. Embora o número de machos tenha sido mantido fixo, foram realizados acréscimos graduais no número de fêmeas. Essa variação impacta diretamente a quantidade de produtos possíveis e, consequentemente, o número de relações de parentesco a serem consideradas pelo algoritmo. Como evidenciado na linha "Relações de parentesco" da Tabela 24, o crescimento no número de pares aumenta o número de relações a serem avaliadas, seguindo a função:

$$\text{Relações de parentesco} = \frac{\text{produtos} \times (\text{produtos} - 1)}{2} \tag{33}$$

Essa função representa a diagonal superior da matriz de coancestralidade que há entre os possíveis produtos. Esse aumento no número de relações representa um desafio computacional, uma vez que impacta diretamente o tempo de execução do algoritmo.

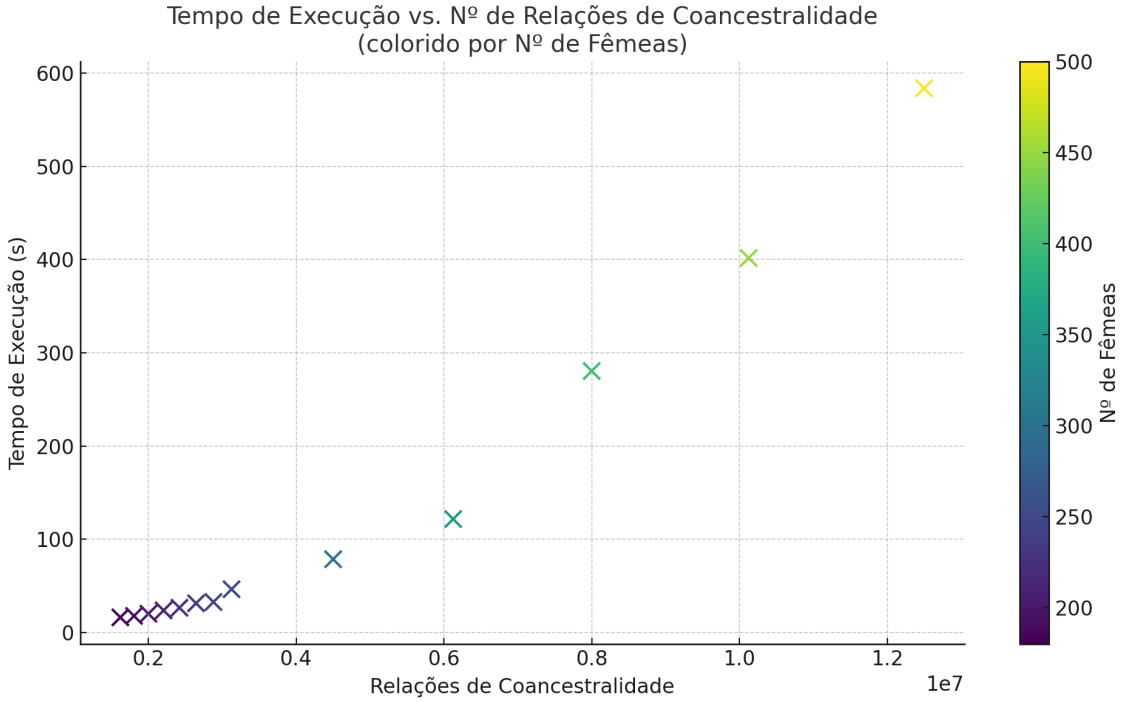
Tabela 24 – Parâmetros utilizados no algoritmo MMAS - Tempo em função dos pares

Nº machos	**Nº fêmeas**	**Produtos possíveis**	**Relações de parentesco**	**Limite por macho**	**Tempo (s)**
10	180	1.800	1.619.100	20	16,25
10	190	1.900	1.804.050	21	17,8
10	200	2.000	1.999.000	22	20,13
10	210	2.100	2.203.950	23	23,71
10	220	2.200	2.418.900	24	26,59
10	230	2.300	2.643.850	25	31,38
10	240	2.400	2.878.800	26	32,71
10	250	2.500	3.123.750	27	46,42
10	300	3.000	4.498.500	32	78,59
10	350	3.500	6.123.250	37	121,68
10	400	4.000	7.998.000	42	280,47
10	450	4.500	10.122.750	47	401,68
10	500	5.000	12.497.500	52	583,69

Fonte: Autor (2025)

A Figura 78 mostra a evolução do tempo de execução conforme aumenta o número de relações de parentesco. É possível observar na figura a formação de uma curva que reflete o grau do polinômio do tempo de execução, dados o número de relações e o tempo de execução necessário. Esse padrão pode representar um gargalo computacional relevante em cenários com rebanhos extensos (na casa dos milhares), já que o algoritmo precisa buscar soluções viáveis percorrendo e avaliando as relações de parentesco entre os produtos.

Figura 78 – Relações de parentesco por tempo de execução



Fonte: Autor, (2025)

Além disso, é importante considerar a quantidade de memória RAM disponível na máquina, pois ela será necessária para processar as matrizes de parentesco entre os produtos. Nos testes realizados, utilizando a máquina cujas configurações estão descritas na Seção 2.3, o limite de memória foi atingido ao trabalhar com 10 machos e 500 fêmeas, ou em qualquer configuração que resultasse em aproximadamente 5.000 possíveis produtos (por exemplo, 20 machos e 250 fêmeas).

A Tabela 25 apresenta um resumo das melhores configurações obtidas para cada função de custo avaliada. São exibidos, para cada abordagem, os parâmetros α e β, o número de formigas e o número de iterações utilizados, bem como os resultados alcançados em termos de consanguinidade média da prole e o tempo total de execução do

Tabela 25 – Melhores configurações por função de custo

Parâmetro	**Coancestralidade média**	**Nº de relações** > 0
Configuração (α, β)	(2, 1)	(1, 2)
Nº de formigas	5	1
Nº de iterações	30	30
Consanguinidade média	0,0045	0,0069
Tempo de execução (s)	~2.900	~3.900

Fonte: Autor, (2025)

algoritmo. Observa-se que a função de custo baseada na coancestralidade média produziu soluções com menor consanguinidade e tempo de execução mais eficiente, destacando-se a configuração com $\alpha = 2$ e $\beta = 1$, 5 formigas e 30 iterações. Já a função que considera o número de relações com coancestralidade estritamente positiva apresentou desempenho inferior nos dois critérios, mesmo quando utilizada a configuração com maior influência da heurística.

6.4 Discussão Geral

Os resultados obtidos demonstram que a abordagem proposta é capaz de identificar conjuntos de acasalamento que mantêm a coancestralidade sob controle, diferentemente de métodos tradicionais de seleção massal ou truncada, que, ao priorizarem exclusivamente o valor genético (como o IECC), tendem a elevar rapidamente a consanguinidade média do rebanho a longo prazo. Enquanto a seleção clássica desconsidera as relações de parentesco entre os pares selecionados, o método ACO penaliza soluções com alta coancestralidade, promovendo a diversidade genética.

No entanto, é importante destacar a limitação de escalabilidade observada. O crescimento quadrático do número de relações de parentesco entre os produtos ($O(n^2)$) impõe restrições práticas ao tempo de execução e uso de memória para rebanhos muito grandes. Embora o uso do banco de dados em grafo mitigue o custo de acesso às relações, a complexidade combinatória do problema permanece. Para aplicações em larga escala (milhares de animais), estratégias adicionais, como o particionamento do rebanho ou o uso de limiares de parentesco para reduzir a densidade do grafo, seriam necessárias, conforme apontado nas sugestões de trabalhos futuros.

7 CONCLUSÃO

7.1 Conclusões

O presente trabalho abordou o problema de acasalamento em rebanhos bovinos com o objetivo de minimizar a coancestralidade e controlar a consanguinidade, mantendo o mérito genético. A formalização do problema e o desenvolvimento de uma solução algorítmica baseada em Otimização por Colônia de Formigas (MMAS), integrada a um banco de dados orientado a grafos, permitiram explorar novas estratégias para o manejo genético.

Os experimentos realizados demonstraram que o algoritmo proposto é capaz de encontrar soluções que reduzem significativamente a coancestralidade média da prole em comparação com métodos de seleção aleatória ou massal. Especificamente, configurações que equilibram a influência do feromônio e da heurística (como $\alpha = 1, \beta = 1$ ou $\alpha = 2, \beta = 1$) mostraram-se mais eficazes. A utilização do banco de dados Neo4j provou ser uma decisão acertada, eliminando a complexidade de consultas relacionais recursivas e permitindo a recuperação eficiente dos coeficientes de parentesco. No entanto, a escalabilidade para rebanhos muito grandes permanece um desafio devido ao crescimento quadrático das relações a serem processadas.

As principais contribuições deste estudo podem ser categorizadas em três vertentes:

- **Contribuição Científica**: A validação da meta-heurística ACO (Max-Min Ant System) como uma abordagem viável e eficiente para o problema de minimização de coancestralidade, capaz de lidar com as restrições complexas de acasalamento.
- **Contribuição Aplicada**: O desenvolvimento de um artefato de software que integra armazenamento em grafos e algoritmos de otimização, oferecendo uma ferramenta potencial para programas de melhoramento genético que buscam sustentabilidade e saúde do rebanho a longo prazo.
- **Contribuição Metodológica**: A demonstração de que bancos de dados orientados a grafos reduzem a distorção de impedância em algoritmos baseados em caminhos e redes, facilitando a modelagem de problemas genealógicos complexos.

7.1.1 Trabalhos Futuros

Embora este estudo tenha avançado na formalização e solução do problema de acasalamento seletivo com restrições de coancestralidade, algumas lacunas persistem e podem ser exploradas em pesquisas futuras. Uma das principais limitações identificadas é a escalabilidade do algoritmo MMAS para rebanhos de maior porte, onde o tempo de execução aumenta rapidamente com o número de relações de parentesco. Futuramente, seria relevante investigar técnicas de paralelização ou otimização computacional, como o uso de computação em nuvem ou GPUs, para reduzir o tempo de processamento em cenários reais de larga escala. Além disso, uma extensão natural seria incorporar uma abordagem multiobjetivo, utilizando algoritmos evolutivos ou híbridos que equilibrem múltiplos critérios, permitindo uma avaliação mais holística das estratégias de acasalamento. Por fim, comparar o MMAS com outras meta-heurísticas, como algoritmos genéticos ou otimização por enxame de partículas, poderia revelar abordagens mais eficientes para problemas semelhantes. Essas investigações não apenas preencheriam as lacunas identificadas, mas também contribuiriam para o avanço da integração entre computação e ciências agrárias, fomentando inovações no melhoramento genético animal.

REFERÊNCIAS

ABNEY, M. A graphical algorithm for fast computation of identity coefficients and generalized kinship coefficients. **Bioinformatics**, Oxford University Press, v. 25, n. 12, p. 1561–1563, 2009.

AGUIAR, M. S. **Análise formal da complexidade de algoritmos genéticos**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 1998.

AGUILAR, I.; MISZTAL, I. Recursive algorithm for inbreeding coefficients assuming nonzero inbreeding of unknown parents. **Journal of Dairy science**, Elsevier, v. 91, n. 4, p. 1669–1672, 2008.

AHMED, I. Z.; SADJADPOUR, H. R.; YOUSEFI, S. Information-assisted dynamic programming for a class of constrained combinatorial problems. **IEEE Access**, v. 10, p. 87816–87831, 2022.

ARKSEY, H.; O'MALLEY, L. Scoping studies: Towards a methodological framework. **Int. J. Social Research Methodology**, Routledge, v. 8, n. 1, p. 19–32, 2005.

ARORA, S.; BARAK, B. **Computational complexity: a modern approach**. [S.l.]: Cambridge University Press, 2009.

BACKUS, V.; GILPIN, M. An efficient algorithm for the additive kinship matrix. **Journal of Heredity**, Oxford University Press, v. 93, n. 6, p. 453–456, 2002.

BAGHEL, M.; AGRAWAL, S.; SILAKARI, S. Survey of metaheuristic algorithms for combinatorial optimization. **International Journal of Computer Applications**, Foundation of Computer Science, v. 58, n. 19, 2012.

BINU, D.; KARIYAPPA, B. S. Ridenn: A new rider optimization algorithm-based neural network for fault diagnosis in analog circuits. **IEEE Transactions on Instrumentation and Measurement**, v. 68, n. 1, p. 2–26, 2019.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys**, v. 35, p. 268–308, 2003. Disponível em: https://doi.org/10.1145/937503.937505.

BOYCE, A. Computation of inbreeding and kinship coefficients on extended pedigrees. **Journal of Heredity**, Oxford University Press, v. 74, n. 6, p. 400–404, 1983.

BROOKSHEAR, J. G. **Ciência da Computação-: Uma Visão Abrangente**. [S.l.]: Bookman Editora, 2013.

CARDOSO, F. Ferramentas e estratégias para o melhoramento genético de bovinos de corte. **Embrapa Pecuária Sul-Documentos (INFOTECA-E)**, Bagé: Embrapa Pecuária Sul, 2009., 2009.

CHAKRAVARTHI, S. S.; KUMAR, G. H. Optimization of network coverage and lifetime of the wireless sensor network based on pareto optimization using non-dominated sorting genetic approach. **Procedia Computer Science**, v. 172, p. 225–228, 2020. ISSN 1877-0509. 9th World Engineering Education Forum (WEEF 2019) Proceedings

: Disruptive Engineering Education for Sustainable Development. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050920313582.

CHARMANTIER, A.; GARANT, D.; KRUUK, L. E. **Quantitative genetics in the wild**. [S.l.]: OUP Oxford, 2014.

COCKERHAM, C. C. Group inbreeding and coancestry. **Genetics**, Oxford University Press, v. 56, n. 1, p. 89, 1967.

COLORNI, A. et al. Distributed optimization by ant colonies. In: **Proceedings of the first European conference on artificial life**. Paris, France: [s.n.], 1991. v. 142, p. 134–142.

CORMEN, T. H. et al. **Algoritmos: teoria e prática**. 2. ed. [S.l.]: Editora Campus, 2002. 296 p.

COSTE, C. F. et al. The kinship matrix: inferring the kinship structure of a population from its demography. **Ecology Letters**, Wiley Online Library, v. 24, n. 12, p. 2750–2762, 2021.

CUNHA, C. B. da; BONASSER, U. de O.; ABRAHÃO, F. T. M. Experimentos computacionais com heurísticas de melhorias para o problema do caixeiro viajante. In: **XVI Congresso da Anpet**. [S.l.: s.n.], 2002.

DAHAN, F. et al. An enhanced ant colony optimization based algorithm to solve qos-aware web service composition. **IEEE Access**, v. 9, p. 34098–34111, 2021.

DEMIR, E. M.; ERKAN, G. Graph databases: A survey on technologies, solutions and applications. **Computer Science Review**, Elsevier, v. 24, p. 1–27, 2017.

DENEUBOURG, J.-L. et al. The self-organizing exploratory pattern of the argentine ant. **Journal of Insect Behavior**, Springer, v. 3, n. 2, p. 159–168, 1990.

DIESTEL, R. **Graph Theory**. [S.l.]: Springer, 2005. (Graduate Texts in Mathematics). ISBN 978-3-642-14278-9.

DORIGO, M.; BIRATTARI, M.; STUTZLE, T. Ant colony optimization. **IEEE computational intelligence magazine**, IEEE, v. 1, n. 4, p. 28–39, 2006.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. In: **IEEE Transactions on Evolutionary Computation**. [S.l.: s.n.], 1997. v. 1, n. 1, p. 53–66.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: Optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, IEEE, v. 26, n. 1, p. 29–41, 1996.

DORIGO, M.; SOCHA, K. An introduction to ant colony optimization. In: **Handbook of approximation algorithms and metaheuristics**. [S.l.]: Chapman and Hall/CRC, 2018. p. 395–408.

EIBEN, A. E. et al. What is an evolutionary algorithm? **Introduction to evolutionary computing**, Springer, p. 25–48, 2015.

ELER, J. **Teorias e métodos em melhoramento genético animal: sistemas de acasalamento**. [S.l.]: Pirassununga: FZEA/USP, 2017.

ELLIOTT, B. et al. Efficiently calculating inbreeding on large pedigrees databases. **Information systems**, v. 34, n. 6, p. 469–492, 2009.

FERNÁNDEZ, J. et al. Management of genetic diversity in small farm animal populations. **Animal**, Elsevier, v. 5, n. 11, p. 1684–1698, 2011.

FERNÁNDEZ, J.; TORO, M.; CABALLERO, A. Fixed contributions designs vs. minimization of global coancestry to control inbreeding in small populations. **Genetics**, Oxford University Press, v. 165, n. 2, p. 885–894, 2003.

FERREIRA, A. P. L. Programação orientada a objetos com grafos. Apresentação no JAI 2006. 2006.

FERREIRA, A. P. L. On the problem of compensatory mating in animal breeding. In: **Anais do VI Workshop-Escola de Informática Teórica (WEIT 2021)**. Bagé, RS: [s.n.], 2021. p. 96–103.

FERREIRA, A. P. L.; RIBEIRO, L. Programação orientada a objeto com grafos. In: ____. **Atualizações em Informática**. Rio de Janeiro: Editora PUC-Rio, 2006. cap. 8, p. 387–453.

FERREIRA, A. P. L.; YOKOO, M. J.-I.; MOTTA, B. E. T. On the problem of optimal mating in animal breeding. In: **Simposio Latinoamericano de Teoría Computacional, Proceedings of the XLVII Conferencia Latinoamericana de Informática (CLEI 2021)**. San José, Costa Rica: IEEE, 2021. p. 1–7. DOI 10.1109/CLEI53233.2021.9640023.

FU, Y. et al. A discrete multi-objective rider optimization algorithm for hybrid flowshop scheduling problem considering makespan, noise and dust pollution. **IEEE Access**, v. 8, p. 88527–88546, 2020.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)**. 1st. ed. [S.l.]: W. H. Freeman, 1979. ISBN 0716710455.

GHOLAMI, K.; THOMAS, A. A linear time algorithm for calculation of multiple pairwise kinship coefficients and the genetic index of familiality. **Computers and Biomedical Research**, Elsevier, v. 27, n. 5, p. 342–350, 1994.

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização combinatória e programação linear: modelos e algoritmos**. [S.l.]: Elsevier, 2005.

GRASSE, P.-P. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. **Insectes sociaux**, Springer, v. 6, n. 1, p. 41–80, 1959.

HALLAUER, A. R.; CARENA, M. J.; FILHO, J. d. M. **Quantitative genetics in maize breeding**. [S.l.]: Springer Science & Business Media, 2010. v. 6.

HECKEL, R. et al. Horizontal and vertical structuring of typed graph transformation systems. **Mathematical Structures in Computer Science**, v. 6, n. 6, p. 613–648, 1996.

HENRYON, M.; SØRENSEN, A.; BERG, P. Mating animals by minimising the covariance between ancestral contributions generates less inbreeding without compromising genetic gain in breeding schemes with truncation selection. **Animal**, Elsevier, v. 3, n. 10, p. 1339–1346, 2009.

HERTZ, A.; WIDMER, M. Guidelines for the use of meta-heuristics in combinatorial optimization. **European Journal of Operational Research**, elsevier, v. 151, p. 247–252, 2003. Disponível em: https://doi.org/10.1016/S0377-2217(02)00823-8.

KALIYAR, R. K. Graph databases: A survey. In: IEEE. **International Conference on Computing, Communication & Automation**. [S.l.], 2015. p. 785–790.

KHANG, J. V. T. A fortran subroutine to compute inbreeding and kinship coefficients according to the number of ancestral generations. **Bioinformatics**, Oxford University Press, v. 5, n. 3, p. 199–204, 1989.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. Keele, 2004.

LI, C. et al. A novel adaptive dynamic programming based on tracking error for nonlinear discrete-time systems. **Automatica**, Elsevier, v. 129, p. 109687, 2021.

LIU, H.; HENRYON, M.; SØRENSEN, A. Mating strategies with genomic information reduce rates of inbreeding in animal breeding schemes without compromising genetic gain. **Animal**, Cambridge University Press, v. 11, n. 4, p. 547–555, 2017.

MOSCATO, P. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. **Caltech Concurrent Computation Program**, 10 2000.

MOTTA, B. E. T. da. **BrangusSelection: algoritmo de otimização para seleção de acasalamentos de bovinos com foco na maximização dos ganhos financeiros**. Dissertação (Mestrado) — Programa de Pós-graduação em Computação Aplicada, Universidade Federal do Pampa, 2021.

MRODE, R. A.; THOMPSON, R. **Linear models for the prediction of animal breeding values**. [S.l.]: CABI publishing, 2005.

MUNN, Z. et al. Systematic review or scoping review? Guidance for authors when choosing between a systematic or scoping review approach. **BMC Medical Research Methodology**, v. 18, 2018. Disponível em: <https://doi.org/10.1186/s12874-018-0611-. >

NEO4J. **The Neo Database**. ano. Online. Disponível em: https://neo4j.com/docs/.

NGATCHOU, P.; ZAREI, A.; EL-SHARKAWI, A. Pareto multi objective optimization. In: IEEE. **Proceedings of the 13th international conference on, intelligent systems application to power systems**. [S.l.], 2005. p. 84–91.

NOMURA, T. A mating system to reduce inbreeding in selection programmes: theoretical basis and modification of compensatory mating. **Journal of Animal Breeding and Genetics**, Wiley Online Library, v. 116, n. 5, p. 351–361, 1999.

NOMURA, T. et al. Optimization of selection and mating schemes in closed broiler lines. **Animal science journal**, Wiley Online Library, v. 73, n. 6, p. 435–443, 2002.

OES, M. R. S. S. et al. Breeding objectives of brangus cattle in brazil. **Journal of Animal Breeding and Genetics**, v. 137, n. 2, p. 177–188, 2020. DOI 10.1111/jbg.12415. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/jbg.12415.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. [S.l.]: Courier Corporation, 1998.

PRODANOV, C. C.; FREITAS, E. C. de. **Metodologia do trabalho científico: métodos e técnicas da Pesquisa e do Trabalho Acadêmico**. 2. ed. Novo Hamburgo: FEEVALE, 2012.

PRYCE, J.; HAYES, B.; GODDARD, M. Novel strategies to minimize progeny inbreeding while maximizing genetic gain using genomic information. **Journal of dairy science**, Elsevier, v. 95, n. 1, p. 377–388, 2012.

RECHENBERG, I. Evolutionsstrategie. **Optimierung technischer Systeme nach Prinzipien derbiologischen Evolution**, Frommann-Holzboog Verlag, 1973.

ROBERT, C. P.; CASELLA, G.; CASELLA, G. **Monte Carlo statistical methods**. [S.l.]: Springer, 1999. v. 2.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph databases: new opportunities for connected data**. [S.l.]: "O'Reilly Media, Inc.", 2015.

RODRIGUEZ, M. A.; NEUBAUER, P. The graph traversal pattern. In: **Graph data management: Techniques and applications**. [S.l.]: IGI global, 2012. p. 29–46.

ROSSO, E. D. da S.; ROSA, A. dos S.; FERREIRA, A. P. L. Estratégia dinâmica para determinação de grau de coancestralidade em um banco de dados orientado a grafos. In: **Anais do VII Workshop-Escola de Informática Teórica**. Porto Alegre: ociedade Brasileira de Computação, 2023. p. 19–28.

ROSSO, E. D. S. **Algoritmo para minimização da endogamia e maximização de índice econômico em sistemas de acasalamento**. 2023. Trabalho de Conclusão de Curso, Engenharia de Computação, Universidade Federal do Pampa.

ROSSO Éric Dias da S.; FERREIRA, A. P. L. Análise experimental da complexidade do problema da minimização da consanguinidade em sistemas de acasalamento. In: **Anais do XIV Congresso Brasileiro de Agroinformática**. Porto Alegre, RS, Brasil: SBC, 2023. p. 32–39. ISSN 2177-9724. Disponível em: https://sol.sbc.org.br/index.php/sbiagro/article/view/26538.

SÁNCHEZ, L.; BIJMA, P.; WOOLLIAMS, J. A. Minimizing inbreeding by managing genetic contributions across generations. **Genetics**, Oxford University Press, v. 164, n. 4, p. 1589–1595, 2003.

SANCHEZ, L.; TORO, M. A.; GARCÍA, C. Improving the efficiency of artificial selection: more selection pressure with less inbreeding. **Genetics**, Oxford University Press, v. 151, n. 3, p. 1103–1114, 1999.

SINGH, P. et al. Multi-criteria decision making monarch butterfly optimization for optimal distributed energy resources mix in distribution networks. **Applied Energy**, v. 278, p. 115723, 2020. ISSN 0306-2619. Disponível em: https://www.sciencedirect.com/science/article/pii/S0306261920312149.

SIPSER, M. Introduction to the theory of computation. **ACM Sigact News**, ACM New York, NY, USA, v. 27, n. 1, p. 27–29, 1996.

SITZENSTOCK, F.; YTOURNEL, F.; SIMIANER, H. A recursive method for computing expected kinship and inbreeding in complex and dynamic breeding programmes. **Journal of Animal Breeding and Genetics**, Wiley Online Library, v. 130, n. 1, p. 55–63, 2013.

SNIEDOVICH, M. **Dynamic programming**. [S.l.]: CRC press, 1991. v. 297.

STÜTZLE, T.; HOOS, H. H. **MAX-MIN Ant System**. [S.l.]: Future Generation Computer Systems, 2000.

TEMPLETON, A. R. **Population genetics and microevolutionary theory**. [S.l.]: John Wiley & Sons, 2021.

TORO, M. et al. Estimation of coancestry in iberian pigs using molecular markers. **Conservation Genetics**, Springer, v. 3, p. 309–320, 2002.

VLECK, L. van. **Selection Index and Introduction to Mixed Model Methods**. CRC-Press, 1993. ISBN 9780849387623. Disponível em: https://books.google.com.br/books?id=mtsqAQAAMAAJ.

WANG, G.-G.; DEB, S.; CUI, Z. Monarch butterfly optimization. **Neural computing and applications**, Springer, v. 31, p. 1995–2014, 2019.

WANG, H.; LI, R.; GONG, W. Minimizing tardiness and makespan for distributed heterogeneous unrelated parallel machine scheduling by knowledge and pareto-based memetic algorithm. **Egyptian Informatics Journal**, v. 24, n. 3, p. 100383, 2023. ISSN 1110-8665. Disponível em: https://www.sciencedirect.com/science/article/pii/S1110866523000312.

WEIGEL, K. Controlling inbreeding in modern breeding programs. **Journal of Dairy Science**, Elsevier, v. 84, p. E177–E184, 2001.

YANG, X.-S. **Nature-inspired metaheuristic algorithms**. [S.l.]: Luniver press, 2010.

YOKOO, M. et al. **Sumário genômico para características de difícil mensuração de animais da raça Brangus-edição setembro de 2020.** [S.l.], 2021.

ZHANG, J. et al. Graph databases: Introduction and fundamentals. **Foundations and Trends® in Databases**, Now Publishers Inc, v. 12, n. 1, p. 1–133, 2018.

ZHAO, H. et al. A decomposition-based many-objective ant colony optimization algorithm with adaptive solution construction and selection approaches. **Swarm and Evolutionary Computation**, Elsevier, v. 68, p. 100977, 2022.

ZHOU, X. et al. Random following ant colony optimization: Continuous and binary variants for global optimization and feature selection. **Applied Soft Computing**, Elsevier, v. 144, p. 110513, 2023.

APÊNDICE A – PROCESSAMENTO DE DADOS GENEALÓGICOS

Este apêndice descreve o código Python desenvolvido para processar dados genealógicos de bovinos da raça Angus, com o objetivo de selecionar animais jovens e seus respectivos ancestrais para posterior análise.

A.1 Objetivo

O código tem como objetivo principal:

- Carregar um arquivo CSV contendo dados genealógicos
- Filtrar animais nascidos no ano de 2024
- Identificar todos os ancestrais desses animais até o nível de bisavós
- Salvar um novo arquivo CSV contendo apenas os animais jovens e seus ancestrais

A.2 Descrição do Código

A.2.1 Carregamento dos Dados

```
import pandas as pd

# Carregar o arquivo CSV
arquivo_csv =
    r"c:/Users/dias_/Desktop/Python_Neo4j/Dados/pedigree-angus.csv"
dados = pd.read_csv(arquivo_csv, delimiter=';')
```

O código utiliza a biblioteca Pandas para manipulação de dados. O arquivo CSV é carregado a partir de um caminho específico no sistema de arquivos, utilizando o caractere ';' como delimitador.

A.2.2 Pré-processamento

```
# Converter a coluna 'Safra' para o formato de data
dados['Safra'] = pd.to_datetime(dados['Safra'], errors='coerce')
```

A coluna 'Safra' (que contém a data de nascimento dos animais) é convertida para o formato de data do Pandas, permitindo operações de filtragem temporal.

A.2.3 Filtragem de Animais Jovens

```
# Filtrar os animais nascidos entre 2024 e 2024
filtro_jovens = (dados['Safra'] >= '2024-01-01') &
 ↪  (dados['Safra'] <= '2024-12-31')
animais_jovens = dados[filtro_jovens]

# Obter IDs dos jovens
ids_jovens = set(animais_jovens['ID'])
```

São selecionados apenas os animais nascidos no ano de 2024, e seus IDs são armazenados em um conjunto (set) para operações posteriores.

A.2.4 Função Recursiva para Obter Ancestrais

```
# Função para obter ancestrais (pais, avós, bisavós)
def obter_ancestrais(ids_atual, nivel, dados):
    if nivel == 0 or not ids_atual:
        return set()

    # Filtrar os ancestrais diretos (pais)
    pais = dados[dados['ID'].isin(ids_atual)]
    ids_pais = set(pais['ID_Pai'].dropna()) |
     ↪  set(pais['ID_Mae'].dropna())

    # Recursão para obter os ancestrais de nível superior
    return ids_pais | obter_ancestrais(ids_pais, nivel - 1,
     ↪  dados)
```

Esta função recursiva obtém todos os ancestrais até um determinado nível (no caso, 3 níveis - pais, avós e bisavós). A função:

- Recebe um conjunto de IDs, o nível atual de profundidade e os dados completos
- Para cada ID, busca os IDs do pai e da mãe

- Chama recursivamente a si mesma para obter os ancestrais do nível superior
- Retorna a união de todos os IDs encontrados

A.2.5 Execução e Salvamento dos Resultados

```
# Obter IDs de todos os ancestrais até os bisavós
ids_ancestrais = obter_ancestrais(ids_jovens, 3, dados)

# Filtrar os dados para incluir apenas os jovens e seus
 ↪ ancestrais
ids_selecionados = ids_jovens | ids_ancestrais
dados_selecionados = dados[dados['ID'].isin(ids_selecionados)]

# Salvar os dados filtrados em um novo arquivo CSV
dados_selecionados.to_csv("c:/Users/dias_/Desktop/Python_Neo4j/Dados/AngusNovo
 ↪ index=False, sep=';')
```

O código final:

- Obtém todos os ancestrais até o nível de bisavós
- Combina os IDs dos animais jovens com os de seus ancestrais
- Filtra o DataFrame original para manter apenas esses IDs
- Salva o resultado em um novo arquivo CSV

A.3 Resultados Esperados

Ao executar este código, será gerado um novo arquivo CSV contendo:

- Todos os animais nascidos em 2024
- Todos os seus pais, avós e bisavós
- Todas as informações genealógicas associadas a esses animais

Este arquivo processado será utilizado como entrada para análises posteriores no banco de dados Neo4j, permitindo a construção de grafos genealógicos e análises de parentesco.

APÊNDICE B – ADEQUAÇÃO DE DADOS GENEALÓGICOS

Este apêndice descreve o código Python desenvolvido para normalizar e corrigir os dados genealógicos previamente filtrados, garantindo consistência na representação dos IDs e tratamento de valores ausentes.

B.1 Objetivo

O código tem como objetivos principais:

- Padronizar a representação dos IDs dos animais e seus progenitores
- Tratar valores ausentes (NA) de forma consistente
- Garantir que todos os pais e mães referenciados existam na base de dados
- Gerar um arquivo CSV corrigido para uso posterior

B.2 Descrição do Código

B.2.1 Carregamento e Pré-processamento Inicial

```
1 import pandas as pd
2
3 # Carregar o CSV
4 df =
   ↪  pd.read_csv("c:/Users/dias_/Desktop/Python_Neo4j/Dados/AngusNovo/pedigree-
   ↪  sep=";")
5
6 # Substituir valores ausentes por 0
7 df.fillna(0, inplace=True)
```

O código inicia carregando o arquivo CSV gerado no processamento anterior, utilizando o Pandas. Valores ausentes (NA) são substituídos por 0, o que facilita a manipulação posterior dos dados.

B.2.2 Padronização dos IDs

```
# Remover o sufixo ".0" dos IDs (animal, pai e mãe)
 ↪ convertendo-os em strings e ajustando
df["ID"] = df["ID"].apply(lambda x: str(int(float(x))) if x != 0
 ↪ else "0")
df["ID_Pai"] = df["ID_Pai"].apply(lambda x: str(int(float(x)))
 ↪ if x != 0 else "0")
df["ID_Mae"] = df["ID_Mae"].apply(lambda x: str(int(float(x)))
 ↪ if x != 0 else "0")

# Converter IDs para string (para evitar problemas com
 ↪ comparações)
df["ID"] = df["ID"].astype(str)
df["ID_Pai"] = df["ID_Pai"].astype(str)
df["ID_Mae"] = df["ID_Mae"].astype(str)
```

Esta seção realiza:

- Conversão dos IDs (que podem estar como floats) para inteiros e depois para strings
- Remoção de sufixos ".0"que podem aparecer em números inteiros representados como floats
- Garantia de que todos os IDs são armazenados como strings para consistência
- Tratamento especial para o valor 0 (representando progenitor desconhecido)

B.2.3 Validação de Referências

```
# Criar um conjunto com todos os IDs conhecidos
ids_conhecidos = set(df["ID"])

# Substituir IDs de pai e mãe não presentes na coluna ID por 0
df["ID_Pai"] = df["ID_Pai"].apply(lambda x: x if x in
 ↪ ids_conhecidos else "0")
df["ID_Mae"] = df["ID_Mae"].apply(lambda x: x if x in
 ↪ ids_conhecidos else "0")
```

Esta parte crítica do código:

- Cria um conjunto com todos os IDs válidos existentes
- Verifica se cada pai e mãe referenciado existe na base de dados
- Substitui por "0"qualquer referência a progenitores inexistentes
- Garante a integridade referencial do pedigree

B.2.4 Salvamento dos Resultados

```
# Salvar o resultado em um novo arquivo CSV
df.to_csv("c:/Users/dias_/Desktop/Python_Neo4j/Dados/AngusNovo/pedigree-angus-
          sep=",",
          index=False)
```

O DataFrame processado é salvo em um novo arquivo CSV, agora:

- Com valores consistentes e padronizados
- Com separador alterado para vírgula
- Sem índice numérico adicional

B.3 Comentários Adicionais

O código comentado no final mostra um exemplo de processamento adicional que poderia ser realizado:

- Remoção de uma coluna específica (índice 0)
- Novo salvamento do arquivo

B.4 Resultados Esperados

Ao executar este código, será gerado um novo arquivo CSV contendo:

- Todos os registros do arquivo original, mas com:
  - IDs padronizados como strings sem sufixos decimais
  - Valores ausentes substituídos por "0"
  - Referências a progenitores inexistentes corrigidas

- Formato consistente para importação no Neo4j
- Dados prontos para análise genealógica sem erros de referência

Este processo de normalização é essencial para garantir a qualidade dos dados antes da construção do grafo genealógico no Neo4j.

APÊNDICE C – CONSTRUÇÃO DO GRAFO GENEALÓGICO NO NEO4J

Este apêndice descreve as consultas Cypher utilizadas para criar e popular o grafo genealógico no Neo4j a partir dos dados processados anteriormente.

C.1 Objetivo

O código tem como objetivos principais:

- Criar nodos representando animais a partir do arquivo CSV
- Estabelecer relações de parentesco (pai e mãe)
- Tratar casos especiais (progenitores desconhecidos)
- Organizar animais sem descendentes em gerações

C.2 Descrição do Código

C.2.1 Criação dos Nodos Iniciais

```
1 LOAD CSV WITH HEADERS FROM 'file:///pedigree-angus-selecao.csv'
   ↪ AS line
2 CREATE (:Animal {ID: line.ID, Sexo: line.Sexo, Safra:
   ↪ line.Safra, IECC: line.IECC})
```

Esta primeira consulta:

- Carrega o arquivo CSV contendo os dados dos animais
- Cria um nodo do tipo `Animal` para cada registro
- Define propriedades básicas: ID, Sexo, Safra (data de nascimento) e IECC

C.2.2 Tratamento de Progenitores

```
1 LOAD CSV WITH HEADERS FROM 'file:///pedigree-angus-selecao.csv'
   ↪ AS line
2 MATCH (a:Animal {ID: line.ID})
3 MERGE (b:Animal {ID: line.ID_Mae})
```

```
4 ON CREATE SET b.ID = line.ID_Mae, b.Sexo = "FEMEA", b.Safra = 0,
   ↪ b.IECC = 0
5 ON MATCH SET a = a
```

```
1 LOAD CSV WITH HEADERS FROM 'file:///pedigree-angus-selecao.csv'
   ↪ AS line
2 MATCH (a:Animal {ID: line.ID})
3 MERGE (b:Animal {ID: line.ID_Pai})
4 ON CREATE SET b.ID = line.ID_Pai, b.Sexo = "MACHO", b.Safra = 0,
   ↪ b.IECC = 0
5 ON MATCH SET a = a
```

Estas consultas realizam:

- Criação de nodos para mães (com sexo definido como "FEMEA")
- Criação de nodos para pais (com sexo definido como "MACHO")
- Atribuição de valores padrão para progenitores não presentes no CSV original
- Utilização de `MERGE` para evitar duplicação de nodos

C.2.3 Limpeza de Dados

```
1 MATCH (n:Animal {ID: "0"})
2 DELETE n
```

Remove todos os nodos representando progenitores desconhecidos (ID = "0"), que foram criados como placeholders durante o processo anterior.

C.2.4 Estabelecimento de Relações Familiares

```
1 LOAD CSV WITH HEADERS FROM 'file:///pedigree-angus-selecao.csv'
   ↪ AS line
2 MATCH (a:Animal {ID: line.ID})
3 MATCH (b:Animal {ID: line.ID_Mae})
4 WHERE b.ID <> "0"
5 CREATE (a)<-[t:MAE_DE]-(b)
```

```
LOAD CSV WITH HEADERS FROM 'file:///pedigree-angus-selecao.csv'
 ↪ AS line
MATCH (a:Animal {ID: line.ID})
MATCH (b:Animal {ID: line.ID_Pai})
WHERE b.ID <> "0"
CREATE (a)<-[t:PAI_DE]-(b)
```

Estas consultas criam as relações de parentesco:

- `MAE_DE`: relação entre mãe e filho(a)
- `PAI_DE`: relação entre pai e filho(a)
- A condição `WHERE b.ID <> "0"` evita criar relações com progenitores desconhecidos
- A direção da seta (<-) indica que o progenitor aponta para o descendente

C.2.5 Organização por Gerações

```
MATCH (a:Animal)
WHERE NOT (a)<-[:PAI_DE]-() AND NOT (a)<-[:MAE_DE]-()
MERGE (g:Geracao {Numero: 0})
CREATE (a)-[:PERTENCE_A]->(g)
```

Esta consulta final:

- Identifica animais que não são pais/mães de nenhum outro (última geração)
- Cria um nodo representando a geração 0
- Conecta esses animais à sua geração correspondente

C.3 Resultados Esperados

Ao executar este conjunto de consultas, será criado no Neo4j:

- Um grafo genealógico completo com todos os animais do dataset
- Relações de parentesco devidamente estabelecidas
- Estrutura organizada por gerações
- Dados limpos e consistentes, sem referências a progenitores desconhecidos

Este grafo estará pronto para consultas e análises de:

- Linhagens familiares
- Padrões de herança genética
- Estrutura populacional
- Análises de coeficientes de endogamia

C.4 Observações Técnicas

- O caminho do arquivo (`file:///`) segue a convenção do Neo4j para importação
- As consultas estão preparadas para CSV com cabeçalho
- O tratamento de progenitores desconhecidos é essencial para a qualidade do grafo
- A organização por gerações facilita análises posteriores

APÊNDICE D – CÁLCULO DE GERAÇÕES GENEALÓGICAS

Este apêndice documenta a implementação de um algoritmo responsável por calcular e registrar a geração de cada animal presente no banco de dados orientado a grafos (Neo4j). O cálculo baseia-se na geração de seus pais e considera como geração inicial (0) os animais sem ascendência registrada.

D.1 Objetivo

O algoritmo tem como objetivo identificar automaticamente a geração de cada animal com base na estrutura do pedigree armazenada em Neo4j, otimizando o cálculo com técnicas de paralelismo e uso de cache.

D.2 Etapas do Algoritmo

A execução do código segue os seguintes passos principais:

1. Estabelecer conexão com o banco de dados Neo4j.
2. Carregar os animais com geração já atribuída (geração 0).
3. Carregar os animais que ainda não possuem geração atribuída.
4. Calcular, de forma recursiva e paralela, a geração de cada animal com base em seus pais.
5. Registrar os resultados no banco de dados.

D.3 Descrição das Funções

`carregar_geracoes_zero(tx)` Carrega os animais da geração 0 (sem pai ou mãe registrados) e armazena no cache global de gerações.

`carregar_animais(tx)` Retorna todos os animais que ainda não possuem relação com geração. Para cada animal, coleta os IDs dos pais, quando disponíveis.

`calcular_geracao_recursive(animal_id, id_to_parents)` Calcula recursivamente a geração de um animal. Se o animal não tiver pais, considera geração 0. Caso contrário, define como a maior geração dos pais acrescida de 1.

`calcular_geracoes_batch(batch, id_to_parents)` Calcula a geração de um lote de animais utilizando a função recursiva.

`calcular_geracoes_paralelo(animais, num_threads)` Divide os animais em lotes e executa o cálculo de suas gerações de forma paralela com múltiplas threads.

`gravar_geracoes_em_lote(tx, batch)` Atualiza o banco de dados, associando cada animal ao nodo da sua geração utilizando `MERGE`.

`calcular_e_gravar_todos_animais()` Função principal do script. Coordena a execução das demais funções: carrega os dados, calcula as gerações de forma paralela e grava os resultados no banco.

D.4 Técnicas Utilizadas

- **Cache compartilhado com `Lock`**: Para evitar recomputações e garantir a consistência das gerações já calculadas em ambiente paralelo.
- **Processamento paralelo com `ThreadPoolExecutor`**: Otimiza o tempo de cálculo ao dividir a carga entre múltiplas threads.
- **Gravação em lotes**: Reduz o número de transações com o banco, agrupando as inserções em blocos de até 10.000 registros.

D.5 Resultados Esperados

Ao final da execução, espera-se que todos os animais estejam corretamente conectados ao respectivo nodo `Geracao` no banco, permitindo análises hierárquicas e a organização do rebanho por profundidade geracional.

D.6 Código-Fonte

O código-fonte completo pode ser consultado na listagem a seguir:

```
1 from neo4j import GraphDatabase
2 import logging
3 from concurrent.futures import ThreadPoolExecutor, as_completed
4 from threading import Lock
5
```

```
# Configura o logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

uri = "bolt://localhost:7687"
username = "neo4j"
password = "12345678"

# Cache compartilhado
geracoes_cache = {}
cache_lock = Lock()

def carregar_animais(tx):
    logger.info("Carregando animais sem geracao calculada do banco.")
    result = tx.run(
        """
        MATCH (a:Animal)
        WHERE NOT (a)-[:PERTENCE_A]->()
        OPTIONAL MATCH (a)<-[:PAI_DE]-(sire:Animal)
        OPTIONAL MATCH (a)<-[:MAE_DE]-(dam:Animal)
        RETURN a.ID AS animalId, sire.ID AS sireID, dam.ID AS damID
        """
    )
    animais = {}
    for record in result:
        animais[record["animalId"]] = {
            "sireID": record["sireID"],
            "damID": record["damID"],
        }
    logger.info(f"Total de animais carregados: {len(animais)}.")
    return animais

def carregar_geracoes_zero(tx):

    logger.info("Carregando geracoes ja registradas no banco.")
    result = tx.run(
        """
        MATCH (a:Animal)-[:PERTENCE_A]->(g:Geracao {Numero: 0})
        RETURN a.ID AS animalId
        """
    )
    with cache_lock:
        for record in result:
            geracoes_cache[record["animalId"]] = 0
    logger.info(f"Geracoes carregadas no cache: {len(geracoes_cache)}.")

def calcular_geracao_recursive(animal_id, id_to_parents):

    with cache_lock:
        if animal_id in geracoes_cache:
            return geracoes_cache[animal_id]

```

```
    if animal_id not in id_to_parents:
        return 0  # Animais sem pais estao na geracao 0

    parents = id_to_parents[animal_id]
    sire_id = parents.get("sireID")
    dam_id = parents.get("damID")

    # Calcula geracoes dos pais recursivamente
    sire_geracao = calcular_geracao_recursive(sire_id, id_to_parents) if sire_id else 0
    dam_geracao = calcular_geracao_recursive(dam_id, id_to_parents) if dam_id else 0

    geracao = max(sire_geracao, dam_geracao) + 1

    with cache_lock:
        geracoes_cache[animal_id] = geracao

    return geracao

def calcular_geracoes_batch(batch, id_to_parents):

    for animal_id in batch:
        calcular_geracao_recursive(animal_id, id_to_parents)

def calcular_geracoes_paralelo(animais, num_threads=4):

    ids = list(animais.keys())
    batch_size = len(ids) // num_threads
    batches = [ids[i:i + batch_size] for i in range(0, len(ids), batch_size)]

    logger.info(f"Iniciando calculo paralelo com {num_threads} threads.")
    with ThreadPoolExecutor(max_workers=num_threads) as executor:
        futures = [executor.submit(calcular_geracoes_batch, batch, animais)
                                                    for batch in batches]
        for future in as_completed(futures):
            future.result()  # Espera pela conclusao de cada thread
    logger.info("Calculo paralelo das geracoes concluido.")

def gravar_geracoes_em_lote(tx, batch):

    tx.run(
        """
        UNWIND $batch AS item
        MATCH (a:Animal {ID: item.animalId})
        MERGE (g:Geracao {Numero: item.geracao})
        MERGE (a)-[:PERTENCE_A]->(g)
        """,
        batch=batch
    )

def calcular_e_gravar_todos_animais():

    driver = GraphDatabase.driver(uri, auth=(username, password))
```

```
110
111     try:
112         with driver.session() as session:
113             # Carregar geracoes ja registradas (geracao 0)
114             session.read_transaction(carregar_geracoes_zero)
115
116             # Carregar animais para calcular
117             animais = session.read_transaction(carregar_animais)
118
119             # Calcular geracoes em paralelo
120             calcular_geracoes_paralelo(animais, num_threads=8)
121
122             # Preparar dados para gravacao
123             batch = [{"animalId": animal_id, "geracao": geracao}
124                 for animal_id, geracao in geracoes_cache.items()]
125
126             # Gravar no banco de dados
127             for i in range(0, len(batch), 10000):
128                 session.write_transaction(gravar_geracoes_em_lote, batch[i:i + 10000])
129
130     finally:
131         driver.close()
132
133 if __name__ == "__main__":
134     calcular_e_gravar_todos_animais()
```

Lista D.1: Algoritmo de cálculo e gravação de gerações

APÊNDICE E – CÁLCULO DE COANCESTRALIDADE

Este apêndice documenta a implementação de um algoritmo desenvolvido para calcular a matriz de coancestralidade entre todos os animais registrados em um banco de dados orientado a grafos (Neo4j). O cálculo é baseado na ancestralidade compartilhada e considera a estrutura genealógica registrada no banco.

E.1 Objetivo

O objetivo do algoritmo é computar, para cada par de animais, o coeficiente de coancestralidade (*kinship coefficient*) e armazenar esse valor no banco de dados por meio de relações direcionadas entre os nodos correspondentes. Para garantir eficiência, utiliza-se uma matriz esparsa para cache de valores e técnicas de otimização no armazenamento.

E.2 Etapas do Algoritmo

A execução do código segue os seguintes passos principais:

1. Estabelecer conexão com o banco de dados Neo4j.
2. Contar o número total de animais no banco.
3. Carregar os dados genealógicos (pais) dos animais.
4. Mapear os IDs dos animais para índices numéricos.
5. Inicializar uma matriz esparsa para armazenar as coancestralidades.
6. Calcular, de forma recursiva, o coeficiente entre todos os pares de animais.
7. Persistir os valores calculados no banco de dados em lotes.

E.3 Descrição das Funções

`Rel_cache.get(index1, index2)` Retorna o valor de coancestralidade entre dois índices se já tiver sido calculado.

`Rel_cache.set(index1, index2, valor)` Armazena o valor de coancestralidade entre dois animais na matriz cache.

`calcular_coancestralidade(...)` Função recursiva responsável por calcular o coeficiente de coancestralidade entre dois animais com base na ancestralidade dos pais.

`carregar_dados_animais(tx)` Consulta o banco para recuperar a lista de animais com seus respectivos pais, ordenando por geração.

`obter_numero_animais(tx)` Retorna o número total de nodos do tipo `Animal`.

`processar_relacoes_otimizado(driver)` Função principal do processo: carrega dados, calcula coeficientes e grava no banco.

`main()` Estabelece conexão com o banco, executa o cálculo e encerra a sessão.

E.4 Técnicas Utilizadas

- **Matriz Esparsa (`dok_matrix`)**: Utilizada para armazenar apenas os coeficientes não nulos, economizando memória.
- **Indexação bidirecional (ID $\leftrightarrow$ índice)**: Permite conversão eficiente entre identificadores de animais e índices numéricos usados na matriz.
- **Recursão com cache**: O cálculo da coancestralidade é recursivo, mas evita recomputações ao armazenar valores já resolvidos.
- **Armazenamento em Lotes**: Os coeficientes calculados são gravados no banco em blocos de até 100.000 pares, otimizando a performance da escrita.
- **Relações direcionadas com atributo `Valor`**: Cada par de animais recebe uma aresta do tipo `COEF_PARENTESCO` contendo o valor calculado.

E.5 Resultados Esperados

Ao final da execução, espera-se que:

- Todos os pares de animais tenham seus coeficientes de coancestralidade calculados e registrados no banco.
- Os valores sejam acessíveis por meio da relação `COEF_PARENTESCO`.
- O sistema esteja preparado para manipular grandes quantidades de animais sem consumir memória excessiva.

E.6 Código-Fonte

```
from scipy.sparse import dok_matrix
from neo4j import GraphDatabase
import logging

# Configura o logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

uri = "bolt://localhost:7690"  # URL do banco Neo4j
username = "neo4j"
password = "12345678"
database = "neo4j"

class RelacionamentoCache:

    def __init__(self, total_animais):

        self.cache = dok_matrix((total_animais, total_animais), dtype=float)

    def get(self, index1, index2):

        if index1 > index2:
            index1, index2 = index2, index1
        return self.cache.get((index1, index2), None)

    def set(self, index1, index2, valor):

        if index1 > index2:
            index1, index2 = index2, index1
        self.cache[index1, index2] = valor

    def total_relacoes(self):

        return len(self.cache)

def obter_numero_animais(tx):

    result = tx.run("""
        MATCH (a:Animal)
        RETURN COUNT(a) AS totalAnimais
    """)
    return result.single()["totalAnimais"]

def carregar_dados_animais(tx):

    logger.info("Carregando dados dos animais do banco de dados...")
    result = tx.run("""
        MATCH (a:Animal)-[:PERTENCE_A]->(gen:Geracao)
        OPTIONAL MATCH (a)<-[r1:PAI_DE]-(sire:Animal)
```

```
        OPTIONAL MATCH (a)<-[r2:MAE_DE]-(dam:Animal)
        RETURN a.ID AS animalId, sire.ID AS sireId, dam.ID AS damId
        ORDER BY gen.Numero
    """)
    return {record["animalId"]: (record["sireId"],
            record["damId"]) for record in result}

def calcular_coancestralidade(index1, index2, relacionamento, animais,
                                                id_to_index, index_to_id):

    valor = relacionamento.get(index1, index2)
    if valor is not None:
        return valor

    animal1 = index_to_id[index1]
    animal2 = index_to_id[index2]

    if index1 == index2:
        valor = 1
    else:
        sire2, dam2 = animais.get(animal2, (None, None))
        valor = sum(0.5 * calcular_coancestralidade(index1,
        id_to_index[parent], relacionamento, animais, id_to_index, index_to_id)
                    for parent in [sire2, dam2] if parent)

    relacionamento.set(index1, index2, valor if valor is not None else 0)
    return valor

def processar_relacoes_otimizado(driver):

    with driver.session() as session:
        total_animais = session.execute_read(obter_numero_animais)
        animais = session.execute_read(carregar_dados_animais)

        id_to_index = {animal_id: idx for idx, animal_id in enumerate(animais.keys())}
        index_to_id = {idx: animal_id for animal_id, idx in id_to_index.items()}

        relacionamento = RelacionamentoCache(total_animais)

        logger.info("Iniciando calculo da matriz de coancestralidade...")
        for idx, animal1 in enumerate(animais.keys()):
            logger.info(f"Processando relacoes para o animal {idx + 1}/{total_animais}")
            for animal2 in list(animais.keys())[idx:]:
                index1 = id_to_index[animal1]
                index2 = id_to_index[animal2]
                calcular_coancestralidade(index1, index2,
                relacionamento, animais, id_to_index, index_to_id)

        logger.info(f"Calculo concluido. Total de relacoes
        armazenadas: {relacionamento.total_relacoes()}")

        logger.info("Persistindo os resultados no banco de dados...")
```

```
        batch = []
        batch_size = 100_000
        for (index1, index2), valor in relacionamento.cache.items():
            if index1 <= index2:

                batch.append({"animal1": index_to_id[index1],
                "animal2": index_to_id[index2], "valor": valor})
                if len(batch) >= batch_size:
                    session.run("""
                        UNWIND $batch AS rel
                        MATCH (a:Animal {ID: rel.animal1}), (b:Animal {ID: rel.animal2})
                        CREATE (a)-[r:COEF_PARENTESCO {Valor: rel.valor}]->(b)
                    """, batch=batch)
                    batch.clear()

        if batch:
            session.run("""
                UNWIND $batch AS rel
                MATCH (a:Animal {ID: rel.animal1}), (b:Animal {ID: rel.animal2})
                CREATE (a)-[r:COEF_PARENTESCO {Valor: rel.valor}]->(b)
            """, batch=batch)

def main():
    driver = GraphDatabase.driver(uri, auth=(username, password), database=database)
    try:
        logger.info("Iniciando o processo de calculo
        e persistencia de coancestralidade...")
        processar_relacoes_otimizado(driver)
        logger.info("Processo concluido com sucesso!")
    finally:
        driver.close()

if __name__ == "__main__":
    main()
```

Lista E.1: Algoritmo de cálculo e gravação de coancestralidade

APÊNDICE F – PRODUTOS

Este apêndice descreve a implementação de um módulo desenvolvido para selecionar pares de animais (machos e fêmeas), gerar novos produtos (prole) a partir desses pares e exportar os dados resultantes para arquivos CSV. O algoritmo também recupera os coeficientes de parentesco entre os pais para calcular a consanguinidade dos produtos.

F.0.1 Objetivo

O módulo tem como propósito:

- Selecionar conjuntos de machos e fêmeas a partir do banco de dados Neo4j, de forma aleatória ou manual.
- Gerar produtos para todas as combinações macho-fêmea, calculando o índice de seleção e a consanguinidade.
- Exportar os dados dos produtos e coeficientes de parentesco para arquivos CSV.

F.1 Descrição das Funções

`Neo4jHandler` Classe que gerencia a conexão com o banco de dados Neo4j, fornecendo métodos para leitura e escrita dentro de transações.

`executar_escrita(func, *args, **kwargs)` Executa funções de escrita no banco de dados dentro de uma transação.

`executar_leitura(func, *args, **kwargs)` Executa funções de leitura no banco de dados dentro de uma transação.

`ProdutoHandler.selecionar_animais(tx, modo, ...)` Seleciona os animais reprodutores do banco de dados conforme o modo definido (aleatório ou manual), retornando listas separadas de machos e fêmeas.

`ProdutoHandler.criar_produtos(tx, machos, femeas)` Gera uma lista de novos produtos com base nas combinações entre machos e fêmeas selecionados, calculando o índice de seleção médio e o coeficiente de consanguinidade para cada par.

`ProdutoHandler.obter_parentesco(tx, machos, femeas)` Obtém os coeficientes de parentesco entre todos os reprodutores fornecidos (machos e fêmeas).

`salvar_em_csv(dados, arquivo_csv, fieldnames)` Salva os dados fornecidos em arquivos CSV, formatando os valores muito pequenos para zero conforme um limite definido (`threshold`).

`main()` Função principal que coordena todo o fluxo da aplicação: interação com o usuario, execução das operações no banco, e salvamento dos resultados.

F.2 Técnicas Utilizadas

- **Transações com o Neo4j**: Todas as operações sobre o banco de dados são encapsuladas em transações seguras utilizando o driver oficial do Neo4j.
- **Seleção flexível de dados**: O sistema suporta dois modos de seleção — aleatória (com sorteio) e manual (via IDs fornecidos pelo usuario), garantindo flexibilidade no uso.
- **Calculo de índice de seleção e consanguinidade**: O índice de seleção do novo animal é calculado como média dos pais, e o coeficiente de consanguinidade é obtido diretamente das relações entre os nodos no banco.
- **Geração combinatória de cruzamentos**: Todos os pares possíveis entre os machos e fêmeas selecionados são combinados para gerar novos produtos.
- **Exportação em CSV**: Os resultados dos cruzamentos e coeficientes de parentesco são armazenados em arquivos CSV para analise posterior.
- **Tratamento numérico com limiar de precisão**: Valores numéricos muito pequenos (abaixo de 10^{-4}) são arredondados para zero na exportação.

F.3 Resultados Esperados

Ao final da execução, espera-se que:

- Os animais reprodutores tenham sido selecionados com sucesso conforme os critérios definidos pelo usuario.
- Todos os cruzamentos possíveis entre machos e fêmeas tenham sido gerados, com

atributos devidamente calculados (ID, índice de seleção, consanguinidade).

- Os dados dos novos produtos estejam salvos em um arquivo CSV.
- Os coeficientes de parentesco entre os reprodutores estejam corretamente extraídos e também armazenados em CSV.
- O sistema opere de forma robusta em diferentes modos de entrada e com diferentes tamanhos de população.

F.3.1 Formato dos Arquivos CSV

`novos_produtos.csv` :

- **ID**: Identificador único do produto (`PaiID_MaeID`)
- **Safra**: Ano fixo de nascimento (2025)
- **Sexo**: Indicador genérico “PRODUTO”
- **PaiID** e **MaeID**
- **IECC**: Índice de seleção
- **Consanguinidade**: Coeficiente de parentesco entre os pais

`parentesco_pais.csv` :

- **Animal_1**, **Animal_2**: Identificadores dos pais
- **Coef**: Valor do coeficiente de parentesco

F.3.2 Código-Fonte Principal

```
import csv
import random
from neo4j import GraphDatabase


class Neo4jHandler:
    """
    Classe responsavel por gerenciar a conex o com o banco de dados Neo4j
    e executar opera  es de leitura e escrita.
    """
    def __init__(self, uri, username, password):
        """
        Inicializa a conex o com o banco Neo4j.
```

```
14
15          Par metros:
16          uri (str): URL do servidor Neo4j.
17          username (str): Nome de usuario para autentica  o.
18          password (str): Senha para autentica  o.
19          """
20          self.driver = GraphDatabase.driver(uri, auth=(username, password))
21
22      def close(self):
23          """
24          Fecha a conex o com o banco de dados Neo4j.
25          """
26          self.driver.close()
27
28      def executar_escrita(self, func, *args, **kwargs):
29
30          with self.driver.session() as session:
31              return session.execute_write(func, *args, **kwargs)
32
33      def executar_leitura(self, func, *args, **kwargs):
34
35          with self.driver.session() as session:
36              return session.execute_read(func, *args, **kwargs)
37
38
39  class ProdutoHandler:
40      """
41      Classe responsavel por manipular dados relacionados aos produtos e animais
42      no banco de dados Neo4j.
43      """
44      @staticmethod
45      def selecionar_animais(tx, modo, num_machos=None, num_femeas=None, ids_animais=None):
46
47          if modo == "aleatorio":
48              if num_machos is None or num_femeas is None:
49                  raise ValueError("Para o modo aleat rio,    necessario
50                  especificar 'num_machos' e 'num_femeas'.")
51
52              # Obter machos e f meas do banco
53              machos = tx.run(
54                  """
55                  MATCH (a:Animal {Sexo: "M"})
56                  WHERE a.Safra <> 0
57                  RETURN a.ID AS ID, a.IECC AS IECC
58                  """
59              ).data()
60
61              femeas = tx.run(
62                  """
63                  MATCH (a:Animal {Sexo: "F"})
64                  WHERE a.Safra <> 0
65                  RETURN a.ID AS ID, a.IECC AS IECC
```

```
                """
            ).data()

            # Selecionar aleatoriamente
            machos_selecionados = random.sample(machos, num_machos)
            femeas_selecionadas = random.sample(femeas, num_femeas)

        elif modo == "manual":
            if ids_animais is None:
                raise ValueError("Para o modo manual,
                   necessario fornecer 'ids_animais'.")

            # Obter informa  es dos IDs fornecidos
            query = """
            UNWIND $ids_animais AS id
            MATCH (a:Animal {ID: id})
            RETURN a.ID AS ID, a.Sexo AS Sexo, a.IECC AS IECC
            """
            animais = tx.run(query, ids_animais=ids_animais).data()

            # Separar em machos e f meas
            machos_selecionados = [animal for animal in animais if animal["Sexo"] == "M"]
            femeas_selecionadas = [animal for animal in animais if animal["Sexo"] == "F"]

            if not machos_selecionados or not femeas_selecionadas:
                raise ValueError("Certifique-se de que os IDs fornecidos
                incluam tanto machos quanto f meas.")

        else:
            raise ValueError("Modo invalido. Use 'aleatorio' ou 'manual'.")

        return machos_selecionados, femeas_selecionadas

    @staticmethod
    def criar_produtos(tx, machos_selecionados, femeas_selecionadas):

        # Obter IDs dos pais
        ids_machos = [macho["ID"] for macho in machos_selecionados]
        ids_femeas = [femea["ID"] for femea in femeas_selecionadas]

        # Buscar coeficiente de parentesco entre machos e f meas
        query = """
        UNWIND $ids_machos AS id1
        UNWIND $ids_femeas AS id2
        MATCH (a1:Animal {ID: id1})-[r:COEF_PARENTESCO]-(a2:Animal {ID: id2})
        RETURN a1.ID AS PaiID, a2.ID AS MaeID, r.Valor AS Coef
        """
        parentesco = tx.run(query, ids_machos=ids_machos, ids_femeas=ids_femeas).data()

        # Criar um dicionario para mapear os coeficientes
        coef_map = {
            (item["PaiID"], item["MaeID"]): item["Coef"]
```

```
            for item in parentesco
        }
        coef_map.update({
            (item["MaeID"], item["PaiID"]): item["Coef"]
            for item in parentesco
        })

        # Gerar os novos animais com a consanguinidade
        novos_animais = [
            {
                "ID": f"{macho['ID']}_{femea['ID']}",
                "Safra": 2025,
                "Sexo": "PRODUTO",
                "PaiID": macho["ID"],
                "MaeID": femea["ID"],
                "IECC": (float(macho["IECC"]) + float(femea["IECC"])) / 2,
                "Consanguinidade": coef_map.get((macho["ID"], femea["ID"]), 0)
            }
            for macho in machos_selecionados
            for femea in femeas_selecionadas
        ]
        return novos_animais

    @staticmethod
    def obter_parentesco(tx, machos, femeas):

        pais = machos + femeas
        ids_pais = [pai['ID'] for pai in pais]

        query = """
        UNWIND $pais AS id1
        UNWIND $pais AS id2
        WITH id1, id2 WHERE id1 <= id2
        MATCH (a1:Animal {ID: id1})-[r:COEF_PARENTESCO]-(a2:Animal {ID: id2})
        RETURN DISTINCT a1.ID AS Animal_1, a2.ID AS Animal_2, r.Valor AS Coef
        """
        resultados = tx.run(query, pais=ids_pais).data()
        return resultados


def salvar_em_csv(dados, arquivo_csv, fieldnames, threshold=1e-4):

    with open(arquivo_csv, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()
        for row in dados:
            row_formatado = {
                chave: (
                    0 if isinstance(valor, float) and abs(valor) < threshold else valor
                )
                for chave, valor in row.items()
            }
```

```
            writer.writerow(row_formatado)
    print(f"Dados salvos em {arquivo_csv}")


def main():

    uri = "bolt://localhost:7690"
    username = "neo4j"
    password = "12345678"

    neo4j_handler = Neo4jHandler(uri, username, password)

    try:
        # Solicitar modo de opera  o ao usuario
        modo = input("Escolha o modo ('aleatorio' ou 'manual'): ").strip().lower()

        if modo == "aleatorio":
            num_machos = int(input("Quantos machos deseja sortear? "))
            num_femeas = int(input("Quantas f meas deseja sortear? "))
            machos_selecionados, femeas_selecionadas = neo4j_handler.executar_escrita(
                ProdutoHandler.selecionar_animais, modo, num_machos, num_femeas
            )

        elif modo == "manual":
            ids_animais = input("Informe os IDs dos animais
            separados por v rgula: ").strip().split(",")
            ids_animais = [id_animal.strip() for id_animal in ids_animais]
            machos_selecionados, femeas_selecionadas = neo4j_handler.executar_escrita(
                ProdutoHandler.selecionar_animais, modo, ids_animais=ids_animais
            )

        else:
            raise ValueError("Modo invalido. Escolha 'aleatorio' ou 'manual'.")

        # Criar novos produtos
        novos_animais = neo4j_handler.executar_escrita(
            ProdutoHandler.criar_produtos, machos_selecionados, femeas_selecionadas
        )

        # Salvar novos produtos em CSV
        salvar_em_csv(
            novos_animais,
            "/novos_produtos.csv",
            ["ID", "Safra", "Sexo", "PaiID", "MaeID", "IECC", "Consanguinidade"]
        )


        # Obter coeficientes de parentesco
        parentesco = neo4j_handler.executar_leitura(
            ProdutoHandler.obter_parentesco, machos_selecionados, femeas_selecionadas
        )

```

```
222         # Salvar coeficientes de parentesco em CSV
223         salvar_em_csv(
224             parentesco,
225             "/parentesco_pais.csv",
226             ["Animal_1", "Animal_2", "Coef"]
227         )
228 
229     finally:
230         neo4j_handler.close()
231 
232 
233 if __name__ == "__main__":
234     main()
```

Lista F.1: Função principal do gerador de produtos

APÊNDICE G – COEFICIENTES DE PARENTESCO ENTRE PRODUTOS

Este apêndice documenta a implementação de um algoritmo responsavel por calcular e registrar os coeficientes de coancestralidade entre animais, com foco nas relações entre pais e produtos, bem como entre os próprios produtos. Os dados são manipulados via arquivos CSV e organizados em memória para otimizar o desempenho da execução.

G.1 Objetivo

O algoritmo tem como objetivo calcular automaticamente os coeficientes de coancestralidade entre pares de animais a partir de dados de pedigree previamente carregados, utilizando estratégias de cache, gravação em buffer e consulta em memória para otimizar a execução.

G.2 Etapas do Algoritmo

A execução do código segue os seguintes passos principais:

1. Inicializar os arquivos CSV de saída com os cabeçalhos adequados.
2. Carregar os coeficientes ja existentes em arquivos anteriores para memória (evita recomputação).
3. Carregar os dados dos produtos e respectivos pais a partir de um arquivo CSV.
4. Calcular os coeficientes de coancestralidade entre pais e seus produtos.
5. Calcular os coeficientes de coancestralidade entre todos os pares de produtos.
6. Armazenar os resultados nos arquivos CSV, utilizando buffers para reduzir acessos ao disco.

G.3 Descrição das Funções

`carregar_csv(caminho)` Carrega um arquivo CSV e retorna os dados como uma lista de dicionarios.

`salvar_csv(caminho, dados, campos)` Salva uma lista de dicionarios em um

arquivo CSV com os campos fornecidos.

`inicializar_csv(caminho, campos)` Cria um novo arquivo CSV com cabeçalho, caso o arquivo ainda não exista.

`carregar_coeficientes_em_memoria(caminhos, tipo)` Lê os coeficientes ja calculados e armazena em memória, para evitar recomputações.

`buscar_coeficiente_em_memoria(animal1, animal2, tipo)` Recupera coeficiente ja existente na memória para um par de animais.

`gravar_buffer_no_csv(tipo)` Grava em disco os coeficientes temporariamente armazenados em memória (buffers), separando por tipo.

`armazenar_coeficiente_no_csv(animal1, animal2, coef, tipo)` Adiciona um coeficiente ao buffer correspondente e o grava quando o limite é atingido.

`calcular_coancestralidade(animal1, animal2, animais, tipo)` Função recursiva que calcula o coeficiente de coancestralidade entre dois animais, com base nos pais registrados.

`main()` Função principal do script. Inicializa os arquivos, carrega os dados e orquestra os calculos de todos os coeficientes entre pais-produtos e entre pares de produtos.

G.4 Técnicas Utilizadas

- **Cache de resultados calculados**: Os coeficientes ja processados são armazenados em um dicionario (`cache`) para evitar calculos repetidos, especialmente em chamadas recursivas.
- **Carregamento em memória**: Os coeficientes previamente calculados são carregados em memória no início do programa, minimizando acessos ao disco.
- **Buffer para escrita**: Os coeficientes são armazenados em buffers antes da gravação no disco, reduzindo o número de operações de I/O. Os dados são gravados em lotes de 100.000 registros.
- **Recursão com memoização**: O calculo de coancestralidade utiliza uma abordagem recursiva, com armazenagem em cache dos resultados intermediarios para ganho de desempenho.

G.5 Resultados Esperados

Ao final da execução, espera-se obter dois arquivos CSV:

- Um arquivo contendo os coeficientes de coancestralidade entre cada produto e seus pais.
- Um arquivo contendo os coeficientes de coancestralidade entre todos os pares de produtos.

Esses resultados podem ser usados para analises genéticas, seleção de animais e planejamento de acasalamentos com menor consanguinidade.

G.6 Código-Fonte

O código-fonte completo pode ser consultado na listagem a seguir:

```
import csv
from itertools import combinations
import logging
import os

# Configura  o do logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Caminhos de arquivos
CAMINHO_ENTRADA_PRODUTOS = '/novos_produtos.csv'
CAMINHO_ENTRADA_PP = '/parentesco_pais.csv'
CAMINHO_SAIDA_1 = '/parentesco_pais_produtos.csv'
CAMINHO_SAIDA_2 = '/parentesco_produtos.csv'

coef_pais_mem = {}
coef_produtos_mem = {}

# Cache para coeficientes calculados
cache = {}

# Lista para armazenar temporariamente os coeficientes antes de gravar no CSV
buffer_pais_produtos = []
buffer_produtos = []


# Fun  es utilitarias
def carregar_csv(caminho):

    with open(caminho, mode='r', encoding='utf-8') as file:
```

```
31          return list(csv.DictReader(file))
32
33  def salvar_csv(caminho, dados, campos):
34
35      with open(caminho, mode='w', newline='', encoding='utf-8') as file:
36          writer = csv.DictWriter(file, fieldnames=campos)
37          writer.writeheader()
38          writer.writerows(dados)
39
40  def inicializar_csv(caminho, campos):
41
42      if not os.path.exists(caminho):
43          with open(caminho, mode='w', newline='', encoding='utf-8') as file:
44              writer = csv.DictWriter(file, fieldnames=campos)
45              writer.writeheader()
46              logger.info(f"Arquivo {caminho} inicializado com cabe alho: {campos}")
47
48  def carregar_coeficientes_em_memoria(caminhos, tipo):
49
50      memoria = {}
51      for caminho in caminhos:
52          try:
53              with open(caminho, mode='r', encoding='utf-8') as file:
54                  reader = csv.DictReader(file)
55                  for row in reader:
56                      a1, a2 = row['Animal_1'], row['Animal_2']
57                      chave = tuple(sorted((a1, a2)))
58                      memoria[chave] = float(row['Coef'])
59          except FileNotFoundError:
60              continue
61      if tipo == 'pais-produto':
62          global coef_pais_mem
63          coef_pais_mem = memoria
64      elif tipo == 'produto-produto':
65          global coef_produtos_mem
66          coef_produtos_mem = memoria
67
68  def buscar_coeficiente_em_memoria(animal1, animal2, tipo):
69
70      chave = tuple(sorted((animal1, animal2)))
71      if tipo == 'pais-produto':
72          return coef_pais_mem.get(chave)
73      elif tipo == 'produto-produto':
74          return coef_produtos_mem.get(chave)
75      return None
76
77  # Fun  o para gravar resultados no arquivo CSV
78  def gravar_buffer_no_csv(tipo):
79
80      if not buffer_pais_produtos and not buffer_produtos:
81          return  # Nada para gravar
82
```

```
    elif tipo == 'pais-produto' and buffer_pais_produtos:
        with open(CAMINHO_SAIDA_1, mode='a', newline='', encoding='utf-8') as file:
            writer = csv.DictWriter(file, fieldnames=['Animal_1', 'Animal_2', 'Coef'])
            writer.writerows(buffer_pais_produtos)
        buffer_pais_produtos.clear()
        logger.info("Coeficientes pais-produto gravados.")

    elif tipo == 'produto-produto' and buffer_produtos:
        with open(CAMINHO_SAIDA_2, mode='a', newline='', encoding='utf-8') as file:
            writer = csv.DictWriter(file, fieldnames=['Animal_1', 'Animal_2', 'Coef'])
            writer.writerows(buffer_produtos)
        buffer_produtos.clear()
        logger.info("Coeficientes entre produtos gravados.")

def armazenar_coeficiente_no_csv(animal1, animal2, coef, tipo):

    registro = {'Animal_1': animal1, 'Animal_2': animal2, 'Coef': coef}

    if tipo == 'pais-produto':
        buffer_pais_produtos.append(registro)
        if len(buffer_pais_produtos) >= 100_000:
            gravar_buffer_no_csv('pais-produto')
    elif tipo == 'produto-produto':
        buffer_produtos.append(registro)
        if len(buffer_produtos) >= 100_000:
            gravar_buffer_no_csv('produto-produto')

# Fun  o para calcular coeficiente de parentesco
def calcular_coancestralidade(animal1, animal2, animais, tipo):
    if (animal1, animal2, tipo) in cache:
        return cache[(animal1, animal2, tipo)]
    if (animal2, animal1, tipo) in cache:
        return cache[(animal2, animal1, tipo)]

    coef = buscar_coeficiente_em_memoria(animal1, animal2, tipo)
    if coef is not None:
        cache[(animal1, animal2, tipo)] = coef
        armazenar_coeficiente_no_csv(animal1, animal2, coef, tipo)
        return coef

    elif animal1 == animal2:
        coef = 1

    else:
        pais = animais.get(animal2)
        if pais and (sire_id := pais['sireID']) and (dam_id := pais['damID']):
            coef = 0.5 * (calcular_coancestralidade(animal1, sire_id,
            animais, 'pais-produto') +
                          calcular_coancestralidade(animal1, dam_id,
                          animais, 'pais-produto'))
        elif pais and (sire_id := pais.get('sireID')):
            coef = 0.5 * calcular_coancestralidade(animal1, sire_id,
```

```
            animais, 'pais-produto')
        elif pais and (dam_id := pais.get('damID')):
            coef = 0.5 * calcular_coancestralidade(animal1, dam_id,
            animais, 'pais-produto')
        else:
            coef = 0

    cache[(animal1, animal2, tipo)] = coef
    armazenar_coeficiente_no_csv(animal1, animal2, coef, tipo)
    return coef


# Fun  o principal
def main():

    inicializar_csv(CAMINHO_SAIDA_1, ['Animal_1', 'Animal_2', 'Coef'])
    inicializar_csv(CAMINHO_SAIDA_2, ['Animal_1', 'Animal_2', 'Coef'])

    carregar_coeficientes_em_memoria([CAMINHO_ENTRADA_PP,
    CAMINHO_SAIDA_1], 'pais-produto')
    carregar_coeficientes_em_memoria([CAMINHO_SAIDA_2], 'produto-produto')

    produtos = carregar_csv(CAMINHO_ENTRADA_PRODUTOS)
    animais = {produto['ID']: {'sireID': produto['PaiID'],
    'damID': produto['MaeID']} for produto in produtos}

    # Coeficiente entre pais e produtos
    for produto in produtos:
        if (sire_id := produto.get('PaiID')):
            coef_sire = calcular_coancestralidade(sire_id,
            produto['ID'], animais, tipo='pais-produto')
            logger.info(f"Coeficiente entre pai {sire_id}
            e produto {produto['ID']}: {coef_sire}")
        if (dam_id := produto.get('MaeID')):
            coef_dam = calcular_coancestralidade(dam_id,
            produto['ID'], animais, tipo='pais-produto')
            logger.info(f"Coeficiente entre  m  e  {dam_id}
            e produto {produto['ID']}: {coef_dam}")

    # Coeficiente entre produtos
    for produto1, produto2 in combinations(produtos, 2):
        coef = calcular_coancestralidade(produto1['ID'],
        produto2['ID'], animais, tipo='produto-produto')
        logger.info(f"Coeficiente entre {produto1['ID']} e {produto2['ID']}: {coef}")

if __name__ == '__main__':
    main()
    # No final, grava o que sobrou no buffer
    gravar_buffer_no_csv('pais-produto')
    gravar_buffer_no_csv('produto-produto')
```

Lista G.1: Algoritmo de calculo de coancestralidade entre pais e produtos

APÊNDICE H – OTIMIZAÇÃO POR SISTEMA DE FORMIGAS MAX-MIN (MMAS)

Este apêndice documenta a implementação de um algoritmo baseado na metaheurística Max-Min Ant System (MMAS), utilizado para a seleção de animais em um sistema de acasalamento. O objetivo principal é encontrar um conjunto de cruzamentos que minimize a coancestralidade entre os produtos, respeitando restrições operacionais impostas ao problema.

H.1 Objetivo

O algoritmo visa otimizar a seleção de pares reprodutivos entre machos e fêmeas, respeitando restrições de uso maximo de cada macho, limite de consanguinidade e exclusividade de uso das fêmeas. Para isso, utiliza uma abordagem probabilística com formigas artificiais (baseada em MMAS), que exploram o espaço de soluções de forma paralela e cooperativa, balanceando exploração e intensificação por meio de atualizações de feromônio.

H.2 Etapas do Algoritmo

A execução do código segue os seguintes passos principais:

1. Carregar os dados dos produtos, contendo identificadores, IECC, consanguinidade e os IDs dos pais.
2. Carregar os coeficientes de parentesco entre os possíveis produtos.
3. Inicializar os níveis de feromônio de cada animal.
4. Verificar a viabilidade mínima de acasalamentos por macho.
5. Para cada iteração:
    1. Gerar múltiplas soluções em paralelo com base em heurísticas e feromônios.
    2. Avaliar as soluções geradas, determinando o custo médio e a consanguinidade.
    3. Atualizar os níveis de feromônio com base na melhor solução da iteração ou global.

4. Armazenar os resultados detalhados da melhor solução em arquivo CSV.
5. Registrar no arquivo resumo as estatísticas da execução.

H.3 Descrição das Funções

`carregar_dados_produtos()` Carrega os dados dos animais candidatos à seleção, extraindo o índice de seleção, consanguinidade e IDs parentais.

`carregar_coeficientes_parentesco()` Carrega os coeficientes de parentesco entre animais, utilizando um dicionario com pares como chave.

`checar_viabilidade()` Verifica se cada macho possui pares viaveis suficientes para atender ao limite de uso estipulado.

`calcular_estatisticas()` Calcula métricas estatísticas basicas como média, mediana, variância, desvio padrão e coeficiente de variação.

`calcular_tau_max_min()` Determina os valores mínimos e maximos permitidos de feromônio, conforme parâmetros do algoritmo.

`salvar_resultados()` Registra a melhor solução encontrada em arquivo, incluindo estatísticas de desempenho e lista de selecionados.

`inicializar_feromonios()` Inicializa os níveis de feromônio para todos os candidatos com valor padrão.

`construir_solucao_individual()` Função executada por cada formiga. Constrói uma solução viavel, considerando restrições, heurística baseada em parentesco e valores de feromônio.

`avaliar_atualizar()` Aplica decaimento global dos feromônios e reforça os feromônios dos animais presentes na melhor solução (global ou da iteração).

`otimizar()` Coordena o processo de otimização global, executando o número definido de iterações, cada uma composta por múltiplas soluções paralelas.

H.4 Técnicas Utilizadas

- **Meta-heurística MMAS**: Utilização do sistema Max-Min Ant System, com alternância de reforço (global ou iterativo) e atualização de feromônio nos nodos.
- **Heurística adaptativa**: A probabilidade de seleção de cada animal é inversamente

proporcional à consanguinidade acumulada com os ja selecionados.

- **Paralelismo com ThreadPoolExecutor**: Soluções são construídas simultaneamente por diferentes threads, melhorando o tempo de execução.
- **Reforço alternado**: A cada iteração, alterna-se o reforço de feromônio entre a melhor solução da iteração e a melhor global, promovendo diversificação e intensificação.
- **Estatísticas descritivas**: A solução final é caracterizada por métricas estatísticas detalhadas do índice de seleção e consanguinidade, facilitando a analise da qualidade genética.
- **Modularidade e reutilização**: As funções são organizadas de forma a permitir reuso e facil adaptação para outros conjuntos de dados ou parâmetros.

H.5 Resultados Esperados

Ao final da execução do algoritmo, espera-se obter:

- Um arquivo CSV contendo os animais selecionados, o custo total da solução (baseado em coancestralidade acumulada) e estatísticas descritivas da população selecionada.
- Um arquivo resumo das execuções com os parâmetros utilizados e as métricas de desempenho alcançadas, permitindo comparações entre diferentes configurações.

Esses resultados são uteis para analises de melhoramento genético, controle da consanguinidade e seleção econômica em programas de acasalamento assistido por algoritmos bioinspirados.

H.6 Código-Fonte

O código-fonte completo pode ser consultado na listagem a seguir:

```
import csv
import logging
import random
from concurrent.futures import ThreadPoolExecutor
import statistics
from collections import defaultdict
import time

```

```
# Configura  o basica de logging
logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")
logger = logging.getLogger(__name__)

# Par metros do MMAS
ALTERNAR_REFORCO = True

# === Fun  es utilitarias ===

def carregar_dados_produtos(caminho_produtos):
    with open(caminho_produtos, mode='r') as file:
        reader = csv.DictReader(file)
        dados = [
            {
                'id': row['ID'],
                'iecc': float(row['IECC']),
                'mid': row['MaeID'],
                'pid': row['PaiID'],
                'consanguinidade': float(row['Consanguinidade'])
            }
            for row in reader
        ]
    maes_diferentes = len(set(item['mid'] for item in dados))
    return dados, maes_diferentes

def carregar_coeficientes_parentesco(caminho_parentesco):
    coeficientes = {}
    with open(caminho_parentesco, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            chave = frozenset([row['Animal_1'], row['Animal_2']])
            coeficientes[chave] = float(row['Coef'])
    return coeficientes

def checar_viabilidade(dados, limite_consanguinidade):
    pares_por_macho = defaultdict(int)
    for d in dados:
        if d['consanguinidade'] <= limite_consanguinidade:
            pares_por_macho[d['pid']] += 1
    return pares_por_macho

def calcular_estatisticas(valores):
    return {
        'media': statistics.mean(valores) if valores else 0,
        'mediana': statistics.median(valores) if valores else 0,
        'moda': statistics.mode(valores) if valores else 0,
        'minimo': min(valores) if valores else 0,
        'maximo': max(valores) if valores else 0,
        'variancia': statistics.variance(valores) if len(valores) > 1 else 0,
        'desvio_padrao': statistics.stdev(valores) if len(valores) > 1 else 0,
        'coeficiente_variacao': (statistics.stdev(valores) /
        statistics.mean(valores) * 100) if len(valores) > 1
```

```
        and statistics.mean(valores) != 0 else 0,
    }

def calcular_tau_max_min(rho, f_best, n, a=0.05):
    tau_max = 1 / (rho * f_best)
    tau_min = tau_max * (1 - a) / ((n / 2 - 1) * a)
    if n > 2 else tau_max * 0.1  # Evita divis o por zero
    return tau_max, tau_min

def salvar_resultados(solucao, custo_medio, iecc_medio, consang_media, caminho_saida):
    logger.info("Salvando resultados.")
    with open(caminho_saida, mode='w', newline='') as file:
        writer = csv.writer(file)

        writer.writerow(["Resumo da Solucao"])
        writer.writerow(["Custo Total da Solucao", f"{custo_medio:.4f}"])
        writer.writerow([])

        # Estat sticas IECC
        writer.writerow(["Estatisticas do Indice de Selecao"])
        writer.writerow(["Metrica", "Valor"])
        for k, v in iecc_medio.items():
            writer.writerow([k.capitalize().replace("_", " "), f"{v:.4f}"])
        writer.writerow([])

        # Estat sticas Consanguinidade
        writer.writerow(["Estatisticas da Consanguinidade"])
        writer.writerow(["Metrica", "Valor"])
        for k, v in consang_media.items():
            writer.writerow([k.capitalize().replace("_", " "), f"{v:.4f}"])
        writer.writerow([])

        # Lista dos animais selecionados
        writer.writerow(["Animais Selecionados"])
        writer.writerow(["Animal_ID"])
        for animal in solucao:
            writer.writerow([animal])

def inicializar_feromonios(dados):
    return {dado['id']: 1.0 for dado in dados}

# === Constru  o de solu  o ===

def construir_solucao_individual(dados, feromonios, coef_parentesco, alpha, beta,
limite_maes, limite_uso_macho, limite_consanguinidade):
    selecionados = set()
    uso_femeas = set()
    uso_machos = {}
    custo_total = iecc_total = cons_total = 0
    iecc_individual = []
    consanguinidade_individual = []

```

```
    while len(uso_femeas) < limite_maes:
        candidatos = [
            d for d in dados
            if d['id'] not in selecionados
            and d['mid'] not in uso_femeas
            and d['consanguinidade'] <= limite_consanguinidade
            and uso_machos.get(d['pid'], 0) < limite_uso_macho
        ]

        if not candidatos:
            logger.debug("Sem candidatos dispon veis.")
            break

        # P r -calculo das somas dos coeficientes para todos os candidatos
        soma_parentesco = {}
        for c in candidatos:
            soma = sum(
                coef_parentesco.get(frozenset([c['id'], s]), 0) for s in selecionados
            ) if selecionados else 0
            soma_parentesco[c['id']] = soma

        # Calculo das probabilidades com heuristica e feromonio
        probabilidades = []
        for c in candidatos:
            heuristica = 1 / (1 + soma_parentesco[c['id']])
            fer = feromonios.get(c['id'], 1.0)
            prob = (fer ** alpha) * (heuristica ** beta)
            probabilidades.append((c, prob))

        total_prob = sum(p for _, p in probabilidades)
        if total_prob == 0:
            break

        probabilidades = [(c, p / total_prob) for c, p in probabilidades]
        escolhido = random.choices(
            [c for c, _ in probabilidades],
            weights=[p for _, p in probabilidades]
        )[0]

        selecionados.add(escolhido['id'])
        uso_femeas.add(escolhido['mid'])
        uso_machos.setdefault(escolhido['pid'], 0)
        uso_machos[escolhido['pid']] += 1

        # Atualiza m tricas
        custo_total += soma_parentesco[escolhido['id']]
        iecc_total += escolhido['iecc']
        cons_total += escolhido['consanguinidade']
        iecc_individual.append(escolhido['iecc'])
        consanguinidade_individual.append(escolhido['consanguinidade'])

    n = len(selecionados)
```

```
    if n == 0:
        return set(), float('inf'), 0.0, 0.0
    return selecionados, (custo_total / n),
    calcular_estatisticas(iecc_individual),
    calcular_estatisticas(consanguinidade_individual)

# Atualizacao de feromonio com reforco alternado e decaimento

def avaliar_atualizar(solucoes, feromonios, rho, q,
melhor_global, iteracao, tau_max, tau_min):
    validas = [s for s in solucoes if len(s[0]) > 0]
    if not validas:
        return feromonios, melhor_global

    reforco_global = (iteracao % 2 == 0) if ALTERNAR_REFORCO else True
    if reforco_global and melhor_global:
        sol_reforco = melhor_global[0]
        custo_reforco = melhor_global[1]
    else:
        melhor_iter = min(validas, key=lambda x: x[1])
        sol_reforco = melhor_iter[0]
        custo_reforco = melhor_iter[1]

    # Decaimento global
    for chave in feromonios:
        feromonios[chave] *= (1 - rho)

    # Refor o nos n s (pares escolhidos)
    for id_sel in sol_reforco:
        delta = q / custo_reforco
        feromonios[id_sel] = min(tau_max, max(tau_min, feromonios[id_sel] + delta))

    return feromonios, (sol_reforco, custo_reforco)

# Fun  o principal de otimiza  o

def otimizar(caminho_produtos, caminho_parentesco,
num_iteracoes, num_formigas, alpha, beta, rho, q,
limite_uso_macho, limite_consanguinidade, caminho_saida):
    logger.info("Iniciando MMAS.")
    dados, limite_maes = carregar_dados_produtos(caminho_produtos)
    coef_parentesco = carregar_coeficientes_parentesco(caminho_parentesco)
    feromonios = inicializar_feromonios(dados)

    melhor_global = None
    melhor_iecc = melhor_consang = None

    # >>> Verifica  o de pares viaveis por macho
    logger.info("Verificando pares viaveis por macho
    considerando a consanguinidade maxima permitida.")
    pares_viaveis = checar_viabilidade(dados, limite_consanguinidade)
    for pid, qtd in sorted(pares_viaveis.items()):
```

```
217         logger.info(f"Macho {pid}: {qtd} pares viaveis
218         (consang <= {limite_consanguinidade})")
219 
220     if any(qtd < limite_uso_macho for qtd in pares_viaveis.values()):
221         logger.warning("Algum macho possui menos pares
222         viaveis do que o limite de uso definido.")
223         logger.warning("Isso pode impedir que todas
224         as f meas sejam atendidas.")
225 
226 
227     for it in range(num_iteracoes):
228         logger.info(f"Itera  o {it + 1}/{num_iteracoes}")
229         with ThreadPoolExecutor(max_workers=4) as executor:
230             solucoes = list(executor.map(
231                 lambda _: construir_solucao_individual(dados,
232                 feromonios, coef_parentesco, alpha, beta,
233                 limite_maes, limite_uso_macho,
234                 limite_consanguinidade),
235                 range(num_formigas)
236             ))
237 
238         iter_melhor = min(solucoes, key=lambda x: x[1]
239         if len(x[0]) > 0 else float('inf'))
240 
241         if not melhor_global or iter_melhor[1] < melhor_global[1]:
242             melhor_global = iter_melhor
243             melhor_iecc = iter_melhor[2]
244             melhor_consang = iter_melhor[3]
245             logger.info(f"Nova melhor: custo=
246             {iter_melhor[1]:.4f}, IECC=
247             {melhor_iecc['media']:.4f}, Consang=
248             {melhor_consang['media']:.4f}")
249 
250         n = len(melhor_global[0]) if melhor_global else 1  # Evita divis o por zero
251         tau_max, tau_min = calcular_tau_max_min(rho, melhor_global[1], n)
252 
253         feromonios, melhor_global = avaliar_atualizar(
254             solucoes, feromonios, rho, q, melhor_global, it, tau_max, tau_min
255         )
256 
257     salvar_resultados(melhor_global[0], melhor_global[1],
258     melhor_iecc, melhor_consang, caminho_saida)
259     return melhor_global[0], melhor_global[1],
260     melhor_iecc, melhor_consang
261 
262 # Execu  o
263 if __name__ == "__main__":
264     caminho_produtos = '/novos_produtos.csv'
265     caminho_parentesco = '/parentesco_produtos.csv'
266     base_saida = '/'
267 
268     # Lista de combina  es de par metros
```

```
    combinacoes = [
        {'num_iteracoes': 1, 'num_formigas': 1, 'alpha': 2.0, 'beta': 1.0}
    ]

    # Par metros fixos
    parametros_fixos = {
        'rho': 0.1,
        'q': 1,
        'limite_uso_macho': 10,
        'limite_consanguinidade': 0.05,
    }

    caminho_resumo = base_saida + 'resumo_experimentos.csv'
    with open(caminho_resumo, mode='w', newline='') as resumo_file:
        writer = csv.writer(resumo_file)
        writer.writerow([
            'Execucao', 'Iteracoes', 'Formigas', 'Alpha', 'Beta',
            'Custo Medio', 'Consang Media', 'Consang Mediana',
            'Consang Min', 'Consang Max', 'Consang Desvio Padrao',
            'Arquivo Resultado', 'Tempo (segundos)'
        ])

        for i, combinacao in enumerate(combinacoes, start=1):
            nome_saida = f'resultado_execucao_{i:02d}.csv'
            caminho_saida = base_saida + nome_saida

            parametros = {**combinacao, **parametros_fixos}

            logger.info(f"\n=== Executando configura  o {i}: {parametros} ===")

            inicio = time.perf_counter()
            selecionados, custo, iecc_stats, consang_stats = otimizar(
                caminho_produtos,
                caminho_parentesco,
                **parametros,
                caminho_saida=caminho_saida
            )
            tempo_total = time.perf_counter() - inicio

            logger.info(
                f"Resultado final da execu  o {i}: "
                f"custo={custo:.4f}, consang_media=
                {consang_stats['media']:.4f}, tempo=
                {tempo_total:.2f} segundos"
            )

            writer.writerow([
                i,
                combinacao['num_iteracoes'],
                combinacao['num_formigas'],
                combinacao['alpha'],
                combinacao['beta'],
```

```
321                 round(custo, 4),
322                 round(consang_stats['media'], 4),
323                 round(consang_stats['mediana'], 4),
324                 round(consang_stats['minimo'], 4),
325                 round(consang_stats['maximo'], 4),
326                 round(consang_stats['desvio_padrao'], 4),
327                 nome_saida,
328                 round(tempo_total, 2)
329             ])
```

Lista H.1: Algoritmo MMAS para seleção de animais com restrições

APÊNDICE I – VERSÃO ALTERNATIVA DO ALGORITMO MMAS: MINIMIZAÇÃO DO NÚMERO DE RELAÇÕES DE COANCESTRALIDADE DIFERENTES DE ZERO

Nesta versão do algoritmo MMAS (*Max-Min Ant System*), o critério de avaliação das soluções é alterado de forma significativa: em vez de calcular um custo baseado na soma dos coeficientes de coancestralidade entre os pares selecionados, utiliza-se como função objetivo o número de pares cuja coancestralidade é **diferente de zero**. Isso representa uma abordagem mais direta à tentativa de minimizar a presença de vínculos genéticos entre os indivíduos selecionados, buscando maximizar o número de pares **geneticamente independentes** (com coeficiente de parentesco igual a zero).

I.1 Definição do custo

Dado um conjunto de indivíduos da prole gerado a partir de um mapeamento p:D* $\rightarrow S^*$, em que cada fêmea $x \in D^*$ é acasalada com um macho $p(x) \in S^*$, define-se o **custo** da solução como a **densidade do grafo de coancestralidade** entre os indivíduos da prole:

$$\text{Custo}(p) = A(\overline{p}) = \sum_{x \in D^*} \sum_{y \in D^*} a([x, p(x)], [y, p(y)]), \tag{34}$$

em que:

$$a(n,m) = \begin{cases} 1, & \text{se } c(n,m) > 0, \\ 0, & \text{caso contrário,} \end{cases}$$

e $c(n,m)$ é o coeficiente de coancestralidade entre os indivíduos n e m (ver Definição 8).

Esse critério de avaliação corresponde ao número de pares de indivíduos da prole com coancestralidade **estritamente positiva**, ou seja, ao número de arestas no grafo de coancestralidade. O objetivo do algoritmo é **minimizar** esse valor, favorecendo soluções compostas por indivíduos não aparentados.

I.2 Aspectos principais do algoritmo

A estrutura geral do algoritmo é mantida conforme a versão tradicional do MMAS, com adaptações nos seguintes pontos-chave:

- **Função objetivo**: contabiliza o número de pares com coancestralidade diferente de zero.
- **Construção da solução**: a heurística associada a cada candidato é inversamente proporcional à soma dos coeficientes de parentesco com os indivíduos já selecionados, como forma de favorecer indivíduos menos aparentados.

I.3 Cálculo do feromônio

Os limites $\tau_{\max}$ e $\tau_{\min}$ são definidos conforme a Equação 35, garantindo que o feromônio se mantenha dentro de um intervalo controlado:

$$\begin{aligned} \tau_{\max} &= \frac{1}{\rho \cdot f_{\text{best}}}, \\ \tau_{\min} &= \tau_{\max} \cdot \frac{1-a}{\left(\frac{n}{2}-1\right) \cdot a}, \end{aligned} \tag{35}$$

em que f_{best} é o custo da melhor solução encontrada até o momento, n é o número de indivíduos selecionados, ρ é a taxa de evaporação do feromônio, e a é um parâmetro constante.

I.4 Código-Fonte

O código-fonte completo pode ser consultado na listagem a seguir:

```
import csv
import logging
import random
from concurrent.futures import ThreadPoolExecutor
import statistics
import time

# Configura o b sica de logging
logging.basicConfig(level=logging.INFO,
format="%(asctime)s - %(levelname)s - %(message)s")
logger = logging.getLogger(__name__)

```

```
ALTERNAR_REFORCO = True

def carregar_dados_produtos(caminho_produtos):
    with open(caminho_produtos, mode='r') as file:
        reader = csv.DictReader(file)
        dados = [
            {
                'id': row['ID'],
                'iecc': float(row['IECC']),
                'mid': row['MaeID'],
                'pid': row['PaiID'],
                'consanguinidade': float(row['Consanguinidade'])
            }
            for row in reader
        ]
    maes_diferentes = len(set(item['mid'] for item in dados))
    return dados, maes_diferentes

def carregar_coeficientes_parentesco(caminho_parentesco):
    coeficientes = {}
    with open(caminho_parentesco, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            chave = tuple(sorted([row['Animal_1'], row['Animal_2']]))
            coeficientes[chave] = float(row['Coef'])
    return coeficientes

def calcular_estatisticas(valores):
    return {
        'media': statistics.mean(valores) if valores else 0,
        'mediana': statistics.median(valores) if valores else 0,
        'moda': statistics.mode(valores) if valores else 0,
        'minimo': min(valores) if valores else 0,
        'maximo': max(valores) if valores else 0,
        'variancia': statistics.variance(valores) if len(valores) > 1 else 0,
        'desvio_padrao': statistics.stdev(valores) if len(valores) > 1 else 0,
        'coeficiente_variacao': (statistics.stdev(valores) /
        statistics.mean(valores) * 100) if len(valores) > 1
        and statistics.mean(valores) != 0 else 0,
    }

def calcular_tau_max_min(rho, f_best, n, a=0.05):
    tau_max = 1 / (rho * f_best)
    tau_min = tau_max * (1 - a) / ((n / 2 - 1) * a) if n > 2 else tau_max * 0.1
    return tau_max, tau_min

def salvar_resultados(solucao, custo_medio, iecc_medio, consang_media, caminho_saida):
    logger.info("Salvando resultados.")
    with open(caminho_saida, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Relacoes de coancestralidade
        diferentes de 0", f"{custo_medio:.0f}"])
```

```
        writer.writerow(["Relacoes de coancestralidade
        iguais a 0", f"{(len(solucao) * (len(solucao)-1))//2
        - int(custo_medio)}"])
        writer.writerow(["Resumo da Solucao"])
        writer.writerow(["Custo Total da Solucao", f"
        {custo_medio:.4f}"])
        writer.writerow([])
        writer.writerow(["Estatisticas do Indice de Selecao"])
        writer.writerow(["Metrica", "Valor"])
        for k, v in iecc_medio.items():
            writer.writerow([k.capitalize().replace("_", " "),
            f"{v:.4f}"])
        writer.writerow([])
        writer.writerow(["Estatisticas da Consanguinidade"])
        writer.writerow(["Metrica", "Valor"])
        for k, v in consang_media.items():
            writer.writerow([k.capitalize().replace("_", " "),
            f"{v:.4f}"])
        writer.writerow([])
        writer.writerow(["Animais Selecionados"])
        writer.writerow(["Animal_ID"])
        for animal in solucao:
            writer.writerow([animal])

def inicializar_feromonios(dados):
    return {dado['id']: 1.0 for dado in dados}

def construir_solucao_individual(dados, feromonios,
coef_parentesco, alpha, beta, limite_maes,
limite_uso_macho, limite_consanguinidade):
    selecionados = set()
    uso_femeas = set()
    uso_machos = {}
    custo_total = iecc_total = cons_total = 0
    consanguinidade_total = 0
    iecc_individual = []
    consanguinidade_individual = []

    while len(uso_femeas) < limite_maes:
        candidatos = [
            d for d in dados
            if d['id'] not in selecionados
            and d['mid'] not in uso_femeas
            and d['consanguinidade'] <= limite_consanguinidade
            and uso_machos.get(d['pid'], 0) < limite_uso_macho
        ]

        if not candidatos:
            logger.info("Sem candidatos disponiveis, modifique as restri  es.")
            break

        probabilidades = []
```

```
        for c in candidatos:
            heuristica = 1.0
            if selecionados:
                heuristica = 1 / (1 +
                sum(coef_parentesco.get(tuple(sorted([c['id'],
                s])), 0) for s in selecionados))
            fer = feromonios.get(c['id'], 1.0)
            prob = (fer ** alpha) * (heuristica ** beta)
            probabilidades.append((c, prob))

        total_prob = sum(p for _, p in probabilidades)
        if total_prob == 0:
            break

        probabilidades = [(c, p / total_prob) for c, p in probabilidades]
        escolhido = random.choices([c for c, _ in probabilidades],
        weights=[p for _, p in probabilidades])[0]

        selecionados.add(escolhido['id'])
        uso_femeas.add(escolhido['mid'])
        uso_machos[escolhido['pid']] = uso_machos.get(escolhido['pid'], 0) + 1

        for s in selecionados:
            if s == escolhido['id']:
                continue
            coef = coef_parentesco.get(tuple(sorted([escolhido['id'], s])), 0)
            if coef != 0:
                custo_total += 1

        iecc_total += escolhido['iecc']
        iecc_individual.append(escolhido['iecc'])
        cons_total += escolhido['consanguinidade']
        consanguinidade_individual.append(escolhido['consanguinidade'])

    n = len(selecionados)
    if n == 0:
        return set(), float('inf'), 0.0, 0.0
    return selecionados, custo_total,
    calcular_estatisticas(iecc_individual),
    calcular_estatisticas(consanguinidade_individual)

def avaliar_atualizar(solucoes, feromonios, rho, q,
melhor_global, iteracao, tau_max, tau_min):
    validas = [s for s in solucoes if len(s[0]) > 0]
    if not validas:
        return feromonios, melhor_global

    reforco_global = (iteracao % 2 == 0) if ALTERNAR_REFORCO else True
    if reforco_global and melhor_global:
        sol_reforco = melhor_global[0]
        custo_reforco = melhor_global[1]
    else:
```

```
169         melhor_iter = min(validas, key=lambda x: x[1])
170         sol_reforco = melhor_iter[0]
171         custo_reforco = melhor_iter[1]
172 
173     for chave in feromonios:
174         feromonios[chave] *= (1 - rho)
175 
176     for id_sel in sol_reforco:
177         delta = q / custo_reforco
178         feromonios[id_sel] = min(tau_max, max(tau_min, feromonios[id_sel] + delta))
179 
180     return feromonios, (sol_reforco, custo_reforco)
181 
182 def otimizar(caminho_produtos, caminho_parentesco,
183 num_iteracoes, num_formigas, alpha, beta, rho, q,
184 limite_uso_macho, limite_consanguinidade, caminho_saida):
185     logger.info("Iniciando MMAS")
186     dados, limite_maes = carregar_dados_produtos(caminho_produtos)
187     coef_parentesco = carregar_coeficientes_parentesco(caminho_parentesco)
188     feromonios = inicializar_feromonios(dados)
189 
190     melhor_global = None
191     melhor_iecc = melhor_consang = None
192 
193     for it in range(num_iteracoes):
194         logger.info(f"Itera  o {it + 1}/{num_iteracoes}")
195         with ThreadPoolExecutor(max_workers=4) as executor:
196             solucoes = list(executor.map(
197                 lambda _: construir_solucao_individual(dados,
198                 feromonios, coef_parentesco, alpha, beta,
199                 limite_maes, limite_uso_macho,
200                 limite_consanguinidade),
201                 range(num_formigas)
202             ))
203 
204         iter_melhor = min(solucoes, key=lambda x: x[1]
205         if len(x[0]) > 0 else float('inf'))
206 
207         if not melhor_global or iter_melhor[1] < melhor_global[1]:
208             melhor_global = iter_melhor
209             melhor_iecc = iter_melhor[2]
210             melhor_consang = iter_melhor[3]
211             logger.info(f"Nova melhor: custo=
212             {iter_melhor[1]:.4f}, IECC=
213             {melhor_iecc['media']:.4f}, Consang=
214             {melhor_consang['media']:.4f}")
215 
216         n = len(melhor_global[0]) if melhor_global else 1  # Evita divis o por zero
217         tau_max, tau_min = calcular_tau_max_min(rho, melhor_global[1], n)
218 
219         feromonios, melhor_global = avaliar_atualizar(
220             solucoes, feromonios, rho, q, melhor_global, it, tau_max, tau_min
```

```
        )

    salvar_resultados(melhor_global[0], melhor_global[1],
    melhor_iecc, melhor_consang, caminho_saida)
    return melhor_global[0], melhor_global[1],
    melhor_iecc, melhor_consang

# Execu  o
if __name__ == "__main__":
    caminho_produtos = '/novos_produtos.csv'
    caminho_parentesco = '/parentesco_produtos.csv'
    base_saida = '/'

    combinacoes = [
        {'num_iteracoes': 30, 'num_formigas': 10,
        'alpha': 1.0, 'beta': 2.0},
        {'num_iteracoes': 30, 'num_formigas': 5,
        'alpha': 1.0, 'beta': 2.0},
        {'num_iteracoes': 30, 'num_formigas': 1,
        'alpha': 1.0, 'beta': 2.0},

    ]

    parametros_fixos = {
        'rho': 0.1,
        'q': 1,
        'limite_uso_macho': 20,
        'limite_consanguinidade': 0.05,
    }

    caminho_resumo = base_saida + 'resumo_experimentos.csv'
    with open(caminho_resumo, mode='w', newline='') as resumo_file:
        writer = csv.writer(resumo_file)
        writer.writerow([
            'Execucao', 'Iteracoes', 'Formigas', 'Alpha', 'Beta',
            'Custo Medio', 'Consang Media', 'Consang Mediana',
            'Consang Min', 'Consang Max', 'Consang Desvio Padrao',
            'Arquivo Resultado', 'Tempo (segundos)'
        ])

        for i, combinacao in enumerate(combinacoes, start=1):
            nome_saida = f'resultado_execucao_{i:02d}.csv'
            caminho_saida = base_saida + nome_saida

            parametros = {**combinacao, **parametros_fixos}
            logger.info(f"\n=== Executando configura  o {i}: {parametros} ===")

            inicio = time.perf_counter()
            selecionados, custo, iecc_stats, consang_stats = otimizar(
                caminho_produtos,
                caminho_parentesco,
                **parametros,
```

```
                caminho_saida=caminho_saida
            )
            tempo_total = time.perf_counter() - inicio

            logger.info(
                f"Resultado final da execu  o {i}: "
                f"custo={custo:.4f},
                consang_media={consang_stats['media']:.4f},
                tempo={tempo_total:.2f} segundos"
            )

            writer.writerow([
                i,
                combinacao['num_iteracoes'],
                combinacao['num_formigas'],
                combinacao['alpha'],
                combinacao['beta'],
                round(custo, 4),
                round(consang_stats['media'], 4),
                round(consang_stats['mediana'], 4),
                round(consang_stats['minimo'], 4),
                round(consang_stats['maximo'], 4),
                round(consang_stats['desvio_padrao'], 4),
                nome_saida,
                round(tempo_total, 2)
            ])
```

Lista I.1: Algoritmo MMAS para seleção de animais com restrições

I.5 Resultados registrados

A cada execução, os seguintes dados são salvos:

- **Número de pares com coancestralidade diferente de zero** (i.e., o custo).
- **Número de pares com coancestralidade igual a zero**, calculado como:

$$\text{Total}_0 = \binom{n}{2} - \text{Custo}(S)$$

- **Estatísticas do índice de seleção**: média, mediana, desvio padrão, etc.
- **Estatísticas da consanguinidade individual dos animais selecionados**.
- **Lista de IDs dos animais selecionados**.

I.6 Vantagens dessa abordagem

Ao utilizar o número de relações genéticas como métrica de avaliação, esta versão do algoritmo enfatiza a diversidade genética da solução. Ela é particularmente útil em cenários em que:

- A penalização por relações aparentadas é desejável, mas os coeficientes absolutos (como 0,015 ou 0,03) têm menos relevância do que a simples presença de parentesco.
- Deseja-se gerar grupos com o maior número possível de indivíduos não aparentados.

I.7 Comparação com a versão baseada em soma dos coeficientes

A principal diferença em relação à versão anterior do MMAS é o critério de custo:

Tabela 26 – Comparativo entre versões do algoritmo MMAS

Critério	**Versão original**	**Versão atual**
Custo	Soma dos coeficientes de parentesco	Número de relações > 0
Objetivo	Minimizar a intensidade do parentesco	Minimizar a quantidade de parentesco
Tipo de penalização	Quantitativa (peso dos coef.)	Binária (existe ou não parentesco)

Fonte: Autor (2025)