Universidade Federal de Pernambuco
Centro de Informática
Departamento de Ciência da Computação

Programa de Pós-Graduação em Ciência da Computação

Uma Metodologia Baseada em Grafos para Detecção de Redundância Estrutural em Arquiteturas Data Mesh

Cayo Felipe Lopes de Oliveira

Dissertação de Mestrado

Recife
Julho de 2025

Universidade Federal de Pernambuco
Centro de Informática
Departamento de Ciência da Computação

Cayo Felipe Lopes de Oliveira

Uma Metodologia Baseada em Grafos para Detecção de Redundância Estrutural em Arquiteturas Data Mesh

Trabalho apresentado ao Programa de Programa de Pós-Graduação em Ciência da Computação do Departamento de Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr. Jamilson Ramalho Dantas*
Co-orientador: *Prof. Dr. Jean Araujo*

Recife
Julho de 2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Oliveira, Cayo Felipe Lopes.
Uma metodologia baseada em grafos para detecção de redundância estrutural em Arquiteturas Data Mesh / Cayo Felipe Lopes de Oliveira. - Recife, 2025.
141f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2025.
Orientação: Jamilson Ramalho Dantas.
Coorientação: Jean Araújo.
Inclui referências e apêndices.

1. Data Mesh; 2. Grafos; 3. Redundância de dados; 4. Algoritmo de grafos; 5. Isomera. I. Dantas, Jamilson Ramalho. II. Araújo, Jean. III. Título.

UFPE-Biblioteca Central

Cayo Felipe Lopes de Oliveira

“Uma Metodologia Baseado em Grafos para Detecção de Redundância Estrutural em Arquiteturas Data Mesh”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 31/07/2025.

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática / UFPE

_____

Prof. Dr. Rubens de Souza Matos Júnior
Departamento de Ciências da Computação/IFS

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática/UFPE
(**orientador)**

Aos meus pais, por cada palavra de incentivo, por me ensinarem o valor dos estudos e por nunca deixarem que eu desistisse dos meus sonhos.
À minha esposa, por estar ao meu lado mesmo quando eu duvidava.
À minha filha, ainda tão pequena, mas já tão inspiradora. Que você cresça sabendo que o conhecimento é um caminho de liberdade.
E a todos que, nos meus dias mais difíceis, seguiram acreditando em mim.

Agradecimentos

A jornada do mestrado foi intensa, desafiadora e transformadora, e não teria sido possível sem o apoio constante de pessoas que estiveram ao meu lado nos momentos mais decisivos.

Agradeço, de forma especial, aos meus amigos e colegas de trabalho, que apoiaram essa empreitada com palavras de encorajamento, escuta generosa e apoio verdadeiro nos momentos em que conciliar estudo, trabalho e vida pessoal parecia impossível.

À minha família, que sempre me acolheu com afeto e palavras de força, renovando minha disposição mesmo nas fases mais difíceis.

Ao meu orientador, Professor Jamilson Dantas, por ter acreditado no potencial deste trabalho quando ele ainda era apenas uma ideia confusa. Sua orientação foi uma verdadeira virada de chave, técnica, acadêmica e pessoal, permitindo que eu reencontrasse propósito e clareza ao longo da pesquisa.

Ao Professor Paulo Maciel, por sua sensibilidade acadêmica e generosidade em compreender o contexto deste trabalho, contribuindo de forma decisiva para que o caminho da pesquisa fosse mais claro e frutífero.

Ao Itaú Unibanco, pelo incentivo institucional e pela oportunidade concreta de conciliar carreira profissional e formação acadêmica.

Por fim, agradeço aos meus pais, por serem minha base inabalável; à minha esposa, pelo amor e paciência nos dias em que estive ausente; e à minha filha, que, mesmo tão pequena, já me inspira a ser melhor a cada dia.

A todos, meu mais sincero obrigado.

Encontrei uma demonstração verdadeiramente maravilhosa para esta proposição, mas esta margem é demasiadamente estreita para contê-la.

—PIERRE DE FERMAT (Último Teorema de Fermat)

Resumo

A adoção do paradigma *Data Mesh* tem impulsionado a descentralização da posse de dados nas organizações, permitindo que cada domínio de negócio gerencie seus próprios produtos de dados. Embora essa abordagem aumente a autonomia e a flexibilidade, ela também intensifica o risco de criação de tabelas estruturalmente semelhantes entre os domínios, gerando redundâncias que comprometem a governança, a rastreabilidade e a eficiência dos recursos. Esta dissertação apresenta uma metodologia para detecção de redundâncias estruturais em arquiteturas de dados distribuídas, fundamentada na modelagem de tabelas como grafos direcionados e na aplicação de algoritmos de isomorfismo de subgrafos. Nesse contexto, três abordagens foram consideradas: o VF2, utilizado como referência consolidada na literatura; o *Node Match*, um algoritmo híbrido desenvolvido neste trabalho com função de pré-filtragem; e modelos supervisionados baseados em redes neurais gráficas (GNN), aplicados à predição de isomorfismos. Como parte das contribuições, também foi desenvolvida a ferramenta *Isomera*, em *Python*, responsável por operacionalizar a metodologia e permitir sua experimentação prática. A proposta organiza um ciclo de experimentação em quatro etapas: geração automatizada de cenários, aplicação dos algoritmos, validação humana supervisionada, necessária para mitigar falsos positivos e assegurar confiabilidade, e avaliação quantitativa dos resultados. A ferramenta *Isomera* possibilita simular arquiteturas sintéticas ou baseadas em benchmarks consolidados, como o TPC-DS, além de oferecer recursos para a execução controlada de experimentos, a comparação entre algoritmos e a análise de métricas como tempo de execução (ET), acurácia (ACC) e frequência de sucesso (SF). Dois estudos de caso ilustraram a aplicação da metodologia: o primeiro, utilizando VF2 e *Node Match*, e o segundo, incorporando redes neurais (GNN), que demonstraram ganhos de acurácia em cenários de maior complexidade, ainda que com maior custo computacional. Adicionalmente, esta dissertação contribui com um artefato científico e experimental que favorece a replicação de testes, a expansão modular com novos algoritmos e a análise sistemática de trade-offs entre performance e precisão. Ao unir flexibilidade, reprodutibilidade e análise crítica, o trabalho oferece uma base sólida para pesquisadores e profissionais que buscam aprimorar a governança de dados em arquiteturas distribuídas.

Palavras-chave: Data Mesh, Redundância Estrutural, Governança de Dados, Isomorfismo de Grafos, VF2, Node Match, GNN, Isomera, TPC-DS.

Abstract

The adoption of the *Data Mesh* paradigm has fostered the decentralization of data ownership within organizations, allowing each business domain to manage its own data products. While this approach increases autonomy and flexibility, it also intensifies the risk of structurally similar tables being created across domains, leading to redundancies that compromise governance, traceability, and resource efficiency. This dissertation presents a methodology for detecting structural redundancies in distributed data architectures, based on modeling tables as directed graphs and applying subgraph isomorphism algorithms. In this context, three approaches were considered: VF2, widely used as a reference in the literature; *Node Match*, a hybrid algorithm developed in this work with a pre-filtering function; and supervised models based on Graph Neural Networks (GNN), applied to isomorphism prediction. As an additional contribution, the *Isomera* tool was developed in *Python*, designed to operationalize the methodology and enable its practical experimentation. The proposed methodology follows a four-step experimental cycle: automated scenario generation, algorithm application, human-supervised validation — necessary to mitigate false positives and ensure reliability — and quantitative evaluation of results. The *Isomera* tool enables the simulation of synthetic architectures or those based on consolidated benchmarks such as TPC-DS, while providing features for controlled experiment execution, algorithm comparison, and metric analysis, including execution time (ET), accuracy (ACC), and success frequency (SF). Two case studies illustrated the application of the methodology: the first using VF2 and *Node Match*, and the second incorporating GNNs, which demonstrated accuracy gains in more complex scenarios, albeit with higher computational cost. Additionally, this dissertation contributes a scientific and experimental artifact that supports test replication, modular expansion with new algorithms, and systematic analysis of trade-offs between performance and accuracy. By combining flexibility, reproducibility, and critical analysis, the work offers a solid foundation for researchers and practitioners seeking to improve data governance in distributed architectures.

Keywords: Data Mesh, Structural Redundancy, Data Governance, Graph Isomorphism, VF2, Node Match, GNN, Isomera, TPC-DS.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

SOR	System of Record.
SOT	System of Transformation.
SPEC	Specialized Processing Engines.
VF2	Algoritmo de isomorfismo de subgrafos VF2.
GNN	Graph Neural Network.
GIN	Graph Isomorphism Network.
MLP	Perceptron Multicamadas.
ReLU	Rectified Linear Unit.
TPC	Transaction Processing Performance Council.
TPC-DS	Transaction Processing Performance Council Decision Support benchmark.
AWS	Amazon Web Services.
S3	Amazon Simple Storage Service.
API	Application Programming Interface.
ERP	Enterprise Resource Planning.
CRM	Customer Relationship Management.
DAG	Directed Acyclic Graph.
JSON	JavaScript Object Notation.
CSV	Comma-Separated Values.
ML	Machine Learning.
ET	Elapsed Time.
ACC	Accuracy.
SF	Success Frequency.

Lista de Símbolos

$G = (V, E)$	Grafo dirigido utilizado neste trabalho.
V	Conjunto de vértices do grafo.
E	Conjunto de arestas do grafo.
$G' = (V', E')$	Subgrafo de G.
(u, v)	Aresta dirigida do vértice u para o vértice v.
$\|V\|, \|E\|$	Cardinalidades (número de elementos) de V e E.
$\rightarrow$	Relação/ligação direcionada de origem para destino.
M	Matriz de dependências/adjacências entre elementos.
A	Matriz de adjacência (direcionada), $A \in \mathbb{N}^{\|V\| \times \|V\|}$.
$\mathbb{N}$	Conjunto dos números naturais.
ℓ	Função de rótulos em $V \cup E$ (não utilizada nesta dissertação).
w	Pesos atribuídos às arestas (não utilizado nesta dissertação).
$f : V \rightarrow V'$	Bijeção de vértices (isomorfismo).
P	Matriz de permutação associada à bijeção f.
$\cup$	União de conjuntos.
$h_v^{(l)}$	Estado oculto (embedding) do vértice v na camada l.
$h_v^{(l+1)}$	Estado oculto atualizado do vértice v na camada $l+1$.
$\varepsilon^{(l)}$	Parâmetro (escalar) aprendível da camada l.
$\mathcal{N}(v)$	Conjunto de vizinhos do vértice v no grafo.
u, v	Vértices do grafo.
l	Índice de camada (inteiro não negativo).

CAPÍTULO 1

Introdução

1.1 Contextualização

O avanço da transformação digital, intensificado pela incorporação estratégica de tecnologias como a inteligência artificial, tem provocado uma profunda reconfiguração nos processos organizacionais, impulsionando a geração massiva de dados em múltiplos contextos e setores. Ao viabilizar automações, personalizações e inovações em escala, este avanço acelerado promove ganhos em eficiência, qualidade e tomada de decisão, mas também impõe desafios gerenciais e culturais para as organizações (KAVAK; RUSU, 2025).

Esse crescimento, impulsionado por demandas analíticas e operacionais cada vez mais complexas, trouxe à tona desafios estruturais ligados à governança, à redundância e à interoperabilidade de dados (BENA *et al.*, 2025). Tradicionalmente, arquiteturas centralizadas como Data Warehouses e Data Lakes dominaram as arquiteturas tecnológicas. Contudo, essas abordagens monolíticas vêm demonstrando limitações frente à crescente heterogeneidade dos negócios e à necessidade de autonomia entre as equipes (GIEß; HUTTERER, 2025).

Nesse cenário, o paradigma do *Data Mesh* surge como uma proposta moderna que busca descentralizar a posse e a responsabilidade sobre os dados, atribuindo aos próprios domínios a missão de gerenciar seus produtos de dados com independência. Essa arquitetura promove maior escalabilidade organizacional ao romper com o gargalo imposto pelas equipes centrais de engenharia de dados e alinhar a tomada de decisões às equipes mais próximas da informação (BODAPATI, 2025).

Em linhas gerais, um domínio de negócio é um grupo de pessoas com profundo conhecimento sobre uma área específica da organização, como vendas, logística ou finanças, sendo, portanto, o responsável natural pela geração e interpretação dos dados relacionados à sua atividade. Ao transferir para os domínios a responsabilidade pelo ciclo de vida dos dados, o *Data Mesh* promove maior contextualização, agilidade e qualidade no tratamento da informação.

Concebido por Zhamak Dehghani, o *Data Mesh* se fundamenta em quatro pilares essenciais: (i) a propriedade dos dados por domínio, que transfere a responsabilidade pela gestão dos dados para os próprios domínios, valorizando o conhecimento específico e contextual de cada equipe sobre os ativos sob sua responsabilidade; (ii) o dado como produto, o que implica tratá-lo com critérios de usabilidade, confiabilidade e qualidade; (iii) uma plataforma de dados de autosserviço, que permite aos domínios criar, transformar e servir dados com mínima dependência técnica; e (iv) a governança federada computacional, que harmoniza o controle de acesso e a padronização sem comprometer a autonomia (DEHGHANI, 2019).

Para situar visualmente essa arquitetura, a Figura 1.1 sintetiza o *Data Mesh*: domínios autônomos (Pagamentos, Finanças, Marketing e *Machine Learning*) compartilham dados, em

geral, em nuvem, enquanto dois times transversais dão suporte: (i) equipe de infraestrutura (*platform team*), responsável por armazenamento, processamento e serviços de autosserviço; e (ii) governança, responsável por políticas, segurança e qualidade.

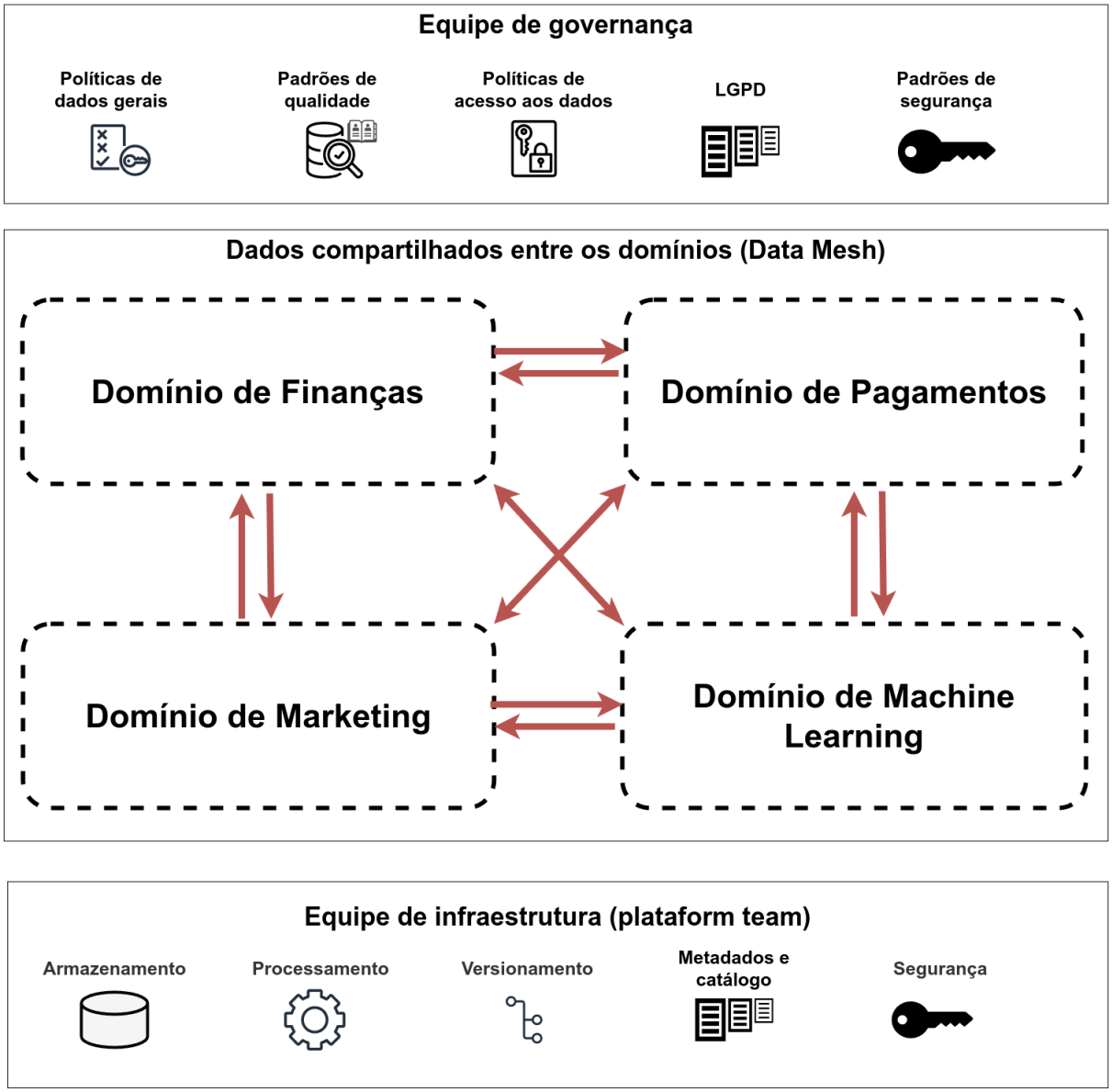


Figura 1.1: Arquitetura *Data Mesh* em alto nível com domínios autônomos, equipe de infraestrutura e equipe de governança.

Neste tipo de arquitetura, há conceitos importantes sobre como as tabelas são criadas e nomeadas ao longo do fluxo de dados. Para orientar a leitura, a Figura 1.2 apresenta, de forma didática, três camadas recorrentes: (i) camada de registro, *System of Record* (SOR), onde residem os dados brutos; (ii) camada de transformação, *System of Transformation* (SOT), na qual os dados são tratados e estruturados; e (iii) camada de processamento especializado, *Speci-*

alized Processing Engines (SPEC), voltada a análises avançadas. Em cenários reais, podem existir dependências cruzadas (por exemplo, SOT a partir de SOT, SPEC a partir de SPEC ou diretamente de SOR); aqui adotamos a organização idealizada apenas para fins didáticos.

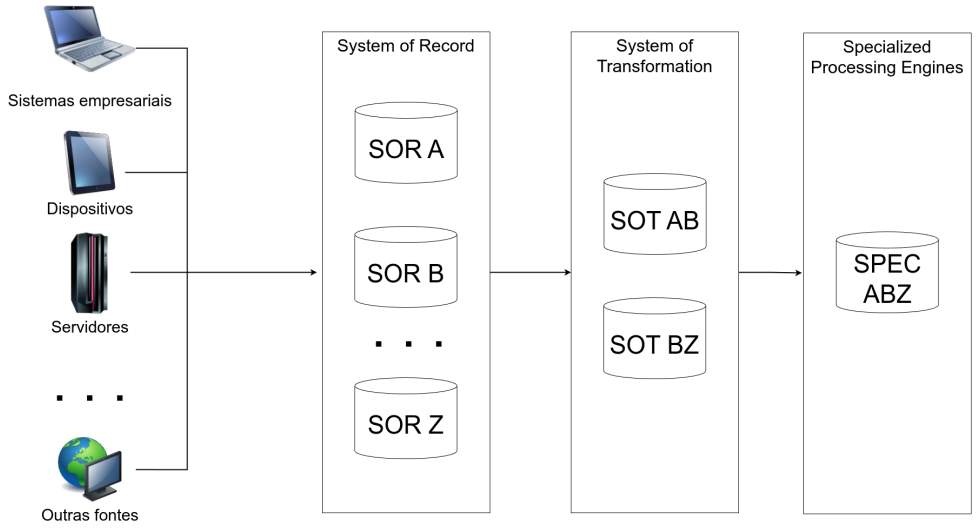


Figura 1.2: Relação idealizada entre camadas: SOR → SOT → SPEC (arquitetura "limpa"). Na prática, podem ocorrer dependências cruzadas.

Cada domínio passa a funcionar como uma unidade lógica autônoma dentro da organização, sendo responsável por criar, manter e compartilhar suas próprias tabelas e ativos. Esses ativos, em geral, são construídos a partir de sistemas de registro, *System of Record* (SOR), responsáveis por armazenar os dados em sua forma original, como ERPs (*Enterprise Resource Planning*), CRMs (*Customer Relationship Management*), e bases de logs e sensores. Essa camada é fundamental para garantir a integridade, a rastreabilidade e a persistência das informações.

A partir dessa base, os domínios criam estruturas intermediárias conhecidas como *System of Transformation* (SOT), onde os dados são submetidos a processos de limpeza, normalização, enriquecimento e agregação, com o objetivo de estruturar informações mais coerentes e utilizáveis para consumo organizacional.

Finalmente, para demandas específicas e análises mais complexas, são utilizadas camadas de processamento especializadas, chamadas *Specialized Processing Engines* (SPEC), nas quais se aplicam técnicas avançadas como aprendizado de máquina, análise preditiva e modelagem estatística, viabilizando a geração de insights estratégicos para os domínios de negócio [1]. Sem padronização ou mecanismos de validação cruzada, essa evolução pode favorecer o surgimento de estruturas redundantes, um problema ainda pouco discutido em soluções práticas de *Data Mesh*.

Para evidenciar esse problema de forma concreta, a Figura 1.3 ilustra um exemplo em que diferentes domínios geram e consomem dados (A–H). Observa-se uma duplicidade entre as tabelas E (Pagamentos) e G (*Machine Learning*), ambas derivadas das bases B e D. Ainda que

[1] A. Hashimoto, SoR, SoT e Spec no contexto de Engenharia de Dados, Alura, 2024. Disponível em: <https://www.alura.com.br/artigos/sor-sot-spec>. Acesso em: 6 out. 2025.

regras de negócio possam diferir entre esses domínios, o conteúdo estrutural dessas tabelas poderia ser consolidado em uma única base, reduzindo redundâncias e custos.

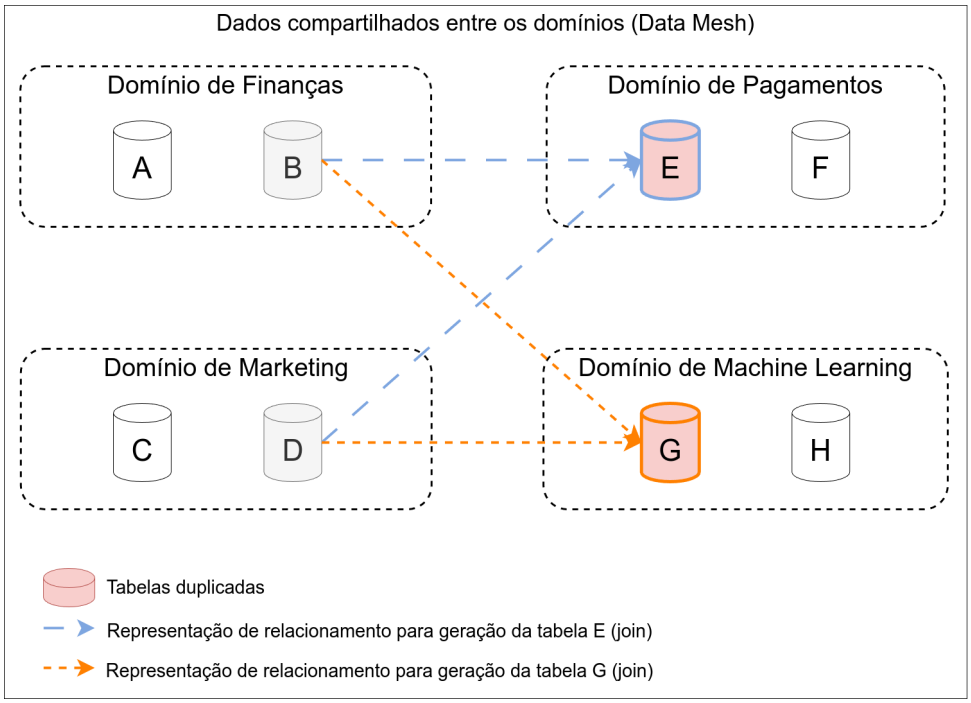


Figura 1.3: Exemplo de duplicidade: E (Pagamentos) e G (ML) derivadas de B e D em domínios distintos, sugerindo redundância potencial.

Embora o *Data Mesh* promova flexibilidade e agilidade, ela também introduz riscos: a proliferação de tabelas estruturalmente semelhantes entre domínios, como vimos na figura anteriors. Essas redundâncias, quando não gerenciadas adequadamente, afetam diretamente a governança de dados, elevam custos operacionais e dificultam a rastreabilidade das informações (DEVI; INAMPUDI; VIJAYABOOPATHY, 2025).

Em ambientes de nuvem, onde os custos estão diretamente associados à escalabilidade e ao volume de armazenamento, a duplicação de estruturas entre domínios pode comprometer significativamente a sustentabilidade financeira da arquitetura de dados. O acúmulo de tabelas redundantes em um cenário descentralizado pode levar a um crescimento desproporcional dos custos de manutenção, superando, em certos casos, o valor agregado pela descentralização dos dados [2].

Estudos recentes apontam que mesmo empresas de grande porte enfrentam desafios ao adotar o *Data Mesh*, em especial no que tange à duplicidade de estruturas, ausência de padrões e dificuldades na consolidação de políticas de qualidade. Casos como o da Netflix ilustram as dificuldades técnicas e organizacionais para alcançar uma descentralização sustentável [3].

[2] H. Rollin, *The Brutal Cost of Data Mesh*, Medium, 2023. Disponível em: <https://medium.com/@hannes.rollin/the-brutal-cost-of-data-mesh-df8cec245506>. Acesso em: 6 out. 2025.

[3] *Netflix*, Netflix Technology Blog, 2022. Disponível em: <https://netflixtechblog.com/data-mesh-a-data-movement-and-processing-platform-netflix-1288bcab2873>. Acesso em: 21 jul. 2025.

O problema se agrava quando múltiplos domínios criam tabelas com esquemas muito similares, mas ligeiramente adaptados às suas realidades locais, dificultando o reconhecimento automático dessas redundâncias por sistemas de governança. Essa fragmentação semântica e estrutural impõe desafios concretos à governança e amplia a necessidade de métodos computacionais que consigam identificar e mitigar tais redundâncias.

Dentre as abordagens emergentes para identificação dessas duplicidades, a representação estrutural por grafos tem se mostrado particularmente promissora. Grafos permitem capturar com precisão as relações entre tabelas, colunas, tipos de dados e chaves, viabilizando a modelagem do ecossistema de dados como uma rede de dependências e estruturas (PATEL; DHARWA, 2017).

Essa representação facilita a aplicação de algoritmos de comparação estrutural, como detecção de isomorfismos, que podem automatizar a identificação de duplicações e padrões recorrentes em larga escala (REN; LI, 2024). Ao explorar tais capacidades, abre-se espaço para práticas de governança mais automatizadas, escaláveis e eficazes, reduzindo os custos operacionais associados ao controle manual de ativos.

Para tornar essa ideia concreta no nosso exemplo, modelamos a arquitetura como um grafo direcionado: cada tabela, (X e Y), por exemplo, são vértices e criamos uma aresta $X \to Y$ quando Y é derivada ou consome dados de X. No cenário da Figura 1.3, os vértices são B, D, E e G e as dependências são $B \to E$, $D \to E$, $B \to G$ e $D \to G$. Formalmente, focamos no subgrafo $G' = \{V, E\}$, e não no grafo completo contendo A, C, F e H, para destacar apenas a porção diretamente ligada à geração de E e G, com $V = \{B, D, E, G\}$ e $E = \{(B,E), (D,E), (B,G), (D,G)\}$. A Figura 1.4 ilustra esse subgrafo.

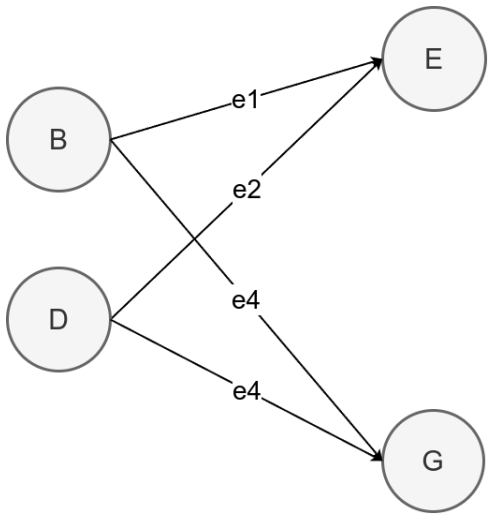


Figura 1.4: Subgrafo G' destacando a geração de E e G a partir de B e D.

Neste cenário, torna-se estratégico o desenvolvimento de soluções computacionais capazes de automatizar a identificação e a validação de redundâncias estruturais em ecossistemas de dados complexos e distribuídos.

Embora esta dissertação tenha sido concebida no contexto de arquiteturas baseadas em *Data Mesh*, a metodologia proposta possui caráter geral e pode ser aplicada a qualquer ambiente em

que as relações entre tabelas possam ser formalmente representadas como grafos. Isso inclui arquiteturas tradicionais, como *Data Warehouses* e *Data Lakes*, bem como sistemas híbridos que combinam características de ambos. Dessa forma, o método aqui apresentado não se restringe a um modelo de governança específico, mas se consolida como uma estrutura analítica adaptável para o estudo, a avaliação e a otimização de diferentes arquiteturas de dados.

1.2 Motivação e Justificativa

A autonomia conferida aos domínios de negócio no paradigma *Data Mesh* representa uma transformação significativa no modo como os dados são concebidos, tratados e disponibilizados dentro das organizações. Contudo, ao mesmo tempo em que permite maior escalabilidade e responsividade às demandas locais, essa descentralização tem exposto lacunas críticas nos mecanismos de controle de qualidade e padronização de estruturas (SERRA, 2024).

Um dos reflexos mais notórios dessa descentralização é o surgimento de tabelas estruturalmente redundantes, artefatos que compartilham esquemas semelhantes, mas que são desenvolvidos de forma paralela e não coordenada por diferentes equipes. Essas redundâncias não apenas aumentam a complexidade dos pipelines de dados, como também geram desperdícios de recursos computacionais em plataformas baseadas em nuvem. Em um cenário onde o armazenamento e o processamento são cobrados por volume e desempenho, a ausência de mecanismos automatizados para detectar tais duplicidades representa uma vulnerabilidade operacional (JI *et al.*, 2012).

Casos reais ilustram esse problema com clareza: instituições como o Itaú Unibanco relataram desafios concretos relacionados ao acúmulo excessivo de arquivos em buckets do Amazon S3, que elevaram significativamente os custos de armazenamento. A empresa foi obrigada a desenvolver indicadores, aplicar regras de ciclo de vida e adotar estratégias específicas de arquivamento para mitigar os impactos financeiros de estruturas pouco otimizadas.[4]

Mais do que um problema de custo, trata-se de uma questão de confiança. Tabelas similares, quando mal documentadas ou desconectadas de um catálogo unificado, podem induzir a interpretações conflitantes sobre o mesmo fenômeno de negócio, comprometendo a consistência analítica e a confiabilidade dos dashboards e modelos que delas derivam. Essa fragmentação dificulta a rastreabilidade de dados e sobrecarrega processos de governança que deveriam ser federados e automatizados, como preconizado no próprio modelo de *Data Mesh* (KAPFERER; BRUNNER; ZELLER, 2021).

Diante disso, esta dissertação se justifica pela necessidade urgente de estruturar uma abordagem sistemática para identificação de redundâncias estruturais em ambientes de dados distribuídos. Ao empregar algoritmos de isomorfismo de grafos, propõe-se não apenas reconhecer similaridades, mas também inferir equivalências estruturais entre tabelas distribuídas.

Essa combinação entre técnicas computacionais e conhecimento especializado visa fortalecer os pilares da governança de dados, promovendo maior transparência, padronização e

[4]R. Lovatti, J. Escoaiella, T. B. dos Santos, H. Papa, O. Correia, K. Oliveira e R. M. Dias, Como o Itaú reduziu custos de armazenamento no Amazon S3, Blog da AWS – Seção Customer Solutions, Amazon Web Services, 14 fev. 2025. Disponível em: <https://aws.amazon.com/pt/blogs/aws-brasil/como-o-itau-reduziu-custos-de-armazenamento-no-amazon-s3/>. Acesso em: 6 out. 2025.

eficiência em arquiteturas modernas orientadas por domínios.

1.3 Trabalhos Relacionados

A literatura recente tem abordado com crescente interesse os desafios oriundos da descentralização de dados em arquiteturas distribuídas. Dehghani foi pioneira ao estabelecer os pilares do *Data Mesh*, apontando para a necessidade de reorganizar a responsabilidade sobre os dados a partir da lógica de domínios (KAPFERER; BRUNNER; ZELLER, 2021). Serra complementa essa visão com uma análise crítica das principais arquiteturas modernas, como *Data Lakehouse* e *Data Fabric*, destacando que, em ambientes distribuídos, a governança tende a se tornar mais frágil sem ferramentas automatizadas de controle (SERRA, 2024).

No contexto de governança e engenharia de dados, Hashimoto destaca como a evolução dos pipelines a partir das fontes SOR pode gerar camadas sucessivas (SOT e SPEC), aumentando a complexidade estrutural dos dados gerados por cada domínio [5]. Quando não há padronização ou mecanismos de validação cruzada, essa evolução pode favorecer o surgimento de estruturas redundantes, um problema ainda pouco discutido em soluções práticas de *Data Mesh*.

Na tentativa de identificar tais redundâncias em arquiteturas não modernizadas em *Data Mesh*, diferentes abordagens baseadas em teoria dos grafos têm sido exploradas. Ren e Li discutem definições fundamentais de isomorfismo de grafos e sua aplicação em bases relacionais (REN; LI, 2024). Já Mancinska et al. exploram variações do problema de isomorfismo em grafos com atributos semânticos, demonstrando que, em determinadas configurações, o reconhecimento estrutural pode ultrapassar os limites da tratabilidade clássica, exigindo abordagens heurísticas para viabilizar soluções computacionais em tempo viável (MANčINSKA; ROBERSON; VARVITSIOTIS, 2024).

Cordella et al. introduziram o algoritmo *VF2*, uma das abordagens determinísticas mais consolidadas para verificação de isomorfismo e subisomorfismo de grafos, cuja eficiência decorre da combinação entre uma representação em espaço de estados (*State Space Representation*) e um conjunto de regras de viabilidade (*feasibility rules*) que reduzem drasticamente o espaço de busca (CORDELLA *et al.*, 2004). Seu diferencial reside na capacidade de integrar, simultaneamente, comparações sintáticas e semânticas entre pares de nós, sendo especialmente eficaz em grafos atribuídos.

Em contextos com grandes volumes de dados, como em empresas ou aplicações científicas, é comum lidar com grafos que apresentam ruídos, falhas ou estruturas incompletas. Nessas situações, métodos que exigem uma correspondência exata entre os grafos tendem a ser pouco eficazes. É nesse cenário que os algoritmos baseados em distância de edição de grafos (*Graph Edit Distance – GED*) ganham destaque, por serem mais flexíveis e tolerantes a imperfeições. A proposta de Zheng et al. se destaca justamente por adotar uma abordagem que combina técnicas para reduzir o esforço computacional durante a busca por grafos semelhantes, mantendo bons níveis de precisão (ZHENG *et al.*, 2015). Os resultados experimentais apresentados pelos autores demonstram que sua solução é capaz de superar métodos tradicionais tanto em agilidade

[5] A. Hashimoto, SoR, SoT e Spec no contexto de Engenharia de Dados, Alura, 2024. Disponível em: <https://www.alura.com.br/artigos/sor-sot-spec>. Acesso em: 6 out. 2025.

quanto na qualidade das respostas, mesmo quando aplicada a bases de dados complexas e diversificadas.

Outros estudos aplicados complementam essa base teórica. Souza explora a biblioteca NetworkX na detecção de similaridade estrutural em bancos relacionais, evidenciando como a modelagem em grafos pode ser útil para refinar consultas e visualizar padrões ocultos (SOUZA; GUEDES, 2023). Elmagarmid et al., por sua vez, discutem estratégias clássicas de deduplicação baseadas em atributos, que embora úteis, não conseguem lidar com a complexidade estrutural de tabelas completas (ELMAGARMID; IPEIROTIS; VERYKIOS, 2007).

No campo das ferramentas e plataformas, Fan et al. investigam algoritmos adaptativos para identificação de isomorfismos parciais em grandes grafos dinâmicos, considerando aspectos como estabilidade topológica e variações incrementais nas estruturas (ZHANG *et al.*, 2023). Esses avanços evidenciam o potencial das representações em grafos como uma base sólida para inspeção estrutural automatizada, especialmente em ambientes descentralizados e altamente mutáveis como os propostos pelo *Data Mesh*.

Yazici e Taşkomaz introduzem o *BF-BigGraph*, um método inovador de isomorfismo de subgrafos que combina a estratégia de busca *best-first* com técnicas de aprendizado de máquina para consultas eficientes em bases de grafos de larga escala (YAZICI; TAşKOMAZ, 2024). Utilizando o algoritmo *Random Forest* como classificador supervisionado, o modelo é capaz de prever ordens de correspondência otimizadas e restringir o espaço de busca, melhorando significativamente o tempo de resposta e o uso de memória em bases com bilhões de nós e arestas. Os experimentos demonstram que o *BF-BigGraph* supera abordagens clássicas em consultas complexas. Esses resultados evidenciam o potencial de metodologias baseadas em *machine learning*.

A proposta de Lu et al. introduz uma abordagem baseada em redes neurais genéticas (GNN) para detecção de dados duplicados em ambientes de integração e mineração de dados. O método combina a capacidade de generalização de redes neurais artificiais com a eficiência de algoritmos genéticos para otimizar tanto a topologia quanto os pesos do modelo antes da fase de aplicação. Essa estratégia permite mapear não linearmente as similaridades entre segmentos de registros, superando limitações dos métodos tradicionais. (LU *et al.*, 2016).

Liu et al. propõem o *G-Finder*, um algoritmo que busca encontrar partes semelhantes entre grafos grandes usando estratégias inteligentes de busca e divisão (LIU *et al.*, 2019). Seu diferencial está em conseguir encontrar estruturas parecidas mesmo quando o grafo possui atributos variados e está incompleto, algo comum em ambientes descentralizados.

Wang et al. propõem o *OblivGM*, um sistema para consultas por isomorfismo de subgrafos atribuídos em grafos armazenados na nuvem, com foco na preservação da privacidade (WANG *et al.*, 2022). A solução utiliza criptografia leve para proteger tanto os dados quanto os padrões de acesso durante a execução das consultas, mesmo com predicados de igualdade e intervalo. Essa abordagem é relevante para contextos descentralizados como o *Data Mesh*, onde a confidencialidade estrutural entre domínios é fundamental.

Outro avanço importante no campo de isomorfismo de subgrafos é o *FIRST* (*Fast Interactive Attributed Subgraph Matching*), proposto por Du et al., que introduz um modelo interativo e eficiente para busca de subgrafos atribuídos (DU; CAO, 2017). O algoritmo é especialmente adequado para cenários exploratórios em que o usuário não possui um padrão de busca definido

a priori e precisa ajustar interativamente a consulta com base nos resultados obtidos. O *FIRST* adota estratégias computacionais eficientes para acelerar o processo de correspondência entre subgrafos em redes heterogêneas, mesmo quando há revisões sucessivas na consulta. Essa capacidade de adaptação torna a abordagem especialmente útil em cenários como o *Data Mesh*, onde a governança dinâmica e a identificação de sobreposições semânticas entre domínios com alta diversidade de atributos são essenciais. A inspiração conceitual do *FIRST* motivou a criação do algoritmo *Node Match*, proposto nesta dissertação, que antecipa à etapa tradicional de isomorfismo (como o *VF2*) um processo de pré-filtragem baseado na similaridade entre nós. Essa estratégia reduz significativamente o espaço de busca ao restringir a execução do isomorfismo apenas aos pares mais promissores, promovendo ganhos relevantes de desempenho em arquiteturas de dados descentralizadas.

A revisão da literatura revela uma diversidade significativa de abordagens para identificação de estruturas semelhantes ou duplicadas em conjuntos de dados. No entanto, grande parte desses estudos concentra-se em contextos mais tradicionais, como bancos relacionais centralizados, sistemas transacionais ou grafos homogêneos com topologias estáticas. Embora ofereçam contribuições valiosas, como algoritmos eficientes, heurísticas adaptativas e modelos com foco em privacidade, esses trabalhos não foram concebidos para enfrentar os desafios específicos de arquiteturas modernas descentralizadas, como o *Data Mesh*.

Nesse sentido, ainda são escassas as soluções voltadas à detecção de duplicidades estruturais em ambientes onde os dados são distribuídos entre domínios independentes, com variações sutis em esquemas e alto volume de ativos gerados. Essa lacuna evidencia a necessidade de métodos práticos e adaptáveis, capazes de operar em cenários orientados por domínio e apoiar a governança descentralizada.

Com base nesses referenciais, esta dissertação propõe um método híbrido para identificação de isomorfismos estruturais, combinando algoritmos baseados em grafos com uma etapa de validação manual. A metodologia foi aplicada em ambientes simulados e reais orientados por domínios, utilizando o benchmark TPC-DS como base de testes (NAMBIAR; POESS, 2006). A fim de destacar os diferenciais da abordagem proposta, realizou-se uma análise comparativa com os principais métodos disponíveis na literatura, considerando critérios como escalabilidade, aplicabilidade prática e aderência ao modelo de dados distribuídos do *Data Mesh*. A Tabela 1.1 apresenta esse comparativo, evidenciando as contribuições centrais de cada estudo e os aspectos que diferenciam esta dissertação.

Tabela 1.1: Comparativo entre os principais trabalhos e esta dissertação quanto aos algoritmos utilizados e à arquitetura de dados analisada.

Autor / Trabalho	**Algoritmos de detecção de isomorfismo**							**Arquitetura de dados**	
	VF2	**Node Match**	**G-FINDER**	**OblivGM**	**FIRST**	**BF-BigGraph**	**GNN**	**Convencional**	**Data Mesh**
Cordella et al. (2001)	x							x	
Liu et al. (2019) [G-Finder]			x					x	
Wang et al. (2022) [OblivGM]				x				x	
Du et al. (2017) [FIRST]					x			x	
Yazici e Taşkomaz (2024) [BF-BigGraph]						x		x	
Lu et al. (2024) [GNN - Dup. Detection]							x	x	
Esta Dissertação (2025)	x	x					x	x	x

Legenda: **x** indica presença e implantação na ferramenta

Vale destacar que, na coluna “Convencional”, estão incluídos trabalhos que operam sobre arquiteturas tradicionais, como bancos de dados relacionais centralizados, pipelines monolíticos ou grafos homogêneos com topologia estática. Diferente disso, a proposta desta dissertação é a única voltada especificamente para arquiteturas baseadas em *Data Mesh*, caracterizadas por descentralização, orientação a domínios e alta variabilidade estrutural entre os nós, mas que também pode ser utilizada por arquiteturas convencionais.

1.4 Objetivos

O objetivo geral desta pesquisa é desenvolver um método baseado em grafos para detectar redundâncias estruturais em tabelas distribuídas no contexto de arquiteturas *Data Mesh*, integrando algoritmos de isomorfismo com validação humana e avaliando a eficiência por meio de métricas computacionais.

Os objetivos específicos são:

- Modelar arquiteturas distribuídas como grafos direcionados, representando relações de linhagem entre tabelas de diferentes domínios;
- Aplicar algoritmos de isomorfismo de subgrafos, como *VF2* e *Node Match*, para detectar estruturas redundantes;
- Desenvolver uma ferramenta em Python (*Isomera*) que simula arquiteturas, executa os algoritmos e coleta métricas;
- Incorporar uma etapa de validação humana para confirmar ou refutar as duplicações detectadas automaticamente;
- Avaliar o desempenho dos algoritmos por meio de métricas como tempo de execução (ET), acurácia (ACC) e *Success Frequency* (SF);
- Aplicar o método proposto a cenários baseados em benchmarks (como o TPC-DS) e arquiteturas sintéticas;
- Analisar os resultados obtidos em diferentes níveis de complexidade arquitetural e propor recomendações para ambientes reais.

Por fim, esta dissertação busca responder duas questões centrais: como identificar tabelas duplicadas em arquiteturas distribuídas orientadas por domínio, como o *Data Mesh*; e como mensurar a eficiência de diferentes abordagens na identificação dessas duplicações em ambientes *Data Mesh*.

1.5 Estrutura da Dissertação

Esta dissertação está organizada em seis capítulos. O Capítulo 1 apresenta o contexto da pesquisa no cenário da engenharia de dados contemporânea, destacando a motivação, os objetivos e a relevância do problema abordado, define as perguntas de pesquisa e as contribuições

esperadas. O Capítulo 2 reúne os principais conceitos que sustentam o trabalho, abordando fundamentos de álgebra linear, teoria dos grafos e detecção de isomorfismos, além de discutir os princípios do paradigma *Data Mesh* e sua relação com a governança e modelagem de arquiteturas distribuídas. O Capítulo 3 detalha o processo metodológico desenvolvido para identificação e validação de redundâncias estruturais em arquiteturas de dados, descrevendo as fases de modelagem em grafos, detecção de isomorfismo, validação supervisionada e avaliação das métricas, estabelecendo a base conceitual para a implementação prática.

O Capítulo 4 apresenta a ferramenta computacional *Isomera*, desenvolvida como artefato de apoio à metodologia proposta, descrevendo seus módulos internos, bibliotecas empregadas (`NetworkX`, `DearPyGui`), interface gráfica, pseudocódigos e fluxo de execução, demonstrando como a metodologia foi operacionalizada e aplicada em experimentos reais. O Capítulo 5 expõe os estudos de caso realizados para validar a metodologia e a ferramenta, incluindo experimentos com diferentes algoritmos de detecção (VF2, Node Match e GNN), cujos resultados são analisados de acordo com métricas de desempenho, precisão e eficiência computacional, permitindo comparar abordagens e discutir implicações práticas. Por fim, o Capítulo 6 resume as principais contribuições teóricas e práticas do trabalho, discute suas limitações e apresenta perspectivas de continuidade, como a integração de novos algoritmos, o uso de técnicas de inteligência artificial generativa e a ampliação do escopo da ferramenta para outras arquiteturas de dados, como *Data Warehouse* e *Data Lake*.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo apresenta os fundamentos teóricos e técnicos que sustentam a metodologia proposta para a identificação de redundâncias estruturais em ambientes *Data Mesh*. São discutidos inicialmente os princípios desse paradigma, seguidos pela representação de dados como grafos e pelas definições formais da teoria dos grafos que servem de base para o estudo. Em seguida, são descritos os algoritmos empregados na detecção de isomorfismos estruturais, incluindo abordagens clássicas e o *Node Match*, desenvolvido no âmbito desta pesquisa, além de modelos baseados em aprendizado de máquina. Por fim, são introduzidos conceitos algébricos, como matrizes de adjacência e operações de permutação, que oferecem suporte à análise de duplicidade estrutural.

2.1 Data Mesh

As arquiteturas modernas de dados têm evoluído substancialmente para atender às crescentes demandas por escalabilidade, flexibilidade e integração entre diferentes domínios organizacionais. Inicialmente, os *Data Warehouses* consolidaram-se como soluções voltadas ao armazenamento estruturado e à análise de dados históricos, promovendo consistência e controle centralizado com base em esquemas rígidos e dados integrados (INMON, 2005; KIMBALL; ROSS, 2013). Posteriormente, os *Data Lakes* emergiram como uma alternativa mais flexível às abordagens tradicionais, ao permitirem o armazenamento de dados estruturados, semiestruturados e não estruturados em sua forma original, sem necessidade de pré-processamento. Essa flexibilidade é viabilizada pelo uso do paradigma *schema-on-read*, no qual a estrutura dos dados é interpretada apenas no momento da leitura, conforme a necessidade de cada aplicação ou análise. Diferentemente do modelo *schema-on-write*, adotado em *Data Warehouses*, que impõe um esquema rígido no momento da ingestão, o *schema-on-read* permite maior agilidade na ingestão de dados heterogêneos e dinamismo na análise. Essa característica tornou os *Data Lakes* especialmente atrativos para cenários de ciência de dados e aprendizado de máquina, nos quais os dados frequentemente precisam ser explorados de modo iterativo e adaptável (KHINE; WANG, 2017).

Apesar da consolidação dos *Data Warehouses* e da ascensão dos *Data Lakes* como estratégias centrais para armazenar e analisar grandes volumes de dados, ambas as abordagens enfrentam limitações significativas à medida que a complexidade dos ecossistemas organizacionais cresce. Os *Data Warehouses*, por adotarem modelos de dados fixos previamente definidos (*schema-on-write*), exigem que os dados sejam transformados e ajustados antes de serem armazenados. Isso implica que qualquer alteração no formato ou na estrutura dos dados, como a

introdução de uma nova coluna em uma tabela de vendas ou a inclusão de um tipo de dado semiestruturado, requer modificações no esquema previamente estabelecido, demandando tempo e esforço técnico consideráveis. Além disso, sua arquitetura centralizada dificulta a adaptação rápida às novas necessidades analíticas dos domínios, limitando a escalabilidade e a agilidade em contextos organizacionais dinâmicos (FANG, 2015). Já os *Data Lakes*, embora promovam maior flexibilidade por meio do armazenamento de dados em estado bruto (*schema-on-read*), enfrentam desafios substanciais quanto à arquitetura, governança, gerenciamento de metadados e modelagem. Ainda não há consenso sobre uma arquitetura ideal para *Data Lakes*, tampouco diretrizes consolidadas para integrar aspectos como controle de qualidade, linhagem dos dados, perfis de acesso e interoperabilidade entre ferramentas (GIEBLER *et al.*, 2019). A ausência de uma estratégia de governança clara e bem definida compromete a usabilidade e a confiabilidade desses repositórios, favorecendo o acúmulo desordenado de dados inválidos, incoerentes ou obsoletos, o chamado "pântano de dados" (*data swamp*), em que o valor potencial do *data lake* se deteriora por falta de controle do ciclo de vida, da semântica e da integridade dos dados (DERAKHSHANNIA *et al.*, 2020).

Diante das limitações das arquiteturas centralizadas, como a rigidez estrutural dos *Data Warehouses* e a ausência de controle e qualidade nos *Data Lakes*, surgiram propostas híbridas capazes de integrar o melhor de ambos os mundos. O *Data Lakehouse*, por exemplo, busca combinar a governança, a confiabilidade e o desempenho analítico dos *Warehouses* com a flexibilidade e a escalabilidade dos *Lakes*, possibilitando processamento analítico direto sobre dados brutos, sem comprometer consistência ou performance (HARBY; ZULKERNINE, 2022). Em paralelo, o *Data Fabric* propõe uma arquitetura de integração inteligente baseada em metadados ativos e conhecimento contextual, conectando automaticamente múltiplas fontes de dados heterogêneas, distribuídas e isoladas por meio de um tecido virtual unificado que favorece descoberta, mapeamento e acesso contínuo. Embora avance na virtualização e na automação de pipelines, o *Data Fabric* não endereça por completo os desafios organizacionais de descentralização da governança, autonomia dos domínios e escalabilidade da produção de dados, aspectos críticos em ecossistemas empresariais complexos (BLOHM *et al.*, 2024).

Nesse cenário, o *Data Mesh* desponta como alternativa arquitetural ao propor a descentralização da responsabilidade sobre os dados, delegando aos domínios de negócio o papel de mantê-los como produtos interoperáveis e confiáveis. Nas arquiteturas monolíticas, típicas de *Data Warehouses* e de muitos *Data Lakes*, todo o ciclo de ingestão, transformação, modelagem e disponibilização de dados é concentrado em uma equipe ou plataforma central. Esse arranjo cria um ponto único de controle, mas também de sobrecarga, já que todos os fluxos precisam atravessar a mesma estrutura técnica para chegar aos consumidores. Em contraste, o *Data Mesh* distribui essas responsabilidades entre os próprios domínios que produzem e consomem os dados, permitindo que cada um administre seu ciclo de ponta a ponta. Essa mudança promove escalabilidade organizacional, autonomia operacional e governança federada, em contraposição aos modelos centralizados que frequentemente geram gargalos e dificultam a adaptação às necessidades específicas de cada área (KANAGARLA, 2024).

Conforme sistematizado por Dehghani e discutido na Seção 1.1, o *Data Mesh* apoia-se em quatro pilares fundamentais que visam superar as limitações impostas pelos tradicionais silos organizacionais e permitir que a escalabilidade do uso de dados acompanhe a estrutura

descentralizada das organizações modernas.

O primeiro pilar, propriedade dos dados por domínio, confronta diretamente a lógica das arquiteturas monolíticas. Nessas arquiteturas, é comum encontrar equipes hiperespecializadas de dados posicionadas entre os domínios produtores (como times de produto) e os consumidores (como times de recomendação ou relatórios executivos), sem conhecimento contextual ou autonomia para responder rapidamente às demandas do negócio (DEHGHANI, 2022). Essa fragmentação gera silos organizacionais: à esquerda os produtores, à direita os consumidores e, no centro, uma equipe de dados sobrecarregada que atua como gargalo. A Figura 2.1 ilustra esse desalinhamento estrutural, no qual a concentração de responsabilidades compromete escalabilidade e tempo de resposta. Ao transferir a responsabilidade para os próprios domínios, o *Data Mesh* rompe com essa centralização e garante que cada área de negócio assuma a gestão dos dados que produz.

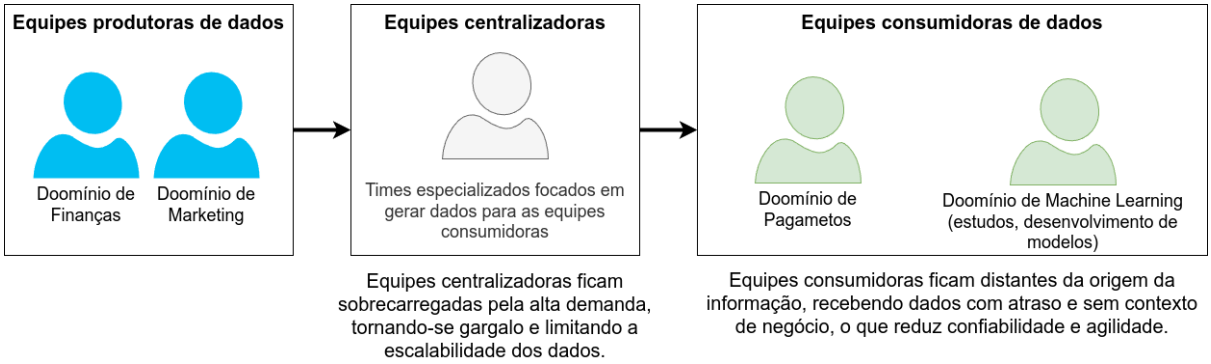


Figura 2.1: Silos em pipelines centralizados: equipe de dados no centro vira gargalo entre produtores e consumidores. Fonte: Dehghani (DEHGHANI, 2022).

O segundo pilar, dado como produto, implica que os domínios de negócio assumam responsabilidade integral pela concepção, qualidade, arquitetura e entrega dos dados que produzem. Esse princípio contrasta fortemente com arquiteturas tradicionais, nas quais as equipes de tecnologia são organizadas em torno de etapas isoladas do ciclo de vida dos dados, ingestão, transformação, modelagem e disponibilização. Como ilustrado na Figura 2.2, esse arranjo gera uma decomposição orientada por atividades, em que cada equipe se responsabiliza apenas por uma fração do processo. Para que um produto de dados seja disponibilizado, é necessário atravessar múltiplas camadas técnicas, envolvendo diferentes grupos altamente especializados.

Essa fragmentação exige coordenação intensa entre equipes distintas, geralmente sem visão completa do objetivo de negócio. Como consequência, o ciclo de valor dos dados torna-se lento e rígido: dependências se acumulam, pequenas alterações exigem longas negociações e o tempo de resposta às demandas organizacionais cresce de forma desproporcional. O acoplamento entre etapas, aliado à ausência de autonomia dos domínios, transforma os pipelines em gargalos permanentes, atrasando a entrega de valor e limitando a escalabilidade em contextos de rápido crescimento.

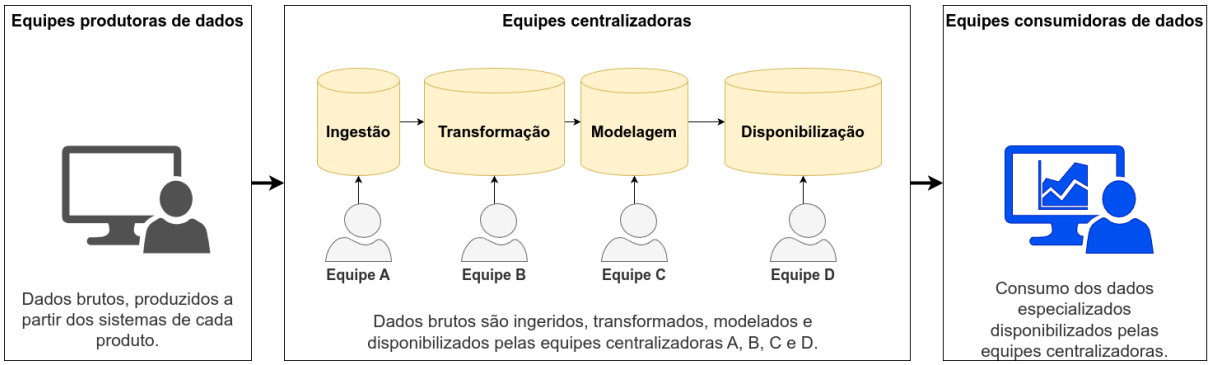


Figura 2.2: Pipeline monolítico orientado por atividades; dependências entre etapas geram atrasos e gargalos. Adaptado de Dehghani (DEHGHANI, 2022).

Em contraponto, o *Data Mesh* propõe uma decomposição organizacional orientada por resultados (*outcomes*), representada na Figura 2.3, correspondente ao pilar de dados como produto. Nesse modelo, cada equipe multidisciplinar passa a ser responsável de ponta a ponta pelos seus próprios produtos de dados, incorporando desde a definição de requisitos de negócio até a entrega para consumo. Na prática, isso significa que o mesmo time que antes limitava-se à sustentação de seus sistemas operacionais, ou que atuava apenas como consumidor de relatórios e dashboards, agora assume a responsabilidade integral pelo pipeline completo: coleta na fonte, transformação e integração, e disponibilização final em camadas especializadas. Essa mudança não é apenas técnica, mas cultural. Exige que os domínios incorporem práticas de qualidade, versionamento e documentação, tratando os dados com o mesmo rigor aplicado a produtos de software. Com isso, o modelo busca alinhar os dados diretamente às necessidades específicas de cada domínio, promovendo maior agilidade, escalabilidade e clareza de responsabilidades na produção e no consumo de dados.

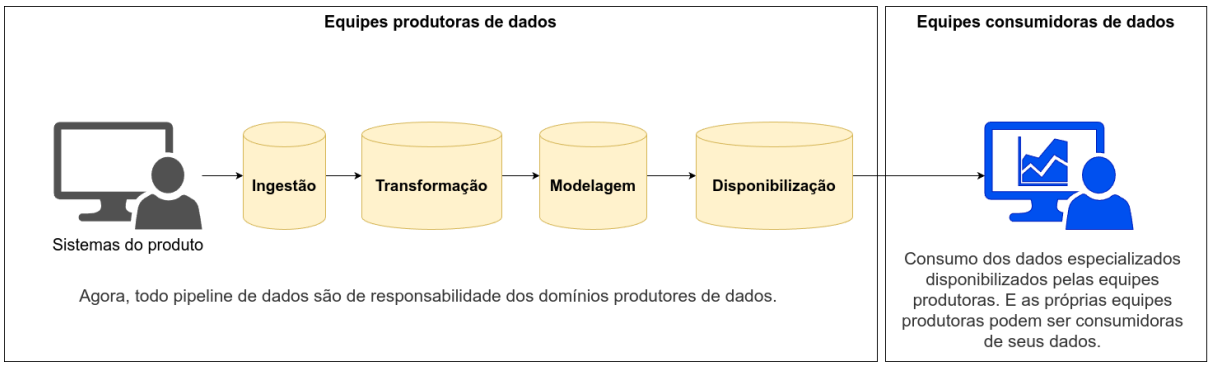


Figura 2.3: Decomposição orientada por resultados no *Data Mesh*: cada domínio opera seu pipeline ponta a ponta com autonomia. Fonte: Dehghani (DEHGHANI, 2022).

O terceiro pilar, denominado plataforma de dados de autosserviço, complementa esse arranjo organizacional e garante que a autonomia dos domínios seja factível. Nesse pilar, a equipe de plataforma (*platform team*) assume a responsabilidade por prover a infraestrutura comum, normalmente baseada em ambientes de nuvem, capaz de padronizar funções essenciais como

armazenamento escalável, processamento distribuído, versionamento de dados, catálogos de metadados, monitoramento e segurança, conforme ilustrado na Figura 2.4. Essa camada de autosserviço funciona como um alicerce compartilhado que reduz a complexidade técnica da descentralização, permitindo que equipes de negócio foquem no valor dos dados e não na manutenção da infraestrutura. Além disso, elimina a dependência de equipes centrais para tarefas básicas, viabilizando que cada domínio projete, publique e mantenha seus fluxos de dados com maior independência. Ao reduzir gargalos históricos de comunicação e coordenação, esse pilar transforma a descentralização em prática sustentável, assegurando que a escalabilidade organizacional em larga escala seja alcançada sem abrir mão de governança e confiabilidade.

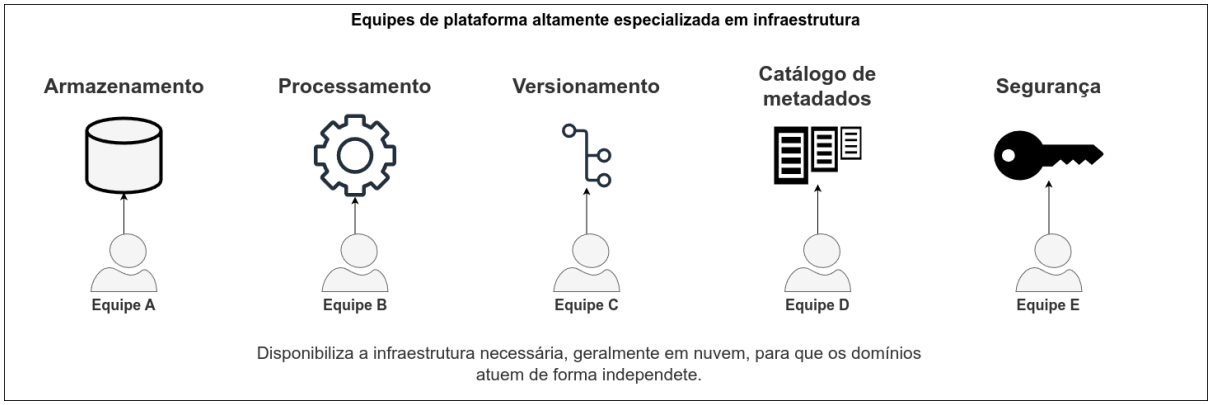


Figura 2.4: Plataforma de autosserviço: infraestrutura comum (armazenamento, processamento, catálogo, segurança) para os domínios. Adaptado de Dehghani (DEHGHANI, 2022).

Na prática, diferentes provedores de nuvem já oferecem recursos que materializam esse pilar. O caso da *Amazon Web Services (AWS)* ilustra bem essa aplicação: como resumido na Tabela 2.1 e ilustrado na Figura 2.5, serviços da AWS oferecem desde armazenamento e processamento até descoberta, governança e aprendizado de máquina, permitindo que os domínios atuem simultaneamente como produtores e consumidores de dados. Em conjunto, esses serviços mitigam parte dos riscos associados à descentralização, ao fornecer uma camada comum que promove interoperabilidade e consistência entre os domínios, sem comprometer sua autonomia[1] .

[1]S. Teles, *Data Mesh: indo além do Data Lake e Data Warehouse*, Blog da comunidade Data Hackers, 2021. Disponível em: <https://medium.com>. Acesso em: 6 out. 2025.

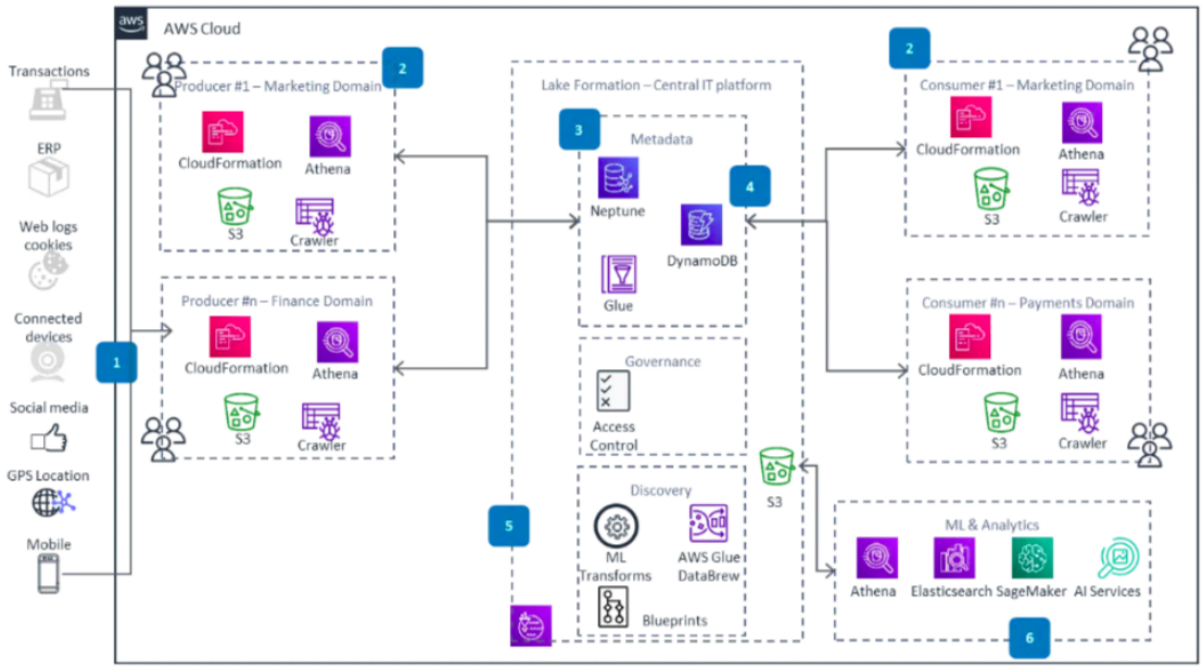


Figura 2.5: Exemplo AWS do pilar de plataforma: Glue, DynamoDB e Lake Formation como autosserviço para domínios. Fonte: Teles

Tabela 2.1: Serviços da AWS aplicados ao pilar de plataforma de autosserviço no *Data Mesh*.

Serviço	Descrição resumida	Contribuição ao *Data Mesh*
Amazon S3	Armazenamento de objetos escalável e distribuído.	Camada base para persistência e compartilhamento de dados entre domínios.
CloudFormation	Infraestrutura como código (IaC).	Automatiza a criação de ambientes padronizados entre domínios.
Athena	Consulta interativa em dados no S3 via SQL.	Facilita o consumo direto de dados sem movimentação entre sistemas.
Crawler	Varredura e catalogação automática de dados.	Detecta esquemas e metadados, promovendo descoberta e integração.
AWS Glue	Serviço de ETL gerenciado.	Orquestra pipelines e integrações entre domínios.
Neptune	Banco de grafos gerenciado.	Representa relações e dependências entre produtos de dados.
DynamoDB	Banco NoSQL de baixa latência.	Suporta aplicações e catálogos distribuídos de dados.
DataBrew	Ferramenta visual de preparação de dados.	Permite limpeza e transformação por equipes de negócio.
Elasticsearch (OpenSearch)	Motor de busca e análise.	Facilita indexação e descoberta de produtos de dados.
SageMaker	Plataforma de aprendizado de máquina.	Permite criação e publicação de modelos de ML como produtos de dados.

O quarto pilar, denominado governança federada computacional, busca responder a um dos maiores desafios do *Data Mesh*: como manter coerência e confiabilidade em um cenário no qual múltiplos domínios atuam de forma autônoma. Em arquiteturas descentralizadas, a liberdade de cada domínio para modelar e operar seus produtos pode levar à fragmentação semântica, à duplicação de métricas e à inconsistência na definição de regras de negócio. Por exemplo, o conceito de "cliente"pode ser calculado de forma distinta por dois domínios, resultando em relatórios divergentes e em decisões estratégicas baseadas em visões incompatíveis. Para mitigar esse risco, como mostrado na Figura 2.6, a governança federada estabelece um mecanismo de coordenação no qual representantes de diferentes domínios definem, de forma colaborativa e automatizada, políticas globais de nomenclatura, qualidade, segurança, acesso e conformidade

regulatória. Assim, cada domínio preserva sua autonomia operacional, mas dentro de um conjunto comum de diretrizes que garantem interoperabilidade e confiança nos dados em escala organizacional.

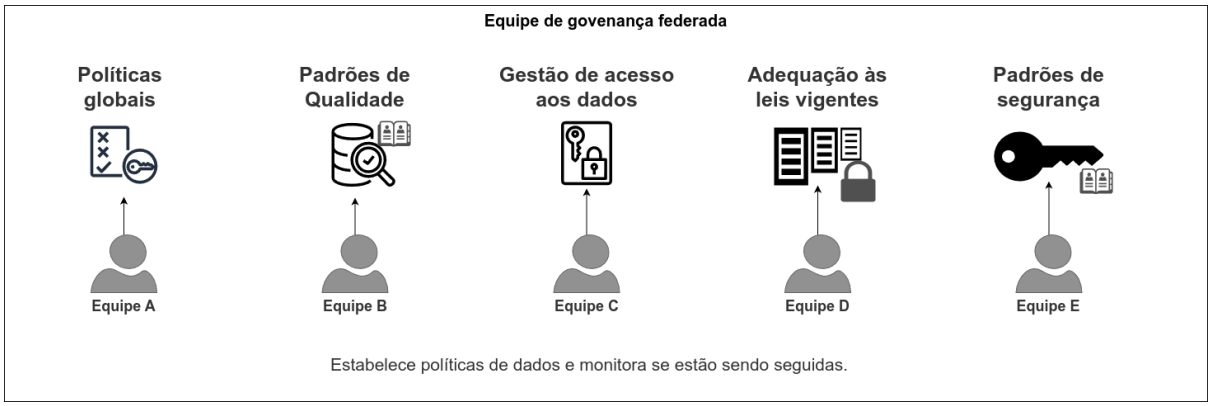


Figura 2.6: Pilar de governança federada no *Data Mesh*: representantes de diferentes domínios colaboram para definir e automatizar políticas globais de dados (qualidade, segurança, nomenclatura e *compliance*), garantindo interoperabilidade sem comprometer a autonomia local. Fonte: Adaptada de Dehghani (DEHGHANI, 2022).

Apesar de sua proposta inovadora e dos avanços proporcionados pelos quatro pilares, a implementação prática do *Data Mesh* ainda enfrenta desafios substanciais, tanto técnicos quanto organizacionais. Entre eles, destacam-se a dificuldade de padronizar processos entre domínios heterogêneos, a necessidade de maturidade tecnológica para garantir autonomia com responsabilidade e o alto custo de manutenção de múltiplos pipelines independentes. Cada domínio passa a definir suas próprias estruturas de dados, regras de negócio e modelos de transformação, o que, embora promova flexibilidade e velocidade, também aumenta o risco de fragmentação semântica, inconsistência de métricas e sobreposição de esforços entre equipes. Em ecossistemas complexos, nos quais dezenas de domínios coexistem, a ausência de mecanismos automáticos de controle e validação interdomínios pode levar à proliferação de tabelas estruturalmente semelhantes, com nomes e propósitos distintos, mas funções equivalentes, um fenômeno conhecido como redundância estrutural.

Esse tipo de redundância, embora muitas vezes imperceptível em estágios iniciais, tende a crescer exponencialmente à medida que novos produtos de dados são criados de forma paralela e autônoma. O resultado é um cenário de governança fragmentada, alto consumo de recursos computacionais e dificuldade de rastreabilidade entre origens e derivadas de dados, comprometendo a confiabilidade das análises e a eficiência operacional do ecossistema como um todo. Assim, o principal problema identificado nesta pesquisa decorre justamente dessa lacuna: a falta de métodos sistemáticos e escaláveis para detectar, analisar e mitigar redundâncias estruturais em arquiteturas distribuídas orientadas a domínios. A seção seguinte (2.1.1) aprofunda essa discussão, apresentando a visão geral da metodologia proposta para identificar tais redundâncias com base na modelagem de tabelas como grafos e na aplicação de algoritmos de isomorfismo estrutural.

A Figura 2.7 apresenta uma visão simplificada do modelo operacional do *Data Mesh*, no qual os domínios de negócio atuam de forma autônoma e responsável na criação de produtos de dados (*data products*), impulsionando aplicações digitais orientadas por dados. Cada domínio conta com equipes multifuncionais, combinando competências de negócio, tecnologia e dados, capazes de projetar, operar e evoluir seus próprios produtos. Esses dados são compartilhados em um ambiente centralizado de plataforma, permitindo sua composição e reutilização por outros domínios. Essa plataforma (*self-serve data platform*) é operada pelo time de tecnologia (*platform team*), que provê serviços técnicos necessários para suportar produção e consumo de dados com autonomia. Complementando essa estrutura, um mecanismo de governança federada, com representantes de múltiplos domínios, define e automatiza políticas globais (nomenclaturas, qualidade, segurança e *compliance*), assegurando padrões coerentes sem abrir mão da descentralização operacional. Esse modelo busca equilibrar liberdade local com coerência organizacional, permitindo escalar a produção e o consumo de dados de forma sustentável.

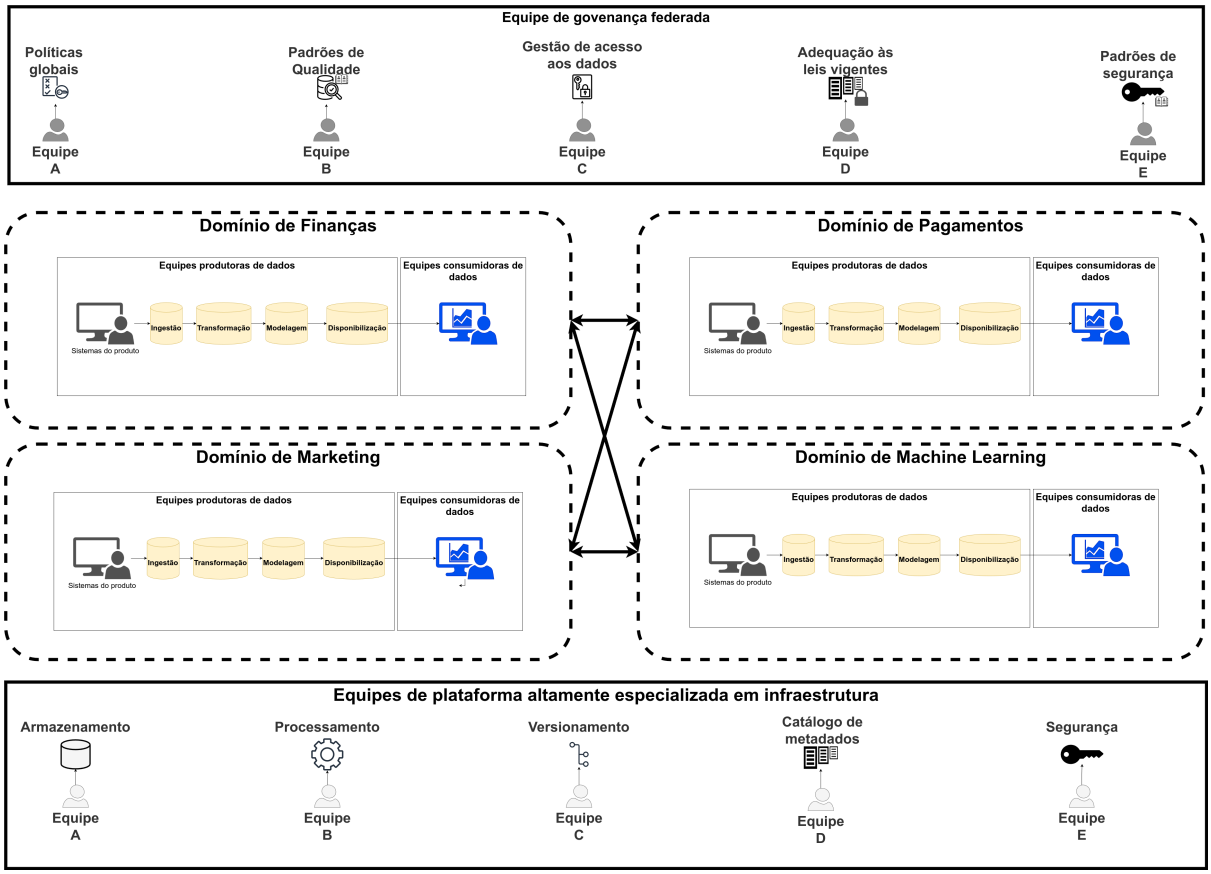


Figura 2.7: Operação simplificada do *Data Mesh*: domínios publicam produtos em plataforma de autosserviço com governança federada. Adaptado de Dehghani (DEHGHANI, 2022).

Assim, o *Data Mesh* não é apenas uma evolução técnica, mas uma reconfiguração sociotécnica que realinha o fluxo de dados com a estrutura de domínio da organização, favorecendo agilidade, qualidade e valor no uso dos dados em larga escala.

2.1.1 Representação Prática do *Data Mesh*

Com base nas figuras anteriores, a Figura 2.8 apresenta uma representação simplificada de uma arquitetura *Data Mesh*, destacando de forma prática como os domínios de negócio se organizam. Cada domínio corresponde a uma área específica da organização, como Marketing, Finanças, Pagamentos e Machine Learning, e assume a responsabilidade integral pelos dados que produz. Esses domínios deixam de atuar de forma restrita como apenas consumidores ou produtores e passam a operar como unidades autônomas, responsáveis por criar e manter seus próprios produtos de dados, além de consumi-los internamente e também acessar dados disponibilizados por ele e por outros domínios.

No exemplo ilustrado, cada domínio possui tabelas próprias, Marketing com A e B, Finanças com C e D, Pagamentos com E e F, e Machine Learning com G e H. Todas essas tabelas são disponibilizadas na infraestrutura em *Data Mesh*, tornando-se acessíveis a outros domínios que delas necessitem. Essa descentralização viabiliza escalabilidade e autonomia, já que cada área administra seu ciclo de dados de ponta a ponta, desde a coleta até a disponibilização para consumo, como mostrado na Figura 2.3. Por outro lado, o modelo também evidencia um risco central desta pesquisa: a possibilidade de redundância estrutural. Diferentes domínios podem criar tabelas semelhantes, com estruturas quase idênticas e finalidades sobrepostas, sem que exista um mecanismo de coordenação explícito. Esse cenário compromete a governança, aumenta os custos de armazenamento e dificulta a rastreabilidade dos dados, problemas que justificam a necessidade de metodologias específicas para detecção e mitigação dessas redundâncias.

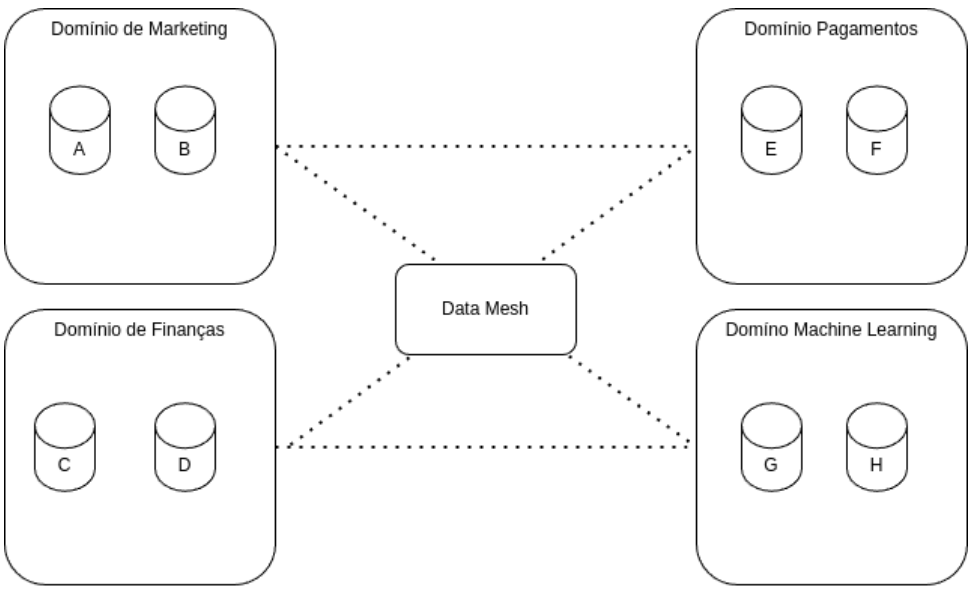


Figura 2.8: Arquitetura *Data Mesh* simplificada com domínios (A–H) e compartilhamento distribuído; autonomia pode gerar redundâncias.

2.1.1.1 Camadas Operacionais em Domínios de *Data Mesh*: SOR, SOT e SPEC

O modelo de dados em um domínio do *Data Mesh* são organizados em três camadas principais que estruturam o fluxo de origem (SOR), transformação (SOT) e consumo de informações (SPEC) [2]. Quando não há padronização ou mecanismos de validação cruzada, essa evolução pode favorecer o surgimento de estruturas redundantes, um problema ainda pouco discutido em soluções práticas de *Data Mesh*. Essa separação permite que cada domínio mantenha clareza sobre o papel e o estágio de maturidade de seus dados, além de favorecer a interoperabilidade com outros domínios da organização.

- SOR (System of Record): São os dados brutos capturados diretamente de sistemas operacionais e transacionais, como ERPs, CRMs, sistemas de pagamento ou plataformas de e-commerce. Por exemplo, o domínio de *Pagamentos* pode possuir um SOR com todas as transações realizadas, contendo atributos como ID da compra, valor, data e método de pagamento. Já o domínio de *Marketing* pode manter um SOR com os registros de campanhas, cliques e leads capturados em plataformas digitais. Esses conjuntos representam a “fonte do sistema” ou "base bruta"e formam a base do pipeline de dados de cada domínio.

- SOT (System of Transformation): Nesta camada, os dados brutos são tratados, limpos e combinados com outras fontes para gerar informações de negócio mais estruturadas. Continuando o exemplo anterior, o domínio de *Pagamentos* pode integrar seu SOR de transações com o SOR de campanhas do domínio de *Marketing*, produzindo um SOT que correlaciona campanhas publicitárias com compras efetivadas, revelando a taxa de conversão por canal. Essa etapa inclui enriquecimentos, validações e junções, sendo essencial para garantir a coerência e a rastreabilidade entre os dados de origem e os resultados intermediários.

- SPEC (Specialized Processing Engines): Representa a camada de consumo especializado, na qual os dados já transformados são utilizados para finalidades analíticas ou operacionais. No mesmo cenário, o domínio de *Machine Learning* pode consumir o SOT gerado por *Pagamentos* e criar uma tabela SPEC usada em modelos de predição de *churn* ou de recomendação de produtos. Da mesma forma, o domínio de *Finanças* pode usar um SPEC derivado para gerar relatórios contábeis e dashboards executivos. Essa camada garante que os dados estejam prontos para uso e expostos como verdadeiros produtos, confiáveis, versionados e documentados.

Seguindo a modelagem da Figura 2.8, a Figura 2.9 exemplifica o funcionamento dessas camadas dentro de um *Data Mesh*. No exemplo visual, os domínios de *Marketing* e *Finanças* geram suas tabelas SOR (B e D), que são consideradas as fontes primárias de dados brutos. O domínio de *Pagamentos*, ao consumir essas informações, cria uma tabela SOT (E) enriquecida, que por sua vez origina uma tabela SPEC (F) voltada a análises avançadas ou uso em APIs de negócio. Esse fluxo evidencia como os domínios atuam simultaneamente como produtores e

[2] A. Hashimoto, SoR, SoT e Spec no contexto de Engenharia de Dados, Alura, 2024. Disponível em: <https://www.alura.com.br/artigos/sor-sot-spec>. Acesso em: 6 out. 2025.

consumidores, formando uma rede colaborativa de dados. Cada domínio mantém sua autonomia técnica e semântica, mas participa ativamente da construção de valor coletivo dentro da organização.

Essa estrutura hierárquica de camadas, ao mesmo tempo simples e poderosa, é o que sustenta a modularidade e a escalabilidade do *Data Mesh*. Contudo, ela também reforça um dos desafios centrais desta pesquisa: quando múltiplos domínios criam suas próprias camadas SOR, SOT e SPEC sem uma visão global consolidada, surgem redundâncias estruturais, diferentes tabelas com esquemas semelhantes e propósitos sobrepostos, que podem comprometer a eficiência e a governança do ecossistema de dados.

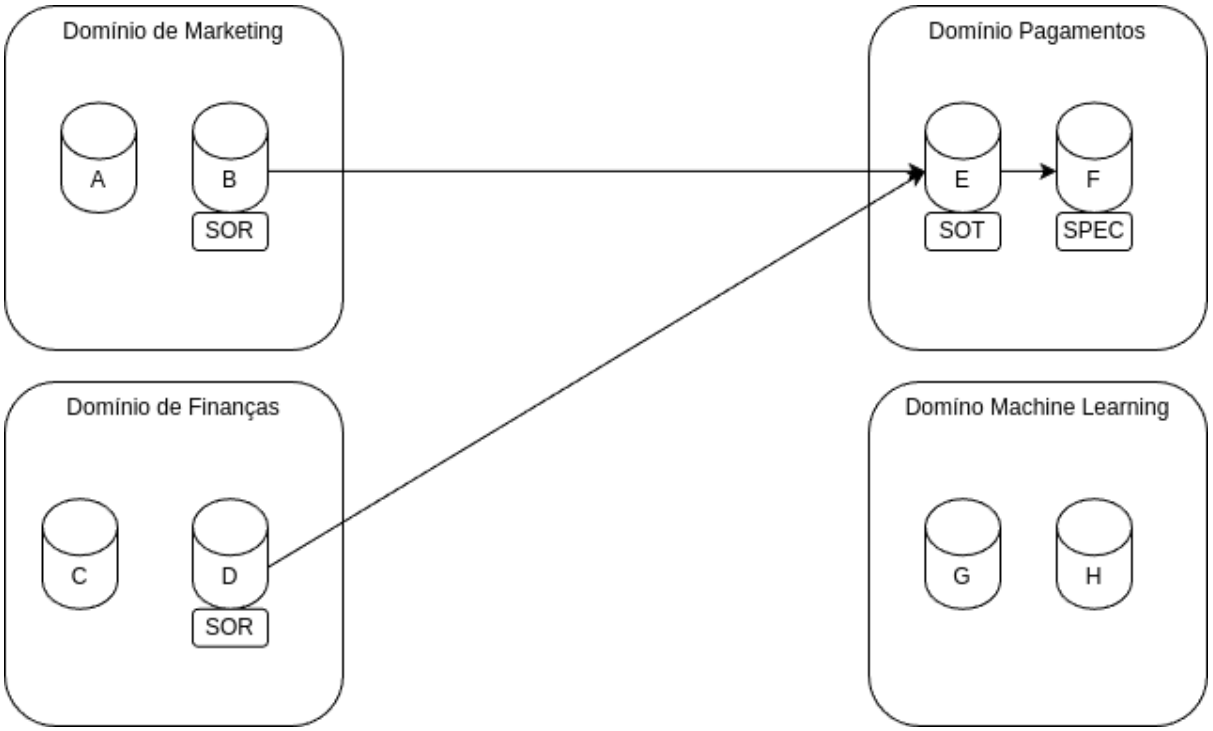


Figura 2.9: Camadas SOR, SOT e SPEC: produção, transformação e consumo distribuídos entre domínios.

2.1.1.2 Redundância Estrutural: Um Problema Emergente

A autonomia conferida pelo *Data Mesh* permite que cada domínio desenvolva seus próprios pipelines de dados de forma independente, respondendo com agilidade às demandas específicas de negócio. Esse princípio, embora essencial para a escalabilidade organizacional, traz consigo um desafio crescente: a ausência de mecanismos de coordenação e visibilidade entre os domínios pode levar à criação de ativos redundantes, especialmente tabelas que compartilham estrutura, semântica e propósito semelhantes, mas são construídas de maneira isolada [3] .

Em um cenário típico, diferentes equipes acessam as mesmas fontes de dados, recriando

[3]S. Teles, *Data Mesh: indo além do Data Lake e Data Warehouse*, Blog da comunidade Data Hackers, 2021. Disponível em: <https://medium.com>. Acesso em: 6 out. 2025.

processos de transformação e modelagem que já existem em outros domínios. Essa duplicação não é fruto de descuido técnico, mas de um efeito colateral natural da descentralização, onde cada domínio, buscando autonomia e velocidade, tende a reproduzir soluções que atendem a problemas locais, sem conhecimento das implementações já existentes em outras áreas. A ausência de catálogos de metadados consolidados, políticas de interoperabilidade ou validações automáticas entre domínios amplifica esse fenômeno, que se torna progressivamente mais difícil de detectar à medida que o ecossistema de dados cresce.

Essa redundância estrutural afeta diretamente a governança e a qualidade das informações, elevando custos de armazenamento e processamento, além de introduzir riscos de inconsistência semântica entre tabelas que deveriam representar o mesmo conceito (JI *et al.*, 2012). Em muitos casos, diferentes domínios partem das mesmas camadas SOR para gerar SOTs e SPECs semelhantes, criando múltiplas versões de um mesmo indicador, como "receita líquida", "número de clientes ativos" ou "taxa de conversão", com lógicas distintas e resultados incongruentes. Essa fragmentação compromete a rastreabilidade (*data lineage*), dificulta auditorias e reduz a confiabilidade dos produtos de dados corporativos.

A Figura 2.10 exemplifica esse problema. Nela, os domínios de Pagamentos e de Machine Learning consomem, de forma independente, os mesmos dados brutos provenientes de Marketing e Finanças para construir suas próprias tabelas intermediárias, denominadas SOT E e SOT G. Embora sigam fluxos técnicos diferentes, ambas possuem estrutura e finalidade semelhantes, representando análises de desempenho transacional. Esse paralelismo revela uma sobreposição de esforços e um risco de divergência analítica entre domínios que, teoricamente, deveriam compartilhar uma visão unificada sobre os mesmos fenômenos de negócio.

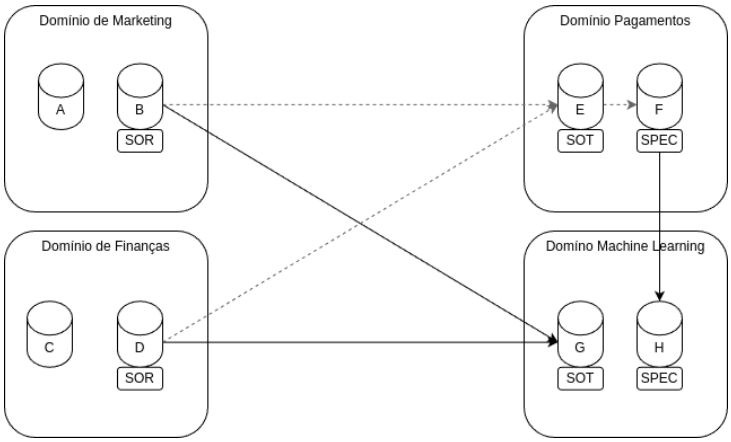


Figura 2.10: Redundância estrutural: domínios distintos criam SOTs semelhantes a partir das mesmas fontes.

A proposta desta dissertação busca enfrentar exatamente esse desafio: identificar e mitigar estruturas redundantes criadas de forma paralela por diferentes domínios dentro de uma arquitetura *Data Mesh*. Para isso, adota-se uma abordagem que combina princípios de governança distribuída com modelagem formal, representando as tabelas como grafos direcionados, como detalhado na Seção 2.3.4. Essa representação permite capturar, de forma abstrata e comparável, os relacionamentos entre colunas, chaves e dependências, viabilizando a aplicação de

algoritmos de isomorfismo de subgrafos para detectar padrões estruturais equivalentes, ainda que nomeados ou organizados de maneiras distintas (PATEL; DHARWA, 2017).

2.2 Fundamentos de Álgebra Linear

A álgebra linear constitui um dos pilares fundamentais para a representação e manipulação de estruturas relacionais e topológicas no âmbito da ciência de dados. Por meio de representações matriciais e transformações vetoriais, torna-se possível modelar tabelas, fluxos de dados e conexões entre elementos em arquiteturas distribuídas, fornecendo uma base matemática sólida para análises estruturais e inferências computacionais (STOLL, 2020). Esta seção apresenta os conceitos algébricos necessários para compreender, posteriormente, a modelagem de dados como grafos e a verificação de equivalências estruturais.

2.2.1 Espaços Vetoriais

Um espaço vetorial é uma estrutura algébrica composta por um conjunto de vetores, sobre o qual estão definidas operações de adição e multiplicação por escalares (GOODFELLOW; BENGIO; COURVILLE, 2016). Em $\mathbb{R}^n$, um vetor pode ser representado como uma n-upla ordenada de números reais:

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \quad v_i \in \mathbb{R}.$$

No contexto computacional, vetores são amplamente utilizados para representar variáveis, atributos, colunas de tabelas ou relações entre objetos. Em uma tabela de dados, por exemplo, cada linha pode ser vista como um vetor que reúne os valores das colunas correspondentes, como identificador, tipo, tamanho ou relação com outras tabelas. Essa representação vetorial permite traduzir estruturas de dados em formas numéricas, manipuláveis por operações matemáticas e algoritmos de aprendizado.

2.2.2 Operações com Vetores

A partir dessa representação, torna-se possível aplicar operações algébricas sobre os vetores que descrevem as entidades de um domínio de dados. Essas operações formam a espinha dorsal de diversas técnicas e algoritmos de análise estrutural, especialmente quando buscamos identificar padrões, semelhanças e redundâncias entre tabelas. Duas operações fundamentais sobre vetores são a adição e a multiplicação por escalar, as quais permitem combinar ou ajustar proporcionalmente diferentes dimensões de informação.

- **Adição de Vetores:** Dado dois vetores $\vec{u}, \vec{v} \in \mathbb{R}^n$, a operação de adição é realizada ele-

mento a elemento, resultando em um novo vetor $\vec{w} \in \mathbb{R}^n$ tal que:

$$\vec{u} + \vec{v} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

Esta operação é comutativa e associativa, permitindo combinações lineares de vetores de maneira consistente.

- **Multiplicação por Escalar:** Seja $\alpha \in \mathbb{R}$ um escalar e $\vec{v} \in \mathbb{R}^n$, a multiplicação escalar resulta em um vetor $\alpha\vec{v} \in \mathbb{R}^n$ cujos elementos são dados por:

$$\alpha\vec{v} = \begin{bmatrix} \alpha v_1 \\ \alpha v_2 \\ \vdots \\ \alpha v_n \end{bmatrix}$$

Essa operação estica ou comprime o vetor original dependendo do valor absoluto de α, e inverte sua direção se $\alpha < 0$.

Essas operações básicas (adição e multiplicação por escalar) fundamentam o conceito de *combinação linear*, no qual vetores podem ser expressos como somas ponderadas de outros vetores (GOODFELLOW; BENGIO; COURVILLE, 2016). Na prática, essa propriedade se manifesta de maneira evidente na multiplicação entre um vetor e uma matriz, tema que será aprofundado na próxima subseção, em que o resultado corresponde a uma combinação linear das colunas da matriz, com os coeficientes determinados pelos elementos do vetor multiplicador (MITRAN, 2023).

2.2.3 Matrizes

Uma matriz é uma extensão natural do conceito de vetor, podendo ser vista como uma estrutura bidimensional que organiza informações em linhas e colunas. Formalmente, uma matriz $A \in \mathbb{R}^{m \times n}$ possui m linhas e n colunas, sendo cada elemento a_{ij} um valor real associado à posição i, j:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

Diferentemente dos vetores, que representam entidades individuais (como colunas de uma tabela ou atributos isolados), as matrizes descrevem conjuntos de relações. No contexto desta

dissertação, elas são utilizadas para codificar estruturas de tabelas e seus relacionamentos, permitindo representar, de forma numérica, como diferentes entidades de um domínio se conectam entre si. Cada célula de uma matriz pode indicar a presença, ausência ou intensidade de uma relação, por exemplo, a existência de uma chave estrangeira entre duas tabelas, uma correspondência entre colunas com tipos de dados semelhantes ou uma dependência funcional entre atributos. Essa capacidade de capturar múltiplas conexões simultaneamente torna as matrizes o ponto de partida para a modelagem em grafos e para a análise de equivalências estruturais.

Um exemplo prático ajuda a compreender esse conceito. Considere o cenário apresentado na Figura 2.10, no qual o domínio de *Pagamentos* constrói uma tabela intermediária SOT E a partir das fontes SOR B (Marketing) e SOR D (Finanças), representados nas Tabelas 2.2 e 2.3, resultando posteriormente em uma tabela final SPEC F.

Tabela 2.2: Tabela SOR_B — Domínio de Marketing

id_produto	**nome_campanha**
1	Verão 2025
2	Inverno 2025
3	Black Friday

Tabela 2.3: Tabela SOR_D — Domínio de Finanças

id_produto	**valor_campanha (R$)**
1	1200
2	800
3	1500
...	...

A tabela SOT_E é formada a partir de uma operação de *join* entre as duas SORs com base no campo *id_produto*, consolidando as informações de nome e valor de cada campanha, como está estruturado na Tabela 2.4. É uma tabela intermediára, conendo apenas um relacionamento, sem sumarizações ou filtros, mas que já pode ser consumida por completo por outros domínios.

Tabela 2.4: Tabela SOT_E — Domínio de Pagamentos

id_produto	**nome_campanha**	**valor_campanha (R$)**
1	Verão 2025	1200
2	Inverno 2025	800
3	Black Friday	1500
...	...	...

Por fim, a tabela SPEC_F é derivada da SOT_E por meio de uma transformação analítica, na qual são calculadas as médias de valor por campanha (tabela 2.5). Essa etapa caracteriza a camada *SPEC*, responsável por consolidar informações agregadas e analíticas que servirão de base para relatórios, dashboards ou modelos preditivos. Nesse caso, a tabela de saída contém apenas as colunas *nome_campanha* e *media_campanha (R$)*, representando o valor médio investido em cada campanha de marketing.

Tabela 2.5: Tabela SPEC_F — Domínio de Pagamentos

nome_campanha	**media_campanha (R$)**
Verão 2025	1150,00
Inverno 2025	950,00
Black Friday	1600,00
...	...

De forma matricial, todo o fluxo descrito, desde a combinação das fontes SOR_B e SOR_D até a formação da tabela analítica SPEC_F, pode ser interpretado como uma sequência de transformações lineares. Cada etapa (*join*, agregação e cálculo de média) corresponde a uma operação que transforma um conjunto de entradas em uma nova estrutura derivada, mantendo a coerência entre as relações de origem e destino.

Nesse contexto, em vez de representar diretamente os valores das tabelas, podemos representar as dependências entre elas. Considerando as quatro tabelas envolvidas no processo (SOR_B, SOR_D, SOT_E e SPEC_F), a relação entre elas pode ser codificada por meio de uma matriz binária, em que o valor 1 indica a existência de uma dependência direta (por exemplo, quando uma tabela é gerada a partir de outra), e o valor 0 indica ausência de relação.

$$M = \begin{bmatrix} & \text{SOR\_B} & \text{SOR\_D} & \text{SOT\_E} & \text{SPEC\_F} \\ \text{SOR\_B} & 0 & 0 & 1 & 0 \\ \text{SOR\_D} & 0 & 0 & 1 & 0 \\ \text{SOT\_E} & 0 & 0 & 0 & 1 \\ \text{SPEC\_F} & 0 & 0 & 0 & 0 \end{bmatrix}$$

Assim, a modelagem matricial dessas relações pode ser expressa como uma *matriz de adjacência*, que chamaremos de M. Essa matriz sintetiza, em formato numérico, as dependências entre as tabelas de um domínio. Cada linha de M representa uma tabela de origem (que fornece dados) e cada coluna representa uma tabela de destino (que consome dados).

Definimos o conjunto de tabelas $T = \{\text{SOR\_B}, \text{SOR\_D}, \text{SOT\_E}, \text{SPEC\_F}\}$ e a matriz de adjacência

$$M : T \times T \to \{0,1\}, \qquad M(i,j) = \begin{cases} 1, & \text{se existe derivação direta de } i \text{ para } j, \\ 0, & \text{caso contrário.} \end{cases}$$

A matriz de adjacência resultante é:

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

A interpretação da matriz M é direta: as entradas M(SOR_B, SOT_E) = 1 e M(SOR_D, SOT_E) = 1 indicam que a tabela SOT_E é derivada simultaneamente das tabelas SOR_B e SOR_D. De forma análoga, M(SOT_E, SPEC_F) = 1 representa que a SPEC_F é criada a partir da SOT_E. As demais posições da matriz mantêm o valor 0, indicando ausência de dependência direta entre as tabelas correspondentes, ou seja, não há fluxo de dados imediato entre essas entidades.

Essa representação em forma de matriz sintetiza o comportamento estrutural do fluxo de dados SOR $\rightarrow$ SOT $\rightarrow$ SPEC, permitindo que operações algébricas sejam aplicadas para identificar padrões, redundâncias e equivalências estruturais. Em outras palavras, essa modelagem matricial constitui a base conceitual desta dissertação: traduzir a lógica relacional entre tabelas de um domínio em uma estrutura formal que possa ser analisada matematicamente e, posteriormente, modelada como um grafo.

2.2.4 Operações com Matrizes

As operações matriciais constituem a base da álgebra linear aplicada e permitem descrever, de forma numérica e estruturada, as dependências entre entidades em um sistema de dados. Cada operação, adição, multiplicação por escalar e multiplicação entre matrizes, carrega propriedades que tornam possível representar transformações, composições e fluxos de informação entre tabelas dentro de um domínio do *Data Mesh*. Formalmente, seja $A = (a_{ij})_{m\times n}$ uma matriz real, $B = (b_{ij})_{m\times n}$ outra matriz da mesma ordem e $k \in \mathbb{R}$ um escalar (BOLDRINI *et al.*, 1986), as principais operações são definidas a seguir.

2.2.4.0.1 Adição e Subtração Duas matrizes de mesma ordem podem ser somadas ou subtraídas elemento a elemento:

$$A + B = (a_{ij} + b_{ij})_{m\times n}, \quad A - B = (a_{ij} - b_{ij})_{m\times n}.$$

Essa operação permite comparar e combinar relações correspondentes, sendo útil para avaliar diferenças entre dependências estruturais de dois domínios ou para integrar fluxos semelhantes de transformação.

2.2.4.0.2 Multiplicação por um Escalar A multiplicação escalar consiste em multiplicar todos os elementos de uma matriz por um número real k:

$$kA = (ka_{ij})_{m\times n}.$$

Essa operação altera proporcionalmente a intensidade das relações representadas na matriz, podendo ser utilizada, por exemplo, para ponderar pesos de influência ou frequência de uso entre tabelas interligadas.

2.2.4.0.3 Multiplicação de Matrizes Dadas duas matrizes $A = (a_{ij})_{m\times n}$ e $B = (b_{jk})_{n\times p}$, o produto $C = AB$ é definido como:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}, \quad \text{para } i = 1,\ldots,m \text{ e } k = 1,\ldots,p.$$

Essa operação combina as relações diretas representadas em A com as de B, revelando conexões indiretas entre entidades, conceito fundamental na análise de fluxos de dados e na modelagem de grafos. A multiplicação matricial é associativa e distributiva, mas não comutativa ($AB \neq BA$), o que é particularmente relevante para representar dependências direcionadas.

2.2.4.0.4 Transposição A transposta de uma matriz $A = (a_{ij})_{m\times n}$ é a matriz $A^T = (a_{ji})_{n\times m}$, obtida pela troca de linhas por colunas:

$$A^T = (a_{ji})_{n\times m}.$$

Essa operação é útil para reorganizar estruturas, inverter a direção de relações ou comparar simetrias entre fluxos de dados.

2.2.4.0.5 Determinante O determinante é um número real associado a uma matriz quadrada que sintetiza, em um único valor, certas propriedades estruturais da matriz. Ele mede o quanto uma transformação linear representada por essa matriz altera o tamanho ou a escala do espaço em que atua, funcionando como um indicador de "preservação" ou "colapso" das relações internas entre variáveis.

Para uma matriz 2×2,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},$$

o determinante é calculado por:

$$\det(A) = a_{11}a_{22} - a_{12}a_{21}.$$

No caso de uma matriz 3×3,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

o determinante é obtido:

$$\det(A) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}.$$

De modo geral, o cálculo do determinante envolve multiplicações diagonais e suas respectivas subtrações, e seu valor indica o grau de independência entre as linhas ou colunas da matriz. Quando $\det(A) \neq 0$, as linhas (ou colunas) são linearmente independentes, e a matriz representa

uma transformação reversível. Por outro lado, se $\det(A) = 0$, ocorre perda de informação, as linhas são combinações lineares umas das outras, e a transformação "achata" o espaço em uma dimensão inferior, tornando impossível recuperar o estado original.

No contexto desta dissertação, o determinante pode ser interpretado como uma medida de preservação estrutural entre tabelas ou camadas de dados. Por exemplo, em uma matriz que representa relações entre tabelas (SOR, SOT e SPEC), um determinante não nulo indicaria que cada conjunto de relações contribui de forma única e independente para o fluxo de dados. Já um determinante igual a zero sugeriria redundância, isto é, que uma ou mais dependências são combinações de outras já existentes, o que, em termos práticos, reflete sobreposição de pipelines ou duplicidade estrutural em um domínio do *Data Mesh*.

2.2.4.0.6 Matriz Inversa Uma matriz quadrada $A \in \mathbb{R}^{n\times n}$ é dita inversível quando o seu determinante é diferente de zero. Nessa condição, existe uma matriz A^{-1}, chamada de inversa de A, que satisfaz:

$$AA^{-1} = A^{-1}A = I_n.$$

A matriz inversa tem o papel de desfazer o efeito da transformação realizada por A. Em outras palavras, se uma matriz representa uma transformação de dados, sua inversa representa o processo de recuperação, ou reconstrução, das informações originais.

2.2.4.0.7 Matriz Identidade A matriz identidade I_n é uma matriz quadrada composta por 1s na diagonal principal e 0s em todas as demais posições:

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}.$$

Ela atua como o elemento neutro da multiplicação matricial, de modo que $AI_n = I_nA = A$ para qualquer matriz A de dimensão compatível. No contexto deste trabalho, o conceito de identidade pode ser interpretado como o estado de equilíbrio de um sistema de dados, no qual nenhuma transformação altera a estrutura original. Essa analogia é útil para compreender operações reversíveis e a preservação de consistência em pipelines de dados, isto é, quando o fluxo de transformação pode ser revertido sem perda de informação.

A relação entre a matriz identidade e a inversa é direta: enquanto I_n representa o ponto de estabilidade, a inversa A^{-1} representa o caminho de volta até ele. Formalmente, para o caso de uma matriz 2×2, a inversa é obtida pela fórmula:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}, \quad \text{onde} \quad \det(A) = a_{11}a_{22} - a_{12}a_{21}.$$

Considere a matriz

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

O primeiro passo é calcular o seu determinante:

$$\det(A) = a_{11}a_{22} - a_{12}a_{21} = 2 \cdot 1 - 1 \cdot 1 = 2 - 1 = 1.$$

Como $\det(A) \neq 0$, A é inversível. Aplicando a fórmula da inversa, temos:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} = \frac{1}{1} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}.$$

Para verificar, multiplicamos A por A^{-1}:

$$AA^{-1} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} (2 \cdot 1 + 1 \cdot (-1)) & (2 \cdot (-1) + 1 \cdot 2) \\ (1 \cdot 1 + 1 \cdot (-1)) & (1 \cdot (-1) + 1 \cdot 2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2.$$

2.2.4.0.8 Cálculo da inversa de uma matriz, regra geral Para uma matriz quadrada $A = (a_{ij})_{n \times n}$, a inversa é dada por

$$A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A),$$

em que $\operatorname{adj}(A)$ é a transposta da matriz dos cofatores. Cada cofator C_{ij} é calculado por

$$C_{ij} = (-1)^{i+j} \det(M_{ij}),$$

sendo M_{ij} a submatriz obtida ao eliminar a linha i e a coluna j de A. Se $\det(A) = 0$, a inversa não existe.

Considere a matriz de adjacência já apresentada na seção 2.2.3

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

O determinante de uma matriz 4×4 pode ser calculado pela expansão de Laplace.

2.2.4.0.9 Expansão de Laplace A expansão de Laplace é um método geral para calcular o determinante de uma matriz de ordem $n \geq 2$. Ela consiste em decompor o determinante em uma soma de produtos entre os elementos de uma linha (ou coluna) e seus respectivos cofatores. O cofator C_{ij} de um elemento a_{ij} é dado por:

$$C_{ij} = (-1)^{i+j} \det(M_{ij}),$$

onde M_{ij} é a submatriz obtida ao eliminar a linha i e a coluna j da matriz original.

Assim, o determinante de uma matriz $A = (a_{ij})_{n \times n}$ pode ser calculado por:

$$\det(A) = \sum_{j=1}^{n} a_{ij} C_{ij},$$

ou seja, expandindo pela linha i, ou de forma equivalente:

$$\det(A) = \sum_{i=1}^{n} a_{ij} C_{ij},$$

ao expandir pela coluna j. Ambas as abordagens produzem o mesmo resultado.

Exemplo. Considere a matriz:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 0 & 4 & 5 \\ 1 & 0 & 6 \end{bmatrix}.$$

O cálculo do determinante pode ser realizado por meio da *expansão de Laplace*, que consiste em desenvolver o determinante em função de uma linha ou coluna escolhida. Expandindo pela primeira linha, a fórmula geral é dada por:

$$\det(A) = \sum_{j=1}^{3} a_{1j} C_{1j}, \quad \text{com} \quad C_{1j} = (-1)^{1+j} \det(M_{1j}),$$

em que M_{1j} representa a submatriz obtida pela eliminação da linha 1 e da coluna j de A, e C_{1j} é o cofator correspondente.

As submatrizes e seus respectivos determinantes são:

$$M_{11} = \begin{bmatrix} 4 & 5 \\ 0 & 6 \end{bmatrix}, \quad \det(M_{11}) = (4 \cdot 6 - 0 \cdot 5) = 24, \quad C_{11} = (+1) \cdot 24 = 24.$$

$$M_{12} = \begin{bmatrix} 0 & 5 \\ 1 & 6 \end{bmatrix}, \quad \det(M_{12}) = (0 \cdot 6 - 1 \cdot 5) = -5, \quad C_{12} = (-1) \cdot (-5) = 5.$$

$$M_{13} = \begin{bmatrix} 0 & 4 \\ 1 & 0 \end{bmatrix}, \quad \det(M_{13}) = (0 \cdot 0 - 1 \cdot 4) = -4, \quad C_{13} = (+1) \cdot (-4) = -4.$$

Substituindo os valores na fórmula da expansão:

$$\det(A) = a_{11} C_{11} + a_{12} C_{12} + a_{13} C_{13},$$

$$\det(A) = 2(24) + 1(5) + 3(-4) = 48 + 5 - 12 = 41.$$

Logo, $\det(A) = 41$.

Esse procedimento ilustra o método da expansão de Laplace, aplicável a matrizes de qualquer dimensão n, em que o determinante é calculado de forma recursiva a partir dos determinantes de submatrizes de ordem inferior.

Retomando a matriz M

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

o objetivo é avaliar sua inversibilidade por meio do determinante.

O determinante de uma matriz 4×4 pode ser calculado pela expansão de Laplace. Expandindo $\det(M)$ pela primeira linha (onde apenas o terceiro elemento é não nulo), tem-se:

$$\det(M) = 0 \cdot C_{11} + 0 \cdot C_{12} + 1 \cdot C_{13} + 0 \cdot C_{14} = C_{13},$$

com $C_{13} = (-1)^{1+3} \det(M_{13}) = (+1) \det(M_{13})$. A submatriz M_{13} é obtida removendo a primeira linha e a terceira coluna:

$$M_{13} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

O determinante de M_{13} é zero, pois a matriz possui pelo menos duas linhas nulas e, portanto, linhas linearmente dependentes:

$$\det(M_{13}) = 0.$$

Logo,

$$\det(M) = \det(M_{13}) = 0.$$

De forma alternativa, como M é estritamente triangular superior (grafo acíclico), trata-se de uma matriz nilpotente ($M^k = 0$ para algum k), o que também implica $\det(M) = 0$.

Em termos simples, "estritamente triangular superior" significa que todos os elementos na diagonal principal e abaixo dela são zero. Isso ocorre quando conseguimos numerar as tabelas (ordenamento topológico) de modo que os dados sempre fluam da linha i para uma coluna j com $j > i$, isto é, sem ciclos. Já "nilpotente" significa que, ao elevar a matriz a uma certa potência k, obtemos a matriz nula: $M^k = 0$. No contexto de matrizes de adjacência, as entradas de M^k contam quantos caminhos de comprimento k existem entre pares de tabelas; se, para um k suficientemente grande, não restam caminhos, então $M^k = 0$, o que caracteriza um grafo acíclico finito.

Uma matriz nilpotente tem todos os autovalores iguais a zero, e como o determinante é o produto dos autovalores, segue $\det(M) = 0$. Isso implica que M não é inversível. No nosso contexto, esse fato tem uma leitura intuitiva: em pipelines acíclicos há agregações e direcionamentos de informação, logo a transformação "uma etapa adiante" não é bijetiva, e não é possível, por uma inversão linear, recuperar de forma única todas as origens a partir dos destinos.

Impacto prático: o determinante nulo sinaliza colapso de volume (perda de graus de liberdade), mas não localiza a redundância. Por isso, para detectar onde a duplicidade acontece, recorremos a ferramentas complementares: (i) potências de M ($M^2, M^3, \ldots$) para revelar dependências indiretas (alcançabilidade multi-etapas), e (ii) testes de equivalência estrutural por permutação, como $B = PAP^\top$, que preservam a estrutura de origens e destinos e permitem comparar padrões de conexões entre tabelas.

Em resumo prático para este trabalho: $\det(M) = 0$ é esperado em fluxos acíclicos e, por si só, não prova duplicidade, pois mesmo sem duplicatas o determinante seria zero devido à triangularidade. A não inversibilidade significa que não conseguimos, por operações lineares, desfazer o fluxo para recuperar de forma única todas as origens a partir dos destinos, logo a

estratégia para achar duplicidades precisa observar padrões estruturais, por exemplo, colunas iguais para duplicidades de destinos (duas SOT recebendo as mesmas entradas) e linhas iguais para duplicidades de fontes (duas SOR apontando para os mesmos destinos).

Como $\det(M) = 0$, a matriz M não é inversível, isto é, não existe M^{-1} tal que $MM^{-1} = I_4$. Portanto, a matriz M não possui inversa.

Agora, se adicionarmos a SOT_G da Figura 2.10, do domínio de *machine learning* na matriz M, teremos a matriz M' dada por

$$M' = \begin{bmatrix} & \text{SOR\_B} & \text{SOR\_D} & \text{SOT\_E} & \text{SOT\_G} & \text{SPEC\_F} \\ \text{SOR\_B} & 0 & 0 & 1 & 1 & 0 \\ \text{SOR\_D} & 0 & 0 & 1 & 1 & 0 \\ \text{SOT\_E} & 0 & 0 & 0 & 0 & 1 \\ \text{SOT\_G} & 0 & 0 & 0 & 0 & 0 \\ \text{SPEC\_F} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Temos a nova matriz de adjacência M' dada por

$$M' = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Neste caso, o $\det(M') = 0$, ou seja, há em M' uma dependência linear global, mas o determinante não indica a localização. Em particular, as colunas 3 e 4 (SOT_E, SOT_G) são idênticas, pois recebem exatamente as mesmas entradas (SOR_B, SOR_D), indicando equivalência estrutural entre SOT_E e SOT_G. Observação: as linhas 1 e 2 (SOR_B, SOR_D) também são idênticas, refletindo destinos iguais.

O que mudou ao passar de 4×4 para 5×5? A matriz continuou representando um grafo acíclico, portanto permaneceu estritamente triangular superior com diagonal nula, por isso o determinante continuou igual a zero, algo esperado. Além disso, ao introduzirmos a SOT_G, criamos deliberadamente uma coluna adicional igual à coluna da SOT_E, tornando visível, nas próprias colunas 3 e 4, a duplicidade de destinos, isto é, duas SOT geradas pelas mesmas fontes. Em termos de diagnóstico, o tamanho maior da matriz não altera o fato de o determinante ser zero, mas fornece um "testemunho" observável da duplicidade, pois colunas idênticas localizam onde está a redundância, e isso pode ser confirmado formalmente por equivalência estrutural via $B = PAP^{\top}$.

2.2.5 Matrizes de Permutação e Interpretação Estrutural

Uma matriz de permutação $P \in \mathbb{R}^{n \times n}$ é uma matriz quadrada obtida a partir da matriz identidade pela troca de linhas (ou colunas). Cada linha e cada coluna contém exatamente um elemento igual a 1, e todos os demais são 0.

Formalmente, se I_n é a matriz identidade de ordem n, então qualquer permutação π dos índices $\{1,2,\ldots,n\}$ gera uma matriz de permutação P tal que:

$$P_{ij} = \begin{cases} 1, & \text{se } j = \pi(i), \\ 0, & \text{caso contrário.} \end{cases}$$

O produto PA reordena as linhas de A, enquanto AP reordena as colunas. Isso ocorre porque a multiplicação à esquerda por P altera a ordem das linhas de A, ao passo que a multiplicação à direita altera a ordem das colunas.

Para ilustrar, considere a matriz

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

e a matriz de permutação

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

que troca a primeira e a segunda linhas (ou colunas) da matriz sobre a qual atua.

2.2.5.0.1 Reordenação de linhas (PA): passo a passo Sejam

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{e} \quad A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

Pela regra padrão de multiplicação matricial, cada elemento do produto é

$$(PA)_{ij} = \sum_{k=1}^{3} p_{ik}\, a_{kj}.$$

Como cada linha de P tem exatamente um único 1 e os demais termos iguais a 0, essa soma reduz-se a selecionar uma única linha de A. Assim, cada linha de PA é uma cópia de alguma linha de A, determinada pela posição do 1 na linha correspondente de P.

Primeira linha de P: $[0\ 1\ 0]$. Para todo j,

$$(PA)_{1j} = 0 \cdot a_{1j} + 1 \cdot a_{2j} + 0 \cdot a_{3j} = a_{2j}.$$

Logo, a 1ª linha de PA é a 2ª linha de A: $[a_{21}\ a_{22}\ a_{23}]$.

Segunda linha de P: $[1\ 0\ 0]$. Para todo j,

$$(PA)_{2j} = 1 \cdot a_{1j} + 0 \cdot a_{2j} + 0 \cdot a_{3j} = a_{1j}.$$

Logo, a 2ª linha de PA é a 1ª linha de A: $[a_{11}\ a_{12}\ a_{13}]$.

Terceira linha de P: $[0\ 0\ 1]$. Para todo j,

$$(PA)_{3j} = 0 \cdot a_{1j} + 0 \cdot a_{2j} + 1 \cdot a_{3j} = a_{3j}.$$

Logo, a 3ª linha de PA é a 3ª linha de A: $[a_{31}\ a_{32}\ a_{33}]$.

Um exemplo pontual de elemento (via produto linha-coluna):

$$(PA)_{12} = \sum_{k=1}^{3} p_{1k}\, a_{k2} = 0 \cdot a_{12} + 1 \cdot a_{22} + 0 \cdot a_{32} = a_{22}.$$

Portanto, o produto final é

$$PA = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

mostrando claramente que P permuta as linhas 1 e 2 de A e mantém a linha 3 inalterada.

2.2.5.0.2 Reordenação de colunas (AP): passo a passo De forma análoga ao caso anterior, a multiplicação de A à direita por P provoca uma *reordenação de colunas*. Cada elemento do produto é obtido por:

$$(AP)_{ij} = \sum_{k=1}^{3} a_{ik}\, p_{kj}.$$

Aqui, cada coluna de AP é formada a partir de uma combinação linear das colunas de A, em que os coeficientes vêm da matriz P. Como cada coluna de P contém exatamente um único 1 e zeros nas demais posições, cada coluna de AP corresponde a uma coluna específica de A, determinada pela posição do 1 em P.

Dadas:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

temos o seguinte raciocínio.

Primeira coluna de P: $[0\ 1\ 0]^T$. Logo,

$$(AP)_{\cdot,1} = 0 \cdot \mathrm{col}_1(A) + 1 \cdot \mathrm{col}_2(A) + 0 \cdot \mathrm{col}_3(A) = \mathrm{col}_2(A).$$

Portanto, a 1ª coluna de AP é a 2ª coluna de A: $[a_{12}, a_{22}, a_{32}]^T$.

Segunda coluna de P: $[1\ 0\ 0]^T$. Assim,

$$(AP)_{\cdot,2} = 1 \cdot \mathrm{col}_1(A) + 0 \cdot \mathrm{col}_2(A) + 0 \cdot \mathrm{col}_3(A) = \mathrm{col}_1(A),$$

isto é, a 2ª coluna de AP é a 1ª coluna de A: $[a_{11}, a_{21}, a_{31}]^T$.

Terceira coluna de P: $[0\ 0\ 1]^T$. Logo,

$$(AP)_{\cdot,3} = 0 \cdot \mathrm{col}_1(A) + 0 \cdot \mathrm{col}_2(A) + 1 \cdot \mathrm{col}_3(A) = \mathrm{col}_3(A),$$

ou seja, a 3ª coluna de AP permanece igual à 3ª de A.

Assim, o produto resultante é:

$$AP = \begin{bmatrix} a_{12} & a_{11} & a_{13} \\ a_{22} & a_{21} & a_{23} \\ a_{32} & a_{31} & a_{33} \end{bmatrix}.$$

Conclui-se que P realiza a troca das colunas 1 e 2 de A, mantendo a coluna 3 inalterada, uma reordenação horizontal que preserva todas as relações internas entre as linhas.

2.2.5.1 Matrizes Isomorfas

Agora, considere a matriz M' da Seção 2.2.4 que expressa as relações entre tabelas de diferentes domínios.

$$M' = \begin{bmatrix} & \text{SOR\_B} & \text{SOR\_D} & \text{SOT\_E} & \text{SOT\_G} & \text{SPEC\_F} \\ \text{SOR\_B} & 0 & 0 & 1 & 1 & 0 \\ \text{SOR\_D} & 0 & 0 & 1 & 1 & 0 \\ \text{SOT\_E} & 0 & 0 & 0 & 0 & 1 \\ \text{SOT\_G} & 0 & 0 & 0 & 0 & 0 \\ \text{SPEC\_F} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Dessa matriz completa, podem ser extraídas duas matrizes de interesse:

$$A = \begin{bmatrix} & \text{SOR\_B} & \text{SOR\_D} & \text{SOT\_E} \\ \text{SOR\_B} & 0 & 0 & 1 \\ \text{SOR\_D} & 0 & 0 & 1 \\ \text{SOT\_E} & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} & \text{SOR\_B} & \text{SOR\_D} & \text{SOT\_G} \\ \text{SOR\_B} & 0 & 0 & 1 \\ \text{SOR\_D} & 0 & 0 & 1 \\ \text{SOT\_G} & 0 & 0 & 0 \end{bmatrix}.$$

As duas matrizes A e B apresentam a mesma estrutura relacional: em ambas, as tabelas SOR_B e SOR_D funcionam como fontes de dados que alimentam uma terceira tabela (SOT_E em A e SOT_G em B), sem que existam dependências adicionais. Isso significa que, do ponto de vista estrutural, SOT_E e SOT_G exercem o mesmo papel dentro do domínio, configurando uma duplicidade funcional.

Essa equivalência pode ser formalmente verificada por meio de uma matriz de permutação P. Se existir P tal que $B = PA$, ou seja, se uma simples reordenação das linhas de A for capaz de gerar B, conclui-se que ambas as matrizes representam a mesma estrutura lógica de dependências. Na prática, isso significa que as origens e relações de geração das SOT permanecem inalteradas, o que indica que SOT_E e SOT_G exercem funções equivalentes, derivando das mesmas fontes (SOR_B e SOR_D), caracterizando assim uma duplicidade estrutural no contexto analisado.

Por exemplo, se existir uma matriz de permutação P tal que:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

então $B = PA$ reordena as linhas 1 e 2 de A, mantendo inalteradas as relações entre as tabelas de origem e destino. Esse resultado mostra que, embora os identificadores ou nomes das tabelas (como SOT_E e SOT_G) possam diferir, a estrutura relacional subjacente é idêntica.

Assim, a existência de uma matriz P que satisfaça $B = PA$ confirma que as duas estruturas são estruturalmente equivalentes, isto é, B pode ser obtida a partir de A apenas por uma reordenação de linhas, sem alterar a lógica de dependência entre as tabelas.

Aplicando $B = PA$, o resultado é:

$$B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Essa estrutura numérica é idêntica à matriz A. Assim, A e B podem ser considerados duplicados, uma vez que as linhas representam as origens de dados e as colunas os destinos. Mesmo com a mudança nos rótulos das tabelas, troca da primeira com a segunda linha, a estrutura de dependência, isto é, as relações de origem e destino, permanece inalterada.

A relação $B = PA$ permite apenas a reordenação das *linhas* de A, isto é, altera as posições das origens das relações, mas mantém fixas as colunas (os destinos). Em outras palavras, PA reorganiza as fontes de dados, mas não ajusta simultaneamente a ordem dos destinos correspondentes, o que pode distorcer a estrutura relacional quando as dependências são simétricas ou bidirecionais.

Para representar de forma completa uma reordenação estrutural coerente, em que tanto as origens (linhas) quanto os destinos (colunas) são permutados de maneira consistente, é necessário incluir a multiplicação também à direita pela transposta de P.

Quando apenas PA é aplicado, a operação reordena exclusivamente as linhas de A, modificando apenas as origens das relações, mas mantendo fixos os destinos. Isso resultaria em uma transformação assimétrica, alterando a interpretação estrutural do sistema.

Por outro lado, a multiplicação pela transposta $P^\top$ à direita garante que as colunas (destinos) sejam reordenadas de forma correspondente às linhas, mantendo a coerência entre as conexões de origem e destino. Dessa forma, cada permutação de linha é acompanhada pela permutação equivalente de coluna, preservando o significado relacional entre os elementos.

Assim, a formulação geral que assegura uma reordenação estrutural completa é dada por:

$$B = PAP^\top.$$

Nessa operação, P reordena as linhas (origens) e $P^\top$ reordena as colunas (destinos) de modo equivalente, garantindo que a estrutura relacional permaneça inalterada, apenas com os rótulos trocados.

Mantemos $B = PA$ como passo pedagógico, por ser uma forma simples de visualizar a reordenação de linhas antes de apresentar a forma completa $B = PAP^\top$, que realiza a permutação coordenada de linhas e colunas.

Em termos práticos, $B = PAP^\top$ assegura que a estrutura relacional subjacente de A seja mantida, apenas com os rótulos reorganizados. Essa é justamente a propriedade necessária para detectar duplicidades estruturais entre tabelas ou domínios, garantindo que duas arquiteturas distintas compartilhem o mesmo padrão de conexões entre suas entidades (TAKAPOUI; BOYD, 2016).

Voltando ao exemplo de A e B, A é

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix},$$

em que as duas primeiras linhas (SOR_B e SOR_D) apontam para a terceira linha (SOT_E).

Definindo a matriz de permutação

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

temos

$$B = PAP^\top.$$

Como $P^\top = P$ para a matriz de troca utilizada e $A' = PA$, então $B = A'P^\top$. A primeira coluna de B é a segunda coluna de A', a segunda coluna de B é a primeira coluna de A' e a terceira coluna de B é a terceira coluna de A'. Neste exemplo, as duas primeiras colunas de A (e de A') são nulas e a terceira coluna é $[1,1,0]^T$; logo, a permutação de colunas não altera o resultado final.

Calculando, obtém-se:

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Nesse caso, o resultado B é estruturalmente idêntico a A. O que ocorreu foi apenas a troca dos rótulos das entidades SOR_B e SOR_D (linhas e colunas 1 e 2), mas a estrutura de dependência foi preservada: ambas continuam alimentando a mesma tabela destino (SOT_E).

Esse exemplo mostra que, quando existe uma matriz P tal que $B = PAP^\top$, podemos afirmar que A e B são estruturalmente equivalentes. Na prática, isso significa que SOT_E e uma eventual SOT_G gerada da mesma forma representam uma duplicidade na arquitetura, pois derivam das mesmas fontes (SOR_B e SOR_D), apenas com rótulos diferentes.

O objetivo da metodologia proposta nesta dissertação é justamente identificar, de forma sistemática, uma matriz de permutação P que satisfaça a igualdade $B = PAP^\top$. Quando tal matriz existe, conclui-se que as estruturas A e B são isomorfas, isto é, equivalentes em termos de suas

origens e destinos, indicando que há uma duplicidade estrutural no *Data Mesh*. Essa verificação permite detectar, de forma algébrica, redundâncias entre domínios distintos, garantindo consistência e otimização na modelagem distribuída de dados.

2.3 Teoria dos Grafos

A teoria dos grafos é um ramo consolidado da matemática discreta que fornece as ferramentas necessárias para representar, modelar e analisar relações entre objetos. Em ciência de dados e, especialmente, em arquiteturas distribuídas como *Data Mesh* como visto na seção 1.1, a modelagem por grafos permite expressar relações entre tabelas, transformações e dependências de forma abstrata, formal e computacionalmente tratável (PATEL; DHARWA, 2017).

2.3.1 Definições e Notação

Um grafo direcionado é um par ordenado

$$G = (V, E),$$

em que V é um conjunto finito e não vazio de vértices (ou nós) e $E \subseteq V \times V$ é um conjunto de arestas direcionadas. Quando necessário, admitimos um *multigrafo direcionado rotulado e ponderado*, denotado por

$$G = (V, E, \ell, w),$$

No modelo geral, pode-se considerar um multigrafo dirigido rotulado e ponderado, $G = (V, E, \ell, w)$, em que $\ell : V \cup E \to \mathscr{L}$ anota nós/arestas e $w : E \to \mathbb{R}_{>0}$ atribui pesos. Nesta dissertação, entretanto, adotamos a forma mínima: grafos dirigidos não rotulados e não ponderados, isto é, trabalhamos com $G = (V, E)$ sem as funções ℓ e w. Os vértices representam tabelas (SOR/SOT/SPEC) e cada aresta indica unicamente a existência de uma dependência/transformação entre duas tabelas, sem tipificação semântica adicional nem peso associado.

Em contextos de modelagem de dados, é possível representar tabelas como vértices e suas inter-relações, como chaves estrangeiras ou dependências de transformação, por meio de arestas direcionadas (ROY-HUBARA *et al.*, 2017). Nesta dissertação, essa representação é formalizada ao modelar SOR, SOT e SPEC como subconjuntos disjuntos de V, isto é, uma partição parcial do conjunto de vértices, enquanto E representa o conjunto de dependências entre essas tabelas. Não assumimos laços (v, v) por padrão, tampouco dependências "de uma tabela em si mesma".

Antes de avançarmos para a seção de classes de grafos, retomamos o exemplo M apresentado anteriormente e explicitamos que a Figura 2.10 pode ser representada por um grafo da forma

$$\begin{aligned} &G = (V, E), \\ &V = \{\text{SOR\_B}, \text{SOR\_D}, \text{SOT\_E}, \text{SPEC\_F}\}, \\ &E = \{(\text{SOR\_B}, \text{SOT\_E}), (\text{SOR\_D}, \text{SOT\_E}), (\text{SOT\_E}, \text{SPEC\_F})\}. \end{aligned}$$

Nesse grafo, o conjunto V representa as tabelas envolvidas no fluxo de dados: as tabelas SOR_B e SOR_D são as fontes de dados (origens), a tabela SOT_E corresponde à camada intermediária que realiza a transformação, e a tabela SPEC_F representa a camada analítica de consumo. Já o conjunto E descreve as dependências entre essas tabelas. Por exemplo, a aresta $(\text{SOR\_B}, \text{SOT\_E})$ indica que existe uma relação direta de derivação entre a tabela SOR_B e a SOT_E, isto é, SOT_E é gerada a partir dos dados de SOR_B. O mesmo ocorre para $(\text{SOR\_D}, \text{SOT\_E})$, enquanto a aresta $(\text{SOT\_E}, \text{SPEC\_F})$ representa a transformação final que produz a tabela analítica. No caso do exemplo M', tem-se um novo grafo, denotado por G', que inclui a tabela SOT_G e, consequentemente, adiciona as arestas $(\text{SOR\_B}, \text{SOT\_G})$ e $(\text{SOR\_D}, \text{SOT\_G})$.

2.3.2 Classes de Grafos

A diversidade de aplicações dos grafos em engenharia de dados torna essencial compreender suas diferentes classificações, uma vez que cada tipo possui propriedades estruturais que impactam diretamente a análise, a modelagem e a execução de algoritmos. No escopo desta dissertação, que lida com a representação de arquiteturas de dados e a identificação de redundâncias estruturais, algumas classes de grafos se destacam de forma especial, como mostrado na Figura 2.11 (FILHO, 2017).

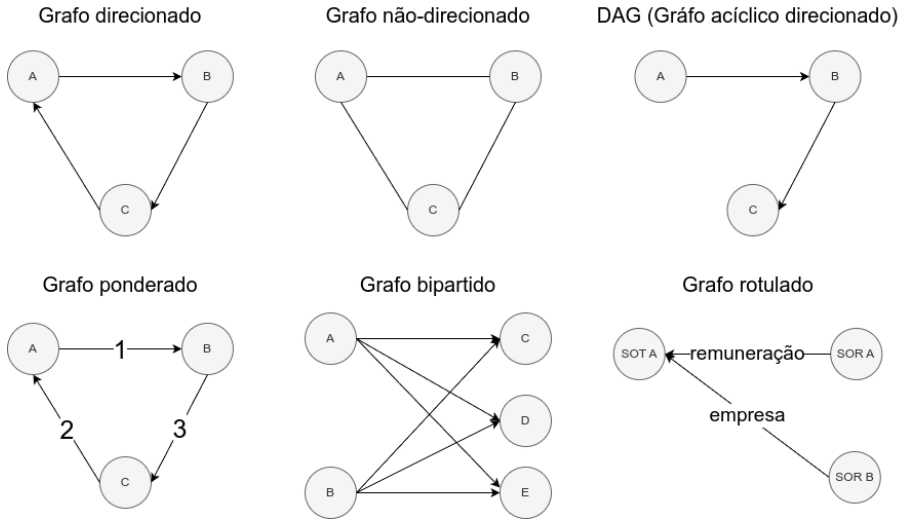


Figura 2.11: Representações visuais dos principais tipos de grafos utilizados em ciência de dados. Fonte: (FILHO, 2017)

- **Grafo Direcionado:** Um grafo orientado pode ser definido como $G = (V, E)$, com $E \subseteq V \times V$, onde cada aresta possui uma direção (FEOFILOFF; KOHAYAKAWA; WAKABAYASHI, 2011). É amplamente utilizado para representar dependências unilaterais, como a derivação de dados entre tabelas em pipelines.

- **Grafo Não Direcionado:** Definido como $G = (V, E)$, com $E \subseteq \{\{u, v\} \mid u, v \in V\}$, ou

seja, cada aresta representa uma relação simétrica. É comum em redes sociais ou conexões bidirecionais (STEEN, 2010).

- **Grafo Direcionado Acíclico (DAG):** Um DAG é um grafo orientado $G = (V, E)$ tal que não contém ciclos, ou seja, não existe qualquer sequência de vértices $v_1, v_2, \ldots, v_k$ com $k > 1$ tal que $(v_i, v_{i+1}) \in E$ para todo i e $v_1 = v_k$. Essa ausência de ciclos assegura a existência de uma ordenação topológica dos vértices, permitindo que tarefas ou transformações associadas a cada nó sejam executadas em sequência lógica sem ambiguidade ou retroalimentação. Conforme demonstrado em (FEOFILOFF, 2020), essa propriedade torna os DAGs fundamentais na análise de dependências e no agendamento de tarefas em sistemas computacionais. Em arquiteturas analíticas modernas, como pipelines de dados e estruturas de *data lineage*, os DAGs são amplamente utilizados para garantir rastreabilidade, consistência e controle de fluxo ao longo do ciclo de vida da informação. Em nosso cenário, não existe uma tabela sendo criada por ela mesma, ou seja, não tem uma SOR, SOT ou SPEC em que a origem são elas mesmas.

- **Grafo Ponderado:** Um grafo ponderado é definido como $G = (V, E, w)$, onde $w : E \to \mathbb{R}^+$ associa a cada aresta um peso positivo. Em contextos distribuídos, um grafo direcionado é dito *balanceado* quando, para todo vértice $v_i \in V$, a soma dos pesos das arestas que chegam em v_i é igual à soma dos pesos das arestas que saem de v_i. Grafos ponderados são amplamente empregados em aplicações como roteamento de dados, balanceamento de carga e sincronização de tarefas em redes de agentes autônomos (RIKOS; HADJICOSTIS, 2018).

- **Grafo Bipartido:** Um grafo bipartido é formalmente definido como $G = (U \cup W, E)$, onde U e W são conjuntos disjuntos de vértices e $E \subseteq U \times W$, ou seja, as arestas ocorrem exclusivamente entre elementos de U e W, sem conexões internas dentro de cada conjunto. Essa estrutura é especialmente eficaz para modelar interações entre dois tipos distintos de entidades. Um exemplo notável de aplicação dessa estrutura está no trabalho de (SOSA; URREGO-LOPEZ; PRIETO, 2024), que analisaram uma rede bipartida composta por usuários e séries de anime. Utilizando técnicas de análise textual e modelos exponenciais aleatórios de grafos (ERGMs), os autores demonstraram que a frequência de certos termos nas descrições das séries influencia diretamente a formação de comunidades entre os usuários. O estudo mostra que descrições com temas como aventura, música ou vida estudantil tendem a promover maior conectividade na rede, enquanto tópicos como ficção científica ou guerra apresentam efeito oposto. Essa abordagem fornece insights relevantes para aprimorar sistemas de recomendação e estratégias de engajamento em plataformas de entretenimento digital.

- **Grafo Rotulado:** Um grafo rotulado é formalmente representado como $G = (V, E, \ell)$, onde V é o conjunto de vértices, $E \subseteq V \times V$ é o conjunto de arestas, e $\ell : V \cup E \to L$ é uma função que atribui rótulos semânticos a vértices e/ou arestas, sendo L o conjunto de possíveis rótulos. A rotulação permite codificar informações contextuais e semânticas sobre os elementos do grafo, favorecendo análises mais precisas e inferências automatizadas.

No contexto de arquiteturas analíticas e *data mesh*, os rótulos nas arestas podem descrever relações semânticas específicas entre tabelas e transformações, como, por exemplo, "remuneração" ou "empresa". Suponha-se que a tabela `SOT_A` seja derivada das tabelas `SOR_A` e `SOR_B`. Ao rotular a aresta `SOR_A → SOT_A` com "remuneração" e `SOR_B → SOT_A` com "empresa", torna-se possível, por meio de mecanismos de busca semântica, recuperar rapidamente todas as origens envolvidas na geração da informação presente em `SOT_A`, a partir de consultas por metadados que envolvam esses conceitos. Essa abordagem é coerente com as diretrizes discutidas por (HOSEINI; THEISSEN-LIPP; QUIX, 2024), que destacam a importância da vinculação de metadados a grafos de conhecimento (*Knowledge Graphs*).

A correta escolha da estrutura gráfica a ser utilizada não apenas influencia a expressividade da modelagem, mas também impacta diretamente a complexidade dos algoritmos aplicáveis e a viabilidade computacional de sua execução. No contexto do *Data Mesh*, a flexibilidade na escolha do tipo de grafo é ainda mais crítica, dado que os domínios são autônomos, heterogêneos e dinâmicos. Por essa razão, esta dissertação prioriza o uso de grafos direcionados, por serem os mais aderentes à realidade dos pipelines distribuídos que se busca modelar e otimizar.

Como ressalta Steen em sua introdução à teoria dos grafos, essa distinção é fundamental para a correta modelagem e análise de redes complexas (STEEN, 2010). Retomando o exemplo apresentado anteriormente, o grafo resultante da matriz M' pode ser formalmente classificado da seguinte forma:

$$
\begin{aligned}
&G' = (V', E'),\\
&V' = \{\text{SOR\_B}, \text{SOR\_D}, \text{SOT\_E}, \text{SOT\_G}, \text{SPEC\_F}\},\\
&E' = \{\, (\text{SOR\_B}, \text{SOT\_E}),\ (\text{SOR\_D}, \text{SOT\_E}),\\
&\qquad (\text{SOR\_B}, \text{SOT\_G}),\ (\text{SOR\_D}, \text{SOT\_G}),\ (\text{SOT\_E}, \text{SPEC\_F})\,\}.
\end{aligned}
$$

Matematicamente, o grafo G' é um grafo direcionado acíclico (DAG), pois cada aresta $(u,v) \in E'$ representa uma dependência orientada entre duas tabelas sem formação de ciclos. Adicionalmente, trata-se de um grafo não ponderado e não rotulado, já que não há pesos (w) associados às arestas nem funções de rótulo (ℓ) aplicadas a vértices ou conexões. Por fim, é possível observar que G' possui múltiplas arestas distintas saindo das mesmas origens (SOR_B, SOR_D), o que caracteriza um grafo com fan-out múltiplo, em que uma mesma tabela de origem alimenta mais de uma transformação (neste caso, SOT_E e SOT_G).

Essa configuração reforça a duplicidade estrutural analisada na Seção 2.2.5.1, evidenciando como a representação gráfica permite identificar redundâncias formais em arquiteturas de dados distribuídas.

2.3.3 Representação Computacional de Grafos

Considere o grafo $G = (V, E)$, com vértices $V = \{v_1, v_2, v_3, v_4\}$ e arestas: $E = \{e_1 = (v_1, v_1), e_2 = (v_1, v_2), e_3 = (v_1, v_3), e_4 = (v_3, v_4), e_5 = (v_2, v_3), e_6 = (v_2, v_3), e_7 = (v_4, v_4)\}$ como mostrado na Figura 2.12. Este grafo é direcionado, pois todas as arestas têm uma direção bem definida entre vértices.

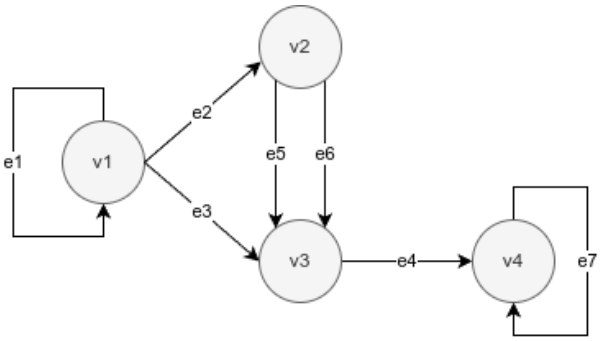


Figura 2.12: Representação do grafo $G = (V, E)$ com múltiplas arestas e laços.

As representações computacionais clássicas para esse grafo são as seguintes:

- **Lista de Adjacência:** Cada vértice é associado a uma lista com os vértices que podem ser alcançados a partir dele por meio de uma aresta direcionada (ou seja, os destinos das arestas de saída):
  - $v_1 \rightarrow [v_1, v_2, v_3]$
  - $v_2 \rightarrow [v_3, v_3]$
  - $v_3 \rightarrow [v_4]$
  - $v_4 \rightarrow [v_4]$

  Essa estrutura é eficiente em termos de espaço, especialmente em grafos esparsos, ou seja, grafos nos quais o número de arestas é significativamente menor que o número máximo possível de conexões entre os vértices.

- **Matriz de Incidência:** A matriz $M \in \mathbb{N}^{4\times 7}$, com linhas representando vértices e colunas representando arestas, é dada por:

$$M = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix}$$

  O valor 2 indica um laço (como em e_1 e e_7). Os valores 1 indicam incidência simples: o vértice está conectado como origem ou destino daquela aresta.

- **Matriz de Adjacência:** Representa as conexões entre os vértices de forma matricial:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nesta notação, A_{ij} indica a quantidade de arestas direcionadas do vértice v_i para o vértice v_j, refletindo explicitamente a orientação das conexões no grafo. Essa forma matricial é particularmente vantajosa para a aplicação de operações algébricas, além de facilitar a análise de propriedades estruturais. Por sua expressividade e aplicabilidade, adotaremos essa representação ao longo desta dissertação.

Voltando ao nosso exemplo M', apresentado anteriormente, a estrutura de dependências entre tabelas pode ser expressa por sua matriz de adjacência, denotada por $A(M')$:

$$A(M') = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Formalmente, $A(M')$ corresponde à representação computacional do grafo $G' = (V', E')$, em que cada linha representa uma tabela de origem (produtora de dados) e cada coluna, uma tabela de destino (consumidora de dados). O valor $A(M')_{ij} = 1$ indica a existência de uma aresta direcionada de v_i para v_j, ou seja, uma dependência direta entre as tabelas correspondentes.

Dessa forma, $A(M')$ sintetiza, em formato matricial, o mesmo conjunto de relações representado graficamente, permitindo que operações algébricas e transformações lineares sejam aplicadas para identificar padrões, redundâncias e equivalências estruturais em arquiteturas de dados. Essa correspondência entre grafo e matriz reforça o caráter direcionado e acíclico da estrutura analisada, servindo como base formal para as análises desenvolvidas nas seções seguintes.

2.3.4 Subgrafos e Equivalência Estrutural

Em teoria dos grafos, o conceito de subgrafo é fundamental para a análise modular de sistemas complexos. Um subgrafo consiste em uma parte de um grafo maior, preservando apenas um subconjunto de seus vértices e as arestas que conectam esses vértices entre si. Formalmente, um subgrafo $G' = (V', E')$ de um grafo $G = (V, E)$ é definido por:

$$V' \subseteq V, \quad E' \subseteq E \cap (V' \times V').$$

Ou seja, V' contém apenas alguns vértices de V, e E' contém todas as arestas que, no grafo original, conectam pares de vértices pertencentes a V'. Essa definição garante que o subgrafo mantenha a coerência estrutural do grafo principal, representando um fragmento funcionalmente consistente de sua topologia.

A identificação e comparação de subgrafos são operações essenciais em tarefas de análise estrutural, pois permitem reconhecer módulos, dependências locais e possíveis redundâncias em arquiteturas complexas. (MESSMER; BUNKE, 2000) propõem, por exemplo, uma técnica baseada em decomposição hierárquica para detecção eficiente de isomorfismos de subgrafos, aplicável em cenários de reconhecimento estrutural e otimização de grafos de grande escala.

Para ilustrar, considere:

$$G = (V,E), \quad V = \{v_1, v_2, v_3, v_4\}, \quad E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}.$$

Um subgrafo possível é:

$$G' = (V',E'), \quad V' = \{v_2, v_3\}, \quad E' = \{(v_2, v_3)\}.$$

Esse subgrafo representa uma etapa intermediária dentro de um fluxo maior de transformações, mantendo apenas a parte relevante da estrutura global, um conceito análogo à extração de dependências parciais em um pipeline de dados.

A Figura 2.13 ilustra como pipelines compostos por tabelas (SOR, SOT, SPEC) podem ser representados como vértices de um grafo, em que as transformações entre eles são modeladas como arestas direcionadas. Essa representação torna explícita a linhagem dos dados e as relações estruturais entre os componentes da arquitetura.

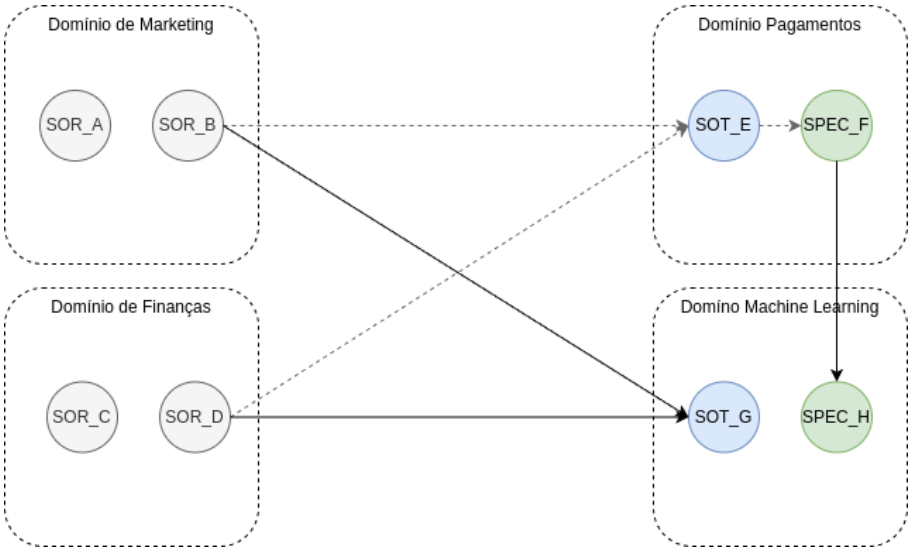
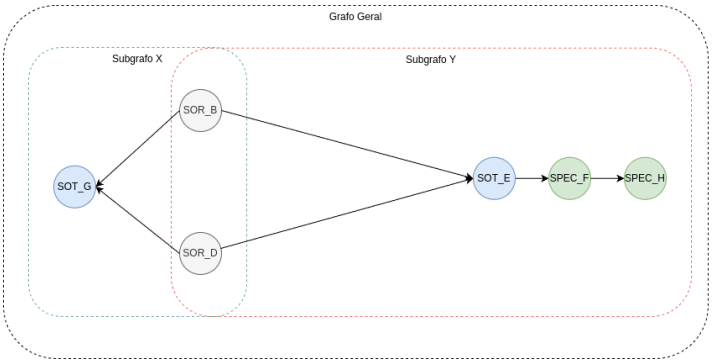


Figura 2.13: Conversão de uma arquitetura de tabelas para grafo

A Figura 2.14 apresenta a construção resultante do grafo geral, evidenciando subgrafos embutidos (como X e Y) e destacando a existência de caminhos equivalentes. Essa estrutura grafo-orientada facilita tanto a visualização quanto a análise algébrica de possíveis duplicidades na arquitetura.

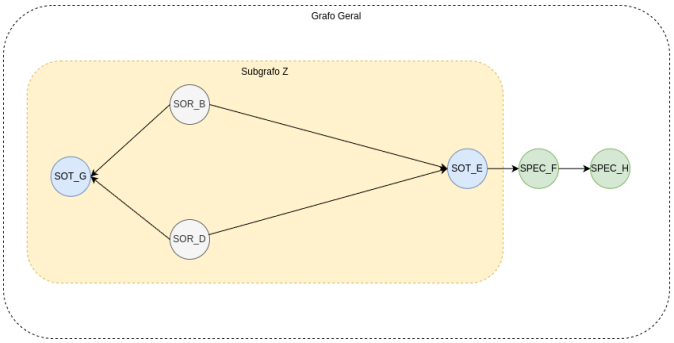


Matriz de Adjacência do Grafo Geral

	SOR_B	SOR_D	SOT_G	SOT_E	SPEC_F	SPEC_H
SOR_B	0	0	1	1	0	0
SOR_D	0	0	1	1	0	0
SOT_G	0	0	0	0	0	0
SOT_E	0	0	0	0	1	0
SPEC_F	0	0	0	0	0	1
SPEC_H	0	0	0	0	0	0

Figura 2.14: Grafo resultante da modelagem das tabelas e subgrafos internos

Por fim, a Figura 2.15 apresenta a matriz de adjacência correspondente ao grafo modelado. Essa matriz é a base para a comparação entre subestruturas por meio de operações algébricas, como permutações, que permitem identificar equivalências formais entre partes distintas da arquitetura.



Matriz de Adjacência do Grafo Geral

	SOR_B	SOR_D	SOT_G	SOT_E	SPEC_F	SPEC_H
SOR_B	0	0	1	1	0	0
SOR_D	0	0	1	1	0	0
SOT_G	0	0	0	0	0	0
SOT_E	0	0	0	0	1	0
SPEC_F	0	0	0	0	0	1
SPEC_H	0	0	0	0	0	0

Figura 2.15: Matriz de Adjacência associada ao grafo da arquitetura

Importância para a Detecção de Redundâncias

A matriz de adjacência constitui o elo formal entre a representação gráfica e a análise algébrica das estruturas de dados. Por meio dela, é possível identificar equivalências estruturais entre diferentes partes de um grafo, independentemente da ordem dos vértices ou da nomenclatura adotada em cada domínio. Na prática, essa análise se traduz na detecção de *subgrafos isomorfos*, ou seja, fragmentos da arquitetura que compartilham a mesma estrutura de dependências entre tabelas.

Retomando o exemplo do grafo G' e de sua matriz de adjacência $A(M')$, apresentada anteriormente, quando decompomos $A(M')$ em duas submatrizes, conforme discutido na Seção 2.2.5.1, estamos, na verdade, isolando dois subgrafos do grafo original. Cada submatriz representa um subconjunto de vértices (tabelas) e as respectivas arestas (dependências) entre eles, permitindo comparar suas estruturas de derivação de forma independente.

No caso da matriz $A(M')$:

$$A(M') = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

podemos extrair dois subgrafos principais, correspondentes às estruturas derivadas de SOT_E e SOT_G:

$$A_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Ambas as matrizes representam subgrafos equivalentes:

$$G_1 = (\{\text{SOR\_B}, \text{SOR\_D}, \text{SOT\_E}\}, \{(\text{SOR\_B}, \text{SOT\_E}), (\text{SOR\_D}, \text{SOT\_E})\}),$$

$$G_2 = (\{\text{SOR\_B}, \text{SOR\_D}, \text{SOT\_G}\}, \{(\text{SOR\_B}, \text{SOT\_G}), (\text{SOR\_D}, \text{SOT\_G})\}).$$

Esses dois subgrafos são estruturalmente idênticos, pois compartilham o mesmo padrão de dependências. Formalmente, existe uma matriz de permutação P tal que:

$$A_2 = PA_1P^\top,$$

2.3.5 Isomorfismo de Grafos

Dando continuidade ao exemplo anterior, em que verificamos que os subgrafos G_1 e G_2 derivados de $A(M')$ possuem a mesma estrutura de dependências, podemos formalizar essa equivalência por meio do conceito de isomorfismo de grafos. O isomorfismo expressa a ideia de que dois grafos diferentes, em rótulos, nomes ou posições, podem representar exatamente a mesma estrutura de conectividade.

Formalmente, dois grafos $G = (V,E)$ e $G' = (V',E')$ são ditos isomorfos se existe uma função bijetiva

$$f : V \to V'$$

tal que, para todo par de vértices $(u,v) \in E$, a aresta correspondente $(f(u), f(v))$ pertence a E'. Essa relação garante que a estrutura de conexões é preservada, mesmo quando os vértices de G e G' apresentam nomes distintos ou posições diferentes na representação.

No plano matricial, essa condição é expressa pela igualdade

$$B = PAP^\top,$$

em que A e B são as matrizes de adjacência dos grafos G e G', e P é uma matriz de permutação que representa a reordenação dos vértices segundo a bijeção f. Se tal matriz P existir, então G e G' são isomorfos, o que significa que compartilham a mesma topologia interna e as mesmas relações de dependência.

Essa propriedade foi demonstrada na Seção 2.2.5.1, quando verificamos que as submatrizes A_1 e A_2, correspondentes às tabelas SOT_E e SOT_G, satisfazem a relação $A_2 = PA_1P^\top$, comprovando que ambas representam estruturas equivalentes do ponto de vista estrutural.

Em termos conceituais, o isomorfismo de grafos permite reconhecer quando duas arquiteturas, ainda que possuam vértices nomeados de maneira diferente, apresentam o mesmo padrão de relações de dependência e fluxo de dados. No contexto desta dissertação, isso significa que diferentes domínios da arquitetura *Data Mesh* podem conter pipelines estruturalmente idênticos, como SOT_E e SOT_G, mesmo que tenham sido desenvolvidos de forma independente.

A verificação de isomorfismo constitui, portanto, o núcleo lógico do método proposto, pois permite detectar redundâncias estruturais, quantificar equivalências entre pipelines e promover o reuso de componentes já validados, contribuindo para a otimização e padronização das arquiteturas distribuídas de dados.

2.3.6 Aplicações Práticas

A teoria dos grafos encontra múltiplas aplicações, particularmente em cenários que envolvem a modelagem e a compreensão de estruturas relacionais complexas. Uma de suas aplicações mais comuns ocorre no mapeamento de linhagem de dados, onde grafos direcionados são utilizados para descrever o fluxo de origem e transformação dos dados ao longo dos pipelines, como mencionado na seção 2.3.3. Essa abordagem permite rastreabilidade e auditoria precisa, sendo amplamente adotada em arquiteturas de dados modernas.

Por exemplo, em ambientes de integração de informações, os grafos auxiliam na modelagem dos processos de ETL (Extract, Transform, Load), em que os nós representam tabelas, e as arestas expressam dependências e fluxos de dados entre etapas. Essa representação gráfica é especialmente valiosa na detecção de gargalos e redundâncias, adotando a identificação de isomorfismos em subgrafos como estratégia para otimização e deduplicação estrutural (MESSMER; BUNKE, 2000).

Outro uso relevante da teoria dos grafos está na organização semântica de bases de conhecimento. (ROY-HUBARA *et al.*, 2017), demonstram como grafos podem representar esquemas de bancos de dados, onde vértices correspondem a entidades e arestas codificam relacionamentos semânticos ou integrações entre atributos. Essa representação gráfica facilita tanto a modelagem conceitual quanto a navegação por dados complexos.

Por fim, uma aplicação recente que ilustra bem o uso de grafos com redes neurais é a detecção de dados duplicados em bases heterogêneas. (LU *et al.*, 2016) propõem um modelo baseado em GNN (Graph Neural Networks) que utiliza a topologia do grafo para otimizar a detecção de registros duplicados. Nesse modelo, os registros são representados como nós, com arestas indicando similaridades semânticas, e a rede aprende a classificar padrões de duplicidade por meio de um processo supervisionado guiado por algoritmos genéticos. A abordagem mostrou resultados superiores a modelos tradicionais, especialmente em termos de precisão e robustez frente a inconsistências de sintaxe.

Em todos esses casos, a modelagem em grafos atua como um recurso estratégico ao proporcionar uma visão estruturada e abstrata dos sistemas. Conforme argumenta (NETTO, 2012), os grafos fornecem uma linguagem matemática de alto poder expressivo, tornando visíveis padrões, relações e dependências que permanecem ocultos em abordagens tabulares tradicionais. Essa capacidade de representar, comparar e transformar estruturas com precisão formal é especialmente valiosa em arquiteturas modernas e distribuídas como o *Data Mesh*, nas quais a complexidade organizacional dos dados tende a crescer continuamente.

2.4 Algoritmos

A detecção de estruturas redundantes em arquiteturas distribuídas pode ser formalizada como um problema de isomorfismo de subgrafos como mencionado na seção 2.3. Neste contexto, diferentes algoritmos podem ser aplicados para identificar se duas estruturas distintas representam essencialmente a mesma topologia de dados. Para simplificação, esta dissertação avalia três abordagens: uma exata, uma heurística e uma baseada em aprendizado de máquina.

2.4.1 *VF2*: Algoritmo Exato de Isomorfismo

O algoritmo *VF2*, proposto por (CORDELLA *et al.*, 2004), é um dos métodos exatos mais eficientes e reconhecidos na detecção de isomorfismo e subisomorfismo de grafos. Seu objetivo é identificar, dados dois grafos direcionados $G_1 = (N_1, E_1)$ e $G_2 = (N_2, E_2)$, se existe uma função de mapeamento entre os nós de G_1 e os de G_2, preservando suas conexões e, quando especificado, seus rótulos e atributos.

Em termos formais, o algoritmo busca uma função de correspondência $M : N_1 \to N_2$ tal que, para todo par de nós $(n_i, n_j) \in E_1$, a imagem $(M(n_i), M(n_j)) \in E_2$, garantindo assim que a estrutura do grafo G_1 esteja contida em G_2 (subisomorfismo) ou seja estruturalmente idêntica a ele (isomorfismo total).

Para alcançar esse objetivo, o *VF2* constrói o mapeamento passo a passo por meio de uma representação por espaço de estados. Cada estado parcial s é uma configuração atual da busca, representando um subconjunto do mapeamento total $M(s) \subseteq N_1 \times N_2$. A cada nova expansão do estado, um novo par (n, m) é adicionado ao mapeamento, com $n \in N_1$ e $m \in N_2$, desde que ainda não tenham sido utilizados.

A função que decide se um par (n, m) pode ser adicionado é chamada de função de viabilidade:

$$F(s, n, m) = F_{\text{estrutural}}(s, n, m) \wedge F_{\text{semântica}}(s, n, m)$$

A primeira condição, $F_{\text{estrutural}}$, verifica se o par respeita as conexões dos grafos: se os predecessores e sucessores de n já mapeados têm suas imagens coerentes com os predecessores e sucessores de m. Já $F_{\text{semântica}}$ valida se os atributos dos nós e arestas envolvidos são compatíveis, quando disponíveis.

As verificações estruturais incluem regras específicas como:

$$R_{\text{pred}}(s, n, m) \Leftrightarrow \forall n' \in \text{Pred}(n), \exists m' \in \text{Pred}(m) : (n', m') \in M(s)$$

$$R_{\text{succ}}(s,n,m) \Leftrightarrow \forall n' \in \text{Succ}(n), \exists m' \in \text{Succ}(m) : (n',m') \in M(s)$$

Ou seja, para que a expansão do estado seja válida, todo predecessor (ou sucessor) de um nó em G_1 já mapeado deve ter uma correspondência coerente no grafo G_2. Isso garante que a estrutura de conectividade local é preservada no mapeamento parcial em construção.

À medida que esse processo de expansão prossegue , validando passo a passo as conexões, o algoritmo busca construir uma função de mapeamento M tal que todas as arestas presentes em G_1 sejam correspondidas por arestas equivalentes em G_2. Quando isso é alcançado e todos os nós de G_1 foram emparelhados com nós de G_2 (no caso de isomorfismo total), ou um subconjunto representativo foi mapeado (no caso de subisomorfismo), pode-se afirmar que G_1 e G_2 são isomorfos.

Formalmente, isso equivale a dizer que existe uma matriz de permutação P tal que:

$$B = PAP^T$$

onde A e B são as matrizes de adjacência dos grafos G_1 e G_2, respectivamente. Essa equivalência matricial confirma que os dois grafos compartilham a mesma estrutura, ainda que os rótulos dos nós sejam diferentes. Dessa forma, o algoritmo *VF2* traduz a tarefa de identificação de isomorfismo em uma sequência de verificações estruturais e semânticas que convergem para um mapeamento globalmente consistente.

Considere os seguintes grafos direcionados simples:

- $G_1 = (N_1, B_1)$, onde $N_1 = \{A, B, C\}$ e $B_1 = \{(A,B), (B,C)\}$
- $G_2 = (N_2, B_2)$, onde $N_2 = \{X, Y, Z\}$ e $B_2 = \{(X,Y), (Y,Z)\}$

O algoritmo *VF2* tentará construir um mapeamento entre os nós de G_1 e G_2, de modo a preservar a estrutura de adjacência. Um possível mapeamento válido seria:

$$M = \{(A \mapsto X), (B \mapsto Y), (C \mapsto Z)\}$$

Verificamos que:
- $(A,B) \in B_1 \Rightarrow (X,Y) \in B_2$; - $(B,C) \in B_1 \Rightarrow (Y,Z) \in B_2$.

Ou seja, todas as conexões entre os nós de G_1 são preservadas na imagem correspondente em G_2, o que satisfaz a condição de compatibilidade estrutural. Admitindo-se compatibilidade entre os atributos dos nós e das arestas também sejam compatíveis (compatibilidade semântica), o *VF2* concluirá corretamente que G_1 e G_2 são isomorfos.

(CORDELLA *et al.*, 2004) testaram o *VF2* em aplicações reais de reconhecimento gráfico, como o processamento de plantas técnicas, símbolos em desenhos CAD e mapas cadastrais. Nessas aplicações, o algoritmo superou o desempenho de outras abordagens. Essa vantagem decorre do uso eficiente de memória e filtragem que tornam o *VF2* particularmente adequado para tarefas de verificação final em sistemas complexos e com grande número de vértices.

2.4.2 Node Match: Filtro Estrutural Inicial por Equivalência de I/O

O algoritmo *Node Match*, desenvolvido nesta dissertação e baseado nos estudos de (ZENG *et al.*, 2009) e (DU; CAO, 2017), tem como objetivo realizar uma filtragem inicial eficiente entre subgrafos, selecionando apenas aqueles que compartilham características externas compatíveis. Ao contrário de métodos exaustivos, como o *VF2*, o Node Match não explora o espaço completo de mapeamentos possíveis. Em vez disso, opera como um mecanismo de pré-seleção baseado na estrutura superficial dos grafos, especialmente seus pontos de entrada e saída.

Sejam dois subgrafos direcionados $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$. O algoritmo *Node Match* inicia identificando os nós com grau de entrada igual a zero, isto é, vértices que não recebem arestas de outros nós, e os nós com grau de saída igual a zero, ou seja, aqueles dos quais não partem arestas. Esses conjuntos representam, respectivamente, os pontos de entrada e saída de cada subgrafo:

$$\text{In}(G) = \{v \in V \mid \deg^-(v) = 0\}, \quad \text{Out}(G) = \{v \in V \mid \deg^+(v) = 0\}$$

A etapa de filtragem estrutural verifica se os dois subgrafos compartilham o mesmo número de nós de entrada e de saída, uma condição necessária (ainda que não suficiente) para a equivalência estrutural:

$$|\text{In}(G_1)| = |\text{In}(G_2)| \quad \text{e} \quad |\text{Out}(G_1)| = |\text{Out}(G_2)|$$

Essa equivalência superficial permite descartar, de forma eficiente, pares de subgrafos estruturalmente incompatíveis logo nas primeiras etapas, evitando a execução desnecessária de algoritmos exatos de isomorfismo em casos inviáveis.

Além disso, são comparadas os rótulos ou nomes dos nós de entrada e saída. Caso os subgrafos apresentem esse padrão de conectividade semelhante, em termos de direção e quantidade de vértices de entrada e saída, são considerados potenciais equivalentes.

Apenas os subgrafos que atendem a essas condições são encaminhados para a etapa de verificação completa com o algoritmo *VF2*. Dessa forma, o *Node Match* atua como um filtro seletivo, descartando antecipadamente comparações entre estruturas incompatíveis e reduzindo significativamente o espaço de busca.

Considere os seguintes grafos direcionados que representam pipelines de processamento de dados:

- **Grafo** $G_A = (V_A, E_A)$: onde $V_A = \{T_1, T_2, T_3\}$ e $E_A = \{(T_1, T_2), (T_2, T_3)\}$;
- **Grafo** $G_B = (V_B, E_B)$: onde $V_B = \{S_1, S_2, S_3\}$ e $E_B = \{(S_1, S_2), (S_2, S_3)\}$.

Ambos os grafos possuem:

1. Um único nó de entrada: T_1 em G_A e S_1 em G_B, ambos com grau de entrada igual a zero;
2. Um único nó de saída: T_3 em G_A e S_3 em G_B, ambos com grau de saída igual a zero;
3. Três vértices dispostos de forma linear, com dois arcos direcionados que formam uma cadeia sequencial.

Se houver compatibilidade entre os rótulos dos vértices $\ell(T_i) = \ell(S_i)$ para $i = 1,2,3$, o algoritmo *Node Match* identifica os dois grafos como estruturalmente equivalentes. Isso permite, então, aplicar o algoritmo *VF2* para a verificação detalhada de isomorfismo entre G_A e G_B, buscando confirmar que existe uma bijeção entre V_A e V_B que preserva a estrutura de conectividade e os rótulos das arestas.

Com isso, o algoritmo contribui diretamente para a escalabilidade da solução proposta, concentrando o esforço computacional apenas nos casos mais promissores.

2.4.3 Redes Neurais em Grafos (GNN

Este trabalho explora, como alternativa às abordagens clássicas de detecção de isomorfismos (baseadas em busca exaustiva ou heurísticas fixas), o uso de *Graph Neural Networks* (GNNs), redes neurais projetadas para operar diretamente sobre grafos. Ao invés de aplicar regras determinísticas para avaliar similaridade estrutural, o modelo baseado em GNN aprende, por meio de exemplos rotulados, a reconhecer padrões isomórficos mesmo diante de pequenas variações topológicas (LU *et al.*, 2016).

Uma GNN é uma arquitetura de rede neural projetada para operar diretamente sobre grafos. Seu funcionamento consiste em aprender uma representação vetorial para cada vértice, chamada de *embedding*, com base na estrutura local do grafo. O processo ocorre de forma iterativa: em cada camada da rede, cada nó atualiza sua representação combinando suas próprias informações com as de seus vizinhos imediatos. Essa atualização é realizada por uma função de agregação seguida de uma transformação não linear, denominada *função de ativação* (como ReLU ou tanh), aplicada dentro de camadas ocultas da rede, como mostrado na Figura 2.16. Ao longo das camadas, a GNN captura padrões estruturais de forma progressiva, acumulando contexto: a primeira camada enxerga a vizinhança direta, a segunda vê vizinhos dos vizinhos, e assim por diante[4].

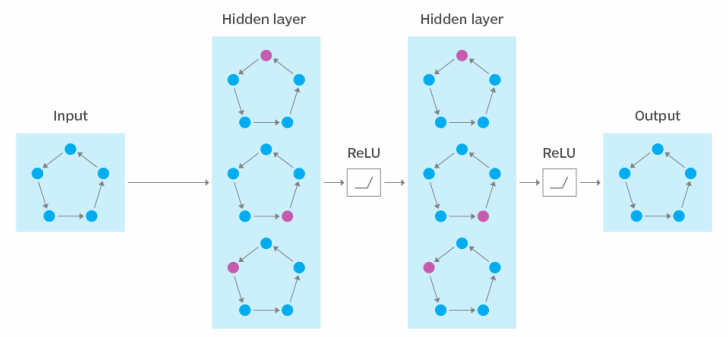


Figura 2.16: Estrutura simplificada de uma GNN. Fonte: Gillis

Neste trabalho, a arquitetura implementada é baseada no modelo *Graph Isomorphism Network (GIN)*, derivada da GNN, proposto por (XU *et al.*, 2019), que se destaca por sua expressividade

[4]A. S. Gillis, *What are graph neural networks (GNNs)?*, TechTarget / The AI Summer, 2024. Disponível em: <https://theaisummer.com/Graph_Neural_Networks/>. Acesso em: 23 jul. 2025.

teórica. A escolha do GIN se justifica por seu desempenho superior em tarefas de comparação estrutural e sua capacidade de generalização.

Seja um grafo direcionado $G = (V, E)$, onde V é o conjunto de vértices (ou nós) e $E \subseteq V \times V$ representa o conjunto de arestas (ligações entre os nós). Em arquiteturas neurais baseadas em grafos, cada vértice $v \in V$ é inicialmente representado por um vetor denso de atributos $h_v^{(0)} \in \mathbb{R}^d$, chamado de *embedding inicial*.

No presente trabalho, para simplificação, assumimos que os vértices não possuem atributos semânticos explícitos, como tipo de dado, nome de tabela ou função na arquitetura. Diante disso, todos os nós recebem o mesmo vetor inicial constante:

$$h_v^{(0)} = \mathbf{1} \in \mathbb{R}^d, \quad \forall v \in V$$

onde $\mathbf{1}$ denota um vetor em $\mathbb{R}^d$ cujas d componentes são iguais a 1. Essa estratégia tem um propósito específico: forçar o modelo a aprender exclusivamente a partir da topologia do grafo, ou seja, das conexões entre os nós, já que o conteúdo de cada nó individualmente é indistinguível dos demais na etapa inicial.

Caso os nós tivessem atributos semânticos relevantes, o vetor $h_v^{(0)}$ poderia ser definido com base neles, o que permitiria à rede neural combinar tanto a estrutura do grafo quanto a semântica local de cada vértice. No entanto, ao nivelar os vetores iniciais, o aprendizado torna-se totalmente dependente da propagação estrutural, o que é particularmente adequado para tarefas de verificação de isomorfismo estrutural.

A atualização dos *embeddings* ao longo das camadas da rede segue o modelo do *Graph Isomorphism Network (GIN)*, proposto por (XU *et al.*, 2019). A fórmula de propagação é definida como:

$$h_v^{(l+1)} = \mathrm{MLP}^{(l)}\left((1 + \varepsilon^{(l)}) \cdot h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l)}\right) \tag{2.1}$$

onde:

- $h_v^{(l)} \in \mathbb{R}^d$ é o embedding do nó v na camada l;
- $\mathcal{N}(v)$ representa o conjunto de vizinhos imediatos de v;
- $\varepsilon^{(l)} \in \mathbb{R}$ é um parâmetro (fixo ou treinável) que regula o peso da contribuição do próprio nó em relação à soma dos vizinhos;
- $\mathrm{MLP}^{(l)} : \mathbb{R}^d \to \mathbb{R}^d$ é uma rede neural chamada *Multi-Layer Perceptron*, composta por camadas totalmente conectadas com ativação não-linear, que transforma a soma agregada em uma nova representação vetorial.

Nas redes neurais aplicadas a grafos (GNNs), cada nó é representado por um vetor (embedding) que é atualizado a partir das informações de sua vizinhança. Esse processo de atualização ocorre por meio de um mecanismo chamado agregação, responsável por coletar os embeddings dos nós vizinhos e combiná-los com o embedding atual do nó central. No caso do *Graph Isomorphism Network* (GIN), essa operação segue dois passos principais:

1. **Agregação Local:** realiza-se a soma dos embeddings dos vizinhos $\sum_{u \in \mathcal{N}(v)} h_u^{(l)}$, adicionando também a representação do próprio nó, ponderada por um fator $(1+\varepsilon^{(l)}) \cdot h_v^{(l)}$, onde $\varepsilon^{(l)}$ é um parâmetro treinável (ou fixo).

2. **Transformação Não-Linear:** o resultado dessa soma é então passado por uma MLP (Multi-Layer Perceptron), que aprende uma nova representação vetorial para o nó, incorporando de forma não-linear as informações estruturais da vizinhança.

Diferentemente de outras arquiteturas de GNN que utilizam médias normalizadas, o GIN utiliza a soma pura como mecanismo de agregação. Essa abordagem foi fundamentada teoricamente por (XU *et al.*, 2019), que demonstraram que a soma possui maior capacidade expressiva para distinguir diferentes estruturas de vizinhança. Isso significa que mesmo grafos com topologias sutilmente distintas podem ser diferenciados com maior precisão, o que torna o GIN especialmente eficaz em tarefas de detecção de isomorfismo estrutural em grafos. Combinado à aplicação de uma MLP com função de ativação não-linear, como a ReLU, o GIN consegue produzir embeddings com alto poder discriminativo.

Ao final de K camadas de propagação, cada nó $v \in V$ terá acumulado informações estruturais de até K passos de vizinhança, representadas no vetor $h_v^{(K)}$. Para transformar todos os embeddings nodais em uma representação única para o grafo como um todo, é aplicada uma operação de agregação global:

$$\mathbf{z}_G = \sum_{v \in V} h_v^{(K)} \tag{2.2}$$

Esse vetor $\mathbf{z}_G \in \mathbb{R}^d$ é chamado de embedding global do grafo, e atua como um resumo vetorial da topologia completa de G.

Para a classificação do isomorfismo entre estes grafos, dado dois subgrafos G_1 e G_2, representados pelos embeddings globais $\mathbf{z}_1$ e $\mathbf{z}_2$, a tarefa de verificação de isomorfismo estrutural é modelada como uma classificação binária. Para isso, os dois vetores são concatenados e processados por uma camada linear com ativação sigmoide:

$$\hat{y} = \sigma\left(\mathbf{W} \cdot [\mathbf{z}_1 \| \mathbf{z}_2] + b\right) \tag{2.3}$$

onde:

- $[\mathbf{z}_1 \| \mathbf{z}_2] \in \mathbb{R}^{2d}$ é a concatenação dos embeddings dos subgrafos;
- $\mathbf{W} \in \mathbb{R}^{1 \times 2d}$ e $b \in \mathbb{R}$ são os parâmetros do classificador denso;
- $\sigma(\cdot)$ é a função sigmoide, que retorna a probabilidade estimada de que $G_1 \cong G_2$.

O modelo é treinado supervisionadamente a partir de pares de subgrafos previamente rotulados: pares positivos ($y = 1$) indicam estruturas isomórficas, enquanto pares negativos ($y = 0$) correspondem a subgrafos estruturalmente distintos. Durante a etapa de inferência, a saída do classificador corresponde a uma probabilidade, sobre a qual se aplica um limiar (threshold) para decidir se o par deve ser considerado equivalente.

Em síntese, o GNN opera como um mecanismo de predição aprendida, capaz de identificar similaridades estruturais com maior flexibilidade que os métodos tradicionais. Embora não substitua algoritmos formais, sua capacidade de generalização o torna particularmente valioso em ambientes *Data Mesh* complexos, onde a escalabilidade, a adaptabilidade e a eficiência são fatores críticos para a governança de dados automatizada.

2.4.4 Comparação Geral dos Algoritmos

Para orientar a seleção do modelo mais apropriado conforme o contexto, complexidade de dados, acurácia e desempenho, organizamos os critérios-chave em um quadro sintético. A Tabela 2.6 sintetiza essas dimensões, facilitando a interpretação dos papéis complementares de cada abordagem na pipeline de detecção de isomorfismos.

Tabela 2.6: Comparativo entre os algoritmos de isomorfismo.

Critério	***VF2***	**Node Match**	**GNN**
Tipo	Exato	Híbrido	Aprendizado Supervisionado
Complexidade	Baixa	Baixa	Alta
Acurácia	Baixa	Baixa	Alta
Desempenho	Baixo	Médio	Alto
Uso ideal	Validação final	Pré-filtragem	Predição baseada em histórico

A comparação entre os três algoritmos evidencia a complementaridade entre abordagens formais, filtragens sintéticas e modelos baseados em aprendizado. O *VF2* se destaca por sua precisão e fundamentação teórica, sendo ideal para etapas finais de verificação onde a garantia formal de isomorfismo é indispensável. No entanto, seu custo computacional elevado limita sua aplicação a conjuntos reduzidos de comparações.

Por outro lado, o algoritmo Node Match propõe uma solução mais leve, voltada à pré-seleção de candidatos com potencial equivalência estrutural. Seu principal mérito reside na eficiência, pois elimina pares obviamente distintos com base em características sintéticas, como grau dos nós, atributos e conectividade direta. Porém, por utilizar o *VF2* depois dessa filtragem, seu custo computacional também é limitado a conjuntos de dados maiores.

A GNN, por sua vez, representa uma abordagem moderna e adaptativa, que aprende diretamente com os dados (aprendizado supervisionado) como reconhecer padrões isomórficos. Embora exija um custo computacional mais alto e infraestrutura adequada, ela se mostra altamente eficaz em contextos com grande volume de dados, oferecendo predições rápidas e escaláveis após o treinamento inicial. Assim, cada técnica assume um papel distinto na pipeline metodológica: Node Match como filtro inicial, GNN como acelerador inteligente e *VF2* como verificador final.

2.5 Métricas de Avaliação

A avaliação da eficácia do método proposto baseia-se em três métricas principais: acurácia (ACC), tempo de execução (ET) e frequência de acertos (*Success Frequency* - SF). Cada métrica mede um aspecto diferente da performance dos algoritmos aplicados (VF2, Node Match e GNN), permitindo uma análise equilibrada entre precisão e eficiência computacional.

A avaliação de modelos classificadores supervisionados, como os utilizados nesta dissertação para detectar isomorfismo entre subgrafos, é fundamentada na análise da matriz de confusão (HASNAIN *et al.*, 2020). Essa matriz resume o desempenho do modelo ao organizar os acertos e erros de classificação em quatro categorias: Verdadeiros Positivos (VP), Falsos Positivos (FP), Verdadeiros Negativos (VN) e Falsos Negativos (FN). A partir dessa estrutura, derivam-se métricas clássicas da aprendizagem supervisionada:

- Acurácia: proporção de classificações corretas entre todas as tentativas; adequada quando há equilíbrio entre as classes e serve como métrica geral de desempenho.

- Precisão: proporção de instâncias classificadas como positivas que são de fato positivas; útil quando o custo de falsos positivos é alto.

- Sensibilidade (ou *recall*): proporção de positivos reais que foram corretamente identificados; importante quando se deseja minimizar falsos negativos.

- F1-Score: média harmônica entre precisão e sensibilidade; equilibra os dois extremos em cenários desbalanceados.

A escolha pela acurácia como métrica principal nesta dissertação se justifica pela necessidade de mensurar o desempenho global de um classificador supervisionado. No entanto, quando não se dispõe de rótulos ou não se deseja formular o problema como uma tarefa de classificação binária, torna-se necessário recorrer a noções contínuas ou não supervisionadas de comparação estrutural.

Nesse contexto, a similaridade entre grafos refere-se ao grau em que duas estruturas compartilham padrões topológicos. Em vez de fornecer uma decisão categórica (isomórfico ou não), essas medidas atribuem um *score* que quantifica o quanto os grafos são semelhantes, mesmo que não sejam idênticos. Essa abordagem é útil em identificação de padrões em grandes bases de grafos.

Diversas técnicas podem ser empregadas para mensurar essa similaridade, incluindo:

- Graph Edit Distance (GED): calcula o custo mínimo necessário para transformar um grafo em outro por meio de operações básicas (adição, remoção ou substituição de nós e arestas) (ZENG *et al.*, 2009).

- Alinhamento Estrutural: busca maximizar a correspondência entre subestruturas dos grafos, sendo amplamente utilizado em bioinformática e reconhecimento de padrões (SHARAN; IDEKER, 2006).

- Kernels de Grafos: projetam os grafos em espaços vetoriais de alta dimensão, utilizando funções especiais para medir similaridade em termos de subestruturas recorrentes (SHERVASHIDZE *et al.*, 2011).
- Métricas Espectrais: comparam os autovalores (espectros) das matrizes associadas aos grafos (como a de adjacência ou a Laplaciana), assumindo que grafos semelhantes possuem espectros semelhantes (WILSON; ZHU, 2008).

Essas métricas são particularmente úteis em tarefas como *clustering* de grafos, onde o objetivo é agrupar estruturas similares sem conhecimento prévio de classes, e *matching* parcial, que visa identificar sobreposições ou subestruturas comuns entre grafos maiores. Ambas as tarefas não exigem pares rotulados de grafos isomórficos, operando em contextos exploratórios.

Neste trabalho, o problema foi formulado explicitamente como uma tarefa de classificação binária, cujo objetivo é prever se dois subgrafos são isomorfos ($y = 1$) ou não ($y = 0$) e, além disso, por simplificação, optou-se por usar a acurácia.

Como fundamento operacional das métricas reportadas, a matriz de confusão organiza acertos e erros em quatro categorias, permitindo visualizar assimetrias entre classes e embasar a escolha de limiares. A Tabela 2.7 apresenta essa estrutura sobre a qual derivamos a acurácia, precisão, sensibilidade e F1-Score decritos anteriormente.

Tabela 2.7: Matriz de Confusão para Isomorfismo de Subgrafos.

Classe Real / Predita	**Não Isomorfo (0)**	**Isomorfo (1)**
Não Isomorfo (0)	Verdadeiro Negativo (TN)	Falso Positivo (FP)
Isomorfo (1)	Falso Negativo (FN)	Verdadeiro Positivo (TP)

- Verdadeiros Positivos (TP): pares isomorfos corretamente identificados como tal;
- Falsos Positivos (FP): pares não isomorfos classificados incorretamente como isomorfos;
- Falsos Negativos (FN): pares isomorfos classificados incorretamente como não isomorfos;
- Verdadeiros Negativos (TN): pares não isomorfos corretamente identificados como tal.

2.5.1 Acurácia (ACC)

Com base na matriz de confusão, define-se a **acurácia** como a proporção total de classificações corretas (positivas e negativas) sobre o total de predições feitas:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

No contexto da identificação de isomorfismo estrutural entre subgrafos, essa métrica indica a capacidade do algoritmo de distinguir corretamente entre estruturas duplicadas e distintas.

Uma acurácia elevada sugere que o modelo está realizando uma discriminação eficaz, enquanto valores baixos indicam presença significativa de erros, sejam falsos positivos (detecção indevida de duplicidade) ou falsos negativos (falha em identificar padrões repetidos).

Essa métrica é particularmente relevante quando decisões de governança e integração de dados são baseadas nos resultados do classificador, uma vez que erros podem gerar inconsistências analíticas ou redundância operacional.

2.5.2 Tempo de Execução (ET)

O tempo de execução representa o tempo necessário para que o algoritmo percorra todos os pares de subgrafos e retorne suas predições. É medido em segundos (s) e calculado com base no tempo total da função de comparação.

$$ET = t_{fim} - t_{incio}$$

A métrica ET é essencial para avaliar a viabilidade prática dos algoritmos, sobretudo em arquiteturas reais com milhares de tabelas distribuídas. Algoritmos com alta acurácia mas tempo de execução inviável não são apropriados para ambientes produtivos com grande escala.

2.5.3 *Success Frequency* (SF): Frequência de sucesso

A métrica *Success Frequency* (SF) foi concebida neste trabalho como uma medida de eficiência algorítmica que integra duas dimensões críticas da avaliação de modelos: a acurácia e o tempo de execução. Sua proposta é simples, mas poderosa: quantificar o número de acertos totais produzidos por segundo de execução. Essa métrica torna-se especialmente relevante no contexto de arquiteturas distribuídas e escaláveis, como o *Data Mesh*, em que tanto a qualidade quanto a velocidade de decisão são vitais.

Seja ACC a acurácia de um algoritmo, definida no intervalo $[0,1]$, e N_{pares} o total de pares de subgrafos analisados durante o processo. Definindo ET como o tempo total de execução (em segundos), a *Success Frequency* (SF) é expressa como:

$$SF = \frac{ACC \cdot N_{pares}}{ET}$$

Unidade: acertos por segundo (s^{-1})

A SF pode ser vista como a derivada do número de acertos em relação ao tempo, i.e., $\frac{d(Acertos)}{dt}$, sob a hipótese de execução determinística e tempo contínuo. Ela assume que os acertos são produzidos uniformemente durante a execução do algoritmo, permitindo uma interpretação em termos de frequência de sucesso temporal.

- Se $SF = 2s^{-1}$, o algoritmo gera, em média, dois pares corretamente identificados por segundo.
- Se $SF \to 0$, o algoritmo ou é impreciso, ou extremamente lento, ou ambos.

Ao contrário de métricas clássicas como Acurácia (ACC) ou Tempo (ET) isoladamente, a SF permite comparar algoritmos com diferentes comportamentos de execução. Por exemplo:

- Algoritmo A: $ACC = 0.95$, $ET = 100\,s$, $N_{pares} = 1000$ resulta em $SF = 9.5$
- Algoritmo B: $ACC = 0.80$, $ET = 20\,s$, $N_{pares} = 1000$ resulta em $SF = 40$

Embora o algoritmo A tenha melhor acurácia, o algoritmo B tem maior eficiência operacional por segundo, podendo ser mais útil em sistemas com alta demanda em tempo real.

No contexto do *Isomera*, a SF é utilizada como a principal métrica de escolha entre algoritmos:

1. Para cada execução do VF2, Node Match ou GNN, o sistema registra:
   - O total de pares avaliados N_{pares};
   - O tempo de execução total ET;
   - O número de acertos confirmados por validação humana $N_{acertos}$.
2. Calcula-se a acurácia $ACC = \frac{N_{acertos}}{N_{pares}}$;
3. A SF é então computada automaticamente para todos os cenários e algoritmos testados:

$$SF = \frac{N_{acertos}}{ET} = \frac{ACC \cdot N_{pares}}{ET}$$

4. Os resultados são organizados em tabelas e visualizações comparativas.

A SF permite à ferramenta *Isomera* recomendar o algoritmo mais apropriado conforme o contexto:

- Cenários com pouca complexidade e alta demanda por velocidade: preferir heurísticas com alta SF mesmo com menor ACC;
- Cenários críticos com alta exigência de precisão: priorizar algoritmos com alta ACC, mesmo com menor SF;
- Cenários de benchmarking ou análise híbrida: comparar modelos por SF permite avaliar a capacidade de escalar com qualidade.

A *Success Frequency* atua como uma métrica integradora, balanceando desempenho computacional com acerto lógico. Ao adotar essa métrica como guia, o método proposto assume um caráter operacional realista e adaptável, essencial para a maturidade de soluções em ambientes de dados altamente distribuídos como o *Data Mesh*.

CAPÍTULO 3

Metodologia

Este capítulo apresenta a metodologia desenvolvida para identificar redundâncias estruturais em arquiteturas baseadas no paradigma *Data Mesh*, por meio da modelagem dos esquemas relacionais como grafos direcionados e da aplicação de algoritmos de detecção de isomorfismo estrutural entre domínios. Cada subseção representa uma etapa da metodologia, organizada em termos de entradas (dados ou artefatos necessários), ações (operações executadas) e saídas (resultados produzidos), garantindo clareza na descrição do processo e reprodutibilidade dos procedimentos.

3.1 Metodologia Proposta

A metodologia proposta tem como propósito identificar e validar redundâncias estruturais em arquiteturas de dados, por meio de um processo fundamentado em modelagem matemática, *detecção* e *validação*. Embora tenha sido concebida com foco em arquiteturas *Data Mesh*, o método é aplicável a qualquer ambiente em que os elementos de uma arquitetura possam ser modelados como grafos direcionados, incluindo *Data Warehouses*, *Data Lakes* e sistemas híbridos. Em todos esses contextos, as tabelas, relações de transformação e dependências entre domínios podem ser representadas por vértices e arestas, tornando o processo de identificação de equivalências estruturais generalizável e reprodutível.

A metodologia foi estruturada para manter um equilíbrio entre rigor teórico e aplicabilidade prática, permitindo que arquiteturas complexas sejam analisadas de forma sistemática e transparente. Ela se apoia em conceitos de álgebra linear e teoria dos grafos, descritos na Seção 2.3.4, e é composta por quatro etapas principais que se interligam de maneira iterativa: modelagem conceitual da arquitetura, detecção de isomorfismos estruturais, validação supervisionada das duplicidades e consolidação dos resultados. Essa estrutura visa garantir reprodutibilidade, clareza no fluxo de decisão e consistência entre as análises teóricas e os resultados obtidos empiricamente.

A Figura 3.1 apresenta uma visão geral do processo metodológico proposto, destacando as quatro etapas centrais e seus fluxos de interação. O diagrama mostra que o processo se inicia com a modelagem em grafos, a partir de três possíveis origens: a geração randômica de grafos sintéticos, o uso de arquiteturas reais baseadas no benchmark TPC-DS ou a modelagem personalizada definida pelo usuário. Em seguida, a metodologia avança para a fase de detecção de isomorfismo, onde são aplicados diferentes algoritmos (VF2, Node Match e GNN) com o objetivo de identificar subgrafos estruturalmente equivalentes.

Os resultados dessa detecção são então encaminhados à etapa de validação supervisionada,

na qual o especialista confirma ou rejeita as correspondências sugeridas pelos algoritmos. Caso os pares validados não apresentem redundâncias estruturais consistentes, o processo retorna à etapa de modelagem, ajustando critérios ou parâmetros de geração dos grafos, o que caracteriza o caráter iterativo do método. Após a validação, realiza-se a avaliação das métricas de desempenho (acurácia, tempo de execução e frequência de sucesso), consolidando os resultados e gerando como produto final um grafo otimizado, sem vértices duplicados, representando uma arquitetura de dados mais enxuta e consistente.

Essa representação visual (Figura 3.1) sintetiza o ciclo contínuo de refinamento que orienta a metodologia, destacando a retroalimentação entre as fases e o alinhamento entre análise algorítmica e validação humana. O fluxo apresentado reforça a natureza reprodutível e expansível do processo, que pode ser aplicado a diferentes tipos de arquitetura de dados, sejam elas *Data Mesh*, *Data Warehouses* ou *Data Lakes*, desde que suas estruturas possam ser expressas como grafos direcionados.

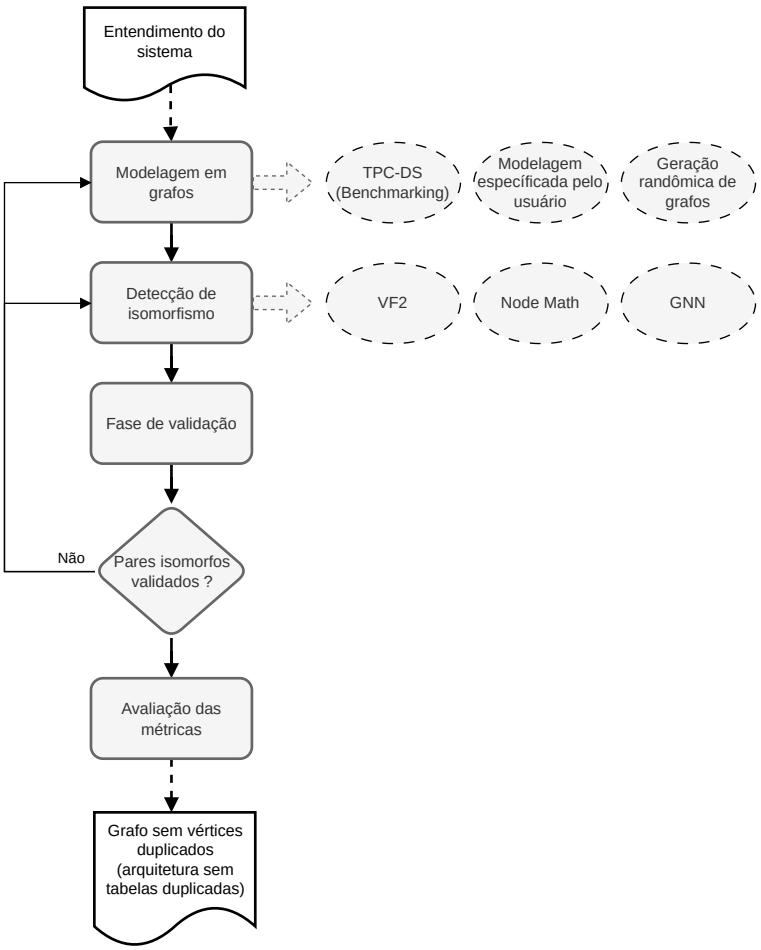


Figura 3.1: Fluxo metodológico para detecção e validação de redundâncias estruturais em arquiteturas de dados.

3.1.1 Entendimento do Sistema

A primeira fase consiste em compreender profundamente o sistema de dados sob análise, identificando seus domínios, tabelas e fluxos de dependência. Essa etapa é fundamental para garantir que o processo de *detecção* e *validação* ocorra sobre uma representação coerente da arquitetura, evitando interpretações equivocadas das conexões entre os elementos do sistema.

É ideal que, antes da execução da metodologia, o pesquisador já disponha de um mapeamento claro das relações entre as tabelas que deseja estudar, isto é, quais dependem de quais, quais servem como origem e quais representam transformações intermediárias ou produtos finais. Quando essa informação não está explicitamente documentada, torna-se necessário realizar o processo de transformação da arquitetura em um grafo dirigido, atividade que demanda a identificação manual das tabelas e de suas dependências lógicas. Esse procedimento foi aplicado no presente trabalho utilizando o benchmark TPC-DS, que será detalhado na Seção 5, onde as tabelas foram segmentadas em domínios e classificadas de acordo com os papéis de origem (*SOR*), transformação (*SOT*) e consumo (*SPEC*).

Essa etapa inicial, portanto, não apenas define o escopo da modelagem como também delimita os limites de observação, assegurando que apenas as relações relevantes sejam consideradas no grafo resultante. O produto é um mapa conceitual estruturado, que serve como base formal para a etapa seguinte de modelagem em grafos e posterior aplicação dos algoritmos de detecção de isomorfismo.

Como perspectiva futura, propõe-se o desenvolvimento de um módulo automatizado para geração de grafos diretamente a partir de esquemas SQL ou modelos de dados relacionais existentes. Tal funcionalidade permitiria converter automaticamente as dependências entre tabelas em representações gráficas, eliminando a necessidade de mapeamento manual e ampliando a escalabilidade da metodologia em ambientes reais. Embora ainda não implementada nesta dissertação, essa capacidade representa um avanço natural do trabalho e o caminho ideal para automatizar a fase inicial de compreensão do sistema.

3.1.2 Modelagem em Grafos

Nesta etapa, a arquitetura de dados é convertida em um grafo direcionado $G = (V, E)$, em que cada vértice $v \in V$ representa uma tabela (SOR, SOT, SPEC) e cada aresta $e \in E$ representa uma relação de dependência/transformação entre elas. Em paralelo, é gerada a matriz de adjacência A, que codifica as conexões de forma binária e serve de base para as etapas de *detecção* e *validação*. O objetivo é produzir uma representação coerente e verificável, pronta para a comparação entre subestruturas.

Tabela 3.1: Entradas, ações e saídas da etapa de modelagem em grafos.

Categoria	**Descrição**
Entradas	Mapa conceitual do sistema (domínios, tabelas e relações de dependência); fontes de dados estruturadas (`JSON`/`CSV`) ou benchmark (ex.: TPC-DS); parâmetros opcionais para geração sintética (número de domínios, limites de tabelas por domínio e proporção entre SOR, SOT e SPEC); convenções estruturais (DAG sem ciclos, ausência de auto-laços e consistência na hierarquia SOR→SOT→SPEC).
Ações	Normalização das entradas e padronização dos papéis SOR, SOT e SPEC; construção do grafo direcionado $G = (V, E)$ utilizando a biblioteca `NetworkX`; geração da matriz de adjacência A e dos dicionários de mapeamento entre vértices e índices; execução de um *procedimento de validação estrutural*, que verifica e corrige inconsistências topológicas.; por fim, os artefatos são persistidos em `JSON`/`CSV`/`PNG` e disponibilizados para inspeção.
Saídas	Grafo direcionado $G = (V, E)$ consistente e pronto para análise; matriz de adjacência $A \in \{0,1\}^{\lvert V\rvert \times \lvert V\rvert}$ com ordenação indexada de V; dicionários de mapeamento (índice ↔ tabela); *relatório de verificação estrutural* contendo: número total de nós e arestas válidos, quantidade de anomalias detectadas, tipo de inconsistência e ação aplicada.

3.1.2.0.1 Entradas. Partem de um mapa conceitual (domínios, tabelas e dependências) e de uma fonte: arquivos `JSON`/`CSV` previamente estruturados, o benchmark TPC-DS, ou uma configuração sintética controlada por parâmetros (domínios e limites mínimo/máximo de tabelas por domínio, além da proporção SOR/SOT/SPEC). As convenções de nomenclatura asseguram unicidade e identificação clara do papel de cada tabela.

3.1.2.0.2 Ações. Após normalizar nomes e papéis, o grafo é construído em `NetworkX` e sua matriz de adjacência A é gerada com ordenação consistente de V. Em seguida, são executadas checagens de sanidade para capturar problemas com a topologia: (i) SPEC sem quaisquer predecessores (isto é, "criadas do nada"); (ii) SOT sem origem; (iii) SOR com entradas (por exemplo, uma SOR derivada de outra SOR). Quando a arquitetura é gerada sinteticamente, os nós/arestas que violam essas regras são removidos automaticamente do grafo final. Quando a arquitetura é importada, as ocorrências são apenas registradas e exibidas ao usuário, preservando fielmente o insumo original. Por fim, os artefatos são persistidos (`JSON`/`CSV`/`PNG`) e disponibilizados para inspeção.

3.1.2.0.3 Saídas. O resultado é um grafo $G = (V, E)$ pronto para análise, a matriz de adjacência A binária e os dicionários de mapeamento índice↔vértice. Acompanha o conjunto um

relatório de checagens que detalha cada inconsistência detectada e a ação adotada (remoção em geração sintética; sinalização em importação), garantindo rastreabilidade e reprodutibilidade para as etapas seguintes de *detecção* e *validação*.

3.1.3 Detecção de Isomorfismo

A etapa de detecção constitui o núcleo analítico da metodologia, sendo responsável por identificar subgrafos estruturalmente equivalentes em diferentes partes da arquitetura de dados. Dois subgrafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são considerados isomorfos se existir uma bijeção $f : V_1 \to V_2$ tal que, para todo par (u, v), vale a condição $(u, v) \in E_1 \Rightarrow (f(u), f(v)) \in E_2$. Em termos matriciais, essa equivalência é expressa pela relação $B = PAP^\top$, em que A e B são as matrizes de adjacência dos subgrafos e P é uma matriz de permutação, como discutido na Seção 2.3.5. Essa formulação garante que a comparação entre estruturas seja independente da ordenação dos vértices, permitindo avaliar a similaridade puramente estrutural entre diferentes partes da arquitetura.

A metodologia proposta não define um algoritmo específico para a detecção de isomorfismos, mas estabelece o processo pelo qual um algoritmo deve ser aplicado e avaliado. Ou seja, a metodologia é agnóstica em relação à técnica utilizada, podendo empregar tanto métodos exatos quanto heurísticos ou baseados em aprendizado de máquina. Neste trabalho, foram implementadas e comparadas três abordagens representativas: (i) o algoritmo VF2, voltado à detecção exata de isomorfismo de subgrafos; (ii) o método Node Match, um algoritmo híbrido com *pré-filtragem* de atributos de nós antes da comparação topológica; e (iii) o modelo GNN (*Graph Neural Networks*), voltado à predição de isomorfismo estrutura por meio de representações vetoriais aprendidas. Essas abordagens foram escolhidas por cobrirem diferentes dimensões de análise, precisão, custo computacional e capacidade de generalização, e podem ser executadas de forma independente ou comparadas dentro da ferramenta *Isomera*.

A natureza modular da metodologia permite a inclusão de novas técnicas de detecção conforme o avanço da pesquisa. Outros algoritmos podem ser incorporados, como métodos probabilísticos, estratégias baseadas em padrões frequentistas ou modelos híbridos que combinem aprendizado supervisionado e heurísticas estruturais. Além disso, há um potencial promissor para integrar abordagens de inteligência artificial generativa, capazes de propor representações intermediárias de grafos ou sugerir automaticamente pares candidatos a isomorfismo. Essas extensões estão previstas como desdobramentos futuros, reforçando o caráter evolutivo e adaptável da metodologia proposta a diferentes contextos analíticos e arquiteturais.

A seguir, a Tabela 3.2 apresenta um resumo técnico das entradas, ações e saídas dessa etapa.

Tabela 3.2: Entradas, ações e saídas da etapa de detecção de isomorfismo.

Categoria	**Descrição**
Entradas	Grafo dirigido $G = (V, E)$ e matriz de adjacência; definição dos subgrafos de interesse (por domínio, tipo de tabela ou profundidade); seleção do algoritmo de detecção (*VF2*, *Node Match* ou *GNN*); parâmetros de comparação, como limite de profundidade e atributos considerados (grau, tipo, domínio).
Ações	1. Segmentação do grafo em subgrafos candidatos a redundância; 2. Aplicação da técnica escolhida: - VF2: detecção exata de isomorfismo de subgrafos; - Node Match: algoritmo híbrido com pré-filtragem por atributos e posterior verificação topológica; - GNN: predição de isomorfismo estrutural via embeddings aprendidos; 3. Cálculo das métricas parciais (número de pares, tempo e similaridade média); 4. Registro dos pares candidatos e salvamento do log de execução.
Saídas	Conjunto de pares de subgrafos potencialmente isomórficos; métricas de desempenho associadas (tempo médio de execução, taxa de correspondência, confiança da predição); relatório de execução contendo parâmetros aplicados e estatísticas de correspondência estrutural.

3.1.3.0.1 Entradas. O processo inicia-se a partir do grafo dirigido gerado na etapa anterior e da sua matriz de adjacência, utilizados como base para as comparações estruturais. São definidos os subgrafos de interesse, por exemplo, grupos de tabelas pertencentes ao mesmo domínio ou com padrões de ligação semelhantes, e escolhida a abordagem de detecção mais adequada à análise pretendida. Essas entradas determinam o escopo da busca e o grau de detalhamento da comparação entre estruturas.

3.1.3.0.2 Ações. O sistema realiza uma segmentação inicial do grafo e, em seguida, aplica o algoritmo de detecção selecionado:

- VF2: realiza a detecção exata de isomorfismo de subgrafos, comparando todas as combinações de vértices e arestas. Ideal para cenários de alta precisão, embora mais custoso computacionalmente.
- Node Match: aplica uma pré-filtragem baseada em atributos (como tipo da tabela, grau de entrada/saída e domínio) antes da análise topológica, reduzindo o espaço de busca e o tempo de execução. É indicado para arquiteturas com grande número de nós e arestas.
- GNN: realiza a predição de isomorfismo estrutural utilizando representações vetoriais (*embeddings*) de cada subgrafo aprendidas por uma rede neural gráfica. Essa abordagem é robusta a pequenas variações estruturais e útil para cenários onde o isomorfismo não é estritamente exato.

3.1.3.0.3 Saídas. Como resultado, obtém-se um conjunto de pares de subgrafos classificados como estruturalmente semelhantes, acompanhados de métricas quantitativas de desempenho e confiança. Esses pares são registrados em relatórios de execução e armazenados para posterior análise na fase de validação, onde serão avaliados quanto à sua equivalência funcional e semântica.

3.1.4 Validação Supervisionada

A validação supervisionada representa uma das fases mais críticas da metodologia, pois é nela que o julgamento humano complementa e refina os resultados obtidos pelos algoritmos de detecção. Mesmo que dois subgrafos apresentem uma equivalência estrutural perfeita, isso não implica necessariamente que desempenhem a mesma função dentro da arquitetura de dados. Em contextos reais, especialmente em sistemas analíticos complexos, uma tabela pode ter o mesmo formato e estrutura de outra, mas servir a propósitos distintos, como domínios diferentes, regras de filtragem específicas ou segmentações de negócio. Por isso, a intervenção manual é indispensável para evitar decisões incorretas, como a exclusão de tabelas relevantes ou a fusão indevida de estruturas que, embora semelhantes, não são redundantes do ponto de vista funcional.

Durante esta etapa, o especialista analisa os pares de subgrafos identificados pelos algoritmos e decide, com base em sua compreensão contextual da arquitetura, se a correspondência representa uma duplicidade funcional real ou não. Essa análise é conduzida diretamente na ferramenta, que apresenta lado a lado os subgrafos candidatos, suas tabelas e dependências, permitindo uma comparação visual precisa e contextualizada.

Nesta fase, existem duas possibilidades principais de decisão: (i) o usuário confirma que um dos pares é realmente duplicado, validando assim a predição feita pelo algoritmo, caracterizando um caso de verdadeiro positivo (TP); ou (ii) o usuário rejeita a sugestão de duplicidade, indicando que, embora estruturalmente semelhantes, os subgrafos exercem papéis distintos no contexto da arquitetura, configurando um falso positivo (FP).

Além desses casos apresentados na interface, há uma terceira situação implícita: os pares que são conhecidos como duplicados (por exemplo, na base de benchmark utilizada) mas que não foram identificados automaticamente pelo algoritmo. Esses pares não são exibidos ao usuário durante a validação, mas são registrados internamente na base de dados, compondo os falsos negativos (FN) utilizados na etapa de cálculo das métricas.

Dessa forma, a validação supervisionada não se limita à simples confirmação ou rejeição das predições do algoritmo, mas atua como o ponto de integração entre a detecção automática e o julgamento humano. É a partir dessas decisões que se constrói a base de verdade (*ground truth*) da metodologia, a qual permitirá, posteriormente, a avaliação quantitativa de desempenho dos algoritmos por meio da matriz de confusão.

Tabela 3.3: Entradas, ações e saídas da etapa de validação supervisionada.

Categoria	**Conteúdo**
Entradas	Conjunto de pares candidatos a isomorfismo estrutural, resultante da etapa de detecção; representações gráficas e matriciais de cada subgrafo
Ações	Apresentação dos pares candidatos ao usuário na interface gráfica; análise contextual de cada par, considerando tanto a estrutura quanto a função; classificação dos pares como duplicados (TP), distintos (FP) ou falsos negativos (FN); registro das decisões na base de verdade (*ground truth*) e atualização da matriz de confusão; escolha dos vértices a serem removidos em caso de duplicidade confirmada.
Saídas	Base de verdade validada, contendo os rótulos atribuídos a cada par analisado; matriz de confusão consolidada com os cenários TP, FP, FN e TN; grafo anotado com as decisões de exclusão ou preservação; relatório de validação contendo as estatísticas e observações do especialista.

3.1.4.0.1 Entradas. A validação supervisionada recebe como entrada o conjunto de pares de subgrafos gerados na fase de detecção. Cada par contém suas representações gráficas e matriciais, além de informações contextuais como o domínio de origem, o tipo de tabela (SOR, SOT ou SPEC) e o grau de conectividade. Esses elementos são fundamentais para que o usuário possa compreender o significado funcional de cada estrutura antes de classificá-la.

3.1.4.0.2 Ações. As ações executadas nesta etapa ocorrem de forma interativa, diretamente na interface da ferramenta. Os pares são apresentados ao usuário lado a lado, permitindo a comparação visual e a análise contextual. O avaliador classifica cada correspondência como duplicada, distinta ou não detectada, decisão que é registrada automaticamente na base de verdade. A partir dessas classificações, a matriz de confusão é atualizada, fornecendo a base quantitativa necessária para calcular as métricas de desempenho dos algoritmos. Quando uma duplicidade é confirmada, o usuário também pode selecionar qual nó deve ser removido, garantindo a integridade da arquitetura final.

3.1.4.0.3 Saídas. O principal produto desta etapa é a base de verdade validada, que representa o resultado consolidado da interação entre o algoritmo e o especialista. Essa base contém todos os rótulos atribuídos aos pares analisados e alimenta diretamente a matriz de confusão. O sistema também gera um relatório de validação com estatísticas, observações e um grafo anotado, destacando os pares confirmados como redundantes. Esses registros não apenas permitem o cálculo posterior de métricas como acurácia, tempo e frequência de acertos, mas também garantem a reprodutibilidade e a rastreabilidade de todas as decisões tomadas.

Por fim, a validação supervisionada cumpre um papel central de controle de qualidade,

assegurando que apenas redundâncias reais sejam tratadas como isomorfismos válidos. Essa etapa protege a arquitetura de remoções indevidas e preserva a coerência semântica e funcional dos dados, funcionando como um elo essencial entre a análise automatizada e o julgamento humano. Ela traduz o equilíbrio entre precisão computacional e entendimento contextual, consolidando a metodologia como uma abordagem confiável, auditável e adaptável para o estudo de arquiteturas de dados complexas.

Critérios e protocolo de validação

Nesta dissertação, o autor atuou como único avaliador especializado, acumulando experiência direta tanto sobre o conjunto de dados quanto sobre os cenários simulados utilizados nos experimentos. A presença de um avaliador com conhecimento profundo do contexto é imprescindível nesta etapa: a decisão sobre duplicidade funcional exige interpretação de nuances de uso, escopo de domínio, regras de filtragem e consumidores, que não são plenamente capturadas pela análise estrutural automática. A opção por um único avaliador decorreu de restrições de escopo e tempo, sendo mitigada por um protocolo explícito de decisão, registro breve de justificativas e rastreabilidade integral no *ground truth*. Para trabalhos futuros, recomenda-se incluir múltiplos avaliadores. A seguir, sintetizam-se as diretrizes adotadas para garantir consistência, auditabilidade e reprodutibilidade:

- Critério de equivalência funcional: a confirmação de duplicidade requer, além da equivalência estrutural, a análise de papel funcional (uso, regras de filtragem, consumidores) do subgrafo no domínio.

- Desempate semântico: em casos de estruturas idênticas com finalidades distintas (por exemplo, segmentações de negócio), o par é rotulado como *não redundante*.

- Documentação da decisão: cada decisão registra justificativa breve e os identificadores dos nós envolvidos, compondo o *ground truth* auditável.

- Conflitos entre avaliadores: quando houver múltiplos avaliadores, recomenda-se regra de maioria simples; empates devem ser resolvidos por um terceiro avaliador com maior senioridade. (Não se aplica aos experimentos desta dissertação.)

3.1.5 Avaliação das Métricas

A etapa de avaliação das métricas encerra o ciclo metodológico, quantificando a eficácia dos algoritmos utilizados na detecção de isomorfismo a partir das classificações obtidas durante a fase de validação. Seu objetivo é traduzir o julgamento humano, consolidado na base de verdade (*ground truth*), em indicadores quantitativos que permitam comparar objetivamente diferentes abordagens, identificando o equilíbrio entre precisão e eficiência computacional. Essa análise é fundamental para avaliar não apenas a qualidade das detecções realizadas, mas também o custo operacional associado a cada algoritmo, oferecendo suporte à escolha do método mais adequado para cada tipo de arquitetura ou volume de dados.

Tabela 3.4: Entradas, ações e saídas da etapa de avaliação das métricas.

Categoria	**Conteúdo**
Entradas	Base de verdade (*ground truth*) contendo decisões do usuário (pares confirmados e rejeitados); matriz de confusão gerada a partir das classificações TP, FP, FN e TN; tempos de execução registrados para cada algoritmo testado.
Ações	Leitura e consolidação das decisões de validação; atualização da matriz de confusão; cálculo das métricas de desempenho (Acurácia, Tempo de Execução e Frequência de Sucesso); comparação cruzada entre algoritmos; geração de relatórios e gráficos de desempenho.
Saídas	Indicadores quantitativos de desempenho; tabelas comparativas e visualizações das métricas; relatório de eficiência por algoritmo; grafo final sem vértices redundantes, representando a arquitetura otimizada.

3.1.5.0.1 Entradas. A avaliação inicia a partir da base de verdade consolidada na etapa anterior, composta pelas decisões do usuário sobre os pares de subgrafos analisados. Essas informações são estruturadas em uma matriz de confusão, na qual se registram os casos de verdadeiro positivo (TP), falso positivo (FP), falso negativo (FN) e verdadeiro negativo (TN). Além disso, são utilizados os tempos de execução medidos durante a aplicação de cada algoritmo de detecção, garantindo que o desempenho seja avaliado tanto sob a ótica da precisão quanto da eficiência.

3.1.5.0.2 Ações. Primeiro, a metodologia realiza a leitura e a consolidação das decisões de validação, atualizando a matriz de confusão de forma automática. Com base nesses dados, são calculadas três métricas principais: (i) Acurácia (ACC), que representa a proporção de classificações corretas em relação ao total avaliado; (ii) Tempo de Execução (ET), correspondente ao tempo médio gasto pelo algoritmo na análise dos pares; e (iii) Frequência de Sucesso (SF), métrica composta que expressa a razão entre o número de acertos e o tempo total de execução, combinando qualidade e velocidade. A seguir, são realizadas comparações cruzadas entre algoritmos, como VF2, Node Match e GNN —, permitindo analisar como diferentes estratégias se comportam sob o mesmo conjunto de dados. Essas comparações são sintetizadas em tabelas e gráficos, auxiliando na interpretação dos resultados e na identificação da abordagem mais eficiente para cada cenário.

3.1.5.0.3 Saídas. O resultado final da avaliação consiste em um conjunto de indicadores quantitativos que descrevem o desempenho dos algoritmos em termos de precisão e eficiência. Essas métricas permitem não apenas comparar métodos distintos, mas também calibrar os parâmetros da metodologia para execuções futuras. Com base nos resultados obtidos, o grafo é reconstruído sem vértices redundantes, refletindo uma arquitetura otimizada e coerente com as

decisões validadas. O relatório final apresenta, de forma consolidada, as métricas calculadas, as análises comparativas e as visualizações correspondentes, permitindo uma interpretação clara e reprodutível do desempenho de cada algoritmo.

3.1.5.0.4 Considerações Finais. Essa fase da metodologia não apenas quantifica resultados, mas também reforça seu caráter científico e auditável. Ao integrar julgamento humano, análise algorítmica e métricas objetivas, o processo garante transparência e confiabilidade, possibilitando reproduzir experimentos, ajustar parâmetros e expandir o estudo para novas abordagens de detecção. Além disso, a estrutura modular adotada permite incorporar métricas adicionais, como precisão, revocação ou F1-Score —, ampliando a capacidade analítica da metodologia e abrindo caminho para investigações futuras.

3.2 Grafo sem Vértices Duplicados

A última fase da metodologia consolida os resultados obtidos nas etapas anteriores, gerando uma nova versão da arquitetura de dados na forma de um grafo orientado sem redundâncias estruturais. Esse grafo representa o produto final do processo, no qual as duplicidades identificadas pelos algoritmos e confirmadas pelo usuário durante a validação são removidas de forma controlada e documentada. O resultado é uma arquitetura racionalizada, mais coesa e semanticamente estável, que preserva apenas os fluxos de transformação e dependência efetivamente válidos, eliminando sobreposições funcionais entre domínios ou camadas de dados.

Tabela 3.5: Entradas, ações e saídas da etapa de consolidação do grafo final.

Categoria	**Conteúdo**
Entradas	Conjunto validado de pares duplicados (base de verdade); grafo original $G = (V, E)$; informações de dependência e metadados associados aos nós e arestas.
Ações	Remoção dos vértices redundantes identificados durante a validação; reconstrução das conexões incidentes preservando a integridade topológica; atualização das relações entre domínios e camadas SOR/SOT/SPEC; geração do grafo otimizado $G' = (V', E')$ e exportação dos resultados (JSON/CSV/PNG); comparação entre a topologia original e a final para mensurar o ganho estrutural.
Saídas	Grafo orientado G' sem redundâncias estruturais; relatório de modificações aplicadas (vértices removidos, arestas ajustadas e domínios afetados); visualização comparativa entre G e G'; indicadores de melhoria topológica (redução de nós, densidade e conectividade média).

3.2.0.0.1 Entradas. A reconstrução do grafo parte da base de verdade consolidada, contendo as duplicidades confirmadas pelo usuário e registradas durante a validação supervisionada. Essas informações são aplicadas sobre o grafo original $G = (V, E)$, que contém os vértices correspondentes às tabelas e as arestas representando suas relações de dependência. Além disso, são consideradas informações auxiliares como o tipo da tabela (SOR, SOT ou SPEC), o domínio de origem e o papel topológico de cada vértice, garantindo que a exclusão de redundâncias preserve a coerência semântica do modelo.

3.2.0.0.2 Ações. O sistema executa a remoção das duplicidades de forma controlada, eliminando apenas os vértices marcados como redundantes e realocando as arestas associadas quando necessário. Essa reconstrução assegura que a estrutura permaneça um grafo direcionado acíclico (DAG), sem laços indevidos ou desconexões não intencionais. Em seguida, o grafo otimizado $G' = (V', E')$ é gerado e exportado em diferentes formatos (JSON, CSV e PNG), acompanhado de um relatório detalhado das modificações realizadas, incluindo a contagem de vértices removidos, arestas reconfiguradas e domínios impactados. Por fim, realiza-se uma comparação entre a topologia original e a resultante, permitindo mensurar o ganho estrutural obtido em termos de redução de redundâncias e melhoria da conectividade média.

3.2.0.0.3 Saídas. O produto final é o grafo otimizado G', livre de duplicidades e semanticamente consistente, representando uma versão consolidada e depurada da arquitetura de dados. Além da estrutura visual e da matriz de adjacência correspondente, são gerados indicadores quantitativos de melhoria, como redução percentual de vértices e diminuição de densidade de arestas, que demonstram o impacto direto da metodologia na racionalização da arquitetura. Esses resultados servem tanto para documentação e auditoria quanto para embasar decisões de reconfiguração de pipelines e políticas de governança de dados.

3.2.0.0.4 Análise dos Resultados. A consolidação do grafo final simboliza o fechamento do ciclo metodológico, unindo a precisão algorítmica à curadoria humana. O modelo resultante não apenas reflete a eliminação de redundâncias estruturais, mas também materializa uma visão otimizada do sistema de dados, onde cada elemento existente tem um papel funcional justificado. Além disso, a comparação entre o grafo original e o otimizado fornece evidências quantitativas e visuais do impacto da metodologia, servindo como referência para ajustes futuros, replicações experimentais e extensões da abordagem em outros tipos de arquitetura, como *Data Warehouses* e *Data Lakes*.

CAPÍTULO 4

Ferramental — Isomera

A ferramenta *Isomera* foi desenvolvida no contexto desta dissertação como um artefato computacional que dá suporte direto à metodologia de detecção de redundâncias estruturais em arquiteturas *Data Mesh*. Seu propósito é operacionalizar, de forma modular, interativa e reprodutível, as etapas de modelagem, comparação e validação de subgrafos que representam dependências entre tabelas.

Neste capítulo, adotamos o nome Isomera para o artefato construído. A escolha remete à ideia de isomeria e ao isomorfismo em grafos: estruturas que, embora nomeadas ou organizadas de modo distinto, compartilham equivalência estrutural. O termo "Isomera" combina "Iso", derivado de isomorfismo, com "mera", inspirado em isomeria na química, reforçando a noção de equivalência estrutural sob diferentes formas.

A concepção do *Isomera* surgiu da necessidade de um ambiente integrado que unificasse a modelagem matemática apresentada nos capítulos anteriores, fundamentada em álgebra linear e teoria dos grafos, com uma implementação prática e visual que permitisse executar e verificar, passo a passo, os algoritmos propostos. Assim, a ferramenta atua como elo entre o arcabouço teórico e a aplicação empírica da metodologia, viabilizando desde a geração do grafo e da matriz de adjacência até a comparação entre algoritmos e a validação supervisionada.

4.1 Arquitetura Geral da Ferramenta

A Figura 4.1 apresenta a arquitetura geral do *Isomera*, organizada em quatro blocos funcionais que podem ser agrupados em duas grandes camadas: *infraestrutura* e *aplicação*. Essa estrutura hierárquica foi projetada para garantir modularidade, escalabilidade e reprodutibilidade, assegurando que o sistema possa ser executado de forma consistente em qualquer ambiente computacional compatível. A arquitetura do *Isomera* reflete um princípio de design minimalista e científico: manter o núcleo da aplicação leve, portátil e autônomo, de modo que o processo de instalação, execução e análise seja inteiramente reproduzível por qualquer pesquisador sem dependências externas complexas.

Um dos aspectos mais relevantes do projeto é a ênfase na reprodutibilidade dos experimentos, princípio essencial na pesquisa científica moderna. O *Isomera* foi desenvolvido de modo a permitir que os mesmos resultados sejam obtidos em qualquer máquina que possua uma instalação padrão de Python (versão 3.11), independentemente de variações no sistema operacional. Isso é alcançado por meio de um controle automatizado de dependências, que garante que todas as bibliotecas utilizadas sejam instaladas localmente, com versões fixadas, dentro de um ambiente virtual isolado. Assim, a ferramenta elimina inconsistências que poderiam surgir em

execuções diferentes, além de permitir auditorias e comparações exatas entre experimentos, o que é crucial em estudos de desempenho de algoritmos e validações empíricas.

Outro ponto de destaque é a leveza operacional da aplicação. O *Isomera* foi intencionalmente projetado para ser eficiente em termos de recursos, tanto de processamento quanto de armazenamento. Todas as operações são realizadas localmente, sem necessidade de servidores, bancos de dados externos ou infraestrutura em nuvem. Os resultados, incluindo logs, matrizes de adjacência, métricas e grafos, são armazenados diretamente em disco, permitindo fácil acesso, replicação e transporte entre ambientes. Apesar de ser uma ferramenta científica com componentes gráficos e computacionais avançados, o sistema mantém um desempenho satisfatório mesmo em máquinas modestas, tornando-o ideal para uso em laboratórios acadêmicos, instituições de ensino e experimentos offline.

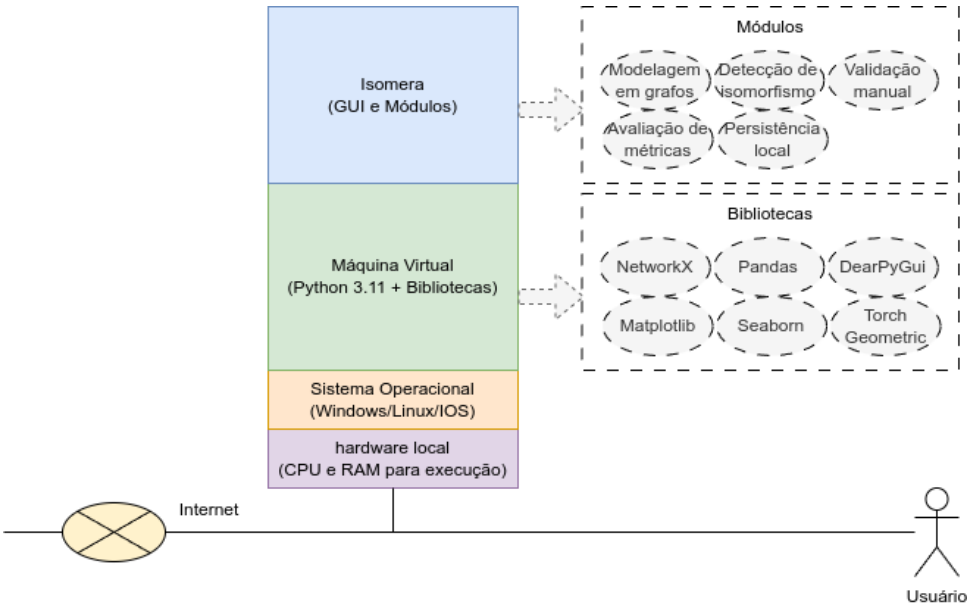


Figura 4.1: Arquitetura em blocos da ferramenta Isomera.

A camada de infraestrutura reúne os elementos responsáveis por prover o ambiente de execução: hardware local (CPU e memória), sistema operacional (Windows, Linux ou macOS) e a máquina virtual Python (versão 3.11), que encapsula as dependências. Como o *Isomera* instala automaticamente as bibliotecas necessárias antes da primeira execução, não exige hardware de alto desempenho nem armazenamento veloz. Toda a operação é realizada de forma local, com resultados gravados em disco e estrutura organizada em diretórios legíveis e padronizados. Embora a biblioteca `DearPyGui` disponha de renderização via GPU, essa funcionalidade ainda não foi habilitada na versão atual, pois os experimentos conduzidos nesta dissertação não demandaram processamento gráfico intensivo. Dessa forma, o foco recai sobre a compatibilidade, a leveza e a autonomia de execução: basta que o computador possua o Python instalado e uma conexão inicial com a internet para baixar as dependências, sendo todo o processamento subsequente independente de rede.

A camada de aplicação concentra o núcleo lógico e funcional da ferramenta, composto pe-

los módulos de modelagem em grafos, detecção de isomorfismo, validação manual e avaliação de métricas. Cada módulo representa uma etapa metodológica do processo de análise, atuando de forma encadeada dentro de um fluxo controlado pela interface gráfica. Essa camada também inclui um módulo transversal de persistência local, responsável por salvar artefatos, logs e resultados de cada execução, permitindo que o usuário retome, revise ou replique experimentos anteriores. A interface, construída com `DearPyGui`, centraliza toda a interação com o sistema, tornando o processo acessível, visual e intuitivo, sem necessidade de executar scripts ou editar código.

Em síntese, a arquitetura do *Isomera* foi desenhada para oferecer um equilíbrio entre rigor científico, eficiência computacional e usabilidade prática. Ela traduz os princípios da metodologia proposta em uma aplicação funcional, reprodutível e extensível, características essenciais para sua utilização tanto em ambientes acadêmicos quanto em projetos de pesquisa aplicada em engenharia de dados.

Camada de Infraestrutura

A arquitetura de execução foi projetada de modo a equilibrar simplicidade e robustez. A ferramenta utiliza o hardware local (CPU e memória RAM disponíveis) para processar as operações de modelagem, comparação e visualização. Não há necessidade de GPU ou processadores dedicados, uma vez que os algoritmos empregados são otimizados para execução em CPU e operam sobre conjuntos de dados moderados, típicos de análises estruturais. Além disso, a instalação é autônoma: na primeira execução, o sistema cria uma máquina virtual Python (versão 3.11) e instala automaticamente as bibliotecas necessárias, isolando-as do sistema principal. Esse mecanismo evita conflitos de dependências e assegura a reprodutibilidade dos resultados, um requisito fundamental para experimentos científicos controlados.

Tabela 4.1: Resumo da camada de infraestrutura da ferramenta *Isomera*.

Componente	**Descrição e função**
Hardware local	Utiliza CPU e RAM disponíveis. O desempenho é suficiente para execução local dos experimentos, sem exigir configurações avançadas.
Sistema operacional	Compatível com Windows, Linux e macOS, assegurando ampla portabilidade e independência de plataforma.
Máquina virtual Python 3.11	Ambiente principal de execução; instala e gerencia automaticamente todas as bibliotecas requeridas, garantindo isolamento e estabilidade.
Renderização GPU	Recurso oferecido pelo `DearPyGui`, mas ainda desativado nesta versão. A renderização via CPU é suficiente para as simulações conduzidas.
Dependências locais	Instaladas automaticamente na primeira execução. Após isso, o uso é totalmente offline, favorecendo reprodutibilidade e controle de versão.

Em síntese, essa camada foi pensada para manter o *Isomera* leve, portátil e de fácil replicação. Não há necessidade de servidores externos, bancos de dados dedicados ou configurações avançadas. O usuário precisa apenas ter o Python instalado e conexão inicial à internet para o download das bibliotecas. A partir daí, todo o funcionamento se torna local e independente, permitindo que os resultados sejam reproduzidos integralmente em qualquer ambiente acadêmico ou corporativo.

Camada da Aplicação

A camada de aplicação representa o núcleo lógico do *Isomera*. Nela, são implementadas as etapas metodológicas descritas nesta dissertação, desde a modelagem em grafos até a avaliação das métricas de desempenho. Cada módulo funciona como uma unidade funcional independente, mas totalmente integrada por meio da interface gráfica, que atua como um orquestrador visual de todo o fluxo computacional.

Os módulos foram concebidos com base nos princípios de clareza, rastreabilidade e modularidade. Isso significa que cada parte da aplicação corresponde a um pseudocódigo formal, apresentado nas seções seguintes, descrevendo de forma transparente as operações realizadas.

Tabela 4.2: Resumo dos módulos principais da camada de aplicação.

Módulo	**Descrição e responsabilidade**
Modelagem em grafos	Converte tabelas e dependências em grafos direcionados e matrizes de adjacência. Implementado com `NetworkX`, constitui a base analítica da ferramenta.
Detecção de isomorfismo	Executa os algoritmos VF2, Node Match e GNN sobre subgrafos, identificando padrões estruturais equivalentes entre diferentes domínios. É o núcleo de processamento da aplicação.
Validação manual	Exibe, via `DearPyGui`, os pares de subgrafos detectados, permitindo que o usuário confirme ou rejeite redundâncias. Essa validação supervisionada cria a base de verdade para as métricas.
Avaliação de métricas	Calcula acurácia (ACC), tempo de execução (ET) e frequência de sucesso (SF), consolidando as decisões do usuário e exportando os resultados em `CSV`.
Persistência local	Armazena logs, grafos, matrizes e resultados em formatos `JSON`, `CSV` e `PNG`, garantindo reprodutibilidade e rastreabilidade completa.

Essa camada foi desenhada para oferecer tanto poder analítico quanto transparência científica. O usuário pode acompanhar visualmente o avanço de cada etapa, inspecionar os grafos gerados, verificar os pares identificados e avaliar o desempenho dos algoritmos em tempo real. Além disso, a estrutura de persistência permite retomar experimentos anteriores, comparar resultados e documentar todas as etapas do processo de forma auditável.

Bibliotecas de Suporte

O *Isomera* fundamenta-se em um conjunto de bibliotecas científicas do ecossistema Python, amplamente reconhecidas por sua estabilidade, integração e suporte à pesquisa aplicada. Essas bibliotecas fornecem uma base técnica sólida que combina desempenho, legibilidade e fácil manutenção, permitindo que a ferramenta se mantenha compatível com práticas modernas de ciência de dados.

Tabela 4.3: Bibliotecas utilizadas na implementação do *Isomera*.

Biblioteca	**Função principal**
`NetworkX`	Estruturação, manipulação e análise de grafos direcionados e multigrafos.
`Pandas`	Organização de matrizes de adjacência, transformação de dados e exportação de resultados tabulares.
`DearPyGui`	Interface gráfica interativa com suporte a visualização de grafos e componentes dinâmicos.
`Matplotlib` e `Seaborn`	Criação de gráficos comparativos e visualizações analíticas de métricas e tempos de execução.
`Torch` e `Torch Geometric`	Implementação das redes neurais gráficas (GNN) e treinamento supervisionado de modelos baseados em embeddings estruturais.

A integração entre essas bibliotecas garante que o *Isomera* mantenha um equilíbrio entre precisão algorítmica e facilidade de uso. Todas as dependências foram escolhidas de forma criteriosa, considerando desempenho, suporte ativo da comunidade e compatibilidade entre versões, o que assegura longevidade e confiabilidade à ferramenta.

Interação com o Usuário

A interação com o usuário é inteiramente conduzida pela interface gráfica desenvolvida com a biblioteca `DearPyGui`. Essa interface foi projetada para oferecer uma experiência fluida, intuitiva e cientificamente rastreável, na qual cada ação do pesquisador é refletida diretamente no ambiente visual. O usuário é capaz de realizar todo o fluxo metodológico, desde o carregamento dos dados até a validação e exportação dos resultados, sem a necessidade de acessar o código-fonte ou executar comandos externos.

A Figura 4.2 e a Tabela 4.4 apresentam o fluxo geral de uso da ferramenta *Isomera*, que organiza as etapas de execução em uma sequência lógica, permitindo compreender a progressão dos dados desde a entrada até a análise final. O diagrama evidencia a modularidade do sistema e a integração entre os algoritmos e a interface, assegurando clareza na execução e reprodutibilidade dos experimentos.

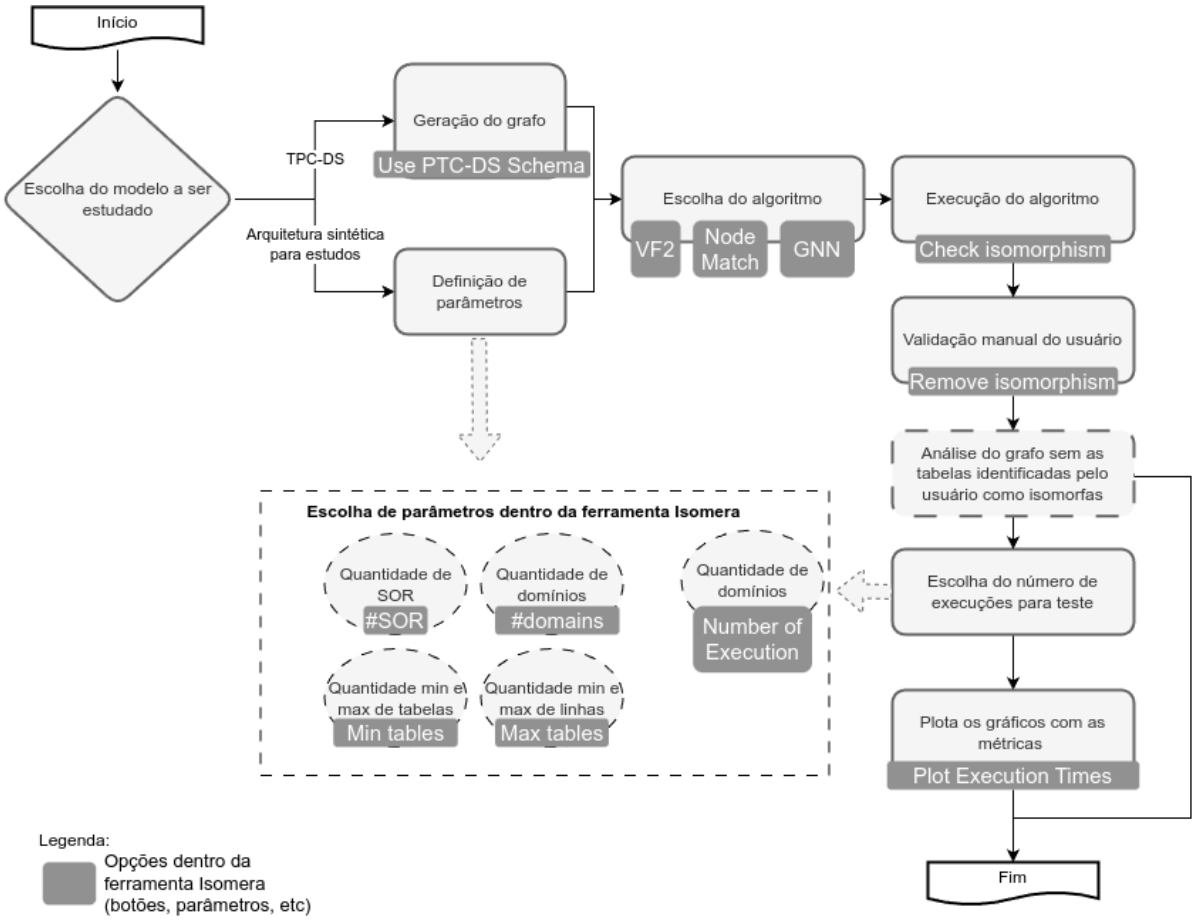


Figura 4.2: Fluxo de uso da ferramenta Isomera.

Tabela 4.4: Resumo das etapas do fluxo de uso da ferramenta *Isomera*.

Etapa	**Descrição resumida**
Carregamento ou geração	Importa arquivos `JSON`/`CSV` ou gera arquitetura sintética com parâmetros ajustáveis.
Modelagem em grafos	Converte tabelas e dependências em grafos direcionados e matrizes de adjacência (`NetworkX`).
Execução de algoritmos	Aplica VF2, Node Match ou GNN e registra os pares candidatos a isomorfismo.
Validação manual	Usuário confirma ou rejeita pares isomórficos; decisões alimentam a base de validação.
Cálculo das métricas	Calcula ACC, ET e SF e exibe resultados em gráficos comparativos.
Exportação e persistência	Salva grafos, métricas e logs em `CSV`, `JSON` e `PNG`.

Durante a execução, o *Isomera* registra automaticamente todos os parâmetros definidos pelo usuário, as ações realizadas e os resultados gerados em cada etapa do processo. Esses registros formam um histórico completo e rastreável de execução, assegurando que qualquer experimento possa ser reproduzido ou auditado posteriormente com precisão. O mecanismo de registro inclui não apenas os valores de entrada e saída, mas também metadados como data e hora de execução, versão das bibliotecas utilizadas e configurações do ambiente computacional. Essa documentação automática elimina a necessidade de anotações manuais e garante que toda informação relevante para a reprodutibilidade esteja preservada de forma estruturada e consultável.

A leveza da ferramenta também é um aspecto central do seu design: todas as operações são processadas localmente, sem dependência de servidores externos, o que garante independência, privacidade e controle total sobre os dados. Mesmo com essa simplicidade operacional, o *Isomera* mantém desempenho eficiente em CPU e armazenamento local, tornando-se adequado tanto para uso em pesquisa aplicada quanto em atividades de ensino. Sua modularidade e transparência reforçam o caráter científico da aplicação, permitindo que resultados sejam facilmente replicados, comparados e compartilhados.

Além disso, a organização interna dos arquivos gerados segue uma estrutura padronizada e intuitiva, facilitando a navegação e a posterior análise dos dados produzidos. Cada execução cria automaticamente um conjunto de artefatos que incluem grafos em formato visual, matrizes numéricas, métricas consolidadas e logs detalhados, todos armazenados em diretórios específicos que refletem a hierarquia lógica do experimento. Essa estruturação permite que o pesquisador localize rapidamente qualquer informação relevante e mantenha organizado o histórico de múltiplas execuções, contribuindo para a manutenção da integridade científica do trabalho e para a clareza na apresentação dos resultados.

A interface do *Isomera* representa a materialização prática da metodologia proposta, unindo modelagem teórica e experimentação visual em um único ambiente. Desenvolvida integralmente com a biblioteca `DearPyGui`, ela oferece uma experiência interativa e intuitiva, que conduz o usuário por todas as etapas, da geração da arquitetura à validação e análise dos resultados, sem necessidade de interação direta com o código ou execução de comandos externos. Essa abordagem garante acessibilidade e reprodutibilidade, permitindo que o processo de detecção e validação de redundâncias estruturais seja realizado de forma simples, guiada e transparente. O layout da interface segue a mesma lógica sequencial da metodologia descrita na dissertação, apresentando os módulos de forma ordenada: modelagem, detecção, validação e métricas, o que facilita a navegação e o entendimento do fluxo completo de análise.

4.2 Interface e Visualização da Ferramenta

A seguir, são apresentadas as principais telas da ferramenta *Isomera*, em sequência lógica, acompanhando o mesmo fluxo metodológico proposto nesta dissertação. As imagens foram organizadas para que o leitor possa compreender passo a passo como a ferramenta funciona na prática, desde a geração do grafo até a visualização final do resultado consolidado sem redundâncias estruturais.

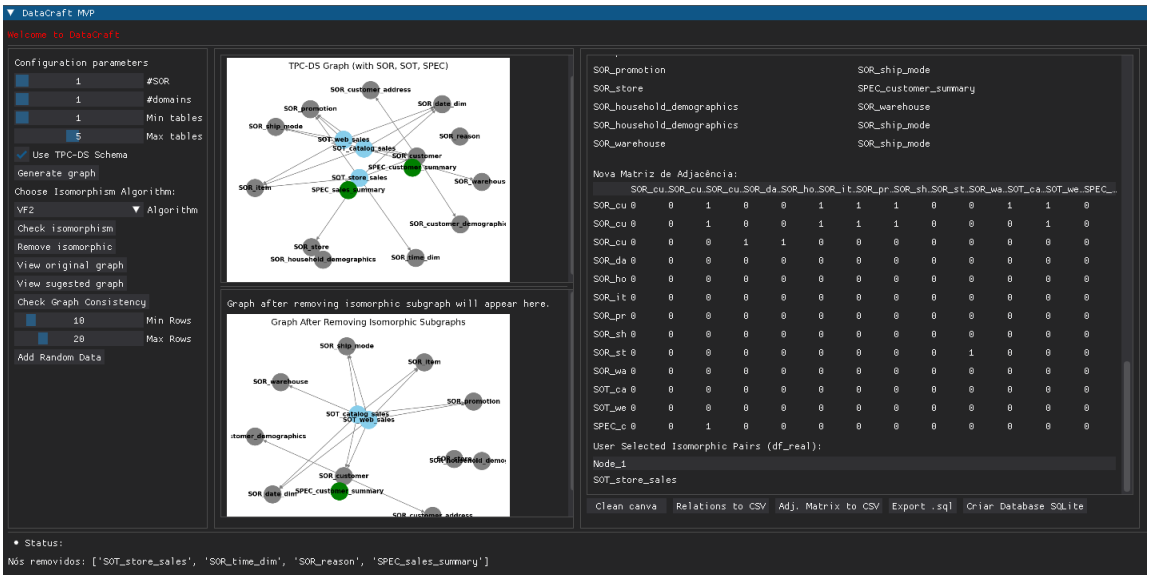


Figura 4.3: Visão geral da ferramenta Isomera (DearPyGui).

A Figura 4.3 apresenta a visão geral da ferramenta *Isomera*, exibindo o conjunto completo de módulos disponíveis na interface gráfica. Nesta tela, o usuário tem acesso aos principais recursos da aplicação, modelagem de grafos, execução dos algoritmos de detecção de isomorfismo, validação manual e análise de métricas. Toda a interação é visual e intuitiva, permitindo a configuração de experimentos, o acompanhamento do progresso e a exportação dos resultados, tudo dentro de um único ambiente gráfico.

A interface foi projetada de modo a guiar o pesquisador em uma jornada completa: do carregamento dos dados até a análise dos resultados. Essa abordagem promove um uso acessível mesmo para usuários sem familiaridade com programação, tornando o *Isomera* uma ferramenta didática e experimental ao mesmo tempo, adequada tanto para pesquisa científica quanto para ensino de conceitos ligados à teoria dos grafos e às arquiteturas *Data Mesh*.

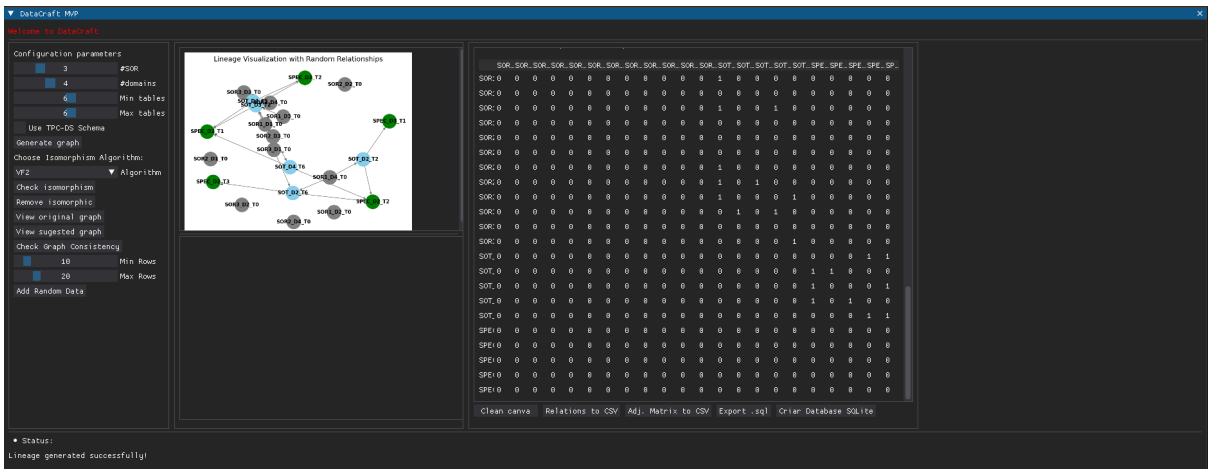


Figura 4.4: Modelagem como grafo e matriz de adjacência.

A Figura 4.4 marca o início do exemplo prático apresentado neste capítulo. Aqui, o usuário define os parâmetros de configuração do experimento, podendo importar uma arquitetura real (como o benchmark TPC-DS) ou gerar automaticamente uma arquitetura sintética diretamente pela interface. Os campos de entrada permitem controlar o número de tabelas, a quantidade de domínios e os limites mínimo e máximo de tabelas e linhas por domínio, parâmetros que determinam a complexidade e densidade do grafo a ser estudado.

No exemplo exibido, os seguintes valores foram configurados para o experimento: 3 tabelas de origem (*SOR*), 4 domínios, 6 tabelas por domínio (mínimo e máximo), 10 linhas mínimas e 20 linhas máximas por tabela. Esses parâmetros definem o tamanho e a conectividade inicial da arquitetura simulada, influenciando diretamente a estrutura final do grafo e o comportamento dos algoritmos de detecção.

À direita da interface, observa-se a matriz de adjacência correspondente ao grafo gerado. Ela representa, em formato numérico, as relações de dependência entre tabelas, permitindo ao usuário verificar visualmente se as conexões foram geradas corretamente. Essa visualização reforça o princípio de transparência da ferramenta, facilitando a compreensão de como cada aresta (transformação) e cada nó (tabela) se traduzem em relações matemáticas dentro da matriz.

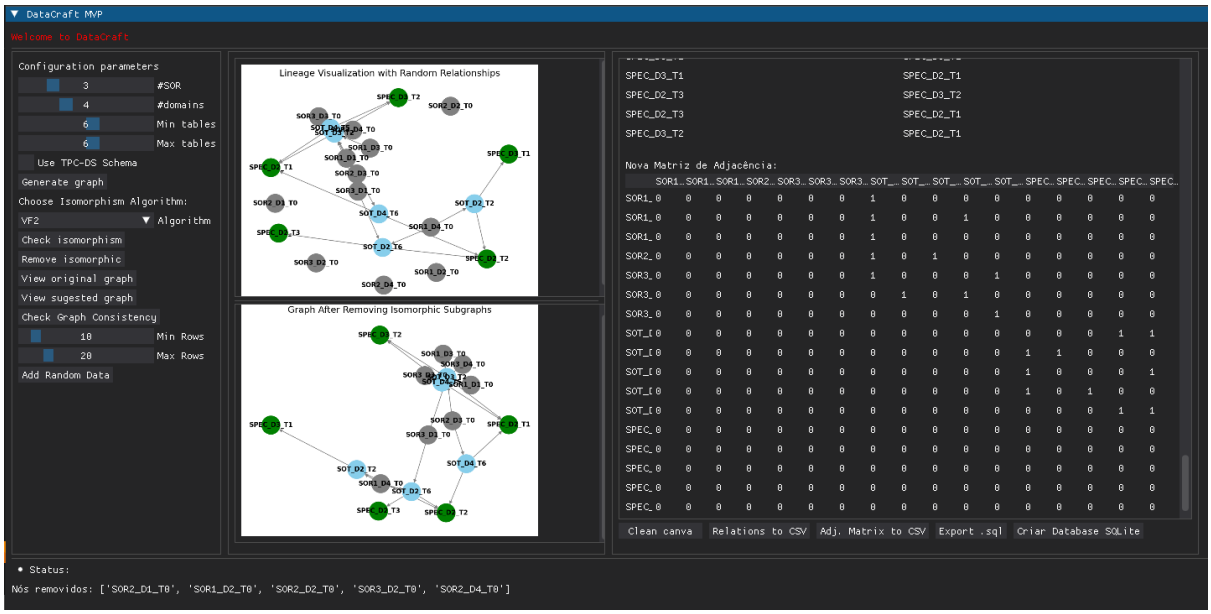


Figura 4.5: Detecção de isomorfismo (VF2).

Na Figura 4.5, o processo de detecção de isomorfismo é iniciado. O usuário seleciona o algoritmo desejado, neste caso, o VF2, e o sistema executa automaticamente a comparação entre os subgrafos gerados a partir da arquitetura modelada. Durante essa execução, o *Isomera* analisa cada par de subestruturas e identifica padrões equivalentes de conectividade, armazenando os resultados intermediários em memória e em disco. Essa etapa corresponde ao núcleo computacional da ferramenta, onde a teoria dos grafos é aplicada diretamente à prática para detectar redundâncias estruturais.

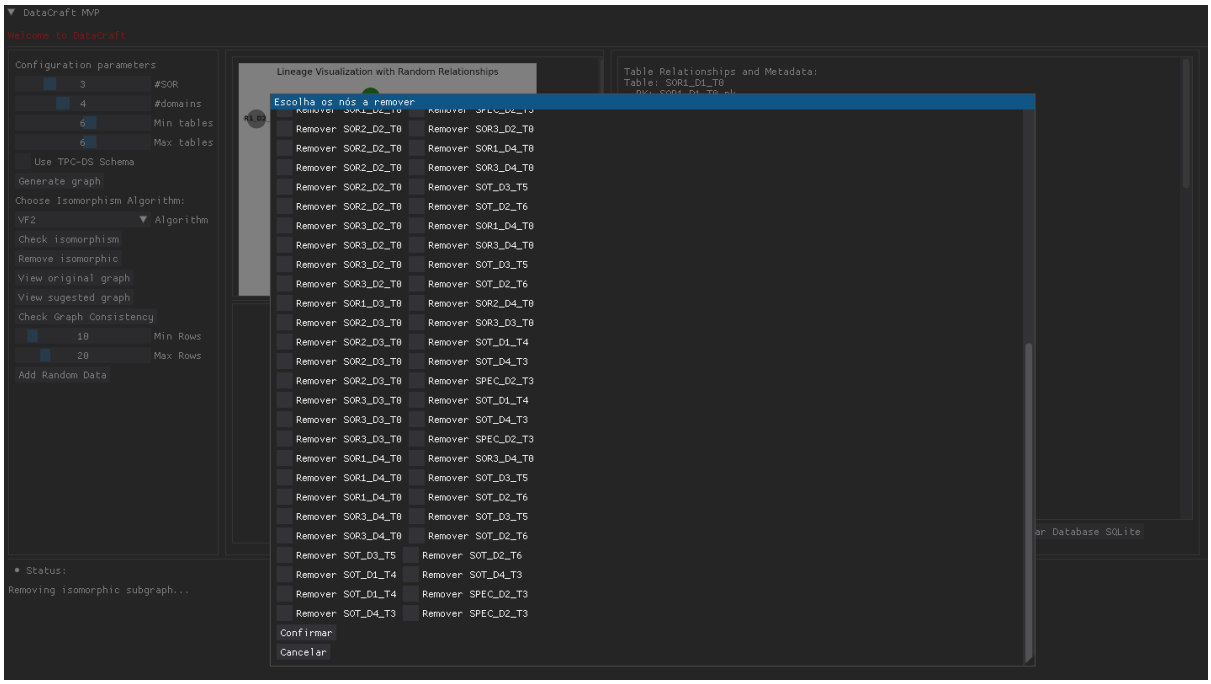


Figura 4.6: Validação manual de pares redundantes.

A Figura 4.6 apresenta a tela de validação manual, etapa essencial do processo. Após a execução dos algoritmos, os pares de subgrafos identificados como equivalentes são exibidos para inspeção visual do usuário. A partir dessa interface, o analista decide se as correspondências encontradas representam redundâncias reais ou apenas semelhanças formais. Essa decisão alimenta automaticamente a base de validação, que é usada nas próximas etapas para o cálculo das métricas de desempenho.

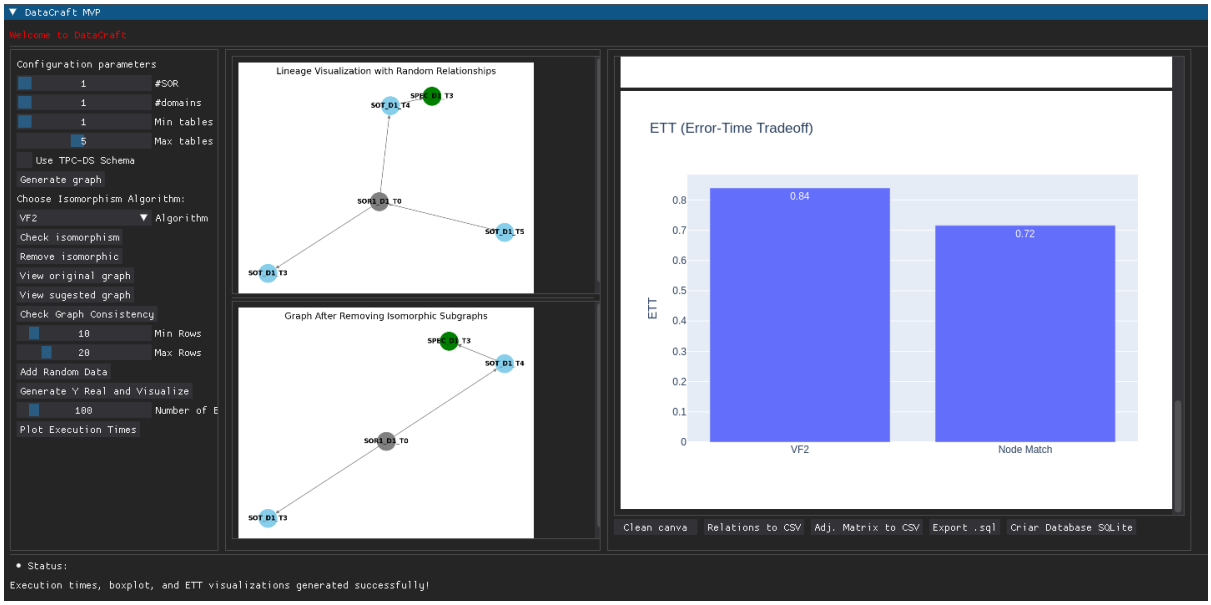


Figura 4.7: Comparação entre algoritmos (métrica SF).

A Figura 4.7 mostra o painel de métricas, responsável por consolidar os resultados experimentais. Aqui, o sistema apresenta indicadores quantitativos como acurácia (ACC), tempo de execução (ET) e frequência de sucesso (SF). Essas métricas permitem comparar o desempenho dos algoritmos (VF2, Node Match e GNN) sob os mesmos parâmetros experimentais, identificando qual abordagem apresenta o melhor equilíbrio entre precisão e eficiência.

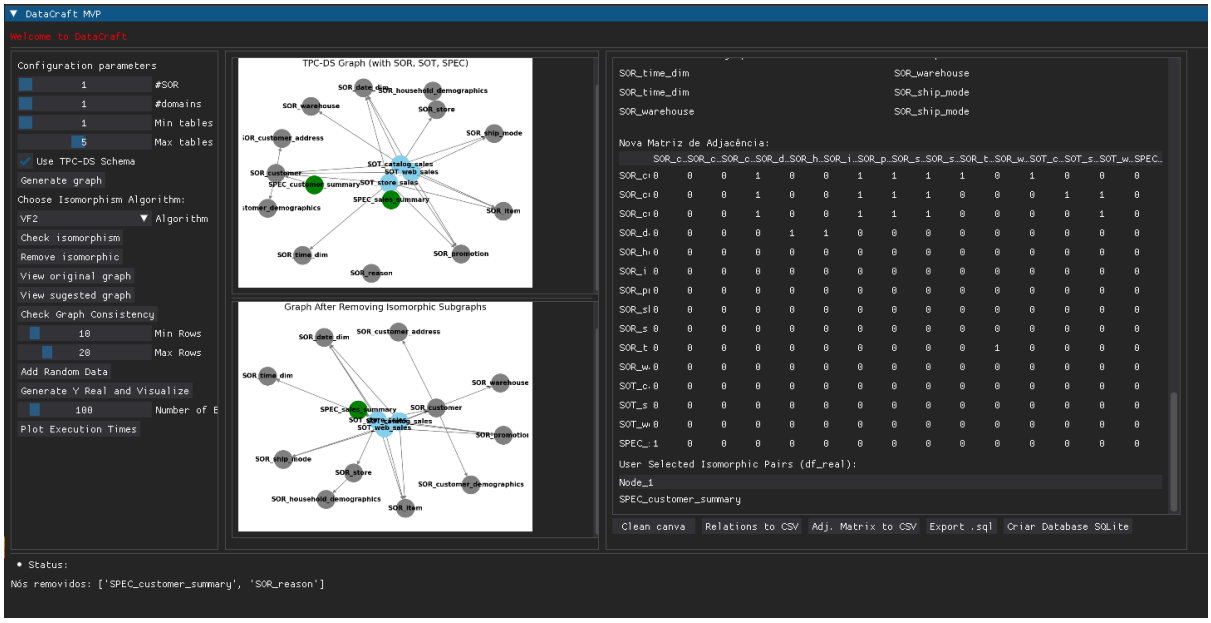


Figura 4.8: Grafo final sem tabelas duplicadas.

Por fim, a Figura 4.8 apresenta o resultado consolidado do experimento: um grafo orientado representando a arquitetura final sem redundâncias estruturais. Essa visualização sintetiza todo o processo realizado, da modelagem inicial à validação, e evidencia como a aplicação dos algoritmos de isomorfismo contribui para identificar e remover duplicidades de forma sistemática. O grafo resultante preserva as relações válidas entre tabelas, ao mesmo tempo em que elimina estruturas equivalentes, resultando em uma representação mais enxuta, coerente e otimizada da arquitetura *Data Mesh*.

4.3 Detalhamento da Solução

A ferramenta *Isomera* foi inteiramente desenvolvida em Python, escolhida por sua ampla adoção em ciência de dados, legibilidade e ecossistema científico consolidado. Essa escolha também favoreceu a integração entre os conceitos matemáticos da metodologia e a implementação de algoritmos de grafos e aprendizado supervisionado.

As principais bibliotecas utilizadas são descritas a seguir.

4.3.1 Escolha da Biblioteca NetworkX

A biblioteca `NetworkX` foi selecionada como núcleo da modelagem e análise estrutural dos grafos que representam as arquiteturas de dados. Segundo (SOUZA; GUEDES, 2023), a `NetworkX` apresenta o melhor equilíbrio entre flexibilidade, integração e estabilidade dentre as principais bibliotecas analisadas (Gephi, Pajek, iGraph e NetworkX).

Enquanto Gephi e Pajek são voltadas à visualização, e iGraph oferece maior desempenho porém menor flexibilidade, a `NetworkX` destacou-se pela capacidade de representar grafos direcionados, multigrafos e estruturas com auto-laços, além de conter implementações estáveis de algoritmos como busca em profundidade, clusterização e detecção de isomorfismos. A ferramenta aproveita especialmente o algoritmo VF2 nativo da biblioteca, amplamente referenciado na literatura para verificação de equivalências estruturais.

4.3.2 Uso de DearPyGui para a Interface Gráfica

A interface gráfica foi construída com a biblioteca `DearPyGui`, escolhida por seu desempenho gráfico acelerado via GPU e suporte a interações em tempo real. De acordo com Langner *et al.* (LANGNER *et al.*, 2025), o `DearPyGui` é particularmente indicado para aplicações científicas que exigem visualizações de alta performance, como a manipulação de grafos e matrizes de grande porte.

Sua arquitetura baseada em renderização direta permite que o usuário interaja com cada etapa da metodologia, geração de grafos, execução de algoritmos e validação manual, sem perda de desempenho. Além disso, a biblioteca permite criar painéis interativos para comparação de métricas e exportação visual dos resultados.

4.3.3 Módulo de Modelagem em Grafos

O primeiro módulo da ferramenta é responsável por transformar tabelas e suas dependências em uma estrutura de grafo direcionado, com vértices representando entidades (SOR, SOT e SPEC) e arestas representando as relações de derivação. Essa modelagem formal constitui a base de toda a metodologia, pois converte o problema de redundância estrutural em um problema de equivalência entre subgrafos.

O processo é automatizado por meio da função representada no pseudocódigo da Figura 4.9, que cria o grafo, define as conexões e gera a matriz de adjacência correspondente. A saída desse módulo é utilizada diretamente na etapa de detecção de isomorfismo.

Algorithm 1 Generate Graph Structure with Mesh-Adaptive TPC-DS Option

Input: Use_TPCDS (boolean), number of domains D, number of SOR nodes S, min/max tables $(T_{\min}, T_{\max})$
Output: Directed graph G
if Use_TPCDS = True **then**
    Load adapted TPC-DS graph
    Rename tables with prefixes `SOR_`, `SOT_`, `SPEC_`
    Add domain-specific node grouping
    Transform schema relationships into graph edges
    return G
end if
Initialize directed graph G
Initialize lists: $sor\_nodes, sot\_nodes, spec\_nodes$
for $d = 1$ to D **do**
    for $s = 1$ to S **do**
        Create node $SOR\_s\_D_d$
        Add node to G
    end for
end for
$T \leftarrow \text{Random}(T_{\min}, T_{\max})$
for $t = 1$ to T **do**
    Select random domain d
    Create node $SOT\_D_d\_T_t$
    Add node to G
    Connect to random SOR nodes
end for
for $t = 1$ to $\text{Random}(1, T)$ **do**
    Select random SOT node
    Create node $SPEC\_T_t$
    Add node to G
    Connect to random SOT nodes
end for
return G

Figura 4.9: Pseudocódigo — módulo de modelagem em grafos.

Esse módulo permite tanto a importação de dados reais (ex.: TPC-DS, PostgreSQL, MySQL) quanto a geração de arquiteturas sintéticas, configuradas com parâmetros de complexidade definidos pelo usuário (número de domínios, número de tabelas e densidade de conexões). A matriz de adjacência gerada representa a estrutura formal que será utilizada nas comparações posteriores.

4.3.4 Módulo de Detecção de Isomorfismo

O segundo módulo executa o núcleo computacional da ferramenta, aplicando algoritmos de detecção de isomorfismo sobre os subgrafos extraídos. Essa etapa identifica pares de estruturas topologicamente equivalentes, revelando redundâncias entre diferentes domínios da arquitetura.

O pseudocódigo da Figura 4.10 apresenta o fluxo de execução deste módulo. Nele, o usuário seleciona o algoritmo desejado, VF2, Node Match ou GNN, e o sistema executa as comparações, registrando os pares isomórficos e os resultados parciais.

Algorithm 2 Isomorphism Detection with Degree Filtering

Input: Directed graph G, Algorithm $A \in \{\text{VF2, Node Match}\}$
Output: List of isomorphic subgraph pairs P
Initialize empty list $P \leftarrow [\,]$
Initialize list of subgraphs $S \leftarrow [\,]$
for each node n in G **do**
    $N \leftarrow$ successors of n
    $g_n \leftarrow$ subgraph induced by $\{n\} \cup N$
    $d_n \leftarrow$ degree of n
    Append (n, g_n, d_n) to S
end for
for each pair (i, j) where $i < j$ in S **do**
    $(n_i, g_i, d_i) \leftarrow S[i]$
    $(n_j, g_j, d_j) \leftarrow S[j]$
    if $d_i \neq d_j$ **then**
        continue
    end if
    if $A = \text{VF2}$ **then**
        if $is\_isomorphic(g_i, g_j)$ **then**
            Append (n_i, n_j) to P
        end if
    else if $A = \text{Node Match}$ **then**
        if $is\_isomorphic(g_i, g_j,\ node\_match = \lambda x, y : x == y)$ **then**
            Append (n_i, n_j) to P
        end if
    end if
end for
return P

Figura 4.10: Pseudocódigo — módulo de detecção de isomorfismo.

O módulo é implementado de forma extensível, permitindo que novos algoritmos sejam incorporados futuramente. Atualmente, o *Isomera* combina a precisão do algoritmo VF2, a eficiência híbrida do Node Match e o poder de generalização das redes neurais gráficas (GNNs),

possibilitando uma análise balanceada entre desempenho e exatidão.

4.3.5 Módulo de Validação Manual

O terceiro módulo incorpora a intervenção humana no processo de detecção, garantindo que as correspondências identificadas pelos algoritmos sejam analisadas em termos de contexto e semântica. A validação manual é realizada diretamente na interface, onde o usuário pode confirmar ou rejeitar cada par de subgrafos sugerido como isomórfico.

A lógica dessa etapa está representada no pseudocódigo da Figura 4.11. Nele, o sistema apresenta os pares detectados, registra as decisões do usuário e atualiza automaticamente a base de validação que servirá para o cálculo das métricas.

Algorithm 3 User-Guided Isomorphism Validation

Input: List of detected pairs P
Output: Validated pairs P_{valid}
Initialize empty list P_{valid}
for each (A, B) in P **do**
    Display A, B to user
    $decision \leftarrow$ GetUserInput("Are these graphs isomorphic? [Y/N]")
    if $decision == Y$ **then**
        Append $(A, B, Isomorphic)$ to P_{valid}
    else
        Append $(A, B, Not\ Isomorphic)$ to P_{valid}
    end if
end for
return P_{valid}

Figura 4.11: Pseudocódigo — módulo de validação manual.

Essa etapa garante que a análise final considere não apenas a equivalência estrutural, mas também a equivalência funcional e semântica das estruturas, o que é fundamental para evitar falsos positivos em ambientes de dados complexos.

4.3.6 Módulo de Avaliação de Métricas

O quarto módulo é responsável por consolidar quantitativamente os resultados obtidos nas etapas anteriores. Com base nas decisões registradas pelo usuário, o sistema calcula automaticamente métricas de desempenho como acurácia (ACC), tempo de execução (ET) e frequência de sucesso (SF), apresentando os resultados de forma visual e comparativa.

A Figura 4.12 mostra o pseudocódigo que representa o funcionamento interno deste módulo.

Algorithm 5 Metrics Evaluation for Isomorphism Detection

Input: Algorithm A, Mode $\in$ {GeneratedGraph, TPCDS}, Scenario Parameters (D, S)
Output: Metrics $(\text{ACC}, \text{ER}, \text{ET}, \text{ETT})$
Initialize $TP \leftarrow 0$, $TN \leftarrow 0$, $FP \leftarrow 0$, $FN \leftarrow 0$
Initialize $ET \leftarrow 0$
for $i = 1$ to 1000 **do**
    if Mode = GeneratedGraph **then**
        Generate graph G_i with D domains and S SOR tables
        $Pair_{real} \leftarrow$ obtained via user validation
    else
        Load TPC-DS benchmark graph G_i
        $Pair_{real} \leftarrow$ predefined benchmark mapping
    end if
    Start timer
    $Pair_{predict} \leftarrow \text{RunAlgorithm}(A, G_i)$
    Stop timer and record $ET_i \leftarrow t^i_{\text{end}} - t^i_{\text{start}}$
    Update $ET \leftarrow ET + ET_i$
    Compare $Pair_{predict}$ and $Pair_{real}$ to update TP, TN, FP, FN
end for
$ET \leftarrow \frac{ET}{1000}$ ▷ Average execution time over 1000 runs
$ACC \leftarrow \frac{TP+TN}{TP+TN+FP+FN}$
$ER \leftarrow 1 - ACC$
$ETT \leftarrow \frac{1-ER}{ET}$
return (ACC, ER, ET, ETT)

Figura 4.12: Pseudocódigo — módulo de avaliação de métricas.

O módulo também exporta os resultados em formatos reutilizáveis (`CSV`, `JSON`, `PNG`), assegurando a reprodutibilidade dos experimentos. Esses dados podem ser reaproveitados para estudos comparativos, auditorias ou replicações futuras da pesquisa.

4.3.7 Módulo de Persistência Local

Por fim, o módulo de persistência garante que todas as informações processadas, grafos, matrizes, logs e resultados, sejam salvas localmente no ambiente de execução. Esse design elimina dependências externas, reforçando a filosofia de reprodutibilidade e controle total dos dados. Todos os artefatos podem ser carregados novamente em execuções futuras, permitindo continuidade e rastreabilidade completa dos experimentos.

CAPÍTULO 5

Estudo de Caso e Avaliação Experimental

Este capítulo apresenta os experimentos realizados para avaliar a eficácia do método proposto na detecção de redundâncias estruturais em arquiteturas *Data Mesh*, por meio da comparação entre três algoritmos de isomorfismo de grafos: VF2, Node Match e Graph Neural Network (GNN). A avaliação se deu com base em métricas de acurácia, tempo de execução e frequência de acertos, conforme definido na Seção 2.5.

5.1 Transformação do TPC-DS em Arquitetura em Grafo

Para fornecer uma base realista de validação, o benchmark analítico TPC-DS, mencionado na Seção 1.3 (NAMBIAR; POESS, 2006), foi reinterpretado como uma arquitetura distribuída baseada em domínios. O TPC-DS contém mais de 20 tabelas com relacionamentos complexos e consultas analíticas típicas de ambientes empresariais.

Embora o TPC-DS tenha sido originalmente concebido para avaliar arquiteturas analíticas centralizadas (por exemplo, *data warehouses*), sua escolha nesta dissertação justifica-se por três razões: (i) disponibilidade pública e ampla aceitação acadêmica, o que favorece a reprodutibilidade; (ii) riqueza estrutural de esquemas e relacionamentos, suficiente para emular dependências e linhagens entre entidades; e (iii) possibilidade de agrupamento funcional em domínios, permitindo sua adaptação ao contexto descentralizado do *Data Mesh*.

Para alinhá-lo a esse contexto, as tabelas foram particionadas em domínios temáticos e organizadas segundo as camadas SOR, SOT e SPEC, o que viabiliza a análise de redundâncias estruturais entre domínios sob um protocolo controlado.

Como limitação, reconhece-se que o TPC-DS não contempla aspectos organizacionais e de governança típicos de um ambiente *Data Mesh* real; ainda assim, sua estrutura fornece uma base adequada para o estudo comparativo de algoritmos sob topologias complexas.

Nesta dissertação, as tabelas do benchmark TPC-DS foram agrupadas em cinco domínios semânticos de negócio, cada um representando uma área funcional distinta da arquitetura analítica. Esse agrupamento foi definido com base na afinidade temática e no uso conjunto das tabelas, resultando na seguinte estrutura de domínios:

- D1 – Customer: engloba dados de clientes, endereços e demografia, com tabelas como SOR_customer, SOR_customer_address e SOR_customer_demographics;
- D2 – Store: concentra informações geográficas e organizacionais, como SOR_store, SOR_region, SOR_nation e SOR_call_center;

- D3 – Catalog: abrange produtos e campanhas, com tabelas como SOR_item, SOR_promotion e SOR_reason;
- D4 – Time: reúne dados temporais de apoio à análise, como SOR_date_dim e SOR_time_dim;
- D5 – Warehouse: trata da estrutura logística, incluindo SOR_warehouse, SOR_ship_mode e SOR_income_band.

Cada domínio foi modelado com três tipos de tabelas, em conformidade com os princípios da arquitetura *Data Mesh*:

- SOR: representam as tabelas brutas extraídas diretamente do TPC-DS, utilizadas como ponto de partida para a modelagem;
- SOT: são derivações intermediárias obtidas por meio de transformações locais dentro de cada domínio, estruturando os dados para consumo interno;
- SPEC: compõem visões analíticas especializadas, que sintetizam informações a partir de múltiplos domínios ou camadas, promovendo o reuso e a integração semântica.

A separação hierárquica entre SOR, SOT e SPEC possibilitou a geração de grafos dirigidos em que os nós representam entidades (tabelas) e as arestas codificam transformações, derivações ou junções entre elas.

A construção desses grafos foi parametrizada quanto ao número de tabelas e domínios, permitindo a geração automática de diferentes cenários experimentais (ver código na Seção A.1).

A Figura 5.1 ilustra essa transformação do TPC-DS em um grafo de arquitetura de dados, estruturado para fins de análise de redundâncias e detecção de padrões isomorfos.

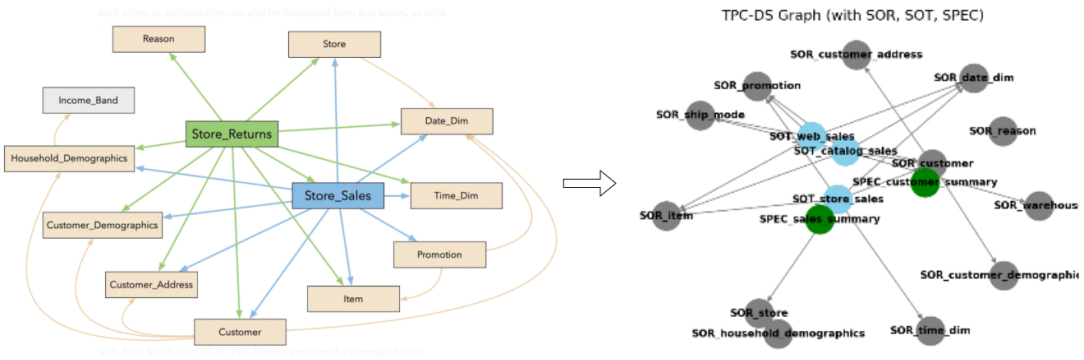


Figura 5.1: Transformação do TPC-DS em um grafo orientado por SOR, SOT e SPEC

Com essa estrutura, tornou-se possível simular pipelines reais de dados, com diferentes graus de acoplamento e redundância estrutural, possibilitando a aplicação dos algoritmos de isomorfismo para avaliar duplicações, similaridades e padrões recorrentes.

5.2 Modelagem de Arquiteturas Sintéticas com Isomera

Para complementar a avaliação com casos reais, foi desenvolvida na ferramenta a possibilidade de gerar cenários sintéticos com diferentes parâmetros e embasados no benchmark reinterpretado do TPC-DS. Essa funcionalidade permite simular arquiteturas com níveis variados de complexidade estrutural, controlando aspectos como densidade de conexões, número de entidades e distribuição entre domínios, o que viabiliza a análise sistemática do comportamento dos algoritmos sob diferentes condições operacionais.

A geração sintética é essencial para validar a metodologia proposta sob condições controladas, uma vez que ambientes reais raramente oferecem a diversidade estrutural e o controle necessário para uma análise comparativa robusta. Por meio dessa funcionalidade, torna-se possível explorar desde arquiteturas simples e isoladas, com poucos domínios e baixa conectividade, até cenários densamente interconectados que emulam ambientes corporativos complexos, nos quais múltiplos domínios compartilham dependências e transformações. A parametrização é realizada de forma flexível, permitindo ao usuário definir as seguintes características estruturais da arquitetura a ser gerada:

- Número de domínios (de 1 a 5);
- Quantidade de tabelas SOR por domínio;
- Faixa de variação para o número de tabelas SOT e SPEC;
- Probabilidade de conexões inter e intra-domínios.

A partir desses parâmetros, o Isomera constrói um grafo direcionado com as seguintes regras: SOR são os nós de origem em cada domínio, SOT conectam-se aleatoriamente a SOR, SPEC derivam de SOT, podendo conectar-se entre domínios.

O resultado é uma matriz de adjacência e uma tabela de linhagem totalmente reprodutível, permitindo simular diferentes níveis de sobreposição estrutural. A Tabela 5.1 apresenta os parâmetros utilizados e a simulação de múltiplos cenários com diferentes complexidades.

Tabela 5.1: Cenários de Experimento e suas Características.

Cenário	SOR	Domínios	Descrição
1	2	1 a 5	Arquiteturas leves com baixa duplicação
2	4	1 a 5	Sobreposição moderada entre domínios
3	8	1 a 5	Complexidade média com redundâncias esperadas
4	16	1 a 5	Arquiteturas densas com alto grau de duplicidade

A Figura 5.2 mostra exemplos visuais de grafos gerados a partir do TPC-DS e dos cenários sintéticos. Esses grafos foram posteriormente utilizados como entrada para os algoritmos de comparação estrutural.

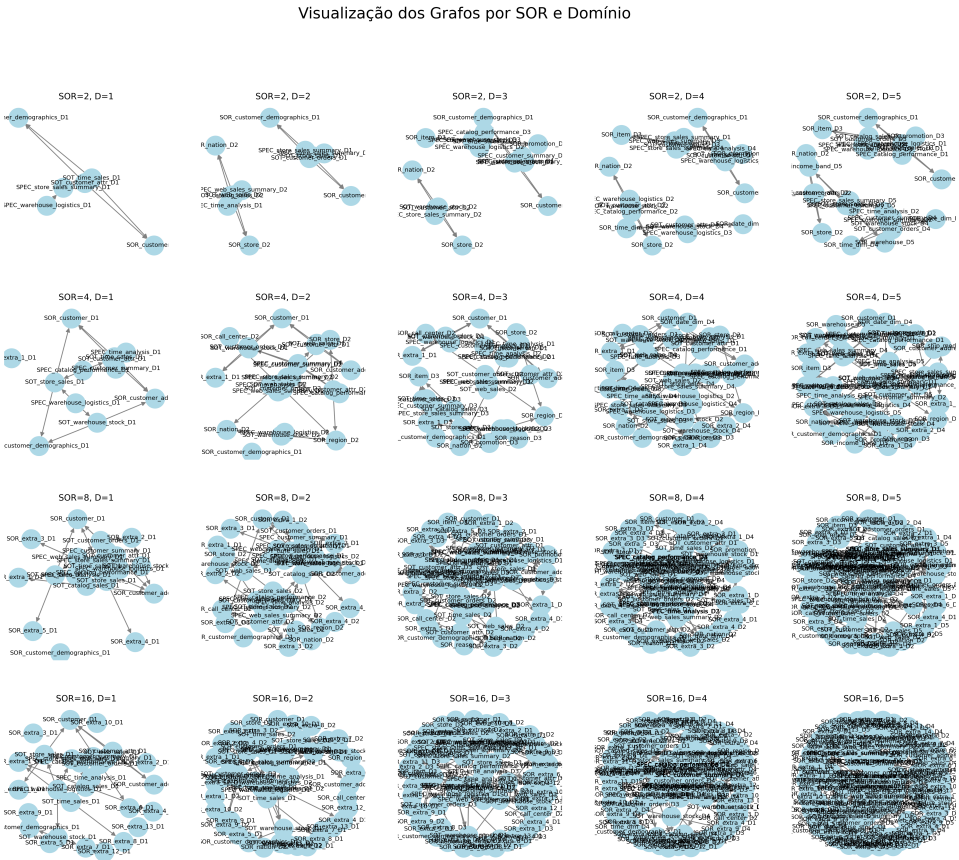


Figura 5.2: Exemplos de grafos gerados por cada cenário na ferramenta Isomera a partir de benchmarks

5.3 Execução dos Testes

Com os grafos gerados, tanto os oriundos da adaptação do TPC-DS quanto os sintetizados artificialmente, foi realizada uma etapa fundamental de validação manual, na qual cada subgrafo representado visualmente na Figura 5.2 foi analisado para identificar pares isomorfos reais. Esta identificação prévia foi essencial para estabelecer um conjunto de referência (ground truth) que pudesse ser utilizado nas métricas de benchmarking.

Ao adotar essa abordagem, o processo de validação pôde ser automatizado nas execuções subsequentes, sem a necessidade de intervenção humana para cada novo teste. Isso garantiu maior reprodutibilidade, robustez na comparação e acelerou significativamente o ciclo de testes da ferramenta *Isomera*.

Os testes foram então executados em ambiente controlado, dentro de uma máquina virtual com recursos computacionais padronizados, e cada algoritmo foi rodado 1000 vezes consecutivas por cenário. Esse número elevado de execuções teve como objetivo mitigar variações naturais de tempo decorrentes de sobrecarga momentânea de CPU, memória ou processos em background, e garantir que o tempo final analisado representasse a média estável de execução.

Para a análise comparativa, foram selecionados três algoritmos: o VF2, o Node Match e

a Graph Neural Network (GNN). A escolha desses algoritmos permite avaliar diferentes abordagens para o problema de detecção de isomorfismo, desde métodos exatos e determinísticos até técnicas baseadas em aprendizado de máquina, possibilitando uma análise abrangente de desempenho sob múltiplas perspectivas, incluindo precisão, velocidade e capacidade de generalização.

A execução seguiu os seguintes passos sistemáticos:

1. Definição do conjunto de pares de subgrafos extraídos dos cenários sintetizados;
2. Aplicação dos algoritmos sobre cada grafo;
3. Registro dos tempos de execução, predições e comparação com os rótulos reais (definidos previamente na validação manual);
4. Cálculo das métricas apresentadas na Seção 2.5: acurácia (ACC), tempo de execução (ET) e frequência de acertos (SF).

A avaliação experimental foi estruturada em dois estudos de caso sequenciais, cada um com objetivos específicos e complementares. Essa divisão permitiu uma análise progressiva e incremental do desempenho dos algoritmos, partindo de abordagens clássicas e evoluindo para técnicas baseadas em aprendizado de máquina.

O Estudo de Caso I focou na comparação entre os algoritmos VF2 e Node Match, os quais estavam inicialmente implementados na primeira versão do *Isomera*. Os resultados dessa primeira fase revelaram limitações de acurácia em cenários de maior complexidade estrutural, especialmente quando o número de domínios e tabelas aumentava significativamente, o que motivou a busca por abordagens mais sofisticadas capazes de capturar padrões estruturais mais complexos.

Em resposta a essas limitações, o Estudo de Caso II introduziu o uso de uma Graph Neural Network (GNN), treinada sobre os mesmos cenários com pares previamente rotulados. Essa introdução marcou uma evolução metodológica importante, ao alavancar técnicas de aprendizado de máquina para predição supervisionada de isomorfismo, permitindo que o modelo capturasse padrões estruturais que algoritmos tradicionais não conseguiam generalizar. Os resultados obtidos nas duas fases são apresentados nas seções seguintes, permitindo uma análise crítica da aplicabilidade de cada abordagem em ambientes distribuídos e heterogêneos.

5.4 Protocolo Experimental e Reprodutibilidade

Para assegurar a reprodutibilidade dos experimentos realizados e permitir comparações futuras sob as mesmas condições, esta seção descreve de forma detalhada o ambiente, os parâmetros de execução, as repetições adotadas e as estratégias de controle de aleatoriedade empregadas durante as simulações com a ferramenta *Isomera*.

Além de especificar o ambiente e os parâmetros, documentamos as definições adotadas para mensuração de tempo (ET), cálculo de métricas e organização dos artefatos produzidos (arquivos CSV/JSON e figuras). Essas informações permitem que terceiros repliquem integralmente os experimentos e verifiquem os resultados a partir dos mesmos insumos e configurações.

Ambiente de Execução

Todos os testes foram conduzidos em ambiente controlado, dentro de uma máquina virtual configurada especificamente para o estudo. A Tabela 5.2 descreve as especificações técnicas do ambiente experimental utilizado.

Tabela 5.2: Especificações do ambiente de execução dos experimentos.

Categoria	**Item**	**Descrição**
Sistema	Sistema operacional	Linux Ubuntu 22.04 LTS (64 bits)
Hardware	Processador	Intel Core i7-12700H (14 núcleos, 3.5 GHz)
Hardware	Memória RAM	16 GB DDR5
Software	Versão do Python	3.11.7 (ambiente virtual isolado)
Biblioteca	NetworkX	Modelagem e análise de grafos em Python.
Biblioteca	Pandas	Manipulação de dados tabulares (DataFrames) e CSV.
Biblioteca	NumPy	Arrays e operações numéricas vetorizadas.
Biblioteca	Matplotlib	Geração de figuras e gráficos estáticos.
Biblioteca	Seaborn	Visualizações estatísticas com estilos aprimorados.
Biblioteca	PyTorch	Treinamento e inferência de redes neurais em CPU.
Biblioteca	Torch Geometric	Operadores e camadas para GNNs (ex.: GIN, pooling).
Execução	Interface de execução	Aplicação local do *Isomera*, modo offline.

Os experimentos foram executados exclusivamente em CPU (sem GPU), em uma máquina virtual dedicada e sem outras cargas concorrentes, com toda a execução em modo offline. As dependências foram isoladas em um ambiente virtual (*venv*) para fixar versões. Os tempos foram medidos com relógio de alta resolução do Python e não houve paralelização explícita nas rotinas de isomorfismo, facilitando a replicação em ambiente equivalente.

Parâmetros Experimentais

A configuração dos experimentos seguiu um conjunto padronizado de parâmetros, garantindo que todos os testes pudessem ser reproduzidos de forma consistente. Esses parâmetros definem o número de execuções, as sementes aleatórias utilizadas e a estrutura do conjunto de referência (*ground truth*). A Tabela 5.3 apresenta o resumo dessas configurações.

Tabela 5.3: Parâmetros experimentais e controles de reprodutibilidade.

Parâmetro	**Descrição**
Número de repetições	Execução em lote para todas as instâncias de cada cenário, com repetições controladas por um conjunto fixo de sementes.
Sementes aleatórias	Valores e locais: `random=42`; `NumPy=42`; `PyTorch=42`; geração/nomeação dos grafos (arquivos `.gml`)=`42`; amostragem de negativos e embaralhamentos=`42`.
Tipos de cenário	TPC-DS com quatro níveis de complexidade (SOR $\in$ {2, 4, 8, 16}).
Ground truth	Construída manualmente na fase de validação supervisionada e armazenada em `JSON` (`validations/real_pairs_*.json`).
Critério de execução	Conjunto de instâncias e número de repetições definidos previamente; não há critério adaptativo de parada.
Persistência dos resultados	Artefatos gerados localmente: tempos (`execution_times.csv`), métricas (`evaluation_metrics.csv`), pares preditos (`predicted_pairs/*.json`) e figuras (`img/*.png`).

Adoções específicas para mensuração de tempo (ET): no Estudo de Caso I (VF2 e Node Match), o ET corresponde ao tempo de processamento por par de grafos no algoritmo avaliado. No Estudo de Caso II (GNN), o ET refere-se apenas ao tempo de inferência por par; o tempo de treinamento é realizado uma única vez por cenário e não é contabilizado no ET das predições.

Os artefatos e seus diretórios estão resumidos na Tabela 5.4. O formato GML (Graph Modelling Language) é um formato textual legível que representa grafos com nós/arestas e atributos.

Tabela 5.4: Artefatos persistidos e diretórios de saída.

Recurso	**Local**
Grafos de entrada (`.gml`)	`benchmark_graphs/`
Modelos GNN (`.pkl`)	`modelos_gnn/`

Com base nesses artefatos, a Tabela 5.5 resume os conjuntos e hiperparâmetros fixados nas execuções de referência. Termos consagrados de aprendizado de máquina são mantidos em inglês quando necessário e definidos brevemente no texto.

Tabela 5.5: Conjuntos e hiperparâmetros fixos utilizados.

Item	**Valor/Descrição**
SOR	$\{2,4,8,16\}$
Domínios	$\{1,2,3,4,5\}$
Semente (grafos/.gml)	`seed = 42`
Repetições	`runs = 1000` (VF2, NodeMatch); `runs = 1000` (GNN — inferência)
GNN — arquitetura	2 camadas GIN (64 unidades) + *global mean pooling*; classificador denso (oculta = 128)
GNN — treinamento	`epochs = 50`; `lr = 0.01` (*learning rate*); `BCEWithLogitsLoss`; otimizador Adam
GNN — limiares	*threshold* de inferência = `0.3`; limiar de treino para contagem de acerto = `0.6`

O limiar (*threshold*) é o ponto de decisão aplicado à probabilidade para classificar um par como duplicado. Como a duplicidade é rara no conjunto, usar 0.3 na inferência reduz a chance de deixar de identificar duplicações verdadeiras, aceitando mais candidatos para verificação. Já 0.6 é empregado somente como critério interno de contagem durante o treino, para uma leitura mais conservadora da evolução por época; esse limiar não interfere na decisão final do modelo.

Sobre o *PyTorch Geometric*, usamos essa biblioteca para construir a rede de grafos: o modelo aprende representações dos subgrafos e, com base nelas, decide se há duplicidade. Em linhas gerais, combinamos camadas que capturam a estrutura do grafo com uma etapa de agregação (*pooling*) que resume o subgrafo em um vetor, seguida de um classificador simples. Nos benchmarks, somente a inferência do modelo é considerada no tempo de execução (ET).

O *learning rate* (lr) define o tamanho do passo nas atualizações dos pesos; adotamos 0.01 por oferecer progresso consistente sem oscilações. O *Adam* é um otimizador que combina momento e ajuste adaptativo para cada parâmetro, o que costuma acelerar a convergência. A *BCEWithLogitsLoss* é uma função de custo para classificação binária que opera diretamente com logits e já incorpora a *sigmoid*, trazendo estabilidade numérica.

A acurácia média de treinamento foi 74,31%, calculada como a média das acurácias finais de cada cenário (D1–D5). Foi necessário treinar um modelo por cenário e salvar um arquivo .pkl para cada um, porque um único modelo global não alcançou boa acurácia; os padrões estruturais variam por cenário e o ajuste específico por domínio trouxe resultados melhores. A Tabela 5.6 apresenta os valores por cenário e os respectivos modelos treinados.

Tabela 5.6: Média de acurácia final por cenário (D1–D5) e respectivos modelos.

Cenário (D)	Média de Acurácia (%)	Modelos Correspondentes
D1	72.99	`graph_SOR2_D1`, `graph_SOR4_D1`, `graph_SOR8_D1`, `graph_SOR16_D1`
D2	75.00	`graph_SOR2_D2`, `graph_SOR4_D2`, `graph_SOR8_D2`, `graph_SOR16_D2`
D3	75.00	`graph_SOR2_D3`, `graph_SOR4_D3`, `graph_SOR8_D3`, `graph_SOR16_D3`
D4	75.00	`graph_SOR2_D4`, `graph_SOR4_D4`, `graph_SOR8_D4`, `graph_SOR16_D4`
D5	73.81	`graph_SOR2_D5`, `graph_SOR4_D5`, `graph_SOR8_D5`, `graph_SOR16_D5`

A adoção desses parâmetros assegura que qualquer pesquisador possa reproduzir integralmente os resultados apresentados nesta dissertação, bastando replicar as mesmas configurações experimentais e utilizar a versão da ferramenta *Isomera* disponibilizada no repositório público do projeto.

5.5 Estudo de Caso I

O primeiro estudo de caso foi conduzido com base nos quatro cenários definidos na Seção 5.2, utilizando grafos derivados de transformações estruturais sobre o benchmark TPC-DS. O principal objetivo foi comparar o desempenho de dois algoritmos de detecção de isomorfismos, VF2 e Node Match, frente a diferentes níveis de complexidade arquitetural, representados pelo número de tabelas de origem (SOR).

5.5.1 Metodologia

A avaliação foi realizada de forma sistemática, com a geração automática de múltiplas instâncias para cada configuração de complexidade, controlando o número de SOR e o grau de conectividade das estruturas de dados. Cada par de grafos foi submetido tanto ao algoritmo VF2 quanto ao Node Match.

As métricas utilizadas para análise, conforme definido na Seção 2.5, foram:

- **Tempo médio de execução (ET)**
- **Acurácia (ACC)**;
- **Frequência de Sucesso (SF)**.

A execução foi repetida com variações controladas de sementes aleatórias para assegurar a reprodutibilidade dos experimentos.

5.5.2 Análise dos resultados

As Figuras 5.3, 5.4 e 5.5 apresentam, respectivamente, a distribuição dos tempos de execução, a acurácia e a evolução da frequência de sucesso (SF) em função da complexidade dos cenários.

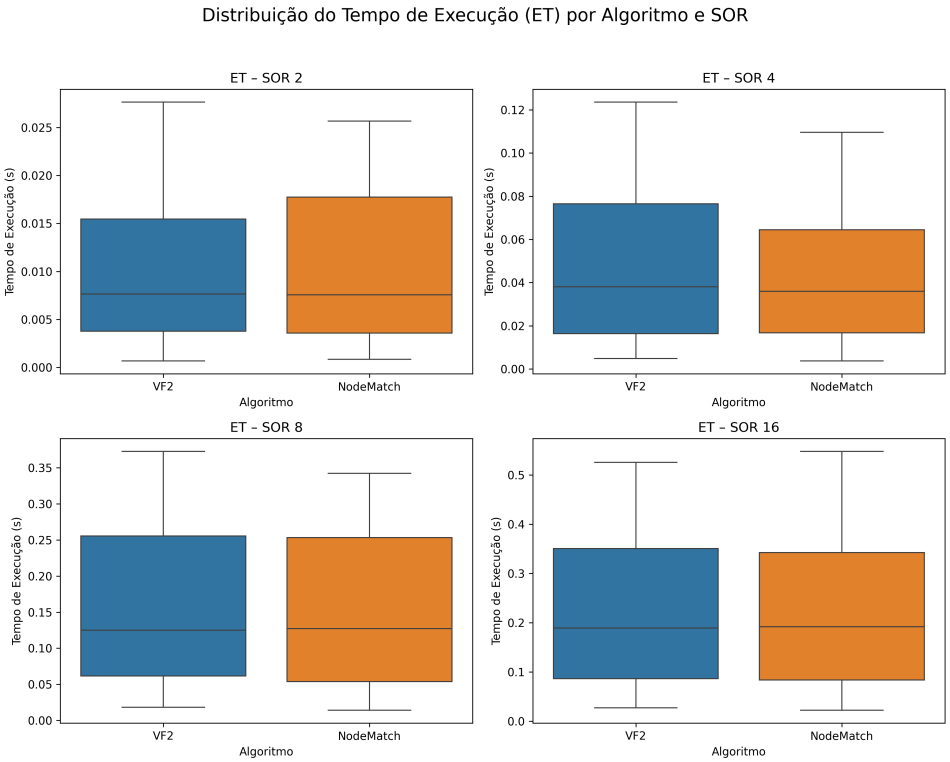


Figura 5.3: Tempo de execução (ET) para diferentes configurações de SOR.

A partir da análise dos resultados, observa-se uma tendência consistente de queda no desempenho dos algoritmos à medida que a complexidade estrutural dos grafos aumenta, conforme representado pelo número de tabelas de origem (SOR).

No que diz respeito ao tempo médio de execução (ET), ambos os métodos apresentaram tempos significativamente baixos nos cenários mais simples. Para SOR = 2, o VF2 registrou aproximadamente 1,10 segundos, enquanto o Node Match atingiu 1,11 segundos. Já no cenário com SOR = 4, os tempos médios foram de 5,18 segundos para o VF2 e 4,61 segundos para o Node Match. Com o aumento da complexidade, observa-se uma elevação gradual nos tempos de processamento: com SOR = 8, o VF2 alcançou 16,65 segundos e o Node Match 15,81 segundos; por fim, com SOR = 16, ambos os algoritmos convergiram para um tempo médio próximo a 24 segundos (VF2: 23,57 s; Node Match: 23,76 s).

Em relação à acurácia (ACC), os resultados foram baixos em todos os cenários, especialmente no mais complexo. Com SOR = 16, tanto o VF2 quanto o Node Match atingiram valores próximos de apenas 4% de acertos (4,04% e 4,29%, respectivamente), refletindo a limitação dessas abordagens na identificação de isomorfismos em arquiteturas altamente densas. Nos demais cenários, o Node Match apresentou desempenho superior ao VF2. Especificamente: para

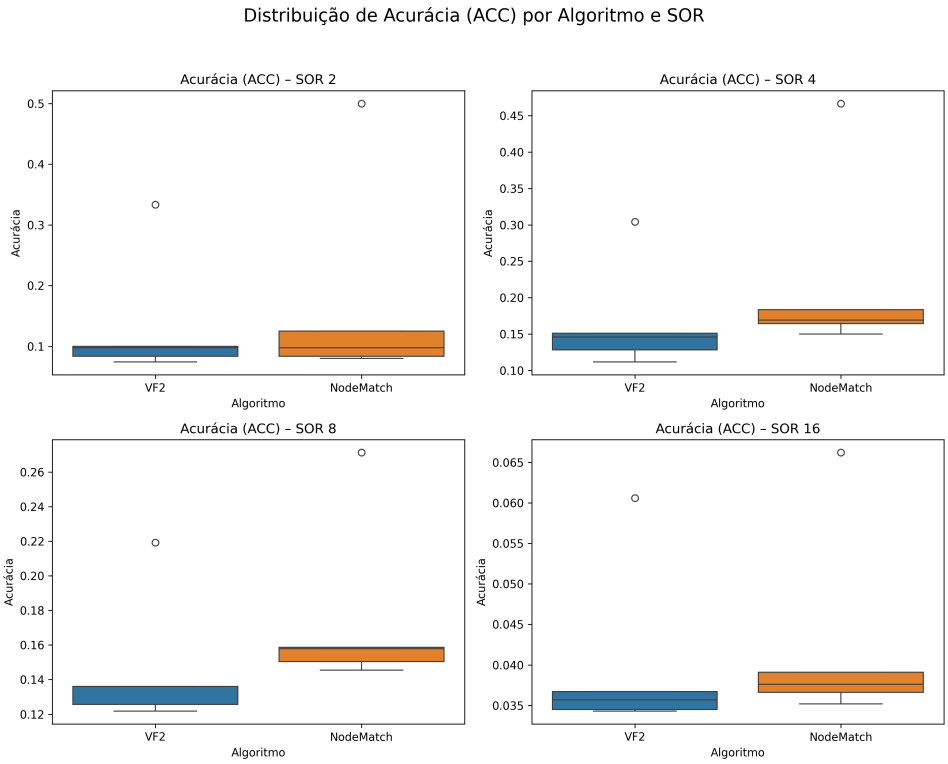


Figura 5.4: Acurácia (ACC) para diferentes configurações de SOR.

SOR = 2, o VF2 obteve 13,77% de acurácia, enquanto o Node Match alcançou 17,72%; com SOR = 4, os valores foram de 16,83% (VF2) e 22,67% (Node Match); em SOR = 8, o VF2 obteve 14,77% e o Node Match 17,67%. Esses resultados sugerem que o Node Match, mesmo sendo heurístico, é mais resiliente diante de alterações estruturais em grafos moderadamente complexos.

A métrica composta de frequência de sucesso (SF), que pondera tempo e acurácia, evidencia com mais clareza a vantagem relativa do Node Match em cenários com menor e média complexidade. No caso de SOR = 2, o Node Match alcançou 168,64 acertos por segundo, enquanto o VF2 registrou 143,65 acertos por segundo. Para SOR = 4, o ganho foi ainda mais expressivo: o Node Match atingiu 358,52 acertos por segundo, contra 220,80 acertos por segundo do VF2. Com SOR = 8, os valores também favoreceram o Node Match, com 220,42 acertos por segundo, frente a 161,91 acertos por segundo do VF2. Entretanto, no cenário mais complexo (SOR = 16), ambos os algoritmos apresentaram quedas drásticas na eficiência: o Node Match obteve 26,16 acertos por segundo, enquanto o VF2 registrou 22,85 acertos por segundo. Esses resultados reforçam que nenhuma das abordagens avaliadas é adequada para lidar com estruturas de larga escala, evidenciando a necessidade de técnicas mais robustas e escaláveis para contextos de maior densidade estrutural.

Esses achados reforçam não apenas a utilidade do framework proposto como ferramenta de análise comparativa, mas também a necessidade de incorporar novos algoritmos capazes de

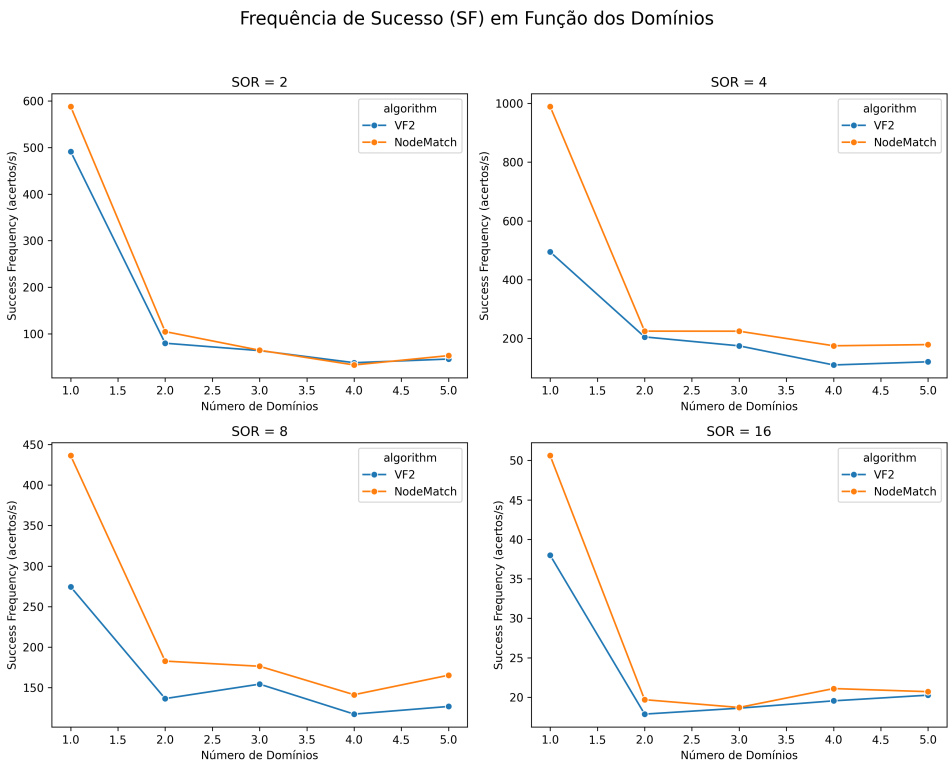


Figura 5.5: Evolução da frequência de sucesso (SF) conforme aumenta a complexidade.

lidar com grafos maiores e mais intrincados — o que motiva o aprofundamento explorado no Estudo de Caso II.

5.5.3 Considerações Finais do Estudo de Caso I

Embora o desempenho dos algoritmos tenha sido limitado, este primeiro estudo de caso cumpre seu papel como etapa metodológica fundamental. Ele permitiu validar o arcabouço de simulação e avaliação proposto, além de evidenciar a necessidade de abordagens mais sofisticadas para cenários de alta complexidade. A partir dessa constatação, o próximo estudo de caso (Seção 5.6) busca ampliar a análise com a introdução de novos algoritmos, incluindo métodos baseados em aprendizado supervisionado e técnicas mais modernas de embeddings estruturais.

5.6 Estudo de Caso II

Dando continuidade à análise iniciada no Estudo de Caso I, o segundo estudo de caso tem como foco a inclusão de um novo algoritmo baseado em aprendizado supervisionado: uma rede neural gráfica (GNN), como descrito na subseção 2.4.3. A principal motivação desta etapa é investigar se, em cenários com maior complexidade estrutural, a abordagem supervisionada pode oferecer ganhos de acurácia e custo-benefício (SF), superando as limitações observadas

nas técnicas clássicas.

5.6.1 Metodologia

Para viabilizar a aplicação do GNN, foi necessário treinar um modelo para cada cenário experimental, com base em exemplos reais de pares isomorfos. Inicialmente, cada grafo foi decomposto em subgrafos locais, considerando um nó central e seus vizinhos diretos. Os pares de subgrafos rotulados como isomorfos (positivos) foram extraídos manualmente, enquanto os pares negativos foram gerados de forma aleatória, respeitando o balanceamento e evitando sobreposição com os positivos.

Cada subgrafo foi convertido para o formato *PyTorch Geometric (PyG)*, com atributos constantes nos nós e arestas dirigidas. Durante o treinamento, a arquitetura GIN recebeu esses pares como entrada, processando cada subgrafo com duas camadas convolucionais seguidas de *mean pooling*. Os vetores gerados foram concatenados e passados por uma rede densa para prever a probabilidade de isomorfismo. O treinamento foi supervisionado, utilizando função de custo *BCEWithLogitsLoss* e otimizador Adam, com 50 épocas de aprendizado por cenário. Os modelos resultantes foram salvos e empregados na fase de inferência com limiar de decisão fixo.

Além das métricas de avaliação em teste, acompanhou-se a acurácia de treinamento por época para monitorar a convergência do modelo. A média da acurácia de treino ao longo das épocas, agregada por cenário, foi de **74,31%**.

Mantendo a mesma configuração experimental descrita na Seção 5.2, os experimentos foram estendidos para incluir o GNN supervisionado, treinado com base em amostras reais de isomorfismos presentes nos dados de entrada. O modelo foi avaliado sob os mesmos critérios anteriores: tempo de execução (ET), acurácia (ACC) e frequência de sucesso (SF), utilizando os mesmos cenários com 2, 4, 8 e 16 tabelas de origem (SOR).

A adição do GNN permite avaliar o potencial de generalização do modelo em grafos não vistos, bem como sua escalabilidade frente à complexidade arquitetural. O código da Seção A.2 mostra o treinamento do GNN e a execução do VF2 e Node Match.

5.6.2 Análise dos Resultados

As Figuras 5.6, 5.7 e 5.8 ilustram, respectivamente, a comparação entre os três algoritmos quanto ao tempo de execução, acurácia e frequência de sucesso, em cada configuração de SOR.

Em termos de tempo de execução, o GNN apresentou desempenho inferior aos algoritmos VF2 e Node Match, com tempos significativamente mais altos em todos os cenários. Para SOR = 2, o GNN registrou 0,17 segundos, contra 0,01 segundos de VF2 e Node Match. Em SOR = 4, o tempo aumentou para 0,58 segundos (GNN), enquanto o VF2 ficou em 0,05 segundos e o Node Match em 0,04 segundos. Com SOR = 8, o tempo médio do GNN chegou a 1,75 segundos, frente a 0,16 segundos do VF2 e 0,15 segundos do Node Match. Finalmente, no cenário mais complexo (SOR = 16), o GNN atingiu 1,78 segundos, contra 0,23 segundos nos dois algoritmos clássicos.

Apesar da maior carga computacional, o GNN apresentou melhoria nas taxas de acerto em cenários complexos. Para SOR = 16, sua acurácia foi de 12,80%, superior aos 4,29% do Node Match e 4,04% do VF2. Para SOR = 8, a acurácia do GNN alcançou 20,87%, frente a

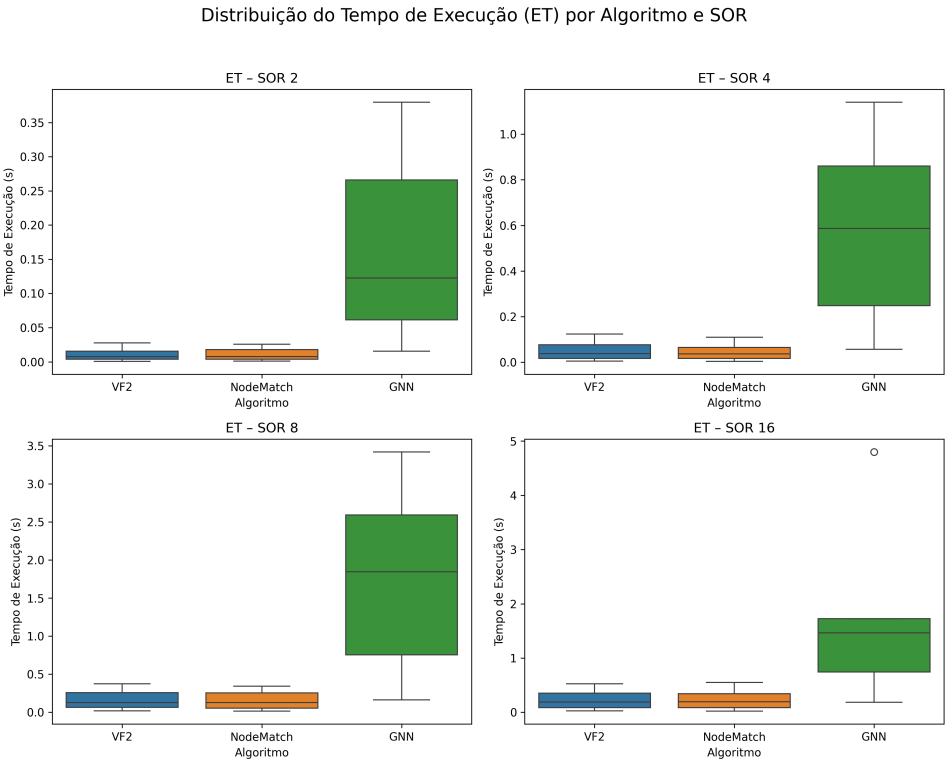


Figura 5.6: Tempo de execução (ET) para diferentes configurações de SOR com inclusão do GNN.

17,67% (Node Match) e 14,77% (VF2). No entanto, nos cenários mais simples, o GNN obteve desempenho inferior: 7,27% de acurácia para SOR = 2 e 15,76% para SOR = 4, ambos abaixo dos resultados dos algoritmos clássicos nesses mesmos cenários.

A métrica de frequência de sucesso (SF), que pondera simultaneamente tempo e acurácia, evidenciou o custo da abordagem supervisionada. O GNN teve SF de apenas 3,99 acertos por segundo para SOR = 2, 15,96 para SOR = 4, 21,93 para SOR = 8 e 11,17 para SOR = 16. Esses valores são significativamente menores que os obtidos pelos demais algoritmos: por exemplo, com SOR = 4, o Node Match atingiu 358,52 acertos por segundo, e o VF2, 220,80. Em SOR = 8, o Node Match chegou a 220,42, contra 21,93 do GNN, ainda que a acurácia do GNN tenha superado as demais em forma isolada.

5.7 Conclusão dos Estudos de Caso

A realização dos estudos de caso permitiu validar a metodologia proposta para avaliação de algoritmos de detecção de redundância estrutural em arquiteturas baseadas em grafos. Utilizando uma base adaptada do benchmark TPC-DS, foi possível estruturar cenários com diferentes níveis de complexidade representados pelo número de tabelas SOR e analisar o desempenho em cada cenário.

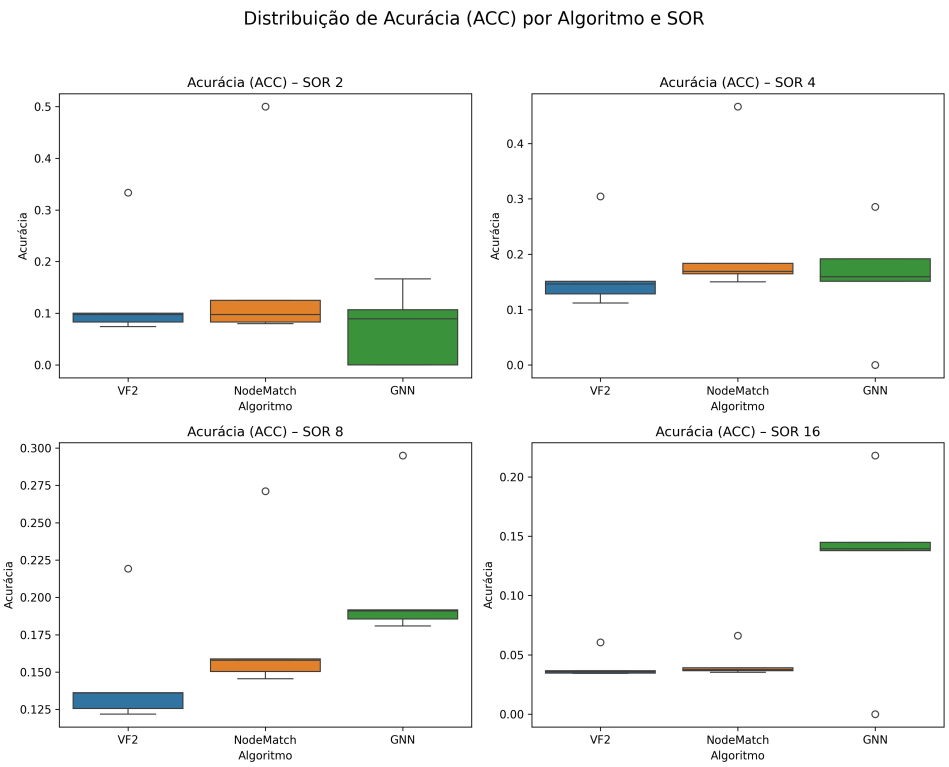


Figura 5.7: Acurácia (ACC) para diferentes configurações de SOR com inclusão do GNN.

No Estudo de Caso I, a comparação entre os algoritmos VF2 e Node Match revelou que, apesar de suas limitações em termos de acurácia, ambos apresentaram bom desempenho computacional nos cenários de baixa complexidade. O Node Match destacou-se pela maior frequência de sucesso (SF) em SOR menores, sugerindo sua utilidade como ferramenta de triagem preliminar. No entanto, ambos os algoritmos demonstraram severa degradação de desempenho à medida que a complexidade estrutural aumentava, tanto em termos de tempo quanto de acerto, especialmente com SOR = 16.

Diante dessa limitação, o Estudo de Caso II incorporou uma abordagem supervisionada baseada em redes neurais (GNN), com o objetivo de investigar ganhos em acurácia nos cenários mais complexos. Os resultados indicaram que o GNN, de fato, obteve melhor desempenho em termos de acurácia para SOR = 8 e SOR = 16, superando consistentemente os métodos anteriores nesse critério. Contudo, o custo computacional da inferência afetou negativamente sua frequência de sucesso (SF), indicando que, embora promissora, a abordagem supervisionada ainda demanda otimizações para aplicações em tempo real.

Do ponto de vista metodológico, os estudos confirmaram a efetividade do arcabouço proposto. A geração automática dos cenários, a definição precisa das métricas e a aplicação sistemática dos algoritmos garantiram reprodutibilidade e comparabilidade dos resultados. Mais do que avaliar algoritmos isolados, a metodologia se mostrou eficaz como ferramenta de experimentação científica voltada à análise da robustez, escalabilidade e custo-benefício de diferentes

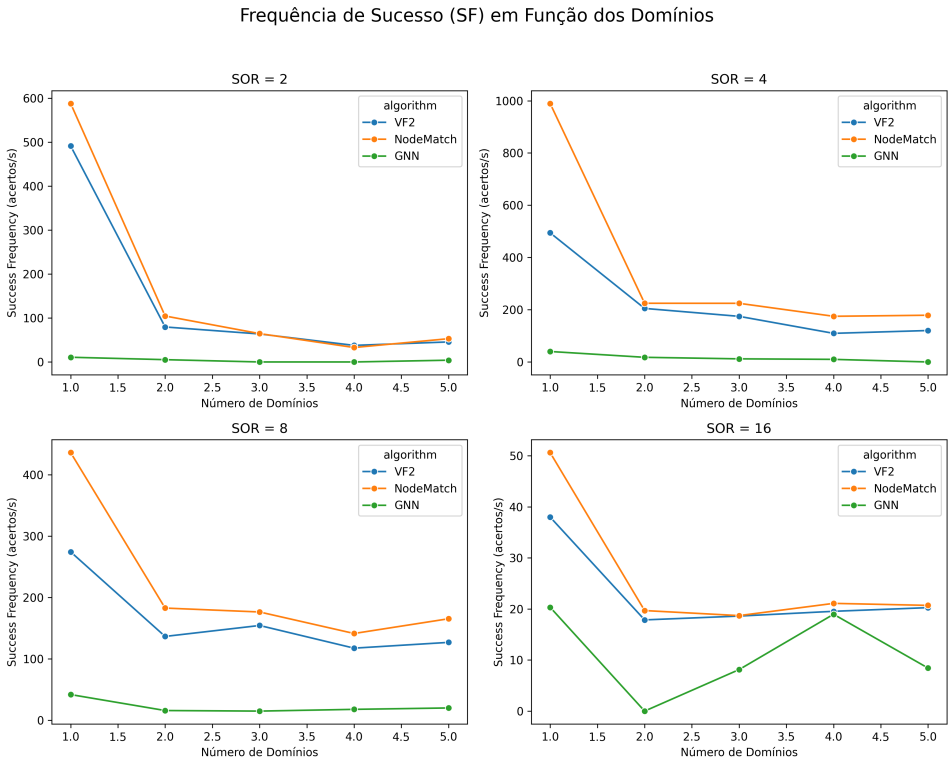


Figura 5.8: Frequência de Sucesso (SF) para diferentes configurações de SOR com inclusão do GNN.

estratégias de detecção de redundâncias.

Assim, os estudos de caso não apenas evidenciam limitações das abordagens tradicionais em arquiteturas complexas, como também sinalizam caminhos para a evolução da técnica, seja por meio de métodos supervisionados otimizados, estratégias híbridas ou mecanismos adaptativos de seleção de algoritmos conforme o grau de complexidade da arquitetura em análise.

5.8 Limitações e Ameaças à Validade

Como todo experimento empírico em ciência de dados, os estudos apresentados nesta dissertação estão sujeitos a limitações que podem afetar a generalização e a interpretação dos resultados. Esta subseção discute as principais ameaças à validade do trabalho, concentrando-se em três dimensões críticas: dependência de validação manual, escalabilidade e custo computacional, e generalização dos resultados para ambientes *Data Mesh* reais.

Dependência de Validação Manual

A principal limitação da metodologia reside na necessidade de validação supervisionada por especialistas para confirmar se as duplicidades detectadas correspondem, de fato, a redundâncias

funcionais. Apesar de os algoritmos de isomorfismo — VF2, Node Match e GNN — serem capazes de identificar similaridades estruturais com precisão, nem sempre a equivalência topológica implica equivalência semântica. Em ambientes corporativos, duas tabelas podem possuir o mesmo formato e conexões, mas desempenhar papéis distintos em contextos de negócio específicos.

Por essa razão, a intervenção humana torna-se essencial para evitar a exclusão de entidades legítimas e preservar a integridade lógica do sistema. Essa dependência, entretanto, introduz uma limitação prática: a escalabilidade do processo de validação. Em arquiteturas de larga escala, a revisão manual de pares de subgrafos torna-se custosa e demorada, podendo afetar o tempo total de análise. Nos trabalhos futuros, propõe-se o uso de abordagens com IA generativa, de modo a reduzir o esforço manual e acelerar ou retirar o processo de confirmação de duplicidades.

Escalabilidade e Custo Computacional

Outra limitação relevante diz respeito ao custo computacional dos algoritmos de detecção, especialmente quando aplicados a grafos de alta densidade. O algoritmo VF2, embora exato, apresenta complexidade exponencial e torna-se inviável em arquiteturas muito grandes. Mesmo o Node Match, um algoritmo híbrido com pré-filtragem, e o modelo supervisionado GNN, com melhor capacidade de generalização, ainda demandam recursos computacionais significativos quando o número de vértices e arestas cresce exponencialmente. Nos trabalhos futuros, propõe-se o uso de abordagens com IA generativa, de modo a reduzir o esforço manual e acelerar ou até eliminar o processo de confirmação de duplicidades.

Durante os experimentos realizados, observou-se que o tempo médio de execução aumentava substancialmente em cenários com mais de 16 tabelas de origem (SOR), o que evidencia o impacto direto da complexidade estrutural sobre a eficiência dos métodos. Como mitigação, recomenda-se, em aplicações práticas, o uso de técnicas de paralelização, particionamento de grafos e indexação de subestruturas recorrentes, de forma a distribuir a carga de processamento e tornar o processo de detecção mais escalável.

Há a necessidade de mais estudos voltados à compreensão detalhada da complexidade computacional envolvida na detecção de isomorfismos, considerando fatores como tamanho, densidade e profundidade topológica dos grafos. Investigações adicionais podem contribuir para o desenvolvimento de mecanismos de estimativa e controle de custo, bem como de técnicas que reduzam o impacto da complexidade, como a decomposição hierárquica de componentes, a compressão de subgrafos equivalentes e a adoção de heurísticas adaptativas para ajuste dinâmico do nível de busca. Essas abordagens podem ampliar a aplicabilidade da metodologia a ambientes corporativos de larga escala, equilibrando precisão analítica e desempenho computacional.

A Tabela 5.7 apresenta as principais ameaças à validade observadas nos estudos experimentais e as respectivas estratégias de mitigação consideradas no escopo desta dissertação.

Tabela 5.7: Síntese das ameaças à validade e estratégias de mitigação

Categoria	**Descrição da Limitação**	**Mitigação / Direção de Aperfeiçoamento**
Dependência de validação manual	A metodologia requer intervenção humana para confirmar se pares identificados pelos algoritmos representam realmente duplicidades funcionais. Essa dependência limita a escalabilidade do processo e pode introduzir vieses subjetivos, principalmente em arquiteturas muito extensas.	Integrar técnicas de IA generativa capazes de sugerir automaticamente classificações preliminares de duplicidade com base em padrões históricos.
Custo computacional	O tempo de execução cresce de forma não linear com o número de vértices e arestas, tornando o processo inviável para grafos de alta densidade. Os algoritmos exatos, como VF2, sofrem com explosão combinatória, e mesmo abordagens heurísticas e supervisionadas demandam recursos intensivos.	Adoção de estratégias de paralelização, particionamento de grafos e indexação de subestruturas recorrentes para distribuir a carga de processamento. Estudos adicionais são necessários para mensurar e reduzir a complexidade estrutural de grafos de grande escala.
Generalização dos resultados	Os experimentos foram conduzidos em cenários controlados — tanto sintéticos quanto baseados no benchmark TPC-DS — que, embora realistas, não abrangem toda a diversidade de arquiteturas corporativas.	Aplicação e validação da metodologia em diferentes contextos, incluindo arquiteturas reais de *Data Mesh*, *Data Lake* e *Data Warehouse*, garantindo maior robustez e capacidade de generalização.
Limitações do *ground truth*	A base de referência utilizada para rotulagem contém apenas pares previamente identificados e validados, restringindo a diversidade de exemplos para treinamento e avaliação supervisionada.	Ampliação da base de pares validados e criação de pipelines automáticos de geração e validação cruzada entre domínios, com monitoramento de consistência semântica usando IA generativa.

CAPÍTULO 6

Conclusão e Trabalhos Futuros

Esta dissertação apresentou uma metodologia sistemática para a detecção de redundâncias estruturais em arquiteturas de dados distribuídas, com ênfase na aplicabilidade ao contexto de *Data Mesh*. O principal foco deste trabalho não foi o desempenho absoluto de um algoritmo específico, mas sim o desenvolvimento de um arcabouço metodológico reprodutível, comparável e extensível, capaz de apoiar pesquisadores e profissionais na avaliação objetiva de diferentes estratégias de detecção.

A proposta foi operacionalizada por meio da ferramenta *Isomera*, construída para permitir simulações automáticas, validações controladas e medições padronizadas em arquiteturas modeladas como grafos direcionados. A metodologia baseia-se em quatro etapas articuladas: (i) geração de grafos estruturais a partir de cenários sintéticos ou benchmarks realistas; (ii) aplicação de algoritmos de isomorfismo (VF2, Node Match ou GNN); (iii) validação supervisionada dos pares redundantes; e (iv) avaliação do desempenho com base em métricas objetivas como tempo de execução, acurácia e frequência de sucesso (SF).

A realização de dois estudos de caso distintos demonstrou o valor da metodologia proposta. O Estudo de Caso I comparou os algoritmos VF2 e Node Match em diferentes configurações de complexidade, revelando limitações previsíveis nas abordagens clássicas — sobretudo em cenários densos, mas, mais importante, validando o funcionamento do arcabouço sob condições controladas e reprodutíveis. Já o Estudo de Caso II introduziu uma rede neural gráfica (GNN) supervisionada como uma alternativa mais robusta para contextos de alta complexidade. Embora o custo computacional tenha sido superior, o aumento de acurácia nos cenários com 8 e 16 SOR mostrou que modelos de aprendizado supervisionado podem capturar padrões estruturais que escapam aos algoritmos tradicionais.

Em resposta às perguntas de pesquisa, demonstrou-se que é possível aplicar algoritmos de isomorfismo para detectar automaticamente estruturas redundantes em arquiteturas orientadas a dados, desde que essas arquiteturas estejam representadas em forma de grafos com modelagem consistente. A comparação entre algoritmos clássicos e modelos baseados em aprendizado supervisionado revelou que diferentes técnicas apresentam vantagens distintas conforme a densidade e o grau de sobreposição entre os domínios. Além disso, observou-se que a intervenção humana na etapa de validação é indispensável para evitar que pares estruturalmente equivalentes, mas semanticamente distintos, sejam erroneamente classificados como duplicidade. Assim, a metodologia proposta mostrou-se adequada para integrar automação e julgamento humano de forma complementar no processo de refinamento arquitetural.

Contudo, o maior resultado desta pesquisa não está nos valores numéricos isolados, mas na consolidação de um processo experimental que une geração sistemática de dados, aplicação modular de algoritmos, supervisão de resultados e mensuração quantitativa. Ao estruturar esse

ciclo completo, o trabalho oferece uma base confiável para replicação científica, comparação entre métodos e avanço incremental no campo da engenharia de dados orientada a grafos.

Adicionalmente à metodologia, o trabalho contribuiu com o desenvolvimento de um artefato computacional completo, escrito em Python, capaz de reproduzir todas as etapas propostas, desde a geração dos grafos até a exportação dos pares detectados e das métricas de desempenho. A ferramenta *Isomera* permite a integração com benchmarks sintéticos (como o TPC-DS) e viabiliza experimentos controlados com diferentes algoritmos e configurações. Seu código modular e interface interativa contribuem não apenas para o uso prático em ambientes corporativos, como também para o ensino, pesquisa e reprodutibilidade acadêmica.

Por fim, espera-se que este trabalho inspire futuras investigações que combinem modelagem algébrica, técnicas de aprendizado de máquina e representação em grafos para o aprimoramento da governança de dados em ecossistemas complexos e descentralizados.

Trabalhos Futuros

Como desdobramentos naturais deste estudo, destacam-se as seguintes linhas de continuidade:

- **Detecção a nível colunar**: estender o modelo atual para identificar redundâncias e equivalências estruturais também em nível de colunas, permitindo uma análise mais granular das dependências entre atributos das tabelas;
- **Geração de grafos a partir de bases reais**: implementar mecanismos automáticos de construção de grafos diretamente a partir de esquemas e metadados de bancos de dados relacionais (PostgreSQL, MySQL, Glue Catalog etc.), possibilitando a validação empírica do método proposto;
- **Aplicação em tempo real e batch**: adaptar a metodologia para ambientes híbridos, permitindo a execução contínua de verificações de redundância tanto em fluxos em tempo real quanto em processamentos em lote;
- **Aprimoramento semântico**: expandir o modelo atual para incorporar ontologias e metadados semânticos, permitindo detectar redundâncias conceituais além das estruturais;
- **Testes com novos algoritmos de aprendizado**: incluir abordagens supervisionadas e não supervisionadas como *autoencoders*, *graph transformers*, e modelos de aprendizado contrastivo;
- **Exploração de IA generativa**: investigar a aplicação de modelos generativos (GNNs generativos, LLMs com raciocínio estrutural) para propor agrupamentos e reduções arquiteturais de forma proativa;
- **Integração com metadados reais**: habilitar a leitura direta de esquemas de bancos de dados (PostgreSQL, MySQL, Glue Catalog etc.), facilitando a aplicação prática da metodologia em pipelines produtivos;

- **Desenvolvimento colaborativo**: evoluir a ferramenta *Isomera* para uma plataforma interativa com validação distribuída, versionamento de testes e exportação de relatórios;
- **Simulação de impacto operacional**: integrar redes estocásticas de Petri para avaliar como a presença ou remoção de redundâncias afeta disponibilidade em arquiteturas *Data Mesh*;
- **Estudo de custos computacionais e escalabilidade**: investigar o comportamento de desempenho e os custos computacionais associados à execução dos algoritmos de detecção de isomorfismos e à manutenção de pipelines em arquiteturas *Data Mesh*, analisando tempo de execução, uso de memória, variação com o aumento do número de vértices e arestas, e limites práticos de escalabilidade em cenários distribuídos;
- **Análise de complexidade computacional**: realizar uma avaliação comparativa entre diferentes algoritmos de isomorfismo de grafos considerando tempo de execução, escalabilidade e uso de memória. Os resultados poderão ser organizados em tabelas de complexidade e benchmarks experimentais para diferentes topologias de grafos.

Referências Bibliográficas

BENA, Y. A. *et al.* Big data governance challenges arising from data generated by intelligent systems technologies: A systematic literature review. **IEEE Access**, IEEE, v. 13, p. 12859–12884, 2025. ISSN 2169-3536. Disponível em: <https://doi.org/10.1109/ACCESS.2025.3528941>.

BLOHM, I. *et al.* Data products, data mesh, and data fabric: New paradigm(s) for data and analytics? **Business & Information Systems Engineering**, Springer, v. 66, n. 5, p. 643–652, 2024. Received: 19 June 2023 / Accepted: 19 April 2024 / Published online: 5 June 2024. Open Access under CC BY 4.0 license. Disponível em: <https://doi.org/10.1007/s12599-024-00876-5>.

BODAPATI, R. V. P. K. Data mesh architecture for scalable business intelligence systems. **Journal of Computer Science and Technology Studies**, v. 7, n. 5, p. 770–777, 2025. Published: 2025-06-05. Open Access under CC BY 4.0 license. Disponível em: <https://doi.org/10.32996/jcsts.2025.7.5.86>.

BOLDRINI, J. L. *et al.* **Álgebra Linear**. 3. ed. São Paulo: Harbra, 1986. ISBN 8529400262.

CORDELLA, L. P. *et al.* A (sub)graph isomorphism algorithm for matching large graphs. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 26, n. 10, p. 1367–1372, 2004. ISSN 0162-8828. Disponível em: <https://doi.org/10.1109/TPAMI.2004.75>.

DEHGHANI, Z. **How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh**. 2019. <https://martinfowler.com/articles/data-monolith-to-mesh.html>. Online; accessed 2025.

DEHGHANI, Z. **Data Mesh: Delivering Data-Driven Value at Scale**. O'Reilly Media, 2022. ISBN 978-1492092391. Disponível em: <https://www.oreilly.com/library/view/data-mesh/9781492092384/>.

DERAKHSHANNIA, M. *et al.* Data lake governance: Towards a systemic and natural ecosystem analogy. **Future Internet**, MDPI, v. 12, n. 8, p. 126, 2020. Disponível em: <https://www.mdpi.com/1999-5903/12/8/126>.

DEVI, C.; INAMPUDI, R. K.; VIJAYABOOPATHY, V. Federated data-mesh quality scoring with great expectations and apache atlas lineage. **Journal of Knowledge Learning and Science Technology**, v. 4, n. 2, p. 92–101, 2025. ISSN 2959-6386. Open Access. Disponível em: <https://doi.org/10.60087/jklst.v4.n2.008>.

DU, B.; CAO, N. First: Fast interactive attributed subgraph matching. ACM, Halifax, NS, Canada, p. 1447–1456, 2017. Applied Data Science Paper. Published: August 13–17, 2017. Disponível em: <https://doi.org/10.1145/3097983.3098040>.

ELMAGARMID, A. K.; IPEIROTIS, P. G.; VERYKIOS, V. S. Duplicate record detection: A survey. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 19, n. 1, p. 1–16, 2007. Published online: 20 Nov. 2006. Disponível em: <https://doi.org/10.1109/TKDE.2007.250581>.

FANG, H. Managing data lakes in big data era: What's a data lake and why has it become popular in data management ecosystem. In: **2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)**. IEEE, 2015. p. 820–824. Disponível em: <https://doi.org/10.1109/CYBER.2015.7288049>.

FEOFILOFF, P. **Ciclos e DAGs — Algoritmos para Grafos**. 2020. <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/cycles-and-dags.html#sec:the-problem>. Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo. Acesso em: 23 jul. 2025.

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. **Uma Introdução Sucinta à Teoria dos Grafos**. São Paulo, Brasil: [s.n.], 2011. Minicurso apresentado na II Bienal da SBM. Disponível em: <https://www.ime.usp.br/~pf/teoriadosgrafos/>. Acesso em: 23 jul. 2025.

FILHO, J. da S. **Criação de uma ferramenta informativa sobre Teoria dos Grafos**. Palmas, TO: [s.n.], 2017. Trabalho de Conclusão de Curso (TCC) – Centro Universitário Luterano de Palmas (CEULP/ULBRA). Orientador: Prof. M.e Fernando Luiz de Oliveira.

GIEBLER, C. *et al.* Leveraging the data lake: Current state and challenges. In: ORDONEZ, C. *et al.* (Ed.). **Big Data Analytics and Knowledge Discovery. DaWaK 2019**. Springer, Cham, 2019. (Lecture Notes in Computer Science, v. 11708), p. 179–188. Disponível em: <https://doi.org/10.1007/978-3-030-27520-4_13>.

GIEß, A.; HUTTERER, A. The future of data management: A delimitation of data platforms, data spaces, data meshes, and data fabrics. **Information Systems and e-Business Management**, 2025. Received 9 September 2024, Revised 16 January 2025, Accepted 4 April 2025, Published 16 July 2025. Disponível em: <https://doi.org/10.1007/s10257-025-00707-4>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. ISBN 9780262035613.

HARBY, A. A.; ZULKERNINE, F. From data warehouse to lakehouse: A comparative review. In: **2022 IEEE International Conference on Big Data (Big Data)**. Osaka, Japan: IEEE, 2022. Disponível em: <https://ieeexplore.ieee.org/document/10020719>.

HASNAIN, M. *et al.* Evaluating trust prediction and confusion matrix measures for web services ranking. **IEEE Access**, v. 8, p. 90847–90859, 2020. Received April 23, 2020; accepted May 2, 2020; published May 12, 2020; current version May 27, 2020. Open Access under CC BY 4.0 license. Disponível em: <https://ieeexplore.ieee.org/document/9091880>.

HOSEINI, S.; THEISSEN-LIPP, J.; QUIX, C. A survey on semantic data management as intersection of ontology-based data access, semantic modeling and data lakes. **Web Semantics: Science, Services and Agents on the World Wide Web**, Elsevier, v. 81, p. 100819, 2024. ISSN 1570-8268. Open Access under CC BY license. Disponível em: <https://doi.org/10.1016/j.websem.2024.100819>.

INMON, W. H. Building the data warehouse. John Wiley & Sons, 2005. Disponível online. Acesso em julho de 2025. Disponível em: <http://www.r-5.org/files/books/computers/databases/warehouses/W_H_Inmon-Building_the_Data_Warehouse-EN.pdf>.

JI, C. *et al.* Big data processing in cloud computing environments. In: **2012 12th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)**. [s.n.], 2012. p. 17–23. Disponível em: <https://doi.org/10.1109/I-SPAN.2012.9>. Acesso em julho de 2025. Disponível em: <https://doi.org/10.1109/I-SPAN.2012.9>.

KANAGARLA, K. P. B. Data mesh: Decentralised data management. **IRACST – International Journal of Computer Networks and Wireless Communications (IJCNWC)**, v. 14, n. 1, January 2024. Disponível em: <https://ssrn.com/abstract=5024895>. Acesso em julho de 2025.

KAPFERER, J.; BRUNNER, M.; ZELLER, M. Decentralized data mesh architectures: Principles and implementation patterns. **Lecture Notes in Computer Science (Springer)**, v. 13011, p. 131–144, 2021.

KAVAK, M.; RUSU, L. Challenges and opportunities of artificial intelligence in digital transformation: A systematic literature review. **Procedia Computer Science**, Elsevier, v. 256, p. 369–377, 2025. Presented at CENTERIS / ProjMAN / HCist 2024. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050925004892>.

KHINE, P.; WANG, Z. Data lake: A new ideology in big data era. In: **Proceedings of the 2017 International Conference on Services Computing**. [S.l.: s.n.], 2017. Available at: <https://www.researchgate.net/publication/321825490_Data_Lake_A_New_Ideology_in_Big_Data_Era>.

KIMBALL, R.; ROSS, M. **The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling**. [S.l.]: John Wiley & Sons, 2013. Disponível em: <https://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/Documentos%20de%20acesso%20remoto/The-Data-Warehouse-Toolkit-2ed_Kimball.pdf>. Acesso em julho de 2025.

LANGNER, J. *et al.* Spectramatcher: A python program for interactive analysis and peak assignment of vibronic spectra. 2025. ArXiv:2505.09060 [physics.chem-ph], 14 páginas, 3 figuras, 1 tabela.

LIU, L. *et al.* G-finder: Approximate attributed subgraph matching. p. 513–522, 2019. Disponível em: <https://ieeexplore.ieee.org/document/9006525>.

LU, H. *et al.* Duplicate data detection using gnn. In: **2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)**. [S.l.: s.n.], 2016. p. 167–170. Disponível em: <https://ieeexplore.ieee.org/document/7529552>.

MANčINSKA, L.; ROBERSON, D. E.; VARVITSIOTIS, A. Graph isomorphism: physical resources, optimization models, and algebraic characterizations. **Mathematical Programming**, v. 205, n. 1, p. 617–660, 2024. Disponível em: <https://doi.org/10.1007/s10107-023-01989-7>.

MESSMER, B.; BUNKE, H. Efficient subgraph isomorphism detection: A decomposition approach. **IEEE Transactions on Knowledge and Data Engineering**, v. 12, n. 2, p. 307–323, 2000. Disponível em: <https://doi.org/10.1109/69.842269>.

MITRAN, S. **Linear Algebra for Data Science**. [S.l.]: University of North Carolina at Chapel Hill, 2023. Disponível em: <http://mitran-lab.amath.unc.edu/courses/MATH347DS/textbook.pdf>.

NAMBIAR, R.; POESS, M. The making of tpc-ds. In: **Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)**. [S.l.]: VLDB Endowment, 2006. p. 1049–1058. Disponível em: <https://dl.acm.org/doi/10.5555/1182635.1164217>.

NETTO, P. O. B. **Grafos: Teoria, Modelos, Algoritmos**. 5. ed. São Paulo: Blucher, 2012. Disponível em: <https://www.editorablucher.com.br/book/9788521206804>. ISBN 978-8521206804.

PATEL, A. A.; DHARWA, J. Graph data: The next frontier in big data modeling for various domains. **Indian Journal of Science and Technology**, v. 10, n. 21, 2017. Disponível em: <https://doi.org/10.17485/ijst/2017/v10i21/112828>.

REN, J.; LI, T. Graph isomorphism—characterization and efficient algorithms. **High-Confidence Computing**, v. 4, n. 4, p. 100224, 2024. ISSN 2667-2952. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2667295224000278>.

RIKOS, A. I.; HADJICOSTIS, C. N. Distributed integer weight balancing in the presence of time delays in directed graphs. **IEEE Transactions on Control of Network Systems**, v. 5, n. 3, p. 1300–1309, 2018. Disponível em: <https://ieeexplore.ieee.org/document/7922507>.

ROY-HUBARA, N. *et al.* Modeling graph database schema. **IT Professional**, IEEE, v. 19, n. 6, p. 34–43, November/December 2017. Disponível em: <https://ieeexplore.ieee.org/document/8123463>.

SERRA, J. **Deciphering Data Architectures: Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh**. O'Reilly Media, 2024. Disponível em: <https://www.oreilly.com/library/view/deciphering-data-architectures/9781098150754/>.

SHARAN, R.; IDEKER, T. Modeling cellular machinery through biological network comparison. **Nature Biotechnology**, Nature Publishing Group, v. 24, n. 4, p. 427–433, 2006. Disponível em: <https://doi.org/10.1038/nbt1196>.

SHERVASHIDZE, N. *et al.* Weisfeiler-lehman graph kernels. **Journal of Machine Learning Research**, JMLR. org, v. 12, p. 2539–2561, 2011. Disponível em: <http://jmlr.org/papers/v12/shervashidze11a.html>.

SOSA, J.; URREGO-LOPEZ, A.; PRIETO, C. Analysis of bipartite networks in anime series: Textual analysis, topic clustering, and modeling. **arXiv preprint arXiv:2411.03333**, 2024. 23 pages, 3 figures, 5 tables. Disponível em: <https://doi.org/10.48550/arXiv.2411.03333>.

SOUZA, B. S. de; GUEDES, L. A. Application of graph theory in the analysis of query similarity and complexity in relational databases. Salvador, Brazil, p. 1–6, 2023. UFRN. Disponível em: <https://sbic.org.br/wp-content/uploads/2023/10/pdf/CBIC_2023_paper084.pdf>.

STEEN, M. van. **Graph Theory and Complex Networks: An Introduction**. The Netherlands: Maarten van Steen, 2010. Available at <https://www.amazon.com/dp/9081540610>. ISBN 9789081540612.

STOLL, M. A literature survey of matrix methods for data science. **GAMM-Mitteilungen**, Wiley-VCH GmbH, v. 43, n. 3, p. e202000013, 2020. ISSN 1522-2608. Open access funding enabled and organized by Projekt DEAL. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/gamm.202000013>.

TAKAPOUI, R.; BOYD, S. **Linear Programming Heuristics for the Graph Isomorphism Problem**. [S.l.], 2016. Available at <https://doi.org/10.48550/arXiv.1611.00711>.

WANG, S. *et al.* Oblivgm: Oblivious attributed subgraph matching as a cloud service. **IEEE Transactions on Information Forensics and Security**, IEEE, v. 17, p. 3583–3598, 2022.

WILSON, R. C.; ZHU, P. A study of graph spectra for comparing graphs and trees. **Pattern Recognition**, v. 41, n. 9, p. 2833–2841, 2008. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0031320308000927>.

XU, K. *et al.* How powerful are graph neural networks? In: **International Conference on Learning Representations (ICLR)**. [s.n.], 2019. Version 3, revised 22 Feb 2019. Disponível em: <https://arxiv.org/abs/1810.00826>.

YAZICI, A.; TAşKOMAZ, E. Bf-biggraph: An efficient subgraph isomorphism approach using machine learning for big graph databases. **Information Systems**, Elsevier, v. 117, p. 102401, 2024. Disponível em: <https://doi.org/10.1016/j.is.2024.102401>.

ZENG, Z. *et al.* Comparing stars: On approximating graph edit distance. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 2, n. 1, p. 25–36, 2009. Disponível em: <https://doi.org/10.14778/1687627.1687631>.

ZHANG, Q. *et al.* Stable subgraph isomorphism search in temporal networks. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 35, n. 6, p. 6405–6420, 2023. Disponível em: <https://doi.org/10.1109/TKDE.2022.3175800>.

ZHENG, W. *et al.* Efficient graph similarity search over large graph databases. **IEEE Transactions on Knowledge and Data Engineering**, v. 27, n. 4, p. 964–978, 2015. Disponível em: <https://ieeexplore.ieee.org/document/6880803>.

APÊNDICE A

Códigos

A.1 Geração de Grafos com Base no TPC-DS

O código abaixo mostra a função desenvolvida em Python para gerar cenários de benchmark com diferentes quantidades de tabelas e domínios, utilizando dados do TPC-DS. A função cria grafos dirigidos representando arquiteturas *Data Mesh* simuladas.

Listing A.1: Função de geração de grafos baseada no TPC-DS

```
import networkx as nx
import random
import os

def build_tpcds_graph_scenario(sor_count=2, domain_count=1, seed=42,
    save_gml=True, output_dir="benchmark_graphs"):
    random.seed(seed)
    G = nx.DiGraph()

    domain_templates = {
        "D1": {"label": "Customer", "SORs": ["SOR_customer", "
            SOR_customer_address", "SOR_customer_demographics"]},
        "D2": {"label": "Store", "SORs": ["SOR_store", "SOR_region", "
            SOR_nation", "SOR_call_center"]},
        "D3": {"label": "Catalog", "SORs": ["SOR_item", "SOR_promotion", "
            SOR_reason"]},
        "D4": {"label": "Time", "SORs": ["SOR_date_dim", "SOR_time_dim"]},
        "D5": {"label": "Warehouse", "SORs": ["SOR_warehouse", "
            SOR_ship_mode", "SOR_income_band"]}
    }

    all_sots = [
        "SOT_customer_attr", "SOT_customer_orders", "SOT_store_sales", "
            SOT_catalog_sales",
        "SOT_web_sales", "SOT_time_sales", "SOT_warehouse_stock"
    ]

    sot_sor_template = {
        "SOT_customer_attr": ["SOR_customer", "SOR_customer_address", "
            SOR_customer_demographics"],
        "SOT_customer_orders": ["SOR_customer", "SOR_item", "SOR_date_dim"
            ],
        "SOT_store_sales": ["SOR_store", "SOR_region", "SOR_promotion"],
        "SOT_catalog_sales": ["SOR_item", "SOR_promotion", "SOR_reason"],
```

```
        "SOT_web_sales": ["SOR_customer", "SOR_item", "SOR_time_dim"],
        "SOT_time_sales": ["SOR_date_dim", "SOR_time_dim"],
        "SOT_warehouse_stock": ["SOR_warehouse", "SOR_ship_mode", "
            SOR_income_band"]
    }

    all_specs = [
        "SPEC_customer_summary", "SPEC_store_sales_summary", "
            SPEC_catalog_performance",
        "SPEC_web_sales_summary", "SPEC_time_analysis", "
            SPEC_warehouse_logistics"
    ]

    spec_template_inputs = {
        "SPEC_customer_summary": [
            "SOT_customer_attr", "SOT_customer_orders",
            "SPEC_store_sales_summary", "SPEC_catalog_performance"
        ],
        "SPEC_store_sales_summary": [
            "SOT_store_sales", "SOT_customer_orders", "
                SPEC_customer_summary"
        ],
        "SPEC_catalog_performance": [
            "SOT_catalog_sales", "SOT_customer_attr", "SOT_store_sales"
        ],
        "SPEC_web_sales_summary": [
            "SOT_web_sales", "SOT_customer_orders", "SPEC_customer_summary"
        ],
        "SPEC_time_analysis": [
            "SOT_time_sales", "SOT_web_sales", "SOT_catalog_sales"
        ],
        "SPEC_warehouse_logistics": [
            "SOT_warehouse_stock", "SOT_store_sales", "
                SPEC_store_sales_summary"
        ]
    }

    all_sor_nodes, all_sot_nodes, all_spec_nodes = [], [], []
    domain_sors_dict, domain_sots_dict = {}, {}

    for d in range(1, domain_count + 1):
        domain_key = f"D{d}"
        template_sors = domain_templates[domain_key]["SORs"]
        selected_sors = []

        if sor_count <= len(template_sors):
            selected_sors = random.sample(template_sors, sor_count)
        else:
            selected_sors = template_sors.copy()
            extras_needed = sor_count - len(template_sors)
            for i in range(extras_needed):
                selected_sors.append(f"SOR_extra_{i+1}")
```

```
74
75          domain_sors = []
76          for sor in selected_sors:
77              node = f"{sor}_{domain_key}"
78              G.add_node(node, type="SOR")
79              domain_sors.append(node)
80              all_sor_nodes.append(node)
81
82          domain_sors_dict[domain_key] = domain_sors
83
84      for domain_key, sor_nodes in domain_sors_dict.items():
85          selected_sots = random.sample(all_sots, min(len(all_sots),
                sor_count))
86          domain_sots = []
87
88          for sot in selected_sots:
89              node = f"{sot}_{domain_key}"
90              G.add_node(node, type="SOT")
91              domain_sots.append(node)
92              all_sot_nodes.append(node)
93
94              template = sot_sor_template.get(sot, [])
95              matches = [f"{s}_{domain_key}" for s in template if f"{s}_{
                    domain_key}" in sor_nodes]
96              edges = matches[:2] if len(matches) >= 2 else random.sample(
                    sor_nodes, min(2, len(sor_nodes)))
97
98              for target in edges:
99                  G.add_edge(node, target)
100
101         domain_sots_dict[domain_key] = domain_sots
102
103     for domain_key, domain_sots in domain_sots_dict.items():
104         selected_specs = random.sample(all_specs, min(len(all_specs),
                sor_count))
105         for spec in selected_specs:
106             node = f"{spec}_{domain_key}"
107             G.add_node(node, type="SPEC")
108             all_spec_nodes.append(node)
109
110             reuse_key = random.choice([k for k in domain_sots_dict if k !=
                    domain_key]) \
111                         if domain_count > 1 and random.random() < 0.3 else
                            domain_key
112
113             input_keys = spec_template_inputs.get(spec, [])
114             candidates = [f"{i}_{reuse_key}" for i in input_keys if f"{i}_{
                    reuse_key}" in all_sot_nodes + all_spec_nodes]
115             if not candidates:
116                 candidates = random.sample(all_sot_nodes, 1)
117
118             for tgt in candidates:
```

```
                G.add_edge(node, tgt)

    if save_gml:
        os.makedirs(output_dir, exist_ok=True)
        filename = f"{output_dir}/graph_SOR{sor_count}_D{domain_count}_seed{seed}.gml"
        nx.write_gml(G, filename)
        print(f" Grafo salvo: {filename}")

    return G

def generate_all_benchmark_graphs():
    for sor in [2, 4, 8, 16]:
        for domain in [1, 2, 3, 4, 5]:
            print(f"Gerando grafo para SOR = {sor}, Domain = {domain}")
            build_tpcds_graph_scenario(sor_count=sor,domain_count=domain, seed=42)

generate_all_benchmark_graphs()
```

A.2 Execução dos algoritmos

Código para execução dos algoritmos VF2, Node Match e GNN. Além do treinamento do GNN.

Listing A.2: Treinamento GNN

```

import os
import json
import time
import pickle
import random
import networkx as nx
import pandas as pd
from itertools import combinations

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.data import Data
from torch_geometric.nn import global_mean_pool

# ================================
# MODELOS GNN
# ================================

class GINLayer(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(GINLayer, self).__init__()
        self.eps = nn.Parameter(torch.zeros(1))
```

```
        self.mlp = nn.Sequential(
            nn.Linear(in_channels, out_channels),
            nn.ReLU(),
            nn.Linear(out_channels, out_channels)
        )

    def forward(self, x, edge_index):
        row, col = edge_index
        agg = torch.zeros_like(x)
        agg.index_add_(0, row, x[col])
        out = self.mlp((1 + self.eps) * x + agg)
        return out

class SubgraphGNN(nn.Module):
    def __init__(self, in_channels=1, hidden_channels=64, out_channels=64):
        super(SubgraphGNN, self).__init__()
        self.gin1 = GINLayer(in_channels, hidden_channels)
        self.gin2 = GINLayer(hidden_channels, out_channels)

    def forward(self, x, edge_index, batch):
        x = self.gin1(x, edge_index)
        x = F.relu(x)
        x = self.gin2(x, edge_index)
        return global_mean_pool(x, batch)

class PairClassifier(nn.Module):
    def __init__(self, emb_size=64):
        super(PairClassifier, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(emb_size * 2, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, emb1, emb2):
        emb1 = emb1.view(1, -1)
        emb2 = emb2.view(1, -1)
        x = torch.cat([emb1, emb2], dim=1)
        return self.fc(x).squeeze(1)

# ================================
# funcoes auxiliares
# ================================

def is_valid_pair(u, v):
    """Retorna True apenas se ambos forem SPEC ou ambos forem SOT."""
    return (
        (str(u).startswith("SPEC") and str(v).startswith("SPEC")) or
        (str(u).startswith("SOT") and str(v).startswith("SOT"))
    )

def extract_subgraphs(G):
```

```
77      subgraphs = {}
78      for node in G.nodes:
79          neighbors = list(G.successors(node))
80          subgraphs[node] = G.subgraph([node] + neighbors).copy()
81      return subgraphs
82
83  def graph_to_pyg_data(G_nx):
84      mapping = {n: i for i, n in enumerate(G_nx.nodes)}
85      edge_index = torch.tensor([[mapping[u], mapping[v]] for u, v in G_nx.
            edges], dtype=torch.long).t().contiguous()
86      x = torch.ones((len(G_nx.nodes), 1))
87      return Data(x=x, edge_index=edge_index)
88
89  def create_datasets_by_scenario(graph_dir="benchmark_graphs",
        validation_dir="validations"):
90      scenario_datasets = {}
91      for filename in os.listdir(validation_dir):
92          if filename.endswith(".json") and filename.startswith("real_pairs_"
                ):
93              scenario_id = filename.replace("real_pairs_", "").replace(".
                    json", "")
94              gml_path = os.path.join(graph_dir, f"{scenario_id}.gml")
95
96              with open(os.path.join(validation_dir, filename)) as f:
97                  real_pairs = json.load(f)
98
99              G = nx.read_gml(gml_path)
100             subgraphs = extract_subgraphs(G)
101             nodes = list(subgraphs.keys())
102             real_set = set(tuple(sorted(p)) for p in real_pairs if
                    is_valid_pair(p[0], p[1]))
103
104             dataset = []
105
106             #  Positivos (pares reais)
107             for u, v in real_set:
108                 g1 = graph_to_pyg_data(subgraphs[u])
109                 g2 = graph_to_pyg_data(subgraphs[v])
110                 dataset.append((g1, g2, 1.0))
111
112             num_positives = len(real_set)
113             num_negatives = num_positives * 3
114
115             # Negativos (pares aleatorios nao-isomorfos)
116             negative_set = set()
117             attempts = 0
118             max_attempts = 5000
119             while len(negative_set) < num_negatives and attempts <
                    max_attempts:
120                 u, v = random.sample(nodes, 2)
121                 pair = tuple(sorted((u, v)))
122                 if is_valid_pair(u, v) and pair not in real_set and pair
```

```
                    not in negative_set:
123                     g1 = graph_to_pyg_data(subgraphs[u])
124                     g2 = graph_to_pyg_data(subgraphs[v])
125                     if g1.edge_index.numel() > 0 and g2.edge_index.numel()
                            > 0:
126                         dataset.append((g1, g2, 0.0))
127                         negative_set.add(pair)
128                 attempts += 1
129
130         print(f" Dataset para {scenario_id}: {len(real_set)} positivos,
                {len(negative_set)} negativos")
131         scenario_datasets[scenario_id] = dataset
132
133     return scenario_datasets
134
135
136 def train_and_save_gnn_model(dataset, model_path="modelos_gnn/gnn_model.pkl
    ", epochs=50, lr=0.01):
137     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
138     gnn = SubgraphGNN().to(device)
139     clf = PairClassifier().to(device)
140     optimizer = torch.optim.Adam(list(gnn.parameters()) + list(clf.
           parameters()), lr=lr)
141     criterion = nn.BCEWithLogitsLoss()
142
143     print(f" Treinando modelo: {model_path} com {len(dataset)} pares...")
144     for epoch in range(epochs):
145         random.shuffle(dataset)
146         total_loss = 0
147         correct = 0
148         total = 0
149         for g1, g2, label in dataset:
150             g1, g2 = g1.to(device), g2.to(device)
151             g1.batch = torch.zeros(g1.num_nodes, dtype=torch.long).to(
                    device)
152             g2.batch = torch.zeros(g2.num_nodes, dtype=torch.long).to(
                    device)
153             emb1 = gnn(g1.x, g1.edge_index, g1.batch).unsqueeze(0)
154             emb2 = gnn(g2.x, g2.edge_index, g2.batch).unsqueeze(0)
155             pred = clf(emb1, emb2)
156             loss = criterion(pred, torch.tensor([label], dtype=torch.float,
                     device=device))
157             optimizer.zero_grad()
158             loss.backward()
159             optimizer.step()
160             total_loss += loss.item()
161
162             # Acuracia simples
163             predicted_label = (torch.sigmoid(pred).item() >= 0.6)  #
                    threshold
164             if predicted_label == bool(label):
165                 correct += 1
```

```
166             total += 1
167
168         acc = 100 * correct / total if total > 0 else 0.0
169         print(f"Epoca {epoch+1}/{epochs} - Loss: {total_loss:.4f} -
                Acuracia: {acc:2f}%")
170
171     os.makedirs(os.path.dirname(model_path), exist_ok=True)
172     with open(model_path, "wb") as f:
173         pickle.dump((gnn.cpu(), clf.cpu()), f)
174     print(f" Modelo salvo: {model_path}")
175
176
177 def train_and_save_models_by_scenario(datasets_dict, output_base_dir="
    modelos_gnn_separados", epochs=2, lr=0.01):
178     os.makedirs(output_base_dir, exist_ok=True)
179     for scenario_id, dataset in datasets_dict.items():
180         model_path = os.path.join(output_base_dir, f"{scenario_id}.pkl")
181         train_and_save_gnn_model(dataset, model_path=model_path, epochs=
                epochs, lr=lr)
182
183 def predict_isomorphism_with_saved_gnn(G, model_path="modelos_gnn/gnn_model
    .pkl", threshold=0.3):
184     if not os.path.exists(model_path):
185         raise FileNotFoundError(f"Modelo GNN ausente em: {model_path}")
186     with open(model_path, "rb") as f:
187         gnn, clf = pickle.load(f)
188
189     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
190     gnn, clf = gnn.to(device), clf.to(device)
191
192     subgraphs = extract_subgraphs(G)
193     nodes = list(subgraphs.keys())
194     isomorphic_pairs = []
195
196     for u, v in combinations(nodes, 2):
197         g1_data = graph_to_pyg_data(subgraphs[u])
198         g2_data = graph_to_pyg_data(subgraphs[v])
199
200         if g1_data.edge_index.dim() < 2 or g1_data.edge_index.shape[1] ==
                0:
201             continue
202         if g2_data.edge_index.dim() < 2 or g2_data.edge_index.shape[1] ==
                0:
203             continue
204
205         g1 = g1_data.to(device)
206         g2 = g2_data.to(device)
207         g1.batch = torch.zeros(g1.num_nodes, dtype=torch.long).to(device)
208         g2.batch = torch.zeros(g2.num_nodes, dtype=torch.long).to(device)
209
210         with torch.no_grad():
211             emb1 = gnn(g1.x, g1.edge_index, g1.batch).unsqueeze(0)
```

```
212             emb2 = gnn(g2.x, g2.edge_index, g2.batch).unsqueeze(0)
213             score = torch.sigmoid(clf(emb1, emb2))
214             prinf(f"[DEBUG] {u} e {v}: score {score.item(}:.4f}")
215             if score.item() >= threshold:
216                 isomorphic_pairs.append((u, v))
217 
218     return isomorphic_pairs
219 
220 def predict_isomorphism_by_scenario(G, scenario_id, model_dir="
       modelos_gnn_separados", threshold=0.3):
221     model_path = os.path.join(model_dir, f"{scenario_id}.pkl")
222     return predict_isomorphism_with_saved_gnn(G, model_path=model_path,
           threshold=threshold)
223 
224 def run_isomorphism_on_benchmarks(
225     input_dir="benchmark_graphs",
226     output_dir="predicted_pairs",
227     algorithms=["VF2", "NodeMatch", "GNN"],
228     runs=1000,
229     time_log="execution_times.csv"
230 ):
231     os.makedirs(output_dir, exist_ok=True)
232 
233     for f in os.listdir(output_dir):
234         os.remove(os.path.join(output_dir, f))
235 
236     if "GNN" in algorithms:
237         datasets_by_scenario = create_datasets_by_scenario()
238         train_and_save_models_by_scenario(datasets_by_scenario)
239 
240     execution_data = []
241     gml_files = [f for f in os.listdir(input_dir) if f.endswith(".gml")]
242 
243     total_steps = len(gml_files) * len(algorithms)
244     current_step = 0
245 
246     for gml_file in gml_files:
247         gml_path = os.path.join(input_dir, gml_file)
248         scenario_id = gml_file.replace(".gml", "")
249         G = nx.read_gml(gml_path)
250 
251         for algo in algorithms:
252             current_step += 1
253             progress = (current_step / total_steps) * 100
254             print(f"\n Progresso: {progress:.2f}% ({current_step}/{
                   total_steps})")
255             print(f" Executando {algo} em {scenario_id}...")
256 
257             total_time = 0
258             all_pairs = set()
259 
260             for _ in range(runs if algo != "GNN" else 1):
```

```
261                 start = time.time()
262                 if algo == "GNN":
263                     pairs = predict_isomorphism_by_scenario(G, scenario_id)
264                 else:
265                     pairs = run_isomorphism(G, algorithm=algo)
266                 end = time.time()
267                 total_time += (end - start)
268                 all_pairs.update(tuple(sorted(p)) for p in pairs)
269 
270             avg_time = total_time / (1 if algo == "GNN" else runs)
271             filename = f"{output_dir}/pairs_{scenario_id}_{algo}.json"
272             with open(filename, "w") as f:
273                 json.dump(sorted(list(all_pairs)), f, indent=4)
274 
275             execution_data.append({
276                 "scenario": scenario_id,
277                 "algorithm": algo,
278                 "runs": 1 if algo == "GNN" else runs,
279                 "avg_time_seconds": round(avg_time, 6)
280             })
281             print(f"Resultado salvo: {filename} | tempo medio: {avg_time:.6
                    f}s")
282 
283     df = pd.DataFrame(execution_data)
284     df.to_csv(time_log, index=False)
285     print(f"\n Tabela de tempos salva em: {time_log}")
286 
287 def run_isomorphism(G, algorithm="VF2"):
288     subgraphs = [(node, G.subgraph([node] + list(G.successors(node)))) for
            node in G.nodes]
289     isomorphic_pairs = []
290     for i in range(len(subgraphs)):
291         for j in range(i + 1, len(subgraphs)):
292             if algorithm == "VF2":
293                 if nx.is_isomorphic(subgraphs[i][1], subgraphs[j][1]):
294                     isomorphic_pairs.append((subgraphs[i][0], subgraphs[j
                            ][0]))
295             elif algorithm == "NodeMatch":
296                 if nx.is_isomorphic(subgraphs[i][1], subgraphs[j][1],
                        node_match=lambda x, y: x == y):
297                     isomorphic_pairs.append((subgraphs[i][0], subgraphs[j
                            ][0]))
298     return isomorphic_pairs
299 
300 
301 #  Rodar
302 run_isomorphism_on_benchmarks()
```