UNIVERSIDADE FEDERAL DE CATALÃO
INSTITUTO DE MATEMÁTICA E TECNOLOGIA
BACHARELADO EM MATEMÁTICA INDUSTRIAL

ISABELLA COSTA MACHADO BENTO

DO DFG À REDE DE PETRI:
DESCOBERTA, CONFORMIDADE E ANÁLISE DE DESEMPENHO EM LOGS REAIS DE PROCESSOS

MONOGRAFIA DE FINAL DE CURSO

CATALÃO – GO, 2026

UNIVERSIDADE FEDERAL DE CATALÃO

INSTITUTO DE MATEMÁTICA E TECNOLOGIA

Av. Dr. Lamartine Pinto de Avelar, número 1120, - Bairro Setor Universitário, Catalão/GO, CEP 75704-020
Telefone: - - https://www.ufcat.edu.br

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA)

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES, DISSERTAÇÕES E TRABALHOS DE CONCLUSÃO DE CURSO NO REPOSITÓRIO INSTITUCIONAL DA UNIVERSIDADE FEDERAL DE CATALÃO (UFCAT)

Na qualidade de titular dos diretos de autor, autorizo a Universidade Federal de Catalão (UFCAT) a disponibilizar, gratuitamente, por meio do Repositório Institucional da Universidade Federal de Catalão (RI/UFCAT), sem ressarcimento dos direitos autorais, de acordo com a Lei 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses, Dissertações e Trabalhos de Conclusão de Curso disponibilizado no RI/UFCAT é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o(a) autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico: Trabalho de Conclusão de Curso

2. Nome completo do autor:

Nome completo do(a) orientador(a):

3. Título do trabalho

DO DFG À REDE DE PETRI: Descoberta, Conformidade e Análise de Desempenho em Logs Reais de Processos

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento: [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:

a) consulta ao(à) autor(a) e ao(à) orientador(a);

b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.

O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Documento assinado eletronicamente por **Isabella Costa Machado Bento**, **Usuário Externo**, em 01/04/2026, às 11:13, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **VASTON GONCALVES DA COSTA**, **Professor(a) do Magistério Superior**, em 02/04/2026, às 08:01, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufcat.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0316873** e o código CRC **FA435298**.

Referência: Processo nº 23852.006305/2025-66 SEI nº 0316873

ISABELLA COSTA MACHADO BENTO

DO DFG À REDE DE PETRI:
DESCOBERTA, CONFORMIDADE E ANÁLISE DE DESEMPENHO EM LOGS REAIS DE PROCESSOS

Monografia apresentada como requisito parcial para a obtenção do título de Bacharel em Matemática Industrial pela Universidade Federal de Catalão.

Orientador:
Vaston G. Costa

CATALÃO – GO

2026

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFCAT.

Bento, Isabella Costa Machado
Do DFG à rede de Petri [manuscrito] : descoberta, conformidade e análise de desempenho em logs de processos reais / Isabella Costa Machado Bento. - 2026.
cxxxiv, 134 f.

Orientador: Prof. Dr. Vaston Gonçalves da Costa.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Catalão, Instituto de Matemática e Tecnologia, Matemática, Catalão, 2026.
Bibliografia. Anexos. Apêndice.
Inclui gráfico, tabelas, algoritmos, lista de tabelas.

1. Mineração de processos. 2. Log de eventos. 3. Análise de dados. 4. Redes de Petri. 5. Directly-follows-graph. I. Costa, Vaston Gonçalves da, orient. II. Título.

CDU 51

UFCAT

UNIVERSIDADE FEDERAL DE CATALÃO

INSTITUTO DE MATEMÁTICA E TECNOLOGIA

ATA DA SESSÃO PÚBLICA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO PARA OBTENÇÃO DO TÍTULO DE BACHAREL EM MATEMÁTICA INDUSTRIAL, A QUE SE SUBMETEU A ALUNA ISABELLA COSTA MACHADO BENTO, ORIENTADA PELO PROF. DR. VASTON GONÇALVES DA COSTA.

Aos nove dias do mês de fevereiro do ano de dois mil e vinte e seis, às quatorze horas, por web conferencia (https://conferenciaweb.rnp.br/sala/vaston-2), reuniu-se a Comissão Examinadora da defesa em epígrafe indicada pela Direção do Instituto de Matemática e Tecnologia, composta pelos: Presidente e Orientador Prof. Dr. Vaston Gonçalves da Costa, Prof. Dr. Marcelo Henrique Stoppa e Prof. M.e. José Salviano Borges, para analisar o trabalho da candidata Isabella Costa Machado Bento**,** apresentado sob o título "DO DFG À REDE DE PETRI: Descoberta, Conformance e Análise de Desempenho em Logs Reais de Processos". O Presidente declarou abertos os trabalhos, a seguir a candidata apresentou o seu trabalho e foi arguida pela Comissão Examinadora. Terminada a exposição e a arguição, a Comissão reuniu-se e deliberou pelo seguinte resultado: **APROVADA**, com nota 9,5 (nove pontos e meio).

Para fazer jus ao título de Bacharel em Matemática Industrial, a versão final do Trabalho de Conclusão de Curso, considerada Aprovada, deverá ser entregue à professora da disciplina até o dia vinte e quatro de fevereiro de dois mil e vinte e seis.

Nada mais havendo a tratar, o Senhor Presidente declara a sessão encerrada, sendo a ata assinada pelos Senhores Membros da Comissão Examinadora.

Documento assinado eletronicamente por **VASTON GONCALVES DA COSTA**, **Professor(a) do Magistério Superior**, em 09/02/2026, às 15:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **MARCELO HENRIQUE STOPPA**, **Professor(a) do Magistério Superior**, em 09/02/2026, às 15:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **José Salviano Borges**, **Técnico(a) de Laboratório**, em 09/02/2026, às 15:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufcat.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0289273** e o código CRC **FBCABA5F**.

Dedico este trabalho à minha família, aos meus amigos e a todas as pessoas que estiveram ao meu lado, oferecendo apoio, incentivo e confiança ao longo da minha trajetória acadêmica.

Agradecimentos

A Deus, por me conceder força, sabedoria e perseverança ao longo de toda essa trajetória acadêmica.

À minha mãe, Joana da C. Lapa, e ao meu pai, Roberto M. Bento, pelo amor incondicional, pelo apoio constante e por acreditarem em mim mesmo nos momentos mais desafiadores. Aos meus irmãos, Duda e Betinho, pelo carinho, incentivo e companheirismo ao longo dessa caminhada.

À minha sobrinha Antonella, a quem me refiro como minha parceira, por ser fonte de alegria, motivação e inspiração nos momentos em que mais precisei.

À minha tia Lila Bento, pelo apoio fundamental e por ter me acompanhado na minha primeira vinda a Catalão, oferecendo cuidado, incentivo e segurança em um momento tão importante da minha vida.

À minha companheira Medea, N., pelo amor, paciência, compreensão e apoio diário, que foram essenciais para que eu conseguisse seguir firme até a conclusão deste trabalho.

Aos meus amigos, pela amizade, apoio, companheirismo e pelos momentos de incentivo e descontração, que tornaram essa jornada mais leve e significativa.

Ao meu orientador Vaston, pela orientação, dedicação, paciência e pelas contribuições valiosas que enriqueceram este trabalho e minha formação acadêmica.

À Universidade Federal de Catalão, por proporcionar um ambiente de aprendizado, crescimento acadêmico e desenvolvimento profissional, contribuindo significativamente para a minha formação.

Por fim, a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho, deixo aqui o meu sincero agradecimento.

“O que importa na vida não é o ponto de partida, mas a caminhada.”

-Cora Coralina

RESUMO

BENTO, I. C. M. *DO DFG À REDE DE PETRI:*
Descoberta, Conformidade e Análise de Desempenho em Logs Reais de Processos. 2026. 132 f. Trabalho Final de Curso (Graduação) – Instituto de Matemática e Tecnologia, Universidade Federal de Catalão, Catalão – GO.

Este Trabalho de Conclusão de Curso investiga a aplicação de técnicas de Mineração de Processos em contextos institucionais, propondo um *pipeline* reprodutível para extração, curadoria e transformação de registros administrativos em *logs* de eventos analisáveis, com ênfase na modelagem inicial via *Directly-Follows Graph* (DFG). Discutem-se desafios típicos de bases reais, como heterogeneidade, incompletude e ausência de identificadores consistentes, bem como a possibilidade de adaptação do procedimento às demais bases descritas no capítulo de bases de dados. Embora o estudo de caso na base da Central de Emendas fosse o objetivo inicial, restrições temporais e exigências de credenciamento impossibilitaram o acesso aos dados no período letivo, postergando esta etapa para trabalhos futuros. Como desdobramentos, apontam-se a aplicação integral sob credenciamento, a institucionalização de rotinas de curadoria/anonimização e o avanço para modelos mais expressivos, como Redes de Petri, incluindo perspectivas de verificação formal com raciocinadores e provadores de teoremas.

Palavras-chaves: Mineração de Processos, *Log* de eventos, *Directly-Follows Graph*, Redes de Petri, Curadoria de dados, Governança de dados.

ABSTRACT

This undergraduate final project investigates the application of Process Mining techniques in institutional settings by proposing a reproducible pipeline for extracting, curating, and transforming administrative records into analyzable event logs, with an initial modeling stage based on the Directly-Follows Graph (DFG). The work discusses typical challenges of real-world datasets – such as heterogeneity, incompleteness, and the lack of consistent identifiers – and emphasizes how the proposed procedure can be adapted to other datasets described in the databases chapter.Although the case study using the Central de Emendas database was the initial goal, time constraints and strict accreditation requirements prevented data access within the project timeframe, leaving this step for future continuation. As next directions, the thesis outlines a complete application under accreditation, the institutionalization of curation/anonymization routines, and the move toward more expressive models such as Petri nets, including perspectives on formal verification with automated reasoners and theorem provers.

Keywords: Process Mining, Event log, Directly-Follows Graph, Petri nets, Data curation, Data governance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

1 Introdução

A Mineração de Processos ocupa, hoje, um espaço singular entre a gestão e a ciência de dados: ela transforma registros de execução, conhecidos como logs de evento, em evidências sobre como os processos realmente acontecem, permitindo descrever fluxos, verificar conformidade e orientar melhorias com base em dados observáveis (AALST, 2019). Em contextos organizacionais e públicos, esse movimento é particularmente relevante: quanto maior a dependência de sistemas digitais (protocolos, atendimentos, trâmites, portais de transparência), maior também o volume de eventos disponíveis para análise. E consequentemente, maior a oportunidade de produzir diagnósticos auditáveis, reprodutíveis e tecnicamente justificáveis.

Na prática, contudo, a transição entre "visualizar" um processo e "explicá-lo formalmente" não é trivial. Representações de alta legibilidade, como os *Directly-Follows Graphs* (DFGs), são úteis para um primeiro entendimento do fluxo observado, destacando sequências frequentes e caminhos alternativos. Entretanto, sua natureza essencialmente descritiva impõe limitações quando se busca capturar concorrência/paralelismo, interpretar estruturas de controle com semântica operacional e, sobretudo, sustentar análises robustas de conformidade e desempenho (LEEMANS; POPPE; WYNN, 2019). Em outras palavras: o DFG é um excelente mapa inicial, mas não é, por si só, uma "máquina" capaz de simular, validar e explicar desvios com rigor.

É nesse ponto que modelos formais, como Redes de Petri, tornam-se centrais. Ao introduzirem estados, marcações e regras de disparo, esses modelos fornecem semântica explícita para o comportamento do processo, viabilizando técnicas consolidadas de análise de conformidade e análises quantitativas de desempenho (ex.: tempos de espera, gargalos, sincronizações) (AALST, 2019; LEEMANS; POPPE; WYNN, 2019). Assim, um caminho metodológico consistente consiste em iniciar pela leitura exploratória e avançar para modelos formais, preservando a comunicabilidade do primeiro sem abrir mão do poder analítico do segundo.

Este trabalho organiza essa passagem como um *pipeline* replicável, com etapas claras de preparação do log, descoberta de modelos, avaliação por métricas e extração de evidências. No desenvolvimento empírico, adota-se um estudo de caso didático com logs de *helpdesk* (tickets), adequado para demonstrar o fluxo de ponta a ponta: da construção e leitura de DFGs até a descoberta e avaliação de Redes de Petri com métricas consolidadas. As bases clássicas (4TU e *BPI Challenges*) e o cenário institucional da Central das Emendas são discutidos como referência e horizonte de aplicação, preservando a coerência com o escopo experimental efetivamente executado no texto.

1.1 Problema, justificativa e objetivos

Esta seção delimita o recorte científico da pesquisa e explicita, com precisão, o porquê e o para quê do estudo. Partindo das lacunas discutidas na literatura, formaliza-se um problema de pesquisa associado às limitações representacionais e diagnósticas dos DFGs, quando aplicados a logs reais, em especial diante de paralelismo e do efeito de filtros de frequência e à necessidade de migrar para modelos com semântica formal, como Redes de Petri, para viabilizar análises robustas de conformidade e desempenho.

1.1.1 Problema de Pesquisa

Como desenvolver e avaliar um pipeline replicável de mineração de processos que, partindo de representações intuitivas (DFGs), úteis como visualização baseline, mas limitadas para capturar concorrência e oferecer diagnóstico formal, evolua para modelos com semântica explícita (Redes de Petri), permitindo (i) análise de conformidade e (ii) análise de desempenho em logs de eventos reais.

O problema emerge da necessidade de transitar, de modo confiável e reprodutível, entre legibilidade e expressividade: preservar o valor comunicativo do DFG, sem perder a capacidade analítica exigida em cenários ruidosos e com múltiplas variantes.

1.1.2 Justificativa Científica

Do ponto de vista científico, a pesquisa justifica-se pela consolidação de um *pipeline* reprodutível que articula o papel dos DFGs como representação inicial (*baseline*) e o papel das Redes de Petri como modelo formal, apto a suportar métricas e diagnósticos de conformidade/desempenho.

Do ponto de vista aplicado, o estudo é motivado pela necessidade de produzir resultados acionáveis: identificar gargalos, desvios e oportunidades de padronização a partir de evidências observadas no log. No estudo de caso de helpdesk, por exemplo, o *pipeline* conduz à obtenção de métricas e sinais quantitativos de desalinhamento e atraso em atividades críticas (ex.: tempos médios elevados em etapas de espera e escalonamento), evidenciando o potencial do método para subsidiar decisões de melhoria contínua.

Finalmente, do ponto de vista institucional e social, a discussão sobre bases públicas e governamentais (como a Central das Emendas) reforça o valor estratégico de métodos transparentes e auditáveis para análise de processos em contextos de interesse público.

1.1.3 Objetivo Geral

Desenvolver e avaliar um pipeline replicável de mineração de processos, iniciando com a obtenção de DFGs como visualização baseline e avançando para a descoberta e análise

de modelos com semântica formal (Redes de Petri), de modo a viabilizar análises robustas de conformidade e desempenho em logs de eventos reais, com foco em reprodutibilidade e evidências quantitativas.

1.1.4 Objetivos Específicos

- Construir e analisar DFGs (frequência e/ou desempenho) para caracterização do fluxo observado e identificação do *happy path* e de caminhos alternativos.
- Investigar o impacto de filtros de frequência sobre a legibilidade do DFG e sobre a visibilidade de comportamentos menos frequentes.
- Descobrir modelos em Redes de Petri por meio de algoritmos clássicos e amplamente utilizados (ex.: *Alpha Miner, Inductive Miner, Heuristics Miner*), visando capturar estruturas não representadas adequadamente em DFGs.
- Avaliar os modelos descobertos por métricas consolidadas de qualidade (ex.: *fitness, precision, generalization* e *simplicity*) e extrair evidências de conformidade e desempenho.
- Identificar gargalos e tempos de espera associados a atividades críticas, discutindo implicações gerenciais a partir dos achados.
- Discutir, de forma prospectiva, a extensão do *pipeline* a bases públicas mais complexas (ex.: *BPI Challenges*) e a bases institucionais/governamentais (ex.: Central das Emendas), apontando desafios e requisitos de dados.

1.2 Estrutura do Trabalho

O restante do documento está organizado da seguinte forma. No Capítulo 2 apresenta-se o referencial teórico, consolidando os fundamentos de Mineração de Processos, DFGs e Redes de Petri, bem como conceitos necessários para conformidade e análise de desempenho. Em seguida, o Capítulo 3 discute a revisão de literatura, situando o trabalho frente a abordagens clássicas e recentes e destacando lacunas que motivam o estudo. O Capítulo 4 descreve a metodologia adotada e o delineamento do pipeline, incluindo ferramentas e critérios de avaliação. No Capítulo 5 caracteriza-se a base utilizada e seu enquadramento em repositórios e desafios clássicos da área.

A etapa empírica inicia-se no Capítulo 6, que apresenta a análise exploratória baseada em DFGs, incluindo variantes, filtros e leitura de frequência/desempenho. O Capítulo 7 discute a transição do *DFG* para modelos executáveis, fundamentando por que Redes de Petri ampliam as possibilidades de verificação e diagnóstico. Na sequência, o capítulo de

implementação detalha a aplicação dos algoritmos de descoberta e das métricas de conformidade e desempenho em Redes de Petri. O Capítulo 9 consolida a ponte metodológica entre a exploração por DFG e a análise formal, destacando os artefatos gerados e sua utilidade para reprodutibilidade. Por fim o capítulo 10. apresenta-se a conclusão, sintetizando resultados, limitações e direções futuras.

2 Referencial Teórico

Este capítulo consolida os fundamentos conceituais e formais que sustentam as análises desenvolvidas ao longo do trabalho. Inicialmente, apresentamos a Mineração de Processos como o elo entre dados observados em logs de eventos e a gestão de processos, enfatizando suas três capacidades centrais: descoberta, conformidade e aprimoramento (AALST, 2011; AALST *et al.*, 2011). Em seguida, introduzimos a teoria dos grafos como base matemática para representar relações de precedência e dependência, preparando o terreno para o *Directly-Follows Graph* (DFG) como representação exploratória intermediária. Por fim, avançamos para as Redes de Petri, formalismo que adiciona semântica operacional e possibilita verificação de propriedades e checagem de conformidade com maior rigor (MURATA, 1989; ROZINAT; van der Aalst, 2008). Esse encadeamento teórico é conectado ao contexto empírico da Central de Emendas, que fornece um cenário realista para observar, modelar e avaliar processos administrativos a partir de dados registrados.

2.1 Mineração de Processos

Segundo Wil van der Aalst (AALST, 2011), amplamente reconhecido como o precursor da disciplina, a mineração de processos atua como o "elo perdido"entre a ciência de dados (mineração de dados, aprendizado de máquina) e o gerenciamento de processos de negócio tradicional (BPM). O autor defende que a técnica permite descobrir, monitorar e melhorar processos reais a partir de fatos registrados em logs de eventos, em vez de se basear em modelos idealizados ou "ficções"criadas manualmente. Aalst estabelece que o insumo fundamental para qualquer análise de mineração são os dados de eventos, que devem referenciar obrigatoriamente um identificador de caso, uma atividade e um registro de data/hora (timestamp). Com base nesses registros, a disciplina subdivide-se em três capacidades essenciais (IBM, s.d.; AALST *et al.*, 2011):

1. Descoberta: produz um modelo de processo a partir de um log bruto, sem o uso de informações a priori.

2. Conformidade: compara um modelo de processo existente com o log para identificar onde a realidade diverge do que foi planejado, permitindo auditorias e verificações de conformidade.

3. Aprimoramento: busca estender ou melhorar o modelo original integrando informações de desempenho extraídas do log, como gargalos, tempos de espera e taxas de serviço.

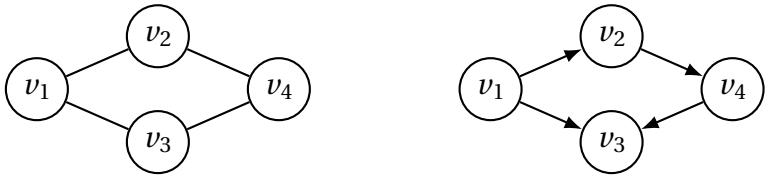
2.2 Grafos

Grafos são estruturas matemáticas empregadas para modelar relações entre objetos ou entidades. De forma formal, um grafo é definido como um par ordenado $G = (V, E)$, em que V representa um conjunto finito de vértices e E um conjunto de arestas, responsáveis por estabelecer conexões entre pares de vértices (WALLIS, 2007). Diferentes tipos de relações, sociais, computacionais, físicas ou organizacionais, podem ser representadas por meio de conexões entre elementos, permitindo análises estruturais e algorítmicas sobre conectividade, caminhos e padrões recorrentes.

Conforme a natureza dessas conexões, os grafos podem ser classificados como não direcionados, quando as arestas não apresentam orientação, ou direcionados, quando existe um sentido definido entre os vértices, caracterizando relações de origem e destino (WALLIS, 2007). A Figura 2.1 exemplifica essas duas categorias.

Além disso, grafos podem ser ponderados, nos quais valores numéricos são associados às arestas para representar custos, distâncias ou intensidades de ligação (BONDY; MURTY, 2008). A Figura 2.2 ilustra um grafo ponderado, no qual os pesos indicam uma medida quantitativa associada às conexões.

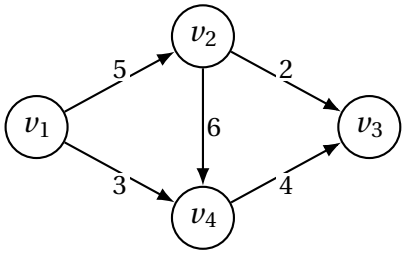
Figura 2.1 – Exemplos ilustrativos de grafos: (a) não direcionado e (b) direcionado.



(a) Grafo não direcionado (b) Grafo direcionado

Fonte: Elaboração própria com base em (WALLIS, 2007).

Figura 2.2 – Exemplo de grafo ponderado, em que os pesos nas arestas representam uma medida quantitativa (por exemplo, custo, distância ou tempo).



Fonte: Elaboração própria com base em (WALLIS, 2007).

A teoria dos grafos possui ampla aplicabilidade em áreas como ciência da computação, engenharia e análise de processos, sendo empregada na representação de redes de comunicação, sistemas logísticos, cadeias de dependência e fluxos de atividades. Sua relevância decorre da capacidade de abstrair sistemas complexos em estruturas matemáticas simples e formalmente analisáveis (WEST, 2001).

No contexto da Mineração de Processos, os grafos assumem papel central ao fornecerem o arcabouço estrutural necessário para representar a dinâmica de execução dos processos organizacionais a partir de logs de eventos. Atividades são modeladas como vértices, enquanto relações de precedência, dependência ou causalidade entre eventos são expressas por meio de arestas direcionadas e, frequentemente, ponderadas por métricas de frequência ou tempo. Essa abordagem gráfica constitui a base conceitual para os DFGs, amplamente utilizados em Mineração de Processos como uma representação intermediária entre os logs de eventos e modelos formais mais expressivos, como Redes de Petri.

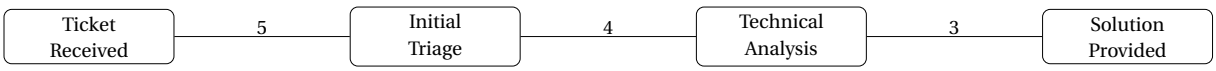
2.3 Directly-Follows Graph (DFG)

O *Directly-Follows Graph* (DFG), ou Grafo de Sequências Diretas, é uma representação gráfica amplamente utilizada na área de Mineração de Processos para modelar o comportamento observado em logs de eventos. Nesse tipo de grafo, os vértices representam atividades de um processo, enquanto as arestas direcionadas indicam que uma atividade ocorre diretamente após outra em pelo menos um caso registrado no log (AALST, 2016).

Um DFG pode ser formalmente definido como um grafo direcionado $G = (V, E)$, em que V corresponde ao conjunto de atividades distintas identificadas no log de eventos e $E \subseteq V \times V$ representa as relações de sucessão direta entre atividades. Uma aresta $(v_i, v_j) \in E$ existe se, em alguma instância do processo, a atividade v_j ocorre imediatamente após v_i. Essas arestas podem ser ponderadas por métricas quantitativas, como frequência de ocorrência ou tempo médio de transição, enriquecendo a análise do fluxo processual.

A Figura 2.3 ilustra um exemplo típico de DFG, no qual as arestas representam exclusivamente relações de precedência direta extraídas do log, sem a imposição de estruturas de controle como paralelismo explícito, sincronização ou condições de disparo. Essa característica distingue o DFG de modelos formais mais expressivos, como Redes de Petri.

Figura 2.3 – Exemplo de Directly-Follows Graph (DFG)

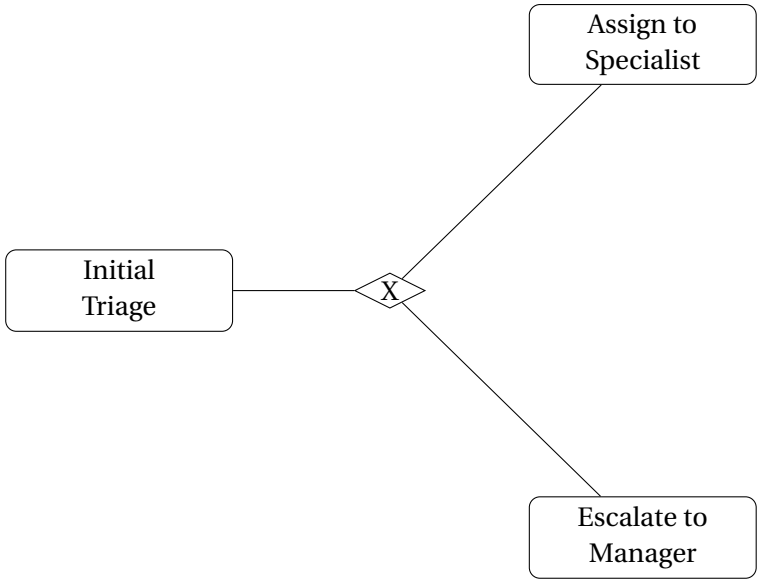


Arestas indicam sucessão direta observada no log, ponderadas pela frequência de ocorrência.

Fonte: Elaboração própria.

Por outro lado, nem toda representação gráfica de um processo constitui um DFG. A Figura 2.4 apresenta um exemplo de grafo que não se enquadra como DFG, pois incorpora elementos semânticos adicionais — como decisões explícitas, gateways lógicos ou dependências não diretamente observáveis no log — que extrapolam a definição estrita de sucessão direta entre atividades.

Figura 2.4 – Exemplo de representação que não caracteriza um DFG



Ao introduzir explicitamente um nó de decisão, o modelo passa a representar escolhas estruturais do processo, enquanto o DFG se limita a capturar relações de precedência direta efetivamente observadas nos registros de execução.

Fonte: Elaboração própria.

Devido à sua simplicidade estrutural e forte aderência empírica aos dados observados, o DFG é amplamente utilizado como uma etapa intermediária na descoberta de modelos de processo. Ele favorece a análise exploratória inicial, a identificação de padrões dominantes, variantes frequentes e possíveis desvios, além de servir como base para algoritmos mais avançados de descoberta, como aqueles que produzem Redes de Petri ou Árvores de Processo (LEEMANS; POPPE; WYNN, 2019; AALST, 2016).

2.4 Redes de Petri no Contexto da Mineração de Processos

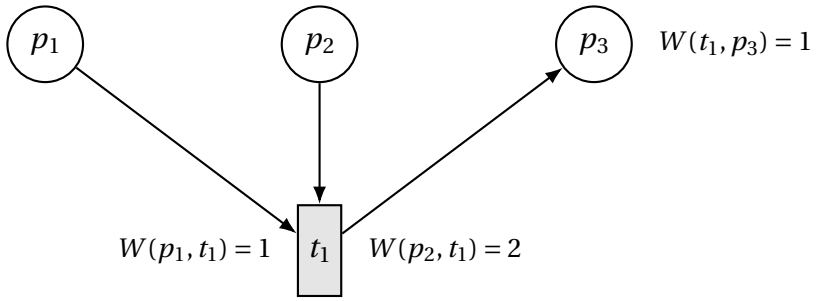
As Redes de Petri constituem um formalismo matemático e gráfico amplamente empregado para a modelagem, análise e verificação de sistemas a eventos discretos, sendo particularmente adequadas para representar processos caracterizados por concorrência, sincronização, paralelismo e compartilhamento de recursos. Esse formalismo foi introduzido por Carl Adam Petri em 1962 e consolidou-se como uma das principais ferramentas para análise formal de processos complexos (PETRI, 1962; MURATA, 1989).

Uma Rede de Petri clássica é definida como uma quíntupla $PN = (P, T, F, W, M_0)$, em que P representa um conjunto finito de lugares, T um conjunto finito de transições, $F \subseteq (P \times T) \cup (T \times P)$ o conjunto de arcos que conectam lugares e transições, W uma função de pesos associada aos arcos, e M_0 a marcação inicial do sistema. Os lugares podem conter marcas (*tokens*), cuja distribuição define o estado corrente do processo modelado, permitindo

uma representação explícita tanto da estrutura do processo quanto de sua dinâmica de execução (MURATA, 1989).

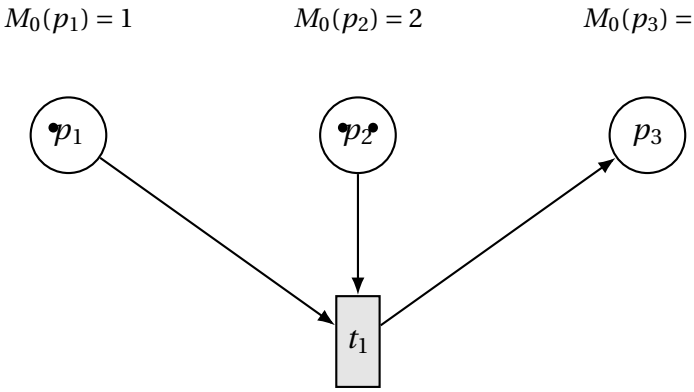
A Figura 2.5 ilustra a estrutura básica de uma Rede de Petri, destacando os conjuntos de lugares, transições e a relação de fluxo entre eles, enquanto a Figura 2.6 exemplifica o conceito de marcação inicial e a dinâmica de disparo das transições.

Figura 2.5 – Exemplo de Rede de Petri destacando os conjuntos de lugares (P), transições (T) e a relação de fluxo (F), com pesos de arco (W) ilustrativos.



Fonte: Elaboração própria com base em (MURATA, 1989)

Figura 2.6 – Exemplo ilustrativo de marcação inicial M_0 por meio de *tokens*.



A habilitação e o disparo de uma transição alteram a marcação, caracterizando a dinâmica da Rede de Petri.

Fonte: Elaboração própria com base em (MURATA, 1989)

A evolução do sistema ocorre por meio do disparo das transições, que estão habilitadas quando todos os seus lugares de entrada contêm um número suficiente de marcas, de acordo com os pesos definidos nos arcos correspondentes. Ao disparar, a transição consome marcas dos lugares de entrada e produz marcas nos lugares de saída, resultando em uma nova marcação do sistema (PETERSON, 1981).

No contexto da Mineração de Processos, as Redes de Petri assumem um papel central como modelo de referência para a representação de processos descobertos a partir de logs de eventos. Algoritmos clássicos, como o*Alpha Miner e o Inductive Miner*, produzem Redes de Petri a partir de dados reais de execução, permitindo a visualização do fluxo do processo e sua verificação formal (AALST, 2016).

Além da descoberta de processos, as Redes de Petri são fundamentais nas análises de conformidade por meio da técnica de *token replay*, permitindo quantificar o grau de aderência entre o comportamento observado e o comportamento permitido pelo modelo (ROZINAT; van der Aalst, 2008). Sua estrutura formal também viabiliza análises de desempenho, possibilitando a identificação de gargalos e a quantificação de tempos médios de espera.

A discussão mais aprofundada sobre o emprego de Redes de Petri na mineração de processos será apresentada na Seção 7. A exposição detalhada das capacidades analíticas das Redes de Petri, após a apresentação prática do uso de DFGs no exemplo do *helpdesk*, permitirá ao leitor compreender com maior clareza as distinções conceituais e as vantagens operacionais oferecidas por essa abordagem. A análise comparativa subsequente, fundamentada na terminologia e nos conceitos práticos já estabelecidos, evidenciará de forma concreta como as Redes de Petri ampliam e aprofundam as possibilidades analíticas em relação aos DFGs.

2.5 Central de Emendas

A Central de Emendas é uma plataforma governamental desenvolvida para centralizar e disponibilizar informações referentes às emendas parlamentares (Central das Emendas, 2025), abrangendo desde a sua proposição até a execução orçamentária e financeira. Por registrar de forma sistemática as etapas do ciclo de vida das emendas, esse sistema constitui uma fonte relevante de dados para análises baseadas em evidências, especialmente no contexto da Mineração de Processos.

No caso da Central de Emendas, cada emenda pode ser interpretada como um caso, enquanto as etapas administrativas correspondem às atividades registradas ao longo do tempo. Dessa forma, os dados disponíveis na plataforma atendem aos requisitos fundamentais para a aplicação de técnicas de Mineração de Processos, tais como identificador de caso, nome da atividade e carimbo temporal.

A partir dos logs extraídos da Central de Emendas, é possível empregar modelos baseados em grafos para a análise exploratória dos processos. O DFG permite visualizar as relações de sucessão direta entre as etapas do processo de execução das emendas, evidenciando os caminhos mais frequentes, variações no fluxo e possíveis gargalos administrativos, servindo como base para análises mais aprofundadas.

Para análises formais e verificações estruturais, os grafos derivados dos logs podem ser transformados em modelos de Redes de Petri, possibilitando representar paralelismo, sincronização e dependências entre atividades, além de permitir a verificação de propriedades como conformidade com normas legais, identificação de bloqueios e avaliação da eficiência do processo.

Nesse contexto, a integração entre a Central de Emendas e a Mineração de Processos configura um arcabouço metodológico robusto para o estudo da gestão pública, contribuindo para o aumento da transparência, a identificação de ineficiências e o aprimoramento dos processos administrativos relacionados à execução das emendas parlamentares.

3 Revisão de Literatura

Este capítulo apresenta a revisão de literatura relacionada ao tema investigado, com o objetivo de situar o leitor no campo científico, evidenciar abordagens predominantes e delimitar o recorte adotado no estudo. A revisão não se limita à descrição de trabalhos isolados, mas busca articular criticamente as contribuições existentes, estabelecendo relações entre teorias, métodos e resultados. Inicialmente, são discutidos os principais conceitos e definições associados ao tema, conforme apresentados na literatura especializada, destacando-se sua relevância e evolução no contexto científico.

3.1 Abordagens Teóricas e Metodológicas na Literatura

A produção científica sobre o tema pode ser organizada segundo diferentes abordagens teóricas e metodológicas, abrangendo desde representações iniciais acessíveis e intuitivas, como o *Directly-Follows Graph* (DFG), até a transição para modelos com semântica formal, como as Redes de Petri, utilizando algoritmos como o *Inductive Miner* e o *Heuristics Miner*. Outros estudos privilegiam o desenvolvimento de extensões recentes e algoritmos de fronteira, como o *Inductive Miner* translúcido, abordagens híbridas e técnicas genéticas, visando lidar com ruído, incertezas, comportamento raro e a incompletude de logs organizacionais reais.

Parte dos estudos adota uma perspectiva na estruturação de modelos por meio da frequência e da causalidade, enfatizando o uso de frequências de relações de atividade translúcidas para filtrar ruído e garantir modelos mais robustos e generalizáveis (BEYEL; AALST, 2025). Outros trabalhos concentram-se na descoberta de modelos DFG acíclicos por meio da partição do log de eventos em sublogs DFG acíclicos, seguida pela fusão das partes, permitindo a repetição de nós de eventos para melhorar a clareza visual e a precisão (SHAIMOV; LOMAZOVA; MITSYUK, 2025).

Em contraste, abordagens que incorporam dados temporais são propostas para lidar com a incompletude dos dados, como a Mineração Genética de Processos Temporizada (T-GPM), na qual a informação de dual-timestamp é utilizada para inferir a provável presença de eventos faltantes a partir de lacunas temporais (EFFENDI; KIM, 2024). Outra linha de pesquisa evidencia a inadequação de mineradores automáticos em cenários de alta complexidade, defendendo a criação de modelos sistemáticos baseados em conhecimento de especialistas e informações organizacionais para fornecer uma compreensão concreta da estrutura do processo aos *stakeholders* (BAKHSHI; HASSANNAYEBI; SADEGHI, 2023).

3.2 Evolução do Tema e Principais Problemas Investigados

Do ponto de vista histórico, observa-se que as pesquisas iniciais concentraram-se em estabelecer os pilares fundamentais da Mineração de Processos (descoberta, verificação de conformidade e aprimoramento) a partir de logs de eventos com atributos obrigatórios básicos (identificador de caso, atividade e registro de data/hora), utilizando representações visuais simples, como o *Directly-Follows Graph (DFG)*, para a exploração inicial dos dados (AALST, 2019; LEEMANS; POPPE; WYNN, 2019), ao passo que estudos mais recentes passaram a explorar abordagens avançadas e extensões de algoritmos (como o *Inductive Miner* translúcido e a mineração genética temporizada), visando lidar com logs de eventos não ideais (incompletude, ruído, comportamento raro) e utilizar informações temporais (dual-timestamp) e dados de atividades habilitadas para aumentar a precisão, a generalização e a robustez dos modelos descobertos (BEYEL; AALST, 2025; EFFENDI; KIM, 2024). Essa evolução revela uma ampliação do escopo investigativo, bem como o surgimento de novos problemas e desafios (AALST, 2011).

Entre os problemas mais recorrentes identificados na literatura destacam-se a incapacidade inerente dos DFGs de representar concorrência ou intercalação de atividades, o que frequentemente gera DFGs Spaghetti com ciclos enganosos, mesmo para processos acíclicos (AALST, 2019; SHAIMOV; LOMAZOVA; MITSYUK, 2025), a falha dos DFGs em fornecer diagnósticos de desempenho confiáveis, especialmente após a aplicação de filtros de frequência, resultando em "lacunas invisíveis"e em tempos médios condicionais enganosos (AALST, 2019), e o desafio de descobrir modelos de processo precisos e robustos a partir de logs de eventos incompletos ou ruidosos, o que afeta diretamente a qualidade e a integridade dos modelos gerados (EFFENDI; KIM, 2024; BEYEL; AALST, 2025).

Para esses problemas, diferentes soluções têm sido propostas, com variados níveis de eficácia e limitações:

- Para a Concorrência e Clareza (DFGs): A solução clássica envolve a migração para modelos com semântica formal, como as Redes de Petri, utilizando algoritmos como o *Inductive Miner* (IM), que garante a solidez do modelo, ou o *Heuristics Miner* (LEEMANS; POPPE; WYNN, 2019). Mais recentemente, foram desenvolvidos algoritmos que particionam logs de eventos a fim de descobrir modelos DFG acíclicos, evitando a formação de ciclos e melhorando a clareza visual, ainda que permitindo a repetição de nós (SHAIMOV; LOMAZOVA; MITSYUK, 2025)

- Para Ruído e Generalização (IM Translúcido): O uso de Logs de Eventos Translúcidos (TELs), que incluem informações sobre atividades habilitadas além das executadas, permite a criação de DFGs enriquecidos e o uso de relações de escolha exclusiva translúcida para filtrar ruídos e comportamento infrequente de forma mais eficaz, resultando em modelos mais robustos e generalizáveis (BEYEL; AALST, 2025).

- Para Incompletude e Precisão (*Genetic Process Mining* Temporizado): O algoritmo de Mineração Genética de Processos Temporizada (T-GPM) utiliza informações de dual-timestamp (tempo de início e fim) para inferir a provável presença e a ordem dos eventos faltantes em logs incompletos, o que demonstrou resultados superiores em fitness e precisão estrutural e comportamental em comparação com algoritmos genéticos tradicionais (EFFENDI; KIM, 2024).

3.3 Lacunas Identificadas na Literatura

Apesar do avanço observado, a literatura analisada apresenta lacunas relevantes. Nota-se, por exemplo, a escassez de estudos que combinam integralmente as três áreas da mineração de processos (descoberta, análise de variantes e verificação de conformidade) em logs reais complexos, como os de setores de saúde (BAKHSHI; HASSANNAYEBI; SADEGHI, 2023), ou que explorem a modelagem de sistemas em ambientes multiusuário com *Directly Follows Graphs* (DFGs), onde o não registro de usuários pode levar a falsas conclusões sobre concorrência e escolha (BEYEL; AALST, 2025).

Bem como limitações metodológicas relacionadas a baixa aplicabilidade em larga escala de logs de eventos translúcidos (TELs) por conta da sua indisponibilidade no mercado (BEYEL; AALST, 2025), à suposição de distribuições de probabilidade restritas (ex.: exponencial) em simulações, o que pode impactar o escopo de aplicação e a precisão do método (SANTIAGO, 2019), e aos problemas de escalabilidade e inflexibilidade que afetam modelos que não preveem atividades paralelas ou concorrentes.

Essas lacunas indicam a necessidade de investigações que aprofundem a precisão e o desempenho de algoritmos avançados na reconstrução de processos em logs que contenham incompletude de dados e ruído, utilizando informações temporais *(dual-timestamp)* ou de atividades habilitadas (EFFENDI; KIM, 2024; BEYEL; AALST, 2025) e que garantam a solidez e a expressividade semântica ao migrar de representações simples (DFGs) para modelos formais (Redes de Petri) (SHAIMOV; LOMAZOVA; MITSYUK, 2025; LEEMANS; POPPE; WYNN, 2019), justificando a realização do presente estudo.

4 Metodologia

4.1 Tipo e Abordagem da Pesquisa

Trata-se de uma pesquisa aplicada, de caráter exploratório e descritivo, com predominância quantitativa, baseada na análise de logs de eventos e na extração de métricas processuais. A dimensão qualitativa aparece na interpretação dos modelos obtidos e na discussão das implicações para melhoria de processos.

4.2 Delineamento Metodológico

O estudo segue um delineamento empírico estruturado em etapas:

1. Seleção de uma base pública de referência (log de *helpdesk*), adequada para experimentação didática e reprodutível.
2. Pré-processamento: padronização de atributos essenciais (*CaseID, Activity, Timestamp*) e ordenação temporal por caso.
3. Descoberta e análise de DFGs (*baseline*), com inspeção de frequências, variantes e pontos de decisão.
4. Descoberta de Redes de Petri por algoritmos clássicos.
5. Conformance checking e análise de desempenho: cálculo de métricas e identificação de gargalos/tempos de espera.
6. Geração de artefatos (figuras e exportações) e consolidação do relatório analítico.

4.3 Materiais e Ferramentas

As análises foram conduzidas em Python, combinando implementações didáticas (construção de DFG com grafos) e o uso de biblioteca de referência na comunidade (*PM4Py*), visando reprodutibilidade e aderência a algoritmos consolidados. Os códigos e artefatos gerados são disponibilizados em apêndices.

4.4 Limitações

As conclusões empíricas deste trabalho refletem o escopo do log efetivamente analisado no estudo de caso (helpdesk) e, portanto, não devem ser interpretadas como gene-

ralizações para todo e qualquer processo institucional. O objetivo primário foi demonstrar, de forma controlada e reprodutível, um *pipeline* de extração, curadoria e modelagem inicial (com DFG) aplicável a cenários reais, evidenciando desafios típicos como heterogeneidade, lacunas de registro e necessidade de padronização.

Embora a proposta tenha sido concebida para evoluir até um estudo de caso completo na base da Central de Emendas, a execução dessa etapa foi limitada por restrições de prazo e, sobretudo, pela indisponibilidade de acesso dentro do período do TCC, dado que a base demanda credenciamento e controle de acesso rigorosos. Assim, a validação direta em dados governamentais/institucionais permanece como extensão prospectiva, condicionada à liberação formal e à verificação de requisitos de qualidade, completude e governança dos dados.

Por fim, ressalta-se que as escolhas metodológicas priorizaram simplicidade, interpretabilidade e viabilidade de implementação no contexto do TCC; modelos mais expressivos (por exemplo, Redes de Petri) e análises formais/automáticas mais avançadas não foram explorados em profundidade nesta etapa, ficando reservados para continuidade futura quando houver acesso aos dados-alvo e tempo adequado para experimentação e comparação sistemática.

5 Base de dados

As bases de dados de eventos desempenham papel central nas pesquisas em Mineração de Processos, pois constituem a principal fonte empírica para a descoberta, análise e avaliação de processos organizacionais a partir de evidências reais. Conforme estabelecido por **Wil van der Aalst**, Mineração de Processos fundamenta-se na análise de logs de eventos registrados por sistemas de informação, os quais permitem reconstruir o comportamento efetivo dos processos, superando limitações de abordagens puramente *model-driven* ou baseadas apenas em entrevistas e documentação formal. Esses registros são tipicamente compostos por identificadores de casos, atividades e carimbos temporais, elementos considerados essenciais para a aplicação das técnicas clássicas da área (AALST, 2016).

Nesse contexto, repositórios públicos de logs de eventos assumem papel estratégico para o avanço científico da área, pois possibilitam a reprodutibilidade dos experimentos, a comparação de métodos e a validação empírica de novas abordagens. Entre os principais repositórios utilizados na literatura destaca-se o **4TU.ResearchData**, amplamente citado em estudos acadêmicos e aplicações experimentais. Esse repositório disponibiliza logs de eventos reais, anonimizados, provenientes de diferentes domínios organizacionais, como instituições financeiras e processos de tecnologia da informação. Conjuntos de dados como o *Event Log of a Dutch Financial Institution* são frequentemente empregados como casos de referência em pesquisas sobre descoberta de processos, análise de conformidade e identificação de gargalos, conforme discutido em (AALST *et al.*, 2011) (AALST, 2016). A disponibilização dos dados nos formatos XES - padrão proposto pela comunidade científica - e CSV favorece sua interoperabilidade com ferramentas consolidadas como ProM e bibliotecas como PM4Py.

Outra base amplamente difundida na literatura é o conjunto de dados de *helpdesk* associado à Universidade de Pisa, frequentemente disponibilizado em repositórios públicos como o *UCI Machine Learning Repository*. Esse conjunto de dados representa o ciclo de vida de chamados de suporte técnico, contendo registros temporais de abertura, encaminhamento e resolução de *tickets*. Sua estrutura relativamente simples o torna particularmente adequado para fins didáticos e estudos exploratórios, sendo amplamente utilizado para ilustrar conceitos fundamentais como descoberta de fluxo de processos, análise de tempos de espera e avaliação de desempenho. Trabalhos introdutórios e educacionais em Mineração de Processos frequentemente utilizam esse tipo de base para demonstrar, de forma controlada, os princípios estabelecidos por (AALST, 2011; AALST, 2016).

Merecem destaque, ainda, os conjuntos de dados associados aos *BPI Challenges* (*Business Process Intelligence Challenges*), iniciativas anuais promovidas pela comunidade científica internacional com o objetivo de fomentar o avanço das técnicas de Mineração de

Processos. Cada edição do desafio disponibiliza logs de eventos reais, oriundos de contextos organizacionais complexos, como processos de concessão de crédito, sistemas judiciais e fluxos administrativos em larga escala. Exemplos amplamente citados incluem o *BPI Challenge 2012*, focado em processos de empréstimos, e o *BPI Challenge 2017*, baseado em logs de sistemas de tribunais. Esses conjuntos de dados, geralmente disponibilizados no formato XES, caracterizam-se por elevada complexidade, múltiplas variantes processuais e comportamentos não estruturados, sendo particularmente adequados para análises avançadas, como detecção de desvios, estudo de exceções, análise de conformidade e otimização de processos, conforme discutido em (INDULSKA *et al.*, 2009).

Em síntese, o uso dessas bases de dados consolidadas - *4TU.ResearchData*, logs de *helpdesk* e *BPI Challenges*, encontra amplo respaldo na literatura clássica de Mineração de Processos e constitui prática recorrente em pesquisas acadêmicas da área. Além de fornecerem um arcabouço empírico robusto para validação metodológica, tais bases permitem a formação de um referencial conceitual e experimental sólido, que pode ser posteriormente estendido a contextos específicos, como a análise de bases governamentais e de transparência pública, a exemplo da Central de Emendas.

6 Mineração de Processos com Logs de *Helpdesk* em formato DFG

A seguir será apresentado um exemplo completo de uso didático com dados de *helpdesk* (apêndice A), desde a base até as análises.

Os dados analisados consistem em logs de eventos extraídos de um sistema de *helpdesk* em operação, contendo 5 instâncias completas de processo *(tickets)*. Cada registro inclui identificador do caso *CaseID*, atividade executada, *timestamp*, recurso responsável e status. Para análise, utilizamos apenas os campos essenciais: *CaseID* e *Activity*, seguindo a abordagem proposta no *Process Mining Framework* de Aalst (2011).

O primeiro passo metodológico envolve a transformação dos logs lineares em sequências de atividades por caso. Para cada *CaseID*, as atividades são ordenadas cronologicamente, gerando trajetórias processuais completas. Este agrupamento temporal é fundamental para preservar a relação de precedência entre eventos, conforme estabelecido por Aalst (2011).

Após a extração das sequências, aplicamos um algoritmo de comparação para identificar variantes únicas. Duas sequências são consideradas idênticas quando apresentam exatamente a mesma ordem e conjunto de atividades. A frequência de cada variante é calculada através de contagem simples, seguindo a fórmula:

$$f_v = \frac{n_v}{N} \times 100\% \tag{6.1}$$

onde f_v representa a frequência da variante v, n_v o número de casos que seguem essa variante, e N o total de casos analisados.

As variantes identificadas são classificadas em categorias temáticas baseadas em padrões reconhecíveis de desvio. Esta classificação segue uma abordagem qualitativa fundamentada na teoria de gestão de processos de negócio (BAKHSHI; HASSANNAYEBI; SADEGHI, 2023), permitindo a interpretação contextual das diferentes trajetórias processuais.

A análise revelou a existência de quatro variantes distintas no processo de atendimento de tickets, conforme sumarizado na Tabela 6.1.

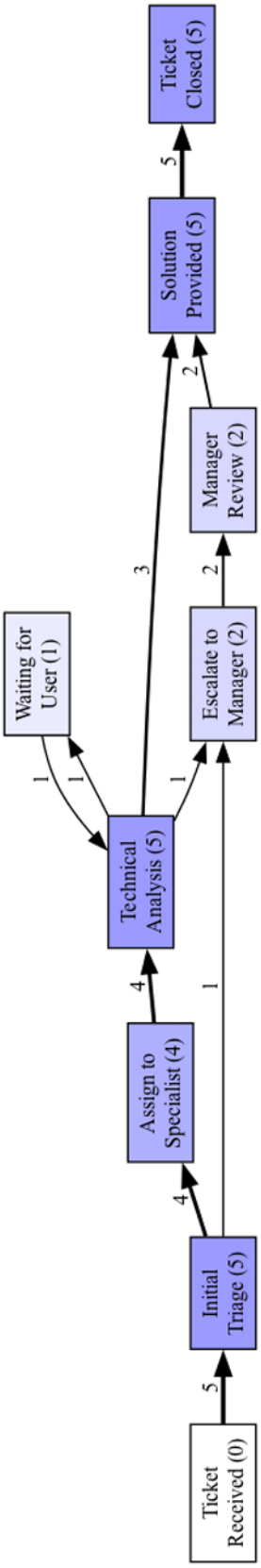
Tabela 6.1 – Análise de Variantes de Processo (5 Tickets)

Distribuição		**Sequência de Atividades**	**Porcentagem**
Tickets	**Frequência**		
2	40%	$R \rightarrow T \rightarrow A \rightarrow An \rightarrow S \rightarrow F$ *(Caminho Normal)*	40%
1	20%	$R \rightarrow T \rightarrow A \rightarrow An \rightarrow W \rightarrow An \rightarrow S \rightarrow F$ *(Com Espera)*	20%
1	20%	$R \rightarrow T \rightarrow E \rightarrow Rv \rightarrow S \rightarrow F$ *(Escalação Direta)*	20%
1	20%	$R \rightarrow T \rightarrow A \rightarrow An \rightarrow E \rightarrow Rv \rightarrow S \rightarrow F$ *(Re-escalação)*	20%
Total		**5 tickets**	**100%**

Legenda: R = Ticket Received, T = Initial Triage, A = Assign to Specialist, An = Technical Analysis, W = Waiting for User, E = Escalate to Manager, Rv = Manager Review, S = Solution Provided, F = Ticket Closed.

As variantes podem ser representadas graficamente através de um DFG, que captura relações de precedência direta entre atividades. A Figura 6.1 ilustra este grafo, onde a espessura das arestas é proporcional à frequência das transições:

Figura 6.1 – Directly-Follows Graph (DFG) das Variantes de Processo



Fonte: elaboração própria a partir do código no apendice C.

Considerando o processo ideal definido pela organização (Caminho Normal), observamos que apenas 40% das instâncias seguem estritamente este fluxo. Os 60% restantes apresentam desvios que, embora possam ser justificáveis contextualmente, indicam uma significativa variação na execução processual.

A presença de múltiplas variantes sugere diferentes fenômenos organizacionais:

- Adaptação contextual: algumas variantes podem representar adaptações legítimas a características específicas dos *tickets*, tais como maior complexidade, nível de urgência ou disponibilidade de recursos.
- Inconsistência operacional: outras variantes podem indicar a existência de falhas na padronização dos procedimentos ou a necessidade de aprimoramento nos processos de capacitação e treinamento das equipes envolvidas.
- Exceções processuais: determinadas variantes podem refletir a gestão de casos excepcionais não previstos explicitamente no modelo de processo padrão, exigindo tratamento diferenciado por parte da organização.

A análise quantitativa das variantes oferece *insights* relevantes para a gestão operacional, conforme descrito a seguir:

- Otimização de processo: a variante "Caminho com Espera" (20% dos casos) sugere a existência de oportunidades para aprimorar a etapa inicial de coleta de informações, contribuindo para a redução de retrabalho e do tempo total de atendimento.
- Capacitação de recursos: a ocorrência da variante "Caminho com Re-escalação" (20%) pode indicar a necessidade de investimentos em treinamento adicional para os especialistas, ampliando sua capacidade de resolução de demandas sem a necessidade de escalonamento.
- Revisão de políticas: a frequência elevada de escalonamentos (40%, considerando conjuntamente as duas variantes que envolvem escalação) pode justificar a revisão dos critérios adotados para o encaminhamento de tickets a gestores, visando maior eficiência e melhor alocação de recursos gerenciais.

É importante reconhecer que esta análise baseia-se em uma amostra limitada (5 casos). Em ambientes reais, recomenda-se a análise de pelo menos 100–200 casos para resultados estatisticamente significativos (WEERDT *et al.*, 2012). Além disso, a abordagem considera apenas a perspectiva do fluxo de controle, não incorporando dimensões como tempo de ciclo, alocação de recursos ou custos associados.

6.1 Análise de DFGs para Mineração de Processos

Nesta seção é desenvolvida a análise dos processos associados à base de dados *Helpdesk* por meio da construção e interpretação de DFGs, cuja definição formal é apresentada na Seção 2.3. A abordagem adotada privilegia um viés computacional, fundamentado na implementação de rotinas em *Python* inicialmente com emprego de bibliotecas amplamente consolidadas para a representação e análise de grafos, como o *NetworkX*. Tal estratégia possibilita a visualização geométrica dos fluxos processuais a partir das relações de precedência direta entre atividades, permitindo uma compreensão estrutural do comportamento observado nos dados, conforme exemplificado no código apresentado no Apêndice B.

Na sequência, explora-se o emprego da biblioteca *PM4Py*, amplamente reconhecida na literatura de Mineração de Processos, para a geração de DFGs sob diferentes perspectivas analíticas. Essa combinação de abordagens possibilita tanto uma compreensão estrutural dos processos, por meio de grafos construídos manualmente, quanto uma análise automatizada e orientada a dados, evidenciando padrões de comportamento, frequências de transição e aspectos relevantes para a interpretação e avaliação dos processos analisados.

6.1.1 Construção de DFG para Helpdesk com NetworkX

Será apresentada a construção de um DFG utilizando a biblioteca *NetworkX* em *Python*, conforme ilustrado no código do Apêndice B. A metodologia adotada envolve a extração das sequências de atividades por caso, seguida da contagem das transições diretas entre atividades consecutivas. A partir dessas contagens, constrói-se um grafo direcionado onde cada nó representa uma atividade e cada aresta indica a transição direta entre atividades, ponderada pela frequência observada.

A construção de DFGs segue uma metodologia sistemática que transforma logs temporais de eventos em representações gráficas analíticas. Dado um log $L = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, onde cada traço $\sigma_k = \langle e_1, e_2, \ldots, e_m \rangle$ representa uma sequência ordenada de eventos, o algoritmo de construção do DFG processa três etapas principais:

1. **Extração de sequências**: Para cada caso k, as atividades são ordenadas temporalmente: $\sigma_k = \langle a_1, a_2, \ldots, a_m \rangle$
2. **Contagem de transições**: Para cada par consecutivo (a_i, a_{i+1}), incrementa-se o contador $c_{i,i+1}$
3. **Construção do grafo**: Cria-se um grafo direcionado onde cada nó representa uma atividade e cada aresta (a_i, a_j) possui peso $w_{ij} = c_{ij}$

Esta abordagem gera dois tipos complementares de DFG, cada um com propósito analítico distinto, conforme detalhado nas subseções seguintes.

Como saída do código, além de uma análise estatística dos dados, são gerados dois tipos de DFGs distintos, cada um enfatizando aspectos diferentes do processo de helpdesk: o DFG de Frequência (Estrutural) e o DFG de Desempenho (Temporal). A seguir, detalha-se a construção e interpretação de cada um desses grafos.

6.1.1.1 Análise do Processo

A seguir apresenta-se um resumo analítico do DFG obtido a partir da base de dados Helpdesk, destacando estatísticas gerais, atividades mais frequentes, fluxos predominantes e pontos de decisão do processo.

Onde as estatísticas básicas foram:

- Total de casos analisados: 5;
- Total de transições observadas: 29;
- Número de transições distintas: 11.

As atividades com maior número de ocorrências ao longo dos casos analisados foram:

- *Ticket Received*: 5 ocorrências (14,7%);
- *Initial Triage*: 5 ocorrências (14,7%);
- *Technical Analysis*: 5 ocorrências (14,7%);
- *Solution Provided*: 5 ocorrências (14,7%);
- *Ticket Closed*: 5 ocorrências (14,7%).

Os fluxos de atividades mais frequentemente observados no DFG foram:

- *Ticket Received* → *Initial Triage*: 5 ocorrências (17,2%);
- *Solution Provided* → *Ticket Closed*: 5 ocorrências (17,2%);
- *Initial Triage* → *Assign to Specialist*: 4 ocorrências (13,8%);
- *Assign to Specialist* → *Technical Analysis*: 4 ocorrências (13,8%);
- *Technical Analysis* → *Solution Provided*: 3 ocorrências (10,3%).

A análise do DFG evidencia a existência de ramificações relevantes no processo, em especial:

- a atividade *Initial Triage*, que pode conduzir às atividades *Assign to Specialist* ou *Escalate to Manager*;
- a atividade *Technical Analysis*, que pode resultar em *Solution Provided*, *Waiting for User* ou *Escalate to Manager*.

No que se refere às atividades de início e encerramento do processo, observa-se total padronização:

- atividade inicial: *Ticket Received*, presente em 100% dos casos analisados;
- atividade final: *Ticket Closed*, também presente em 100% dos casos.

Além da análise baseada em frequência, procedeu-se à geração de DFGs enriquecidos com métricas temporais, permitindo avaliar o desempenho das transições entre atividades e identificar atrasos relevantes ao longo do processo.

Mostrando assim a análise de conformidade realizada sobre o DFG revelou a existência de desvios em relação ao comportamento esperado do processo. Foram identificadas três transições não previstas no modelo de referência, conforme descrito a seguir:

- *Technical Analysis* → *Waiting for User*: 1 ocorrência;
- *Waiting for User* → *Technical Analysis*: 1 ocorrência;
- *Technical Analysis* → *Escalate to Manager*: 1 ocorrência.

Apesar desses desvios pontuais, a taxa global de conformidade observada foi de 89,7%, indicando elevado grau de aderência entre o comportamento registrado nos dados e o fluxo processual predominante.

Já na análise temporal permitiu a identificação de gargalos significativos no processo, totalizando cinco gargalos principais, caracterizados por elevados tempos médios de transição:

- *Escalate to Manager* → *Manager Review*: tempo médio de 562,5 minutos, em 2 ocorrências;
- *Solution Provided* → *Ticket Closed*: tempo médio de 234,0 minutos, em 5 ocorrências;
- *Waiting for User* → *Technical Analysis*: tempo médio de 180,0 minutos, em 1 ocorrência;
- *Manager Review* → *Solution Provided*: tempo médio de 90,0 minutos, em 2 ocorrências;
- *Technical Analysis* → *Solution Provided*: tempo médio de 80,0 minutos, em 3 ocorrências.

Esses resultados evidenciam pontos críticos do processo que impactam diretamente o tempo total de atendimento, reforçando a necessidade de intervenções gerenciais e operacionais direcionadas.

Assim como resultado da análise, foram gerados arquivos auxiliares contendo informações detalhadas para posterior inspeção e validação dos achados:

- `dfg_analysis.csv`, contendo 11 transições analisadas;
- `cases_paths.csv`, contendo os caminhos processuais de 5 casos distintos.

6.1.2 DFG de Frequência (Estrutural)

O DFG de frequência, ilustrado na Figura 6.2, focaliza a análise estrutural do processo, representando a distribuição probabilística das transições entre atividades. A espessura das arestas é proporcional à frequência relativa, calculada como:

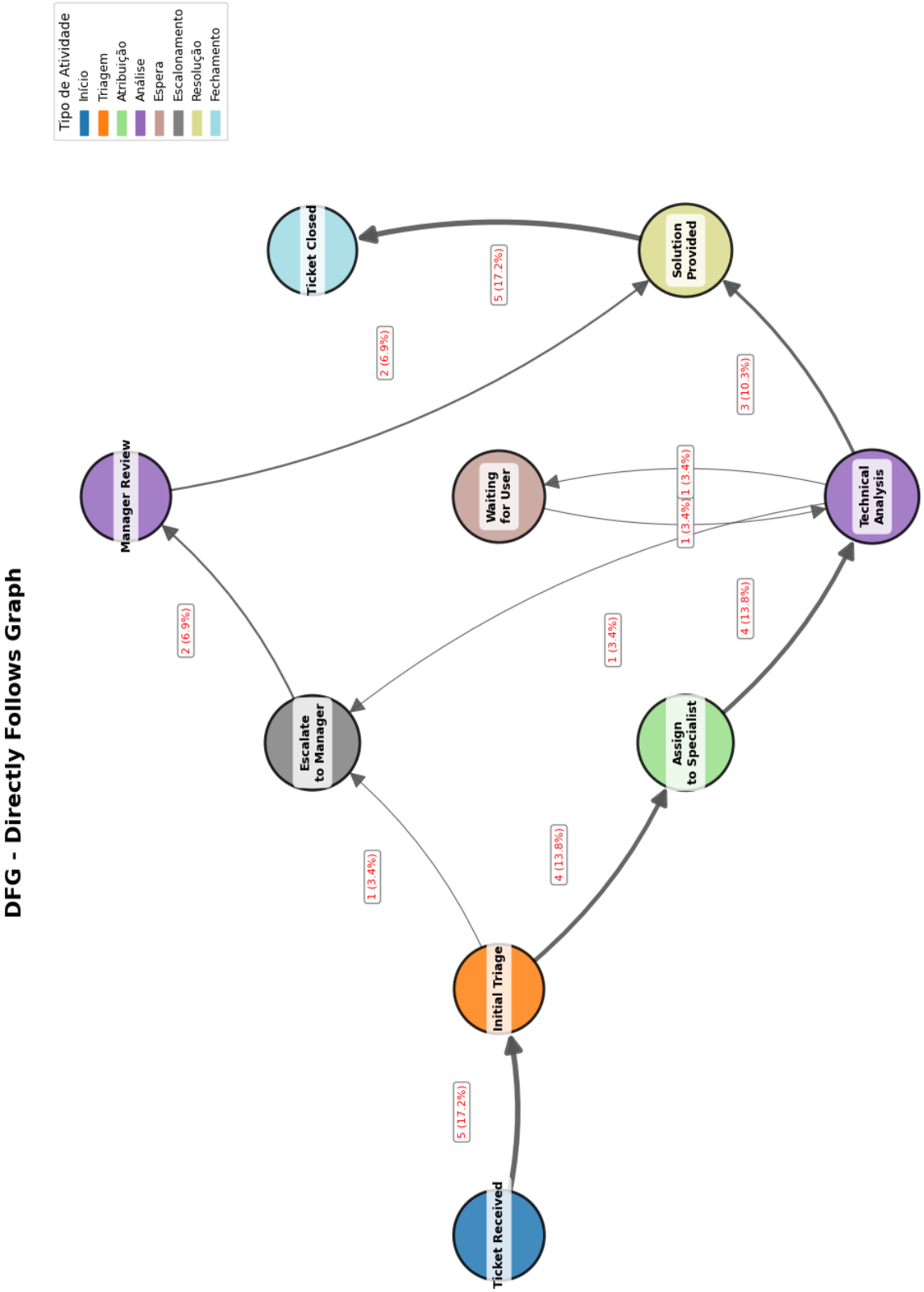
$$f_{ij} = \frac{c_{ij}}{\sum_{(p,q)\in E} c_{pq}} \times 100\% \tag{6.2}$$

onde c_{ij} denota o número de ocorrências da transição da atividade i para j.

Este grafo responde a questões fundamentais sobre a estrutura do processo:

- Identificação de caminhos predominantes (arestas mais espessas)
- Detecção de pontos de decisão (nós com múltiplas arestas de saída)
- Quantificação da variabilidade processual (distribuição de frequências)
- Análise de conformidade com fluxos esperados

Figura 6.2 – DFG de Frequência do processo de helpdesk. A espessura das arestas representa a frequência relativa das transições, enquanto os rótulos indicam contagem absoluta e porcentagem.



Fonte: elaboração própria a partir do código no apendice B.

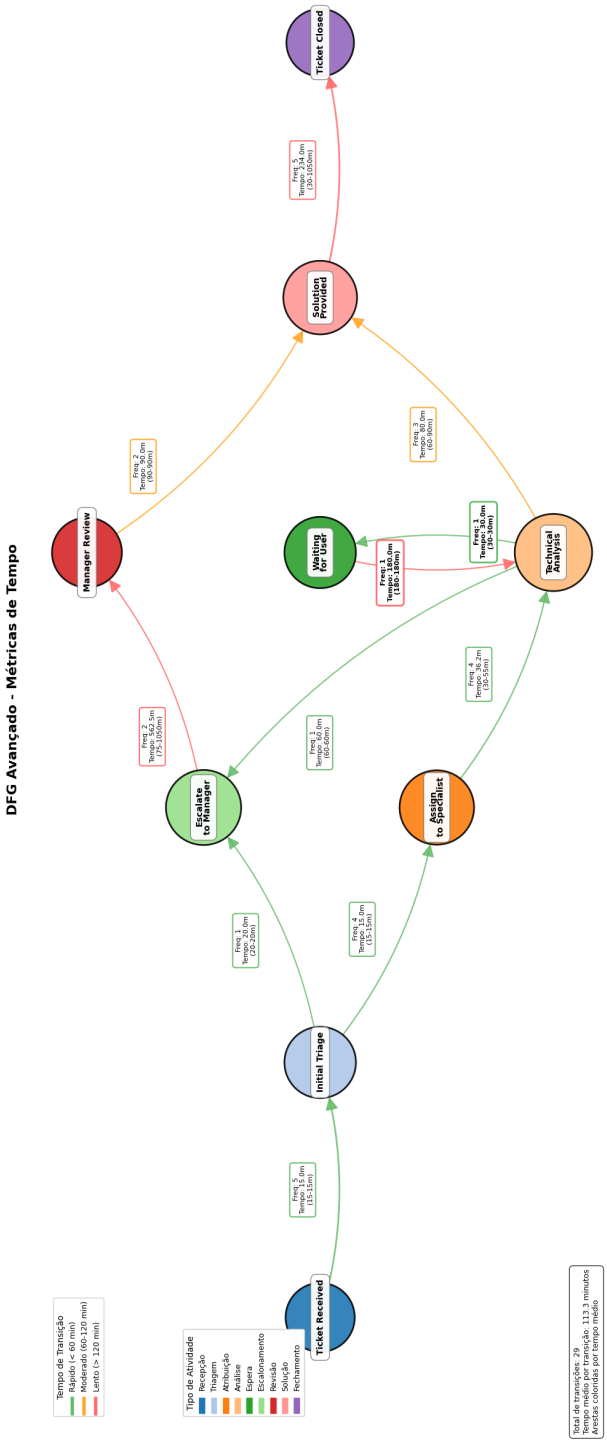
6.1.3 DFG de Desempenho (Temporal)

O DFG de desempenho, apresentado na Figura 6.3, concentra-se na dimensão temporal do processo, codificando informações sobre tempos de transição através de esquemas de cores e rótulos detalhados. O tempo médio de transição $\bar{t}_{ij}$ é calculado como:

$$\bar{t}_{ij} = \frac{1}{n_{ij}} \sum_{k=1}^{n_{ij}} t_{ijk} \tag{6.3}$$

onde t_{ijk} representa o tempo da k-ésima ocorrência da transição $i \rightarrow j$, e n_{ij} é o número total de ocorrências.

Figura 6.3 – DFG de Desempenho do processo de helpdesk.



Legenda: As cores das arestas indicam categorias temporais: verde (rápido: < 60 min), laranja (moderado: 60-120 min), vermelho (lento: > 120 min). Os rótulos apresentam frequência, tempo médio e intervalo.

Fonte: elaboração própria a partir do código no apendice B.

O esquema de cores segue uma codificação semafórica:

- Verde: Transições rápidas ($\bar{t} < 60$ minutos)
- Laranja: Transições moderadas ($60 \leq \bar{t} \leq 120$ minutos)
- Vermelho: Transições lentas ($\bar{t} > 120$ minutos)

Esta visualização permite identificar:

- Gargalos: Arestas vermelhas indicam atrasos significativos
- Variação temporal: Intervalos amplos sugerem inconsistência processual
- Pontos de espera: Transições com tempos anormalmente elevados
- Eficiência comparativa Contraste entre caminhos alternativos

6.1.4 Interpretação Conjunta dos Grafos

A análise integrada dos dois DFGs proporciona insights holísticos sobre o processo, conforme exemplificado na Tabela 6.2.

Tabela 6.2 – Interpretação conjunta de DFG de Frequência e Desempenho

Cenário	DFG de Frequência	DFG de Desempenho
Fluxo principal eficiente	Aresta grossa (alta frequência)	Aresta verde (baixo tempo)
Bottleneck crítico	Aresta grossa (alta frequência)	Aresta vermelha (alto tempo)
Caso excepcional problemático	Aresta fina (baixa frequência)	Aresta vermelha (alto tempo)
Alternativa eficiente subutilizada	Aresta fina (baixa frequência)	Aresta verde (baixo tempo)

A interpretação conjunta segue a lógica matricial:

$$\begin{aligned} \text{Prioridade de intervenção} &= f(\text{Frequência}, \text{Tempo}) \\ &= \begin{cases} \text{Alta}, & \text{se frequência E tempo forem altos;} \\ \text{Média}, & \text{se frequência OU tempo for alto;} \\ \text{Baixa}, & \text{se frequência E tempo forem baixos.} \end{cases} \end{aligned} \tag{6.4}$$

Esta abordagem bidimensional permite priorizar intervenções processuais com base tanto no impacto quantitativo (frequência) quanto qualitativo (tempo).

6.1.5 Estudo de Caso: Processo de *Helpdesk*

A aplicação da metodologia proposta ao processo de *helpdesk* revelou *insights* relevantes acerca de sua estrutura e desempenho. A base de dados analisada é composta por cinco casos completos, totalizando 34 eventos distribuídos em nove atividades distintas, o que permite uma análise exploratória consistente do comportamento processual observado.

A análise estrutural, a partir do Grafo de Sequência Direta baseado em frequência (Figura 6.2), evidencia um padrão de início fortemente padronizado, no qual todos os casos se iniciam pela sequência *"Ticket Received"* seguida de *"Initial Triage"*. A partir dessa etapa, observa-se um ponto de decisão crítico: 60% dos casos seguem o fluxo *"Assign to Specialist"*, enquanto 40% são imediatamente escalonados para *"Escalate to Manager"*. Adicionalmente, identifica-se a ocorrência de retrabalho em 20% dos casos, caracterizado pelo ciclo *"Technical Analysis"* → *"Waiting for User"* → *"Technical Analysis"*, indicando interrupções no fluxo normal do processo. Apesar dessas variações, a finalização mostra-se consistente, com todos os casos encerrando-se pela sequência *"Solution Provided"* seguida de *"Ticket Closed"*.

Sob a perspectiva de desempenho, o DFG temporal (Figura 6.3) permite a identificação de gargalos específicos ao longo do processo. O principal gargalo está associado à transição entre "*Waiting for User*" e "*Technical Analysis*", apresentando um tempo médio de aproximadamente 180 minutos, o que corresponde a três horas de espera. Observa-se ainda um atraso significativo na transição entre "*Manager Review*" e "*Solution Provided*", com tempo médio de cerca de 120 minutos. Em contraste, o caminho principal do processo, composto pelas atividades "*Assign to Specialist*", "*Technical Analysis*" e "*Solution Provided*", apresenta desempenho mais eficiente, com tempo médio inferior a 90 minutos.

A análise conjunta dos aspectos estruturais e temporais sugere implicações gerenciais relevantes. O tempo excessivo associado à etapa "*Waiting for User*", ainda que presente em apenas 20% dos casos, impacta de forma significativa o desempenho global do processo, indicando a necessidade de ações voltadas à redução do tempo de espera, como a melhoria da coleta inicial de informações ou a implementação de mecanismos automáticos de lembrete. Além disso, a elevada frequência de escalonamentos para gestores, observada em 40% dos casos, aponta para a conveniência de revisar os critérios de escalonamento e investir na capacitação dos especialistas, reduzindo a dependência da intervenção gerencial. Por fim, o fato de apenas 40% dos casos seguirem o caminho considerado ideal reforça a importância de maior padronização do processo, por meio do estabelecimento de diretrizes mais claras para as etapas de triagem inicial e análise técnica.

6.1.6 Construção de DFG para Helpdesk com *PM4Py*

O PM4Py (*Process Mining for Python*) é uma biblioteca *Python* de código aberto amplamente reconhecida pela comunidade científica de mineração de processos, projetada

para a implementação de algoritmos padronizados e técnicas analíticas consolidadas na literatura (AALST, 2016). Diferentemente de implementações customizadas, o PM4Py disponibiliza versões validadas de algoritmos fundamentais, garantindo robustez metodológica, reprodutibilidade científica e aderência às boas práticas estabelecidas na área.

Nesta seção apresenta-se a aplicação metodológica do *PM4Py* à análise do processo de *helpdesk*, em conformidade com as diretrizes estabelecidas no *Process Mining Manifesto* (AALST *et al.*, 2011), integrando aspectos estruturais, temporais e de conformidade do processo analisado. O código completo encontrasse no apêndice C

6.1.7 Resultados da Análise com PM4Py

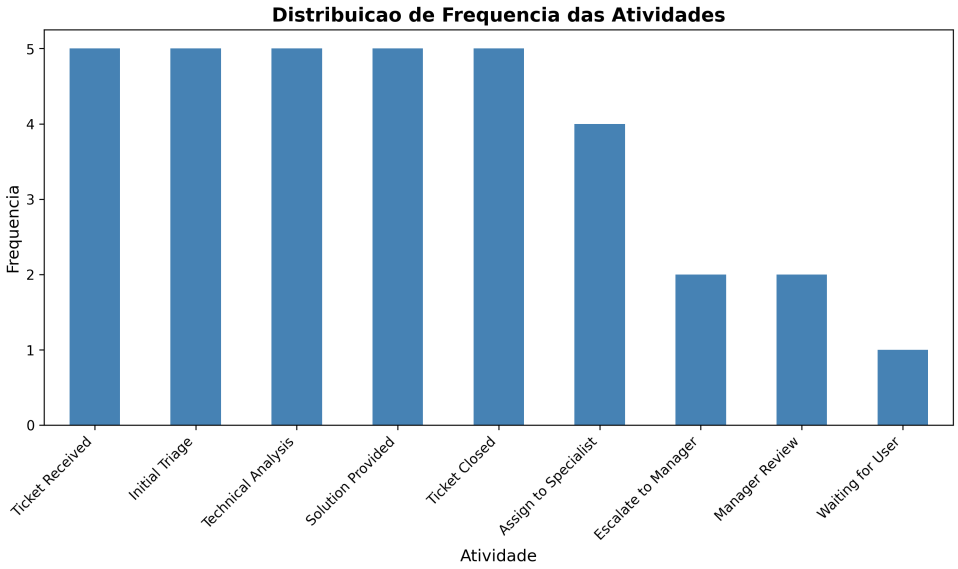
A Tabela 6.3 apresenta as principais estatísticas descritivas do processo de *helpdesk* extraídas com o auxílio do *PM4Py*.

Tabela 6.3 – Estatísticas descritivas do processo de helpdesk obtidas com o PM4Py

Métrica	**Valor**	**Interpretação**
Casos únicos	5	Adequado para análise exploratória
Eventos totais	34	Média de 6,8 eventos por caso
Atividades únicas	9	Complexidade moderada
Variantes únicas	4	Variabilidade processual relevante

A análise de frequência das atividades, ilustrada na Figura 6.4, evidencia padrões importantes de execução do processo.

Figura 6.4 – Distribuição de frequência das atividades no processo de helpdesk



Fonte: elaboração própria a partir do código no apêndice C

Observam-se como principais achados:

1. a presença obrigatória das atividades `Ticket Received` e `Ticket Closed` em 100% dos casos;
2. o papel central da atividade `Technical Analysis`, presente em 80% dos casos; e
3. a ocorrência condicional das atividades `Waiting for User` (20%) e `Escalate` *to Manager* (40%).

Foram gerados dois DGFs complementares, apresentados nas Figuras 6.5 e 6.6: um baseado em frequência e outro enriquecido com métricas de desempenho temporal.

A análise conjunta dos grafos evidencia que as transições mais espessas correspondem aos caminhos predominantes, enquanto as cores do DFG de desempenho indicam tempos médios elevados em determinadas transições, destacando gargalos relevantes.

6.1.8 Comparação entre o Uso do *PM4Py* e Implementações Customizadas

A análise comparativa entre o uso da biblioteca *PM4Py* e o desenvolvimento de implementações customizadas revela diferenças relevantes tanto do ponto de vista metodológico quanto prático. O *PM4Py* destaca-se por oferecer um conjunto abrangente de algoritmos de Mineração de Processos implementados de forma otimizada, testada e validada pela comunidade científica, assegurando maior confiabilidade dos resultados obtidos. Além disso, a biblioteca segue padrões amplamente aceitos na indústria e na literatura especializada em Mineração de Processos , incorporando boas práticas consolidadas e métricas avançadas, como *fitness*, precisão e generalização, que são fundamentais para análises mais rigorosas de conformidade e desempenho.

Outro aspecto relevante do *PM4Py* é a disponibilidade de múltiplos algoritmos de descoberta de processos, tais como *Alpha Miner, Heuristics Miner* e *Inductive Miner*, permitindo comparar diferentes perspectivas analíticas sobre o mesmo conjunto de dados. A padronização dos formatos de entrada e saída, incluindo XES para logs de eventos e formatos gráficos como PNG e SVG, também contribui para a interoperabilidade com outras ferramentas e para a reprodutibilidade dos experimentos. Ademais, o *PM4Py* é projetado para lidar de forma eficiente com grandes volumes de dados, sendo adequado para cenários reais de produção e aplicações industriais.

Por outro lado, implementações customizadas apresentam vantagens específicas, especialmente em contextos exploratórios. O desenvolvimento manual de algoritmos de descoberta e visualização de DFGs possibilita controle total sobre o processo de análise e sobre a forma de representação gráfica, permitindo personalizações que nem sempre são triviais em bibliotecas padronizadas. Além disso, a ausência de dependências externas reduz o *overhead* computacional e facilita a compreensão detalhada dos mecanismos internos da mineração de processos.

Do ponto de vista da implementação, as diferenças entre as abordagens manifestam-se de forma clara no código. Enquanto o *PM4Py* abstrai etapas complexas, como a criação do log de eventos, a geração de DFGs, a identificação de variantes e a avaliação de conformidade por meio de *token replay*, as implementações customizadas exigem manipulação direta de estruturas de dados, cálculos manuais de estatísticas e verificações explícitas das relações de precedência. Essa diferença implica um maior esforço de desenvolvimento, mas também favorece o entendimento conceitual dos algoritmos envolvidos.

Diante desse cenário, recomenda-se o uso do *PM4Py* em projetos acadêmicos e industriais que envolvam dados reais de produção, necessidade de métricas padronizadas e exigência de robustez metodológica. Em contrapartida, implementações customizadas mostram-se mais adequadas em contextos educacionais, em análises exploratórias com pequenos conjuntos de dados ou no desenvolvimento de protótipos rápidos que demandem visualizações altamente específicas. Assim, ambas as abordagens podem ser vistas como complementares.

6.1.9 Tutorial de Uso do Script `dfg_pm4py.py` para Mineração de Processos com PM4Py

Esta subseção apresenta um tutorial prático para execução do *script* `dfg_pm4py.py`, responsável por implementar uma análise interativa de mineração de processos com a biblioteca *PM4Py*. O objetivo do script é automatizar tarefas típicas da área, incluindo descoberta de Grafos de DFGs, inspeção de variantes, análise de conformidade via *token replay* e identificação de gargalos temporais, além de exportar resultados em formatos adequados para auditoria e reprodutibilidade.

Requisitos e Preparação do Ambiente

Antes da execução, recomenda-se garantir que o ambiente Python esteja devidamente configurado. Em termos mínimos, o script requer: (i) *Python* 3.9+; (ii) bibliotecas `pandas` e *pm4py*. Para visualizações e exportações gráficas, recomenda-se a instalação do Graphviz (`dot`) no sistema operacional e a presença do pacote `graphviz` no ambiente Python, quando aplicável.

O arquivo de log de eventos deve estar disponível no mesmo diretório do script (ou no caminho configurado no próprio código), tipicamente em formato CSV, contendo ao menos: identificador de caso (`CaseID`), nome da atividade (`Activity`) e carimbo de tempo (`Timestamp`). A existência de atributos complementares (`Resource`, `Status`) é desejável, pois permite análises mais ricas (por exemplo, recursos envolvidos e etapas do ciclo de vida).

Execução do Script

A execução é realizada via terminal, posicionando-se no diretório onde se encontra o arquivo `dfg_pm4py.py`. Em seguida, executa-se:

```
python dfg_pm4py.py
```

Após iniciar, o programa apresenta um menu interativo. O usuário escolhe uma das opções disponíveis digitando `1`, `2` ou `3` e pressionando `Enter`.

Menu Interativo e Funcionalidades

Opção 1 — Análise completa com PM4Py Ao selecionar `1`, o script executa uma análise completa baseada em *PM4Py*. De forma resumida, essa opção realiza:

- Descoberta de DFG por frequência, evidenciando os caminhos mais recorrentes entre atividades;
- Descoberta de DFG por desempenho, incorporando métricas temporais (por exemplo, tempo médio entre transições);
- Visualização dos grafos (quando o Graphviz estiver disponível), com mensagens de progresso no terminal;
- Extração de estatísticas do processo, como número de casos, eventos, atividades distintas e distribuição de frequências;
- Identificação de variantes do processo e suas frequências, permitindo analisar diversidade de trajetórias;
- Análise de conformidade via *token replay*, produzindo métricas como *fitness*;
- Identificação de *bottlenecks* a partir de métricas temporais, destacando atividades e/ou transições com maior tempo médio.

Durante a execução, a saída no terminal inclui resumos, contagens, percentuais e mensagens indicando o avanço da análise, facilitando acompanhamento e depuração. Ao final, o script exporta arquivos estruturados (CSV/JSON), descritos na Subseção 6.1.9.

Opção 2 — Comparação PM4Py vs implementação customizada Ao selecionar `2`, o script executa um procedimento de validação cruzada comparando resultados obtidos pelo PM4Py com uma implementação customizada do DFG (por exemplo, `build_directly_follows_graph`). O objetivo é verificar a consistência estrutural das transições identificadas, imprimindo no terminal:

- transições comuns entre ambas as abordagens (interseção);

- transições exclusivas do PM4Py;
- transições exclusivas do método customizado.

Essa opção é particularmente útil para fins didáticos (explicação do algoritmo) e para validação metodológica (conferência da consistência dos resultados em múltiplas implementações).

Opção 3 — Sair A opção 3 finaliza o programa sem executar novas análises.

Arquivos Gerados pela Análise e Exportação

Ao executar a análise completa (Opção 1), o script exporta resultados em arquivos que favorecem reprodutibilidade, inspeção manual e integração com outras ferramentas. Os principais artefatos gerados são:

- `pm4py_dfg_analysis.csv`: tabela de transições do DFG, contendo colunas típicas como `From`, `To`, `Frequency` e `Avg_Time_Seconds`;
- `pm4py_variants.csv`: variantes do processo e contagem de casos por variante;
- `pm4py_conformance.json`: métricas de conformidade derivadas do *token replay* (por exemplo, *fitness* e indicadores associados);
- `pm4py_bottlenecks.csv`: atividades com tempo médio elevado, caracterizadas como gargalos (*bottlenecks*).

Boas Práticas de Uso e Interpretação

Para maximizar a utilidade do script em contexto acadêmico, recomenda-se: (i) registrar a versão das bibliotecas utilizadas (PM4Py, pandas e dependências); (ii) manter o log de eventos original preservado e versionado; (iii) relatar explicitamente parâmetros e filtros aplicados (por exemplo, limiares de frequência ou cortes temporais); e (iv) interpretar resultados de conformidade e *bottlenecks* considerando limitações do conjunto de dados, especialmente quando se trata de bases reduzidas ou com ruído.

Em síntese, o `dfg_pm4py.py` fornece uma interface operacional simples para execução de análises típicas de mineração de processos, combinando geração de modelos, extração de métricas e exportação padronizada de resultados, o que favorece transparência metodológica e reprodutibilidade dos achados reportados.

6.2 Conclusão da seção

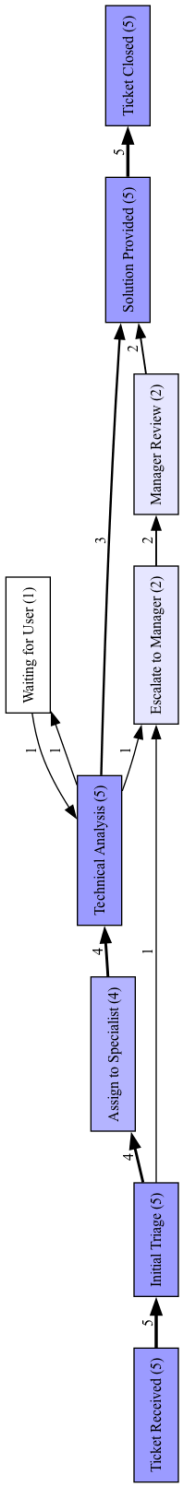
Nesta seção, apresentamos duas abordagens computacionais para a análise em Mineração de Processos. A primeira se apoiou em bibliotecas consolidadas, porém generalistas,

voltadas à análise de grafos e à visualização de estruturas de dependência; a segunda empregou a biblioteca *pm4py*, concebida especificamente para tarefas típicas da área. Em ambos os casos, o foco recaiu sobre a representação do comportamento observado no log por meio de DFGs, recurso particularmente útil como etapa inicial de exploração e de comunicação dos padrões mais frequentes.

Entretanto, como discutido na Seção 2.4, o DFG deve ser entendido como uma abstração de sucessões imediatas, sem semântica operacional completa. Para análises que exigem maior fidelidade comportamental, como a explicitação de concorrência, sincronizações, escolhas exclusivas e verificação de propriedades do processo, o paradigma de Redes de Petri se apresenta como uma evolução conceitual e metodológica mais apropriada.

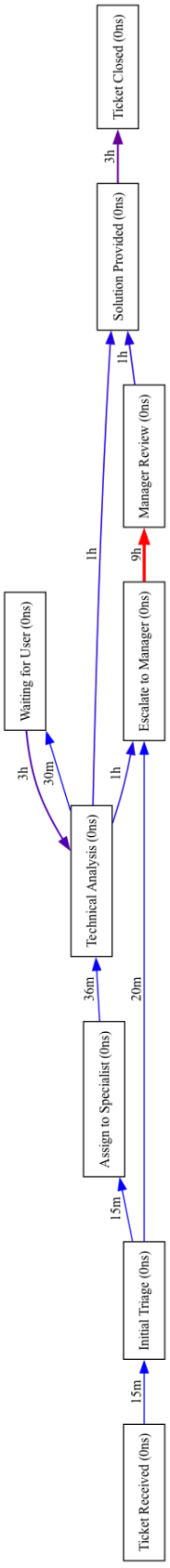
Dessa forma, na seção seguinte aprofundaremos os argumentos introduzidos em 2.4, estabelecendo o fundamento necessário para, na seção posterior, aplicar algoritmos de descoberta e análise que representem a Mineração de Processos diretamente no formalismo de Redes de Petri.

Figura 6.5 – Grafo de Sequência Direta (DFG) do processo de helpdesk gerado pelo PM4Py, baseado na frequência das transições.



Fonte: elaboração própria a partir do código no apêndice C

Figura 6.6 – Grafo de Sequência Direta (DFG) do processo de helpdesk gerado pelo PM4Py, enriquecido com métricas de desempenho temporal.



Fonte: elaboração própria a partir do código no apêndice C

7 Redes de Petri além do DFG na Mineração de Processos

Conforme antecipado nas Seções 2.4 e 6, DFG oferece uma visão inicial valiosa dos fluxos processuais, mas apresenta limitações significativas que justificam a adoção de formalismos mais robustos. A análise prática realizada na Seção 6 demonstrou empiricamente tanto a utilidade quanto as restrições dos DFGs na representação do processo de *helpdesk*. Esta experiência concreta fornece o contexto necessário para uma discussão mais profunda sobre as deficiências conceituais dos DFGs e as vantagens comparativas das Redes de Petri, tema introduzido teoricamente na Seção 2.4 e que será agora detalhado.

Na prática cotidiana da Mineração de Processos, o DFG costuma ser a primeira janela para o log: um retrato rápido das sucessões imediatas observadas. Entretanto, esse retrato não é um modelo comportamental completo; é uma abstração útil, mas perigosa quando interpretada como se tivesse semântica de execução. Em particular, o DFG tende a exagerar possibilidades (aceitando traços não suportados pelo processo real), sofre com concorrência (gerando "espaguete" e laços artificiais) e é frequentemente simplificado por limiares de frequência que criam "lacunas invisíveis" o que pode induzir conclusões equivocadas sobre conformidade e desempenho (AALST, 2019).

A evolução conceitual que tornou o Mineração de Processos uma disciplina mais analítica do que meramente descritiva passou, de forma decisiva, pela adoção de Redes de Petri como linguagem de descoberta e verificação. A razão é simples: enquanto o DFG se comporta como um "mapa de estradas", a Rede de Petri atua como um "sistema de controle de tráfego": explicita estados, regras de disparo e sincronizações, permitindo falar com rigor sobre o que pode (ou não pode) acontecer (MURATA, 1989; AALST; WEIJTERS; MARUSTER, 2004).

7.1 DFG versus Rede de Petri

Dado um log de eventos com casos $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, o DFG captura relações de "seguir diretamente" (quantas vezes a é imediatamente seguido por b). Essa representação é valiosa para exploração inicial, porém o grafo resultante não incorpora, em sua estrutura, conceitos fundamentais de estado, habilitação (*enablement*), sincronização e exclusividade(AALST, 2019).

Uma Rede de Petri $N = (P, T, F, M_0)$ descreve lugares (P), transições (T), arcos (F) e uma marcação inicial (M_0). O "jogo de tokens" fornece uma semântica operacional precisa: transi-

ções disparam somente quando habilitadas pela marcação, alterando o estado do sistema (MURATA, 1989). Esse detalhe muda tudo: com semântica, vêm propriedades verificáveis e técnicas de diagnóstico.

Tabela 7.1 – Capacidades típicas: DFG vs Redes de Petri.

Comportamento	**DFG**	**Rede de Petri**
Paralelismo/concorrência	Fraco/ambíguo	Explícito (AND-split/join via lugares)
Sincronização	Ausente	Natural (junções por pré-condições)
Escolhas exclusivas	Múltiplos arcos sem semântica	Modelável com estruturas de decisão
Loops complexos/aninhados	Pode "inflar" possibilidades	Controlados por condições de habilitação
Recursos/estados intermediários	Não modela estado	Lugares podem codificar estado/recursos

O ponto central é que o DFG mostra ocorrências locais , mas não define regras globais de execução. Redes de Petri, ao contrário, permitem capturar simultaneidade com sincronização (por exemplo, "verificar estoque" e "calcular frete" em paralelo, juntando antes de "processar pagamento") de modo que o modelo não "invente" combinações indevidas (MURATA, 1989; AALST, 2019).

O marco histórico foi o α-algorithm, que consolidou a ideia de extrair relações do log para construir uma Rede de Petri (*workflow net*) (AALST; WEIJTERS; MARUSTER, 2004). A partir daí, uma linhagem inteira de métodos passou a equilibrar qualidade e robustez:

- Heuristics Miner: melhora a tolerância a ruído usando medidas de dependência baseadas em frequência, mitigando fragilidades do α em logs imperfeitos (WEIJTERS; AALST; MEDEIROS, 2006).
- Inductive Miner: busca modelos bloco-estruturados e "corretos por construção", favorecendo propriedades como *soundness* em muitas variantes (LEEMANS; FAHLAND; AALST, 2013).
- Split Miner: combina filtragem de DFG com identificação de gateways para obter modelos simples e acurados, com garantias (AUGUSTO *et al.*, 2017).

Essa linha do tempo ilustra um fato: o DFG pode ser um ponto de partida, mas os avanços mais sólidos emergem quando se chega a modelos com semântica executável, pois é aí que a "qualidade" deixa de ser estética e se torna verificável.

Com Redes de Petri, a pergunta muda de "o que aparece no log?" para "o que é permitido pelo processo?", habilitando checagem de conformidade por técnicas como *token-based replay*, na qual cada caso é reexecutado no modelo e são contabilizados desvios, permitindo

extrair métricas de ajuste e adequação ao comportamento observado (ROZINAT; van der Aalst, 2008). Em domínios regulados, essa transição é decisiva, pois Redes de Petri permitem comprovar violações de ordem obrigatória, separação de deveres ou sincronizações críticas, quando a semântica do processo exige isso.

Ao explicitar estados, Redes de Petri suportam análises como vivacidade, *deadlocks*, limitação, alcançabilidade e *soundness* em modelos de workflow, conectando a Mineração de Processos à Verificação Formal e à Análise de Sistemas Concorrentes (MURATA, 1989).

Outro problema prático é a supergeneralização, quando a representação sugere que "tudo pode seguir tudo". DFGs, especialmente em presença de concorrência e simplificações por frequência, podem induzir modelos excessivamente permissivos, enquanto Redes de Petri, ao introduzir condições de habilitação e sincronização, permitem impor restrições estruturais que preservam a inteligibilidade do processo e reduzem falsos positivos de caminho (AALST, 2019).

A maturidade do formalismo reflete-se no ecossistema, com ferramentas como o ProM viabilizando descoberta, conformidade e análise sobre modelos formais, enquanto DFGs permanecem como ponto de entrada visual. Apesar disso, os ganhos mais profundos continuam associados a modelos com semântica e verificabilidade (DONGEN *et al.*, 2005; AALST, 2019).

Em síntese, DFGs são adequados para exploração rápida, mas não devem ser confundidos com modelos executáveis. Redes de Petri introduzem semântica operacional, propriedades verificáveis, descoberta com garantias e conformidade mensurável, permitindo que a Mineração de Processos avance do "desenho do caminho" para a "engenharia do processo" (AALST; WEIJTERS; MARUSTER, 2004; ROZINAT; van der Aalst, 2008; AALST, 2019).

8 Implementação da Análise com Redes de Petri

8.1 Estrutura do Código

O código (vide apêndice D) implementa uma análise completa de processos utilizando Redes de Petri sobre dados de helpdesk. A implementação segue uma abordagem estruturada:

1. **Carregamento e preparação de dados**: Conversão do log CSV para formato PM4Py
2. **Descoberta de modelos**: Aplicação de três algoritmos de mineração
3. **Análise de conformidade**: Avaliação do alinhamento entre log e modelo
4. **Análise estrutural**: Verificação de propriedades da rede
5. **Simulação**: Geração de casos sintéticos
6. **Identificação de bottlenecks**: Análise de desempenho
7. **Relatório final**: Consolidação dos resultados

8.2 Algorítmos Implementados

8.2.1 Alpha Miner

Algoritmo clássico que constrói Redes de Petri baseado em relações de precedência entre atividades. Retorna uma tupla contendo:

- `net`: Rede de Petri descoberta
- `initial_marking`: Marcação inicial
- `final_marking`: Marcação final

8.2.2 Inductive Miner

Algoritmo moderno que garante *soundness* (correção estrutural) do modelo. Menos sensível a ruídos que o Alpha Miner.

8.2.3 Heuristics Miner

Algoritmo robusto a ruídos que primeiro gera uma *Heuristics Net* e depois converte para Rede de Petri.

8.3 Métricas de Conformidade Calculadas

- **Fitness**: Mede quanto o log se encaixa no modelo (0-1)
- **Precision**: Mede quão específico é o modelo
- **Generalization**: Avalia a capacidade de generalização
- **Tokens faltando**: Quantidade de tokens ausentes durante replay
- **Tokens restantes**: Tokens que permanecem apos execução

8.4 Visualizações Geradas

O código gera quatro arquivos de visualização:

1. `helpdesk_petri_alpha.png`: Modelo Alpha Miner
2. `helpdesk_petri_inductive.png`: Modelo Inductive Miner
3. `helpdesk_petri_heuristics.png`: Modelo Heuristics Miner
4. `helpdesk_heuristics_net.png`: Heuristics Net

8.5 Resultados Exportados

- `resultados_petri.json`: Métricas em formato JSON
- Saida textual: Estatísticas e análises detalhadas
- Imagens PNG: Modelos visuais para documentação

8.6 Requisitos Técnicos

```
Python 3.7+
pm4py[all]
pandas
numpy
matplotlib
```

8.6.1 Execução

Para executar a análise:

```
python petri_helpdesk_analysis.py
```

A saída inclui estatísticas do processo, métricas de conformidade, identificação de bottlenecks e recomendações para melhoria.

9 Do DFG à Rede de Petri

A análise por DFG, desenvolvida no Capítulo 6, cumpre um papel exploratório: explicita as sucessões imediatas mais frequentes e fornece uma leitura rápida de variabilidade e possíveis gargalos. Entretanto, conforme discutido em 7, o DFG não constitui um modelo comportamental com semântica operacional. Em particular, ele não codifica estado, habilitação, sincronização e exclusividade de forma verificável, o que limita análises que dependem de execução e verificação formal.

Dessa limitação decorre a necessidade de uma etapa metodológica adicional: a construção automatizada de modelos em Redes de Petri a partir do log. Essa decisão não é meramente estética; trata-se de permitir que o processo seja analisado como um sistema executável , habilitando: (i) descoberta de modelo com semântica; (ii) comparação entre algoritmos clássicos (robustez vs simplicidade); (iii) *conformance checking* (replay com métricas de ajuste); (iv) diagnóstico de gargalos com base em tempos médios observados; e (v) exportação de artefatos auditáveis (figuras, tabelas e JSON) para reprodutibilidade.

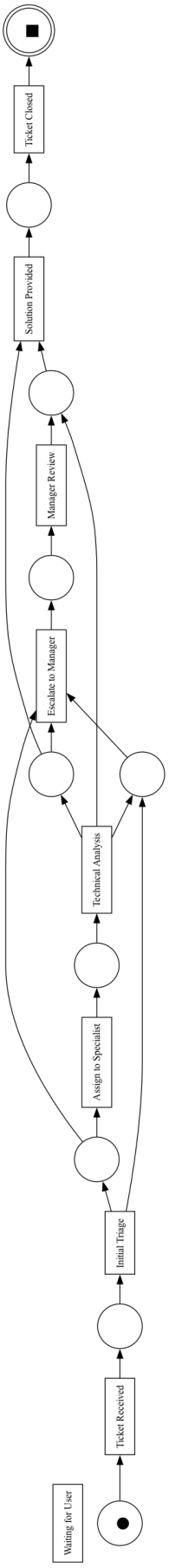
Assim, o script desenvolvido (Apêndice D) materializa a transição metodológica proposta: parte-se do log e obtêm-se modelos formais, permitindo que as conclusões deixem de ser apenas descritivas (DFG) e passem a ser também diagnósticas e verificáveis (Rede de Petri).

9.1 Modelos descobertos: Alpha Miner, Heuristics Miner e Inductive Miner

Para evidenciar como diferentes pressupostos geram modelos distintos, empregaram-se três algoritmos clássicos de descoberta. O *Alpha Miner* tende a produzir modelos compactos em logs com baixo ruído; o *Heuristics Miner* explora frequências para filtrar relações fracas; e o *Inductive Miner* busca modelos bloco-estruturados, frequentemente favorecendo propriedades de correção estrutural.

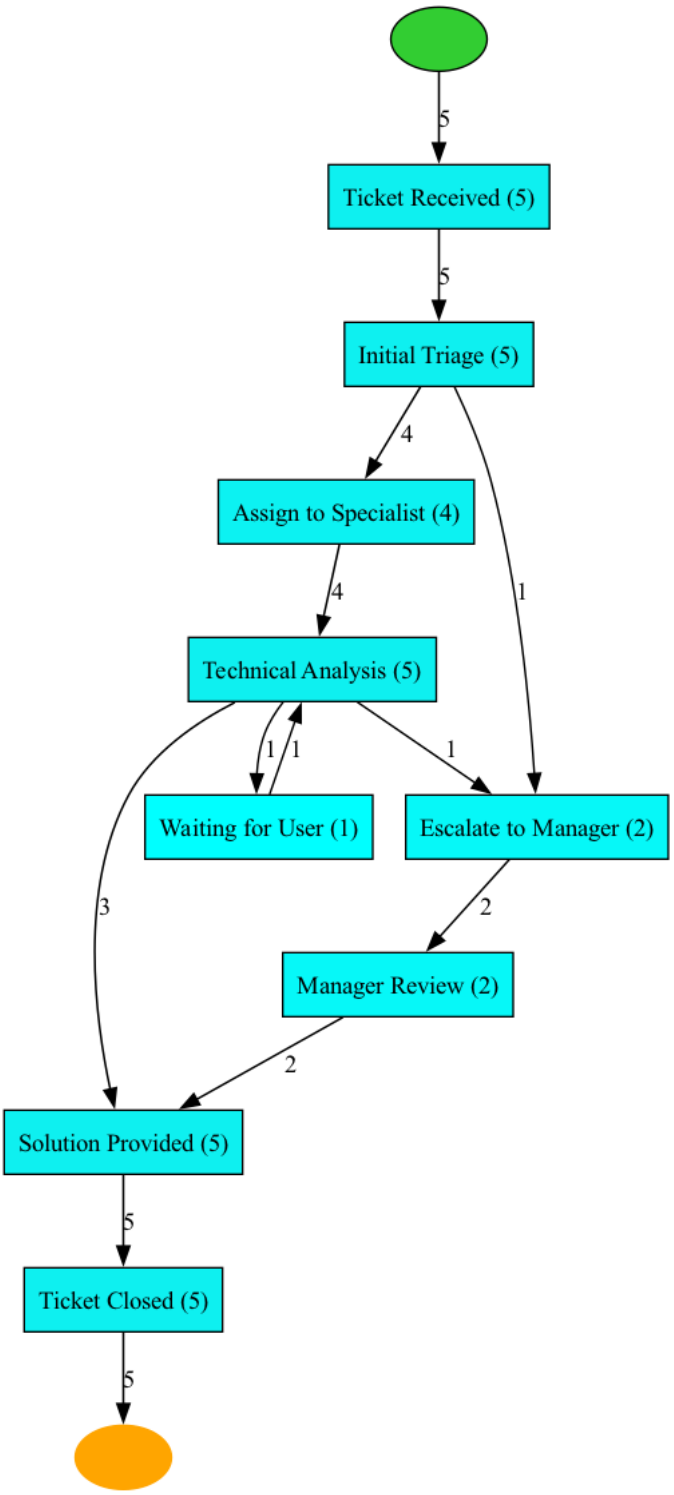
Os modelos resultantes são apresentados nas Figuras 9.1, 9.2 e 9.3. Em particular, a comparação entre as Figuras 9.1 e 9.3 é útil para discutir a diferença entre (i) uma descoberta mais direta baseada em relações de precedência e (ii) uma descoberta orientada a estruturas bem-formadas; já a Figura 9.2 explicita o papel de medidas de dependência/frequência na filtragem de relações pouco relevantes.

Figura 9.1 – Rede de Petri descoberta para o processo de helpdesk via *Alpha Miner*.



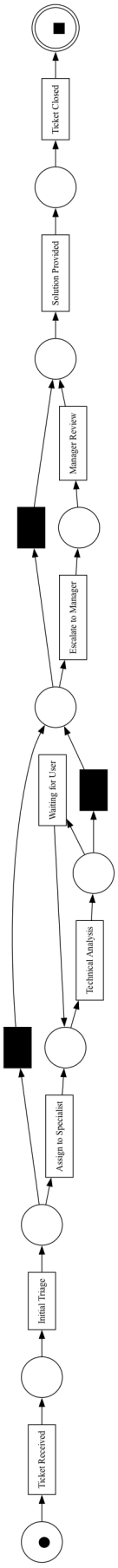
Fonte: elaboração própria.

Figura 9.2 – Heuristics Net descoberta para o processo de helpdesk via *Heuristics Miner* (ênfase em frequência/-dependência).



Fonte: elaboração própria.

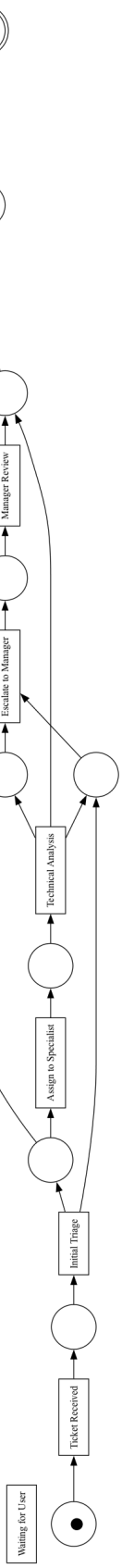
Figura 9.3 – Rede de Petri descoberta para o processo de helpdesk via *Inductive Miner*.



Fonte: elaboração própria.

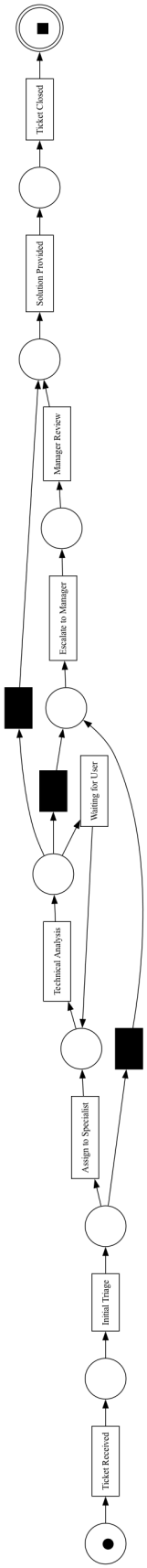
Além dos modelos descobertos, foi gerada uma visualização enriquecida, útil para comunicação e inspeção do fluxo. A Figura 9.4 consolida a leitura do processo em um modelo executável, servindo como referência visual para interpretar, ao longo da checagem de conformidade, onde se concentram desvios e reencaminhamentos. Complementarmente, a Figura 9.5 apresenta uma rede derivada a partir de relações heurísticas, reforçando a interpretação baseada em evidências de frequência.

Figura 9.4 – Rede de Petri do helpdesk em visualização enriquecida (saída do pipeline de análise).



Fonte: elaboração própria.

Figura 9.5 – Rede de Petri derivada a partir de relações heurísticas (saída do pipeline).



Fonte: elaboração própria.

Como prática de transparência metodológica, os resultados sumarizados do experimento foram exportados em formato JSON. Esse artefato registra estatísticas gerais do log, métricas de conformidade e um resumo analítico (por exemplo, gargalos por tempo médio), permitindo auditoria independente e repetição do experimento em novas amostras.

O arquivo `resultados_petri.json` é apresentado no código 9.1. Sua existência é especialmente útil para: (i) versionar resultados junto ao código; (ii) comparar execuções (por exemplo, com diferentes filtros/limiares); e (iii) integrar a análise com rotinas externas de relatório.

Código 9.1 – Saída do experimento (`resultados_petri.json`).

```
{
  "statistics": {
    "cases": 5,
    "fitness": 0.8430349257326375,
    "problematic_cases": 5
  },
  "bottlenecks": {
    "Manager Review": 90.0,
    "Technical Analysis": 66.0,
    "Waiting for User": 180.0,
    "Solution Provided": 234.0,
    "Escalate to Manager": 562.5
  },
  "report_summary": {
    "cases": 5,
    "fitness": 0.8430349257326375,
    "variants": 4,
    "bottlenecks": 5,
    "potential_deadlocks": 0
  }
}
```

Em síntese, a etapa em Redes de Petri fecha o arco metodológico iniciado no DFG: o que antes era apenas "rastro de sucessões" torna-se um modelo executável, permitindo diagnóstico com base em semântica, e não apenas em adjacência observada.

9.2 Análise e Discussão da Saída do Script `petri_helpdesk.py`

Esta seção interpreta e discute criticamente os resultados produzidos pelo script `petri_helpdesk.py`, cuja saída integral de execução em terminal encontra-se no Anexo A. A intenção aqui é transformar a listagem técnica do anexo em uma narrativa analítica: o que foi medido, como os números se relacionam com o processo de *helpdesk* e quais implicações

metodológicas e gerenciais emergem quando se troca o DFG por um modelo executável em Rede de Petri.

9.2.1 Visão geral do pipeline executado

Conforme indicado no Anexo A, o script segue um pipeline em oito etapas: (1) carregamento do log; (2) descoberta de modelos (*Alpha Miner, Inductive Miner e Heuristics Miner*); (3) análise de conformidade; (4) análise estrutural do modelo; (5) simulação; (6) identificação de gargalos; (7) geração de visualizações avançadas; e (8) relatório executivo final.

A execução confirma que o experimento se apoia em uma amostra reduzida, porém completa, composta por 5 casos e 34 eventos. Essa característica é adequada para fins didáticos e exploratórios, mas impõe um alerta: métricas de descoberta e conformidade tendem a ser mais instáveis em bases pequenas, pois cada traço individual tem impacto proporcionalmente elevado sobre o modelo e sobre os indicadores.

9.2.2 Descoberta de modelos: diferenças estruturais relevantes

A etapa de descoberta produz três modelos com naturezas distintas, evidenciando por contraste o porquê de se empregar mais de um algoritmo em Mineração de Processos.

O modelo descoberto via *Alpha Miner* apresenta, segundo o Anexo A, 9 transições (atividades visíveis), 10 lugares(estados) e 22 arcos (relações), além de marcações inicial e final explícitas. Em termos interpretativos, trata-se de um modelo compacto e diretamente ligado às relações de precedência observadas, adequado para logs pequenos e com baixo ruído. Como efeito colateral, o *Alpha Miner* costuma ser sensível a variações e comportamentos infrequentes, podendo gerar modelos que exigem cuidado na leitura de conformidade.

O *Inductive Miner* gera um modelo com 12 transições e 10 lugares, destacando-se pela mensagem *"Soundness garantida pelo algoritmo"* (Anexo A). Do ponto de vista metodológico, isso é crucial: a descoberta tende a impor uma organização estrutural que evita inconsistências clássicas (como impasses inevitáveis) e favorece modelos bem-formados. Em termos práticos, a diferença no número de transições sugere que o *Inductive Miner* pode estar introduzindo estruturas auxiliares (por exemplo, transições silenciosas/invisíveis) para acomodar escolhas e sincronizações de forma consistente.

O *Heuristics Miner* resulta em uma *Heuristics Net* com 9 nós e indicação de alta robustez a ruído. Embora não seja uma Rede de Petri no sentido estrito, ela é especialmente útil para realçar relações sustentadas por frequência e dependência, filtrando conexões frágeis. Na prática, essa rede funciona como um intermediário entre o DFG (muito descritivo) e a Rede de Petri (executável), servindo como evidência de quais relações são mais estáveis sob o ponto de vista estatístico.

9.2.3 Conformidade

A análise de conformidade por *token-based replay* reporta as métricas: *fitness* = 0,843, *precision* = 0,558 e *generalization* = 0,428 (Anexo A). Essas três medidas, em conjunto, descrevem um equilíbrio delicado do modelo descoberto:

- *Fitness* (0,843): o modelo consegue reproduzir parte substancial do comportamento observado, mas com desvios. Isso é consistente com a existência de *tokens* faltando (5) e *tokens* restantes (11), sinalizando que, ao simular os casos do log sobre o modelo, em alguns momentos o modelo precisaria de *tokens* que não estavam disponíveis (desvio por falta) e, em outros, termina com *tokens* que não foram consumidos até o final (desvio por sobra). Em termos intuitivos: há aderência global razoável, mas há *mismatch* local importante.

- Casos conformes (0/5): apesar do *fitness* relativamente alto, nenhum caso foi classificado como plenamente conforme. Essa combinação é comum quando se adota uma noção estrita de conformidade por traço (um pequeno desvio já invalida o caso), ao mesmo tempo em que o ajuste global *(fitness)* tolera desvios pontuais. Em bases pequenas, esse efeito fica amplificado: um único comportamento raro pode forçar o modelo a escolhas que, ao serem *replayed*, geram penalidades em todos os traços.

- Precisão (0,558): indica que o modelo não é altamente específico; isto é, ele ainda permite comportamentos que não aparecem no log. Essa leitura é coerente com a etapa de simulação, em que o *script* explicitamente observa que a simulação explora caminhos possíveis no modelo e pode produzir comportamentos não observados no logc (Anexo A).

- Generalização (0,428): sugere baixa capacidade de generalizar a partir de poucos exemplos sem perder qualidade. Novamente, este resultado é compatível com a amostra reduzida: com apenas 5 casos, é difícil distinguir ruído de padrão e, portanto, o modelo tende a oscilar entre ser permissivo demais (reduzindo precisão) ou ajustar-se demais aos poucos traços (reduzindo generalização quando medido por técnicas padrão).

Assim essa seção confirma que todos os 5 casos apresentam problemas e lista traços com 0, 1 e 2 *tokens* faltando (Anexo A). Em termos narrativos para o TCC: o modelo é suficientemente expressivo para capturar o fluxo principal, mas evidencia que o log contém variações que desafiam uma reconstrução perfeita com o formalismo e/ou com a configuração empregada, reforçando o argumento de que Redes de Petri não apenas desenham o processo, mas testam o processo.

9.2.4 Estrutura do modelo

Na análise estrutural, o *script* identifica 4 sequências e 2 pontos de escolha (XOR/OR), reportando 0 *loops* identificados e nenhum *deadlock* potencial (Anexo A). Essa saída é particularmente útil para costurar com a narrativa do capítulo: os DFGs já indicavam ramificações em *Initial Triage* e *Technical Analysis*; aqui, tais ramificações deixam de ser apenas "arestas alternativas" e passam a ser tratadas como construções do modelo, aproximando a leitura de uma semântica de execução.

A verificação de *soundness* realizada no *script* confirma a presença das marcações inicial e final, mas é importante registrar que tal verificação (como apresentada na saída) é uma checagem mínima: assegura a existência das marcações, porém não equivale, sozinha, a uma prova completa de *soundness*. Ainda assim, o contraste com o *Inductive Miner* é didaticamente relevante, pois ali a própria estratégia de descoberta declara garantia estrutural, reforçando a diferença entre "detectar marcações" e "construir um modelo com garantias".

9.2.5 Simulação

A simulação do processo reporta 10 casos simulados e 10 variantes únicas, acompanhada de trajetórias extremamente longas com repetição de *Waiting for User* (Anexo A). Esse resultado deve ser interpretado com cautela e, ao mesmo tempo, pode ser usado a favor do argumento central do capítulo: modelos executáveis permitem *ress-test* do comportamento permitido.

Em termos metodológicos, a simulação não tem como objetivo "reproduzir fielmente" o log original, e sim explorar o espaço de comportamentos possíveis do modelo. Quando o modelo possui baixa precisão (0,558), é esperado que o gerador encontre trajetórias que não ocorreram nos 5 casos observados. Assim, a simulação serve como um holofote: ela evidencia onde o modelo pode estar permissivo demais e onde seria necessário impor restrições adicionais.

9.2.6 Gargalos análise

A análise de gargalos lista transições com alto tempo médio, destacando: *Escalate to Manager* (562,5 min), *Solution Provided* (234,0 min), *Waiting for User* (180,0 min), *Manager Review* (90,0 min) e *Technical Analysis* (66,0 min) (Anexo A). Mesmo em amostra reduzida, o ordenamento já sugere uma leitura gerencial consistente:

- *Escalate to Manager* é o gargalo mais crítico: quando há escalonamento, o tempo associado explode, sugerindo fila/espera por decisão gerencial ou restrição de capacidade de gestor.

- *Waiting for User* aparece como atraso expressivo: ainda que não seja a etapa mais frequente, é uma etapa dominante em tempo quando ocorre, reforçando a hipótese de melhoria por coleta inicial de informações, padronização de solicitação de dados e lembretes automáticos.
- *Solution Provided* com tempo alto indica que "resolver e formalizar a solução" pode estar agregando espera (aprovação, documentação, validação), e não apenas execução técnica.

Além disso, o *script* identifica lugares de espera/estoque e pontos de sincronização do tipo "2 entradas / 1 saída", o que permite uma leitura compatível com a intuição de Redes de Petri: gargalos não são apenas "transições lentas", mas podem refletir acúmulo em lugares (fila/estoque), restrições de capacidade e necessidade de sincronização antes de prosseguir.

9.2.7 Síntese interpretativa

Em conjunto, a saída do Anexo A sustenta três conclusões úteis:

1. A passagem de DFG para Rede de Petri muda o tipo de pergunta que a análise consegue responder: deixa-se de apenas visualizar sucessões para testar execução, conformidade e permissividade.
2. As métricas de conformidade apontam aderência global, mas não conformidade estrita: *fitness* razoável com 0/5 casos conformes indica desvios distribuídos e/ou modelo permissivo, o que é instrutivo para discutir limites de amostras pequenas e escolhas do algoritmo de descoberta.
3. Os gargalos aparecem como alvos de intervenção priorizáveis: escalonamento e espera do usuário dominam o tempo e sugerem ações de melhoria diretamente associáveis à prática de *helpdesk*.

Portanto, a saída do *script* não deve ser lida como "um relatório automático", mas como um artefato experimental: ela evidencia, simultaneamente, resultados (gargalos, escolhas, ausência de *deadlock* potencial) e tensões metodológicas (precisão, generalização e simulação com trajetórias não observadas), que são precisamente o tipo de discussão que justifica a adoção de Redes de Petri além do DFG.

10 Considerações Finais

Ao longo deste trabalho, foi construído um percurso que parte da motivação institucional e do enquadramento conceitual da Mineração de Processos, avançando para a discussão sobre a natureza dos registros de eventos, seus desafios de qualidade (*missing data*, granularidade, ruídos e inconsistências) e os cuidados necessários para que inferências sejam, de fato, sustentadas pelo que está observado no log. Em seguida, foram explorados modelos e técnicas de representação do comportamento processual, com destaque para estruturas orientadas à sequência de execução, como o DFG, e para a necessidade de distinguir, com clareza metodológica, aquilo que emerge empiricamente dos dados daquilo que seria próprio de uma modelagem normativa. Esse encadeamento permitiu organizar um *pipeline* analítico coerente, capaz de apoiar análises comparáveis e tecnicamente justificadas.

Com essa base, o trabalho consolidou um conjunto de procedimentos voltados à modelagem e análise de processos, enfatizando a consistência entre fundamentação teórica, organização dos dados e possibilidades de avaliação do comportamento observado. Foram delineados conceitos, etapas e critérios que permitem estruturar uma análise rigorosa, desde a preparação e validação de registros, passando pela construção de representações derivadas do log, até a leitura crítica dos resultados, preservando, sempre que necessário, a distinção entre o que é inferido a partir dos registros e aquilo que seria próprio de uma especificação formal do processo. O arcabouço apresentado foi descrito de modo a favorecer sua portabilidade, podendo ser adaptado a outras fontes de dados discutidas no capítulo de bases de dados, desde que elas permitam a reconstrução de eventos com identificadores de caso, carimbos temporais e atividades ou estados relevantes.

Embora o arcabouço metodológico e os procedimentos descritos sejam diretamente aplicáveis ao contexto institucional visado, a execução de um estudo empírico completo sobre a base de dados da Central de Emendas ficou condicionada a fatores externos ao desenho metodológico, em especial o tempo disponível e o processo de credenciamento e autorização de acesso, reconhecidamente rigoroso. Dessa forma, a aplicação ponta a ponta do *pipeline* delineado, incluindo extração, curadoria, construção de logs de eventos e análise por meio de representações como DFG e modelos correlatos, permanece como etapa natural de continuidade. Sua realização futura não se limita a uma demonstração de viabilidade, mas constitui uma etapa de validação metodológica, com potencial para produzir resultados diretamente acionáveis, como identificação de gargalos, variações de fluxo, pontos de retrabalho e comparações entre diferentes janelas temporais.

Em perspectiva de continuidade, destaca-se ainda a relevância de consolidar um *pipeline* institucional de extração, curadoria e, quando aplicável, anonimização de dados, com

ênfase em rastreabilidade, reprodutibilidade e versionamento. Tal formalização transforma uma atividade pontual de pesquisa em procedimento sustentável e auditável, permitindo que o núcleo metodológico aqui proposto seja estendido às demais bases institucionais discutidas ao longo do trabalho, respeitando suas particularidades sem comprometer a comparabilidade das análises.

Por fim, o aprofundamento da modelagem constitui um avanço natural, com a incorporação de representações mais expressivas, como Redes de Petri, permitindo investigar aspectos estruturais do processo, alternativas de fluxo, sincronizações e padrões recorrentes, bem como a análise de mudanças de comportamento ao longo do tempo (*drift*). Em um nível mais avançado, a integração entre Mineração de Processos e Métodos Formais, por meio da verificação de propriedades em modelos derivados de logs, abre espaço para análises com maior grau de robustez e auditabilidade, especialmente relevantes em cenários institucionais sensíveis. Nesse sentido, o valor do estudo reside não apenas na aplicação imediata, mas na oferta de um fundamento técnico que sustente decisões futuras com maior confiabilidade, comparabilidade e transparência.

REFERÊNCIAS

AALST, W. M. P. V. D. *Process Mining: Data Science in Action*. 2. ed. Berlin, Heidelberg: Springer, 2016. ISBN 978-3-662-49850-7. Disponível em: <https://doi.org/10.1007/978-3-662-49851-4>. Citado 3 vezes nas páginas 22, 23 e 31.

AALST, W. M. P. van der. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin, Heidelberg: Springer, 2011. ISBN 978-3-642-19344-6. Disponível em: <https://doi.org/10.1007/978-3-642-19345-3>. Citado 4 vezes nas páginas 19, 27, 31 e 33.

____. *Process Mining: Data Science in Action*. [S.l.]: Springer, 2016. Citado 2 vezes nas páginas 21 e 46.

AALST, W. M. P. van der *et al. Process Mining Manifesto*. Eindhoven/Berlin/Como, 2011. Disponível em <https://www.tf-pm.org/upload/1580738212409.pdf>. Disponível em: <https://www.tf-pm.org/upload/1580738212409.pdf>. Citado 3 vezes nas páginas 19, 31 e 46.

AALST, W. M. V. D. A practitioner's guide to process mining: Limitations of the directly-follows graph. *Procedia Computer Science*, v. 164, p. 321–328, 2019. ISSN 1877-0509. CENTERIS 2019 - International Conference on ENTERprise Information Systems / ProjMAN 2019 - International Conference on Project MANagement / HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2019. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050919322367>. Citado 5 vezes nas páginas 15, 27, 54, 55 e 56.

AALST, W. M. van der; WEIJTERS, A.; MARUSTER, L. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 16, n. 9, p. 1128–1142, 2004. Citado 3 vezes nas páginas 54, 55 e 56.

AUGUSTO, A. *et al.* Split miner: Discovering accurate and simple business process models from event logs. In: *2017 IEEE International Conference on Data Mining (ICDM)*. [S.l.: s.n.], 2017. p. 1–10. Citado na página 55.

BAKHSHI, A.; HASSANNAYEBI, E.; SADEGHI, A. H. *Optimizing Sepsis Care through Heuristics Methods in Process Mining: A Trajectory Analysis*. 2023. Disponível em: <https://arxiv.org/abs/2303.14328>. Citado 3 vezes nas páginas 26, 28 e 33.

BEYEL, H. H.; AALST, W. M. P. Using translucent activity relationships frequencies to enhance process discovery. *Process Science*, v. 2, n. 1, p. 15, 2025. ISSN 2948-2178. Published online: July 9, 2025. Disponível em: <https://doi.org/10.1007/s44311-025-00010-y>. Citado 3 vezes nas páginas 26, 27 e 28.

BONDY, J. A.; MURTY, U. S. R. *Graph Theory*. New York: Springer, 2008. Citado na página 20.

Central das Emendas. *Central das Emendas — plataforma de dados e transparência sobre emendas parlamentares*. 2025. Acessado em: 13 nov. 2025. Disponível em: <https://www.centraldasemendas.info/>. Citado na página 24.

DONGEN, B. F. van *et al.* The prom framework: A new era in process mining tool support. In: CIARDO, G.; DARONDEAU, P. (Ed.). *Applications and Theory of Petri Nets 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 444–454. ISBN 978-3-540-31559-9. Citado na página 56.

EFFENDI, Y. A.; KIM, M. Timed genetic process mining for robust tracking of processes under incomplete event log conditions. *Electronics*, v. 13, n. 18, 2024. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/13/18/3752>. Citado 3 vezes nas páginas 26, 27 e 28.

IBM. *O que é Process Mining? Uma introdução*. s.d. International Business Machines Corporation. Disponível em: <https://www.ibm.com/br-pt/think/topics/process-mining>. Acesso em: 12 nov. 2025. Citado na página 19.

INDULSKA, M. *et al.* Business process modeling: Current issues and future challenges. In: ECK, P. van; GORDIJN, J.; WIERINGA, R. (Ed.). *Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 501–514. ISBN 978-3-642-02144-2. Citado na página 32.

LEEMANS, S. J.; FAHLAND, D.; AALST, W. M. van der. Discovering block-structured process models from event logs-a constructive approach. In: SPRINGER. *International Conference on Applications and Theory of Petri Nets and Concurrency*. [S.l.], 2013. p. 311–329. Citado na página 55.

LEEMANS, S. J.; POPPE, E.; WYNN, M. T. Directly follows-based process mining: Exploration & a case study. In: *2019 International Conference on Process Mining (ICPM)*. [s.n.], 2019. p. 25–32. Disponível em: <https://doi.org/10.1109/ICPM.2019.00015>. Citado 4 vezes nas páginas 15, 22, 27 e 28.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989. Citado 6 vezes nas páginas 19, 22, 23, 54, 55 e 56.

PETERSON, J. L. *Petri Net Theory and the Modeling of Systems*. [S.l.]: Prentice Hall, 1981. Citado na página 23.

PETRI, C. A. *Kommunikation mit Automaten*. Tese (Doutorado) — Universität Hamburg, 1962. Citado na página 22.

ROZINAT, A.; van der Aalst, W. Conformance checking of processes based on monitoring real behavior. *Information Systems*, v. 33, n. 1, p. 64–95, 2008. ISSN 0306-4379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S030643790700049X>. Citado 3 vezes nas páginas 19, 24 e 56.

SANTIAGO, B. M. *Uso de redes de petri na modelagem de alocação de recursos em mineração de processos*. 61 p. Dissertação (Dissertação de Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro, 2019. Ilustrações coloridas. Inclui bibliografia. Citado na página 28.

SHAIMOV, N.; LOMAZOVA, I.; MITSYUK, A. *Discovering Directly-Follows Graph Model for Acyclic Processes*. 2025. Disponível em: <https://arxiv.org/abs/2502.00499>. Citado 3 vezes nas páginas 26, 27 e 28.

WALLIS, W. *A Beginner's Guide to Graph Theory*. [S.l.: s.n.], 2007. ISBN 978-0-8176-4484-0. Citado na página 20.

WEERDT, J. D. *et al.* A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Information Systems*, v. 37, n. 7, p. 654–676, 2012. ISSN 0306-4379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437912000464>. Acesso em: 2024-01-15. Citado na página 36.

WEIJTERS, A.; AALST, W.; MEDEIROS, A. *Process Mining with the Heuristics Miner-algorithm*. [S.l.: s.n.], 2006. v. 166. Citado na página 55.

WEST, D. B. *Introduction to Graph Theory*. 2. ed. Upper Saddle River: Prentice Hall, 2001. Citado na página 20.

APÊNDICE A – Base de dados Helpdesk

```
CaseID,Activity,Resource,Timestamp,Status
1001,Ticket Received,System,2024-01-01 09:00:00,Open
1001,Initial Triage,Agent_John,2024-01-01 09:15:00,In Progress
1001,Assign to Specialist,Agent_John,2024-01-01 09:30:00,In Progress
1001,Technical Analysis,Specialist_Maria,2024-01-01 10:00:00,In
    Progress
1001,Solution Provided,Specialist_Maria,2024-01-01 11:30:00,Resolved
1001,Ticket Closed,Agent_John,2024-01-01 12:00:00,Closed
1002,Ticket Received,System,2024-01-01 09:05:00,Open
1002,Initial Triage,Agent_Sarah,2024-01-01 09:20:00,In Progress
1002,Assign to Specialist,Agent_Sarah,2024-01-01 09:35:00,In Progress
1002,Technical Analysis,Specialist_Maria,2024-01-01 10:30:00,In
    Progress
1002,Waiting for User,Specialist_Maria,2024-01-01 11:00:00,Waiting
1002,Technical Analysis,Specialist_Maria,2024-01-01 14:00:00,In
    Progress
1002,Solution Provided,Specialist_Maria,2024-01-01 15:30:00,Resolved
1002,Ticket Closed,Agent_Sarah,2024-01-02 09:00:00,Closed
1003,Ticket Received,System,2024-01-01 09:10:00,Open
1003,Initial Triage,Agent_John,2024-01-01 09:25:00,In Progress
1003,Escalate to Manager,Agent_John,2024-01-01 09:45:00,In Progress
1003,Manager Review,Manager_David,2024-01-01 11:00:00,In Progress
1003,Solution Provided,Manager_David,2024-01-01 12:30:00,Resolved
1003,Ticket Closed,Agent_John,2024-01-01 13:00:00,Closed
1004,Ticket Received,System,2024-01-02 10:00:00,Open
1004,Initial Triage,Agent_Sarah,2024-01-02 10:15:00,In Progress
1004,Assign to Specialist,Agent_Sarah,2024-01-02 10:30:00,In Progress
1004,Technical Analysis,Specialist_Maria,2024-01-02 11:00:00,In
    Progress
1004,Solution Provided,Specialist_Maria,2024-01-02 12:00:00,Resolved
1004,Ticket Closed,Agent_Sarah,2024-01-02 12:30:00,Closed
1005,Ticket Received,System,2024-01-02 14:00:00,Open
1005,Initial Triage,Agent_John,2024-01-02 14:15:00,In Progress
1005,Assign to Specialist,Agent_John,2024-01-02 14:30:00,In Progress
1005,Technical Analysis,Specialist_Carlos,2024-01-02 15:00:00,In
```

```
    Progress
1005,Escalate to Manager,Specialist_Carlos,2024-01-02 16:00:00,In
    Progress
1005,Manager Review,Manager_David,2024-01-03 09:30:00,In Progress
1005,Solution Provided,Manager_David,2024-01-03 11:00:00,Resolved
1005,Ticket Closed,Agent_John,2024-01-03 11:30:00,Closed
```

APÊNDICE B – Representação Helpdesk como DFG em Python

```
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from collections import Counter, defaultdict
from datetime import datetime, timedelta

# ==================== FUNCOES AUXILIARES ====================
def parse_log_to_df(log_content):
    """Converte string de log em DataFrame"""
    from io import StringIO
    df = pd.read_csv(StringIO(log_content))
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])
    return df

def extract_cases(df):
    """Extrai sequencias de atividades por caso"""
    cases = {}
    for case_id, group in df.groupby('CaseID'):
        # Ordenar por timestamp e extrair atividades
        activities = group.sort_values('Timestamp')['Activity'].tolist
            ()
        cases[case_id] = activities
    return cases

def build_directly_follows_graph(df):
    """Constroi DFG a partir do DataFrame"""
    # Agrupar por caso e ordenar por timestamp
    df_sorted = df.sort_values(['CaseID', 'Timestamp'])

    # Calcular relacoes diretamente-segue
    dfg = Counter()
    total_flows = 0
```

```
    for case_id, group in df_sorted.groupby('CaseID'):
        activities = group['Activity'].tolist()

        # Para cada par consecutivo
        for i in range(len(activities) - 1):
            from_act = activities[i]
            to_act = activities[i + 1]
            dfg[(from_act, to_act)] += 1
            total_flows += 1

    return dfg, total_flows

def build_dfg_with_metrics(df):
    """Constroi DFG com metricas adicionais"""
    df_sorted = df.sort_values(['CaseID', 'Timestamp'])

    dfg_counts = Counter()
    dfg_times = defaultdict(list)
    activity_durations = defaultdict(list)

    for case_id, group in df_sorted.groupby('CaseID'):
        group = group.sort_values('Timestamp')
        activities = group['Activity'].tolist()
        timestamps = group['Timestamp'].tolist()

        # Calcular duracoes das atividades
        for i in range(len(activities)):
            if i < len(activities) - 1:
                duration = (timestamps[i + 1] - timestamps[i]).
                    total_seconds() / 60  # em minutos
                dfg_times[(activities[i], activities[i + 1])].append(
                    duration)

        # Contar transicoes
        for i in range(len(activities) - 1):
            dfg_counts[(activities[i], activities[i + 1])] += 1

    return dfg_counts, dfg_times
```

```
# ==================== VISUALIZACAO DFG ====================
def visualize_dfg_simple(dfg, total_flows, title="DFG - Directly Follows Graph"):
    """Visualizacao simples do DFG usando networkx"""
    G = nx.DiGraph()

    # Adicionar arestas com pesos
    for (from_act, to_act), count in dfg.items():
        weight = count
        frequency = (count / total_flows) * 100

        G.add_edge(from_act, to_act,
                   weight=weight,
                   frequency=frequency,
                   label=f"{count}\n({frequency:.1f}%)")

    # Layout
    plt.figure(figsize=(12, 8))
    pos = nx.spring_layout(G, k=2, iterations=50)

    # Desenhar nos
    node_colors = ['#4285F4' for _ in G.nodes()]  # Azul Google
    node_sizes = [2000 + G.degree(node) * 300 for node in G.nodes()]

    nx.draw_networkx_nodes(G, pos,
                           node_color=node_colors,
                           node_size=node_sizes,
                           alpha=0.9,
                           edgecolors='black',
                           linewidths=2)

    # Desenhar rotulos dos nos
    nx.draw_networkx_labels(G, pos,
                            font_size=10,
                            font_weight='bold',
                            font_color='white')

    # Desenhar arestas com espessura proporcional a frequencia
    edges = G.edges()
```

```
    widths = [G[u][v]['weight'] * 0.5 for u, v in edges]
    edge_labels = nx.get_edge_attributes(G, 'label')

    nx.draw_networkx_edges(G, pos,
                           edge_color='#555555',
                           width=widths,
                           alpha=0.7,
                           arrowsize=20,
                           arrowstyle='-|>',
                           connectionstyle='arc3,rad=0.1')

    # Adicionar rotulos nas arestas
    nx.draw_networkx_edge_labels(G, pos,
                                 edge_labels=edge_labels,
                                 font_size=8,
                                 font_color='red',
                                 bbox=dict(alpha=0.7, boxstyle='round'))

    plt.title(title, fontsize=14, fontweight='bold', pad=20)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

    # Retornar metricas
    return G

def visualize_dfg_advanced(dfg_counts, dfg_times):
    """Visualizacao avancada do DFG com metricas de tempo"""
    G = nx.DiGraph()

    # Calcular metricas agregadas
    edge_metrics = {}
    for (from_act, to_act), times in dfg_times.items():
        if times:
            avg_time = sum(times) / len(times)
            min_time = min(times)
            max_time = max(times)
            count = dfg_counts.get((from_act, to_act), 0)
```

```
        edge_metrics[(from_act, to_act)] = {
            'count': count,
            'avg_time': avg_time,
            'min_time': min_time,
            'max_time': max_time
        }

# Adicionar arestas ao grafo
for (from_act, to_act), metrics in edge_metrics.items():
    label = f"Freq: {metrics['count']}\n"
    label += f"Tempo: {metrics['avg_time']:.1f}m\n"
    label += f"({metrics['min_time']:.0f}-{metrics['max_time']:.0f}
        m)"

    G.add_edge(from_act, to_act,
               weight=metrics['count'],
               avg_time=metrics['avg_time'],
               label=label)

# Layout melhorado
plt.figure(figsize=(14, 10))

# Usar layout hierarquico para processos sequenciais
try:
    pos = nx.multipartite_layout(G, subset_key='subset', align='
        horizontal')
except:
    pos = nx.spring_layout(G, k=3, iterations=100)

# Desenhar nos
nodes = list(G.nodes())
node_colors = plt.cm.Set3(range(len(nodes)))

nx.draw_networkx_nodes(G, pos,
                       node_color=node_colors,
                       node_size=2500,
                       alpha=0.8,
                       edgecolors='black',
                       linewidths=2,
```

```
                        node_shape='s')

# Rotulos dos nos
nx.draw_networkx_labels(G, pos,
                        font_size=9,
                        font_weight='bold',
                        bbox=dict(boxstyle='round,pad=0.3',
                                  facecolor='white',
                                  alpha=0.8))

# Desenhar arestas com cores baseadas no tempo
edges = G.edges()
edge_colors = []
edge_widths = []

for u, v in edges:
    avg_time = G[u][v]['avg_time']
    # Cor baseada no tempo (vermelho = lento, verde = rapido)
    if avg_time > 120:  # Mais de 2 horas
        color = '#FF6B6B'  # Vermelho
    elif avg_time > 60:  # 1-2 horas
        color = '#FFA726'  # Laranja
    else:
        color = '#66BB6A'  # Verde

    edge_colors.append(color)
    edge_widths.append(G[u][v]['weight'] * 0.3)

nx.draw_networkx_edges(G, pos,
                       edge_color=edge_colors,
                       width=edge_widths,
                       alpha=0.7,
                       arrowsize=25,
                       arrowstyle='fancy',
                       connectionstyle='arc3,rad=0.2')

# Rotulos das arestas
edge_labels = nx.get_edge_attributes(G, 'label')
nx.draw_networkx_edge_labels(G, pos,
```

```
                                edge_labels=edge_labels,
                                font_size=7,
                                bbox=dict(boxstyle='round,pad=0.5',
                                          facecolor='white',
                                          alpha=0.9))

    # Legenda de cores
    legend_elements = [
        plt.Line2D([0], [0], color='#66BB6A', lw=4, label='Rapido (< 60
            min)'),
        plt.Line2D([0], [0], color='#FFA726', lw=4, label='Moderado
            (60-120 min)'),
        plt.Line2D([0], [0], color='#FF6B6B', lw=4, label='Lento (> 120
            min)')
    ]

    plt.legend(handles=legend_elements, loc='upper right', fontsize=9)
    plt.title('DFG com Metricas de Tempo e Frequencia', fontsize=14,
        fontweight='bold', pad=20)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

    return G, edge_metrics

def generate_process_summary(dfg, total_flows, cases):
    """Gera um resumo textual do processo baseado no DFG"""

    print("=" * 60)
    print("ANALISE DO PROCESSO - RESUMO DO DFG")
    print("=" * 60)

    # 1. Estatisticas basicas
    print(f"\n ESTATISTICAS BASICAS:")
    print(f"    Total de casos: {len(cases)}")
    print(f"    Total de transicoes: {total_flows}")
    print(f"    Transicoes unicas: {len(dfg)}")

    # 2. Atividades mais frequentes
```

```
activity_counts = Counter()
for (from_act, _), count in dfg.items():
    activity_counts[from_act] += count

# Adicionar ultima atividade
for case_activities in cases.values():
    if case_activities:
        activity_counts[case_activities[-1]] += 1

print(f"\n ATIVIDADES MAIS FREQUENTES:")
for activity, count in activity_counts.most_common(5):
    percentage = (count / (total_flows + len(cases))) * 100
    print(f"   {activity}: {count} ocorrencias ({percentage:.1f}%)"
        )

# 3. Fluxos mais comuns
print(f"\n FLUXOS MAIS COMUNS:")
for (from_act, to_act), count in dfg.most_common(5):
    percentage = (count / total_flows) * 100
    print(f"   {from_act} -> {to_act}: {count} vezes ({percentage
        :.1f}%)")

# 4. Pontos de decisao
print(f"\n PONTOS DE DECISAO (RAMIFICACOES):")
from_activities = defaultdict(list)
for (from_act, to_act) in dfg.keys():
    from_activities[from_act].append(to_act)

for activity, next_activities in from_activities.items():
    if len(next_activities) > 1:
        print(f"   {activity} pode levar para: {', '.join(
            next_activities)}")

# 5. Inicio e fim do processo
start_activities = Counter([activities[0] for activities in cases.
    values() if activities])
end_activities = Counter([activities[-1] for activities in cases.
    values() if activities])
```

```
    print(f"\n ATIVIDADES DE INICIO:")
    for activity, count in start_activities.most_common(3):
        percentage = (count / len(cases)) * 100
        print(f"   {activity}: {count} casos ({percentage:.1f}%)")

    print(f"\n ATIVIDADES DE FIM:")
    for activity, count in end_activities.most_common(3):
        percentage = (count / len(cases)) * 100
        print(f"   {activity}: {count} casos ({percentage:.1f}%)")

    print("\n" + "=" * 60)

# ==================== ANALISE DE CONFORMIDADE COM DFG
    ====================
def check_conformance_dfg(dfg, expected_paths):
    """Verifica conformidade com caminhos esperados no DFG"""

    print("=" * 60)
    print("ANALISE DE CONFORMIDADE - DFG")
    print("=" * 60)

    deviations = []

    for (from_act, to_act), count in dfg.items():
        # Verificar se esta transicao e esperada
        is_expected = False
        for expected_path in expected_paths:
            for i in range(len(expected_path) - 1):
                if expected_path[i] == from_act and expected_path[i +
                    1] == to_act:
                    is_expected = True
                    break

        if not is_expected:
            deviations.append((from_act, to_act, count))

    if deviations:
        print(f"\n  DESVIOS ENCONTRADOS: {len(deviations)} transicoes
            nao esperadas")
```

```
        for from_act, to_act, count in deviations:
            print(f"   {from_act} -> {to_act}: {count} ocorrencias")
    else:
        print(f"\n TODAS AS TRANSICOES SAO CONFORMES!")

    # Calcular taxa de conformidade
    total_transitions = sum(dfg.values())
    unexpected_count = sum(count for _, _, count in deviations)
    conformance_rate = ((total_transitions - unexpected_count) /
        total_transitions) * 100

    print(f"\n TAXA DE CONFORMIDADE: {conformance_rate:.1f}%")
    print("=" * 60)

    return conformance_rate, deviations

# ================== FUNCAO PRINCIPAL ==================
def main():
    """Funcao principal para demonstracao didatica"""

    # 1. Carregar dados (usando o log do helpdesk fornecido
        anteriormente)
    log_content = """CaseID,Activity,Resource,Timestamp,Status
1001,Ticket Received,System,2024-01-01 09:00:00,Open
1001,Initial Triage,Agent_John,2024-01-01 09:15:00,In Progress
1001,Assign to Specialist,Agent_John,2024-01-01 09:30:00,In Progress
1001,Technical Analysis,Specialist_Maria,2024-01-01 10:00:00,In
    Progress
1001,Solution Provided,Specialist_Maria,2024-01-01 11:30:00,Resolved
1001,Ticket Closed,Agent_John,2024-01-01 12:00:00,Closed
1002,Ticket Received,System,2024-01-01 09:05:00,Open
1002,Initial Triage,Agent_Sarah,2024-01-01 09:20:00,In Progress
1002,Assign to Specialist,Agent_Sarah,2024-01-01 09:35:00,In Progress
1002,Technical Analysis,Specialist_Maria,2024-01-01 10:30:00,In
    Progress
1002,Waiting for User,Specialist_Maria,2024-01-01 11:00:00,Waiting
1002,Technical Analysis,Specialist_Maria,2024-01-01 14:00:00,In
    Progress
1002,Solution Provided,Specialist_Maria,2024-01-01 15:30:00,Resolved
```

```
1002,Ticket␣Closed,Agent_Sarah,2024-01-02␣09:00:00,Closed
1003,Ticket␣Received,System,2024-01-01␣09:10:00,Open
1003,Initial␣Triage,Agent_John,2024-01-01␣09:25:00,In␣Progress
1003,Escalate␣to␣Manager,Agent_John,2024-01-01␣09:45:00,In␣Progress
1003,Manager␣Review,Manager_David,2024-01-01␣11:00:00,In␣Progress
1003,Solution␣Provided,Manager_David,2024-01-01␣12:30:00,Resolved
1003,Ticket␣Closed,Agent_John,2024-01-01␣13:00:00,Closed
1004,Ticket␣Received,System,2024-01-02␣10:00:00,Open
1004,Initial␣Triage,Agent_Sarah,2024-01-02␣10:15:00,In␣Progress
1004,Assign␣to␣Specialist,Agent_Sarah,2024-01-02␣10:30:00,In␣Progress
1004,Technical␣Analysis,Specialist_Maria,2024-01-02␣11:00:00,In␣
    Progress
1004,Solution␣Provided,Specialist_Maria,2024-01-02␣12:00:00,Resolved
1004,Ticket␣Closed,Agent_Sarah,2024-01-02␣12:30:00,Closed
1005,Ticket␣Received,System,2024-01-02␣14:00:00,Open
1005,Initial␣Triage,Agent_John,2024-01-02␣14:15:00,In␣Progress
1005,Assign␣to␣Specialist,Agent_John,2024-01-02␣14:30:00,In␣Progress
1005,Technical␣Analysis,Specialist_Carlos,2024-01-02␣15:00:00,In␣
    Progress
1005,Escalate␣to␣Manager,Specialist_Carlos,2024-01-02␣16:00:00,In␣
    Progress
1005,Manager␣Review,Manager_David,2024-01-03␣09:30:00,In␣Progress
1005,Solution␣Provided,Manager_David,2024-01-03␣11:00:00,Resolved
1005,Ticket␣Closed,Agent_John,2024-01-03␣11:30:00,Closed"""

    df = parse_log_to_df(log_content)

    # 2. Extrair casos
    cases = extract_cases(df)

    # 3. Construir DFG basico
    dfg, total_flows = build_directly_follows_graph(df)

    # 4. Visualizar DFG simples
    print("␣GERANDO␣VISUALIZACAO␣DFG␣SIMPLES...")
    G_simple = visualize_dfg_simple(dfg, total_flows)

    # 5. Gerar resumo do processo
    generate_process_summary(dfg, total_flows, cases)
```

```
    # 6. Construir DFG com metricas de tempo
    print("\n GERANDO DFG COM METRICAS DE TEMPO...")
    dfg_counts, dfg_times = build_dfg_with_metrics(df)
    G_advanced, edge_metrics = visualize_dfg_advanced(dfg_counts,
        dfg_times)

    # 7. Analise de conformidade
    expected_paths = [
        ['Ticket Received', 'Initial Triage', 'Assign to Specialist',
         'Technical Analysis', 'Solution Provided', 'Ticket Closed'],
        ['Ticket Received', 'Initial Triage', 'Escalate to Manager',
         'Manager Review', 'Solution Provided', 'Ticket Closed']
    ]

    conformance_rate, deviations = check_conformance_dfg(dfg,
        expected_paths)

    # 8. Analise de bottlenecks (com base nos tempos)
    print("\n IDENTIFICANDO BOTTLENECKS...")
    bottlenecks = []
    for (from_act, to_act), metrics in edge_metrics.items():
        if metrics['avg_time'] > 60:  # Mais de 1 hora
            bottlenecks.append((from_act, to_act, metrics['avg_time'],
                metrics['count']))

    if bottlenecks:
        print(f" {len(bottlenecks)} BOTTLENECKS IDENTIFICADOS:")
        for from_act, to_act, avg_time, count in sorted(bottlenecks,
            key=lambda x: x[2], reverse=True):
            print(f"    {from_act} -> {to_act}: {avg_time:.1f} min (
                media) em {count} ocorrencias")
    else:
        print(" Nenhum bottleneck significativo identificado!")

    # 9. Exportar resultados
    export_results(dfg, edge_metrics, cases)

def export_results(dfg, edge_metrics, cases):
```

```
    """Exporta resultados para analise posterior"""

    # Criar DataFrame com fluxos
    dfg_list = []
    for (from_act, to_act), count in dfg.items():
        metrics = edge_metrics.get((from_act, to_act), {})
        dfg_list.append({
            'From': from_act,
            'To': to_act,
            'Frequency': count,
            'Avg_Time_Min': metrics.get('avg_time', 0),
            'Min_Time': metrics.get('min_time', 0),
            'Max_Time': metrics.get('max_time', 0)
        })

    dfg_df = pd.DataFrame(dfg_list)

    # Exportar para CSV
    dfg_df.to_csv('dfg_analysis.csv', index=False, encoding='utf-8')

    # Exportar casos
    cases_df = pd.DataFrame([
        {'CaseID': case_id, 'Path': ' -> '.join(activities)}
        for case_id, activities in cases.items()
    ])
    cases_df.to_csv('cases_paths.csv', index=False, encoding='utf-8')

    print(f"\n Resultados exportados:")
    print(f"   dfg_analysis.csv: {len(dfg_df)} transicoes analisadas")
    print(f"   cases_paths.csv: {len(cases_df)} casos processados")

# ==================== EXECUCAO ====================
if __name__ == "__main__":
    print(" PROCESS MINING - DEMONSTRACAO DIDATICA COM DFG")
    print("=" * 60)
    main()
%
```

APÊNDICE C – Representação Helpdesk como DFG em Python com pm4py

```
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import pm4py
from pm4py.objects.log.util import dataframe_utils
from pm4py.objects.conversion.log import converter as log_converter
# importar funcao customizada de DFG (caso queira comparar)
try:
    from dfg_helpdesk import build_directly_follows_graph
except Exception:
    build_directly_follows_graph = None
from io import StringIO

# ==================== FUNCOES PM4Py ====================
def create_pm4py_log(df):
    """Cria log no formato PM4Py a partir de DataFrame"""
    # Renomear colunas para padrao PM4Py
    df = df.rename(columns={
        'CaseID': 'case:concept:name',
        'Activity': 'concept:name',
        'Timestamp': 'time:timestamp',
        'Resource': 'org:resource',
        'Status': 'lifecycle:transition'
    })

    # Converter para log PM4Py
    log = log_converter.apply(df)
    return log, df

def discover_dfg_pm4py(log):
    """Descobre DFG usando PM4Py"""
    from pm4py.algo.discovery.dfg import algorithm as dfg_discovery
```

```
    # Descobrir DFG (frequencia)
    # O retorno de dfg_discovery.apply pode variar entre versoes do
        PM4Py
    # (ex.: retornar 3 ou 4 elementos). Tratamos de forma robusta.
    res = dfg_discovery.apply(log)
    if isinstance(res, tuple) or isinstance(res, list):
        if len(res) >= 3:
            dfg_freq = res[0]
            start_activities = res[1]
            end_activities = res[2]
        elif len(res) == 2:
            dfg_freq = res[0]
            start_activities = res[1]
            end_activities = {}
        else:
            dfg_freq = res[0]
            start_activities = {}
            end_activities = {}
    else:
        # se retornou apenas um dicionario
        dfg_freq = res
        start_activities = {}
        end_activities = {}

    # Descobrir DFG (desempenho)
    perf_res = dfg_discovery.apply(log, variant=dfg_discovery.Variants.
        PERFORMANCE)
    if isinstance(perf_res, tuple) or isinstance(perf_res, list):
        dfg_perf = perf_res[0]
    else:
        dfg_perf = perf_res

    return dfg_freq, dfg_perf, start_activities, end_activities

def visualize_dfg_pm4py(dfg_freq, dfg_perf, log):
    """Visualiza DFG usando visualizacao nativa do PM4Py"""
    from pm4py.visualization.dfg import visualizer as dfg_visualizer

    print("GERANDO VISUALIZACOES COM PM4Py...")
```

```
    # Visualizacao de frequencia
    print("\n1. DFG por Frequencia:")
    gviz_freq = dfg_visualizer.apply(dfg_freq,
                                     log=log,
                                     variant=dfg_visualizer.Variants.
                                         FREQUENCY)
    dfg_visualizer.view(gviz_freq)

    # Visualizacao de desempenho
    print("\n2. DFG por Desempenho (tempo):")
    gviz_perf = dfg_visualizer.apply(dfg_perf,
                                     log=log,
                                     variant=dfg_visualizer.Variants.
                                         PERFORMANCE)
    dfg_visualizer.view(gviz_perf)

    return gviz_freq, gviz_perf

def discover_process_model_pm4py(log):
    """Descobre modelo de processo usando PM4Py"""
    from pm4py.algo.discovery.alpha import algorithm as alpha_miner
    from pm4py.algo.discovery.heuristics import algorithm as
        heuristics_miner
    from pm4py.algo.discovery.inductive import algorithm as
        inductive_miner
    from pm4py.visualization.petri_net import visualizer as
        pn_visualizer

    print("\nDESCOBRINDO MODELOS DE PROCESSO...")

    # 1. Alpha Miner (basico)
    print("1. Alpha Miner (classico):")
    net_alpha, initial_marking_alpha, final_marking_alpha = alpha_miner
        .apply(log)
    gviz_alpha = pn_visualizer.apply(net_alpha, initial_marking_alpha,
        final_marking_alpha)
    pn_visualizer.view(gviz_alpha)
```

```
    # 2. Heuristics Miner (robusto a ruidos)
    print("\n2. Heuristics Miner (robusto):")
    heu_net = heuristics_miner.apply_heu(log)
    from pm4py.visualization.heuristics_net import visualizer as
        hn_visualizer
    gviz_heu = hn_visualizer.apply(heu_net)
    hn_visualizer.view(gviz_heu)

    # 3. Inductive Miner (moderno)
    print("\n3. Inductive Miner (moderno):")
    net_inductive, initial_marking_inductive, final_marking_inductive =
        inductive_miner.apply(log)
    gviz_inductive = pn_visualizer.apply(net_inductive,
        initial_marking_inductive, final_marking_inductive)
    pn_visualizer.view(gviz_inductive)

    return {
        'alpha': (net_alpha, initial_marking_alpha, final_marking_alpha
            ),
        'heuristics': heu_net,
        'inductive': (net_inductive, initial_marking_inductive,
            final_marking_inductive)
    }

def analyze_process_stats_pm4py(log, df):
    """Analise de estatisticas do processo usando PM4Py"""
    from pm4py.statistics.traces.generic.log import case_statistics
    from pm4py.statistics.start_activities.log import get as
        start_activities_get
    from pm4py.statistics.end_activities.log import get as
        end_activities_get
    from pm4py.statistics.attributes.log import get as attributes_get

    print("\nESTATISTICAS DO PROCESSO (PM4Py):")
    print("=" * 60)

    # 1. Estatisticas basicas
    print(f"1. ESTATISTICAS BASICAS:")
    print(f"   - Total de casos: {len(log)}")
```

```
print(f"   - Total de eventos: {sum(len(trace) for trace in log)}")

# 2. Atividades de inicio
start_activities = start_activities_get.get_start_activities(log)
print(f"\n2. ATIVIDADES DE INICIO:")
for activity, count in start_activities.items():
    percentage = (count / len(log)) * 100
    print(f"   - {activity}: {count} casos ({percentage:.1f}%)")

# 3. Atividades de fim
end_activities = end_activities_get.get_end_activities(log)
print(f"\n3. ATIVIDADES DE FIM:")
for activity, count in end_activities.items():
    percentage = (count / len(log)) * 100
    print(f"   - {activity}: {count} casos ({percentage:.1f}%)")

# 4. Atividades mais frequentes
activities = attributes_get.get_attribute_values(log, "concept:name
    ")
print(f"\n4. ATIVIDADES MAIS FREQUENTES:")
for activity, count in sorted(activities.items(), key=lambda x: x
    [1], reverse=True)[:5]:
    total_events = sum(activities.values())
    percentage = (count / total_events) * 100
    print(f"   - {activity}: {count} ocorrencias ({percentage:.1f
        }%)")

# 5. Duracao dos casos
case_durations = case_statistics.get_all_case_durations(log,
                parameters={case_statistics.Parameters.
                    TIMESTAMP_KEY: "time:timestamp"})
if case_durations:
    avg_duration = sum(case_durations) / len(case_durations)
    min_duration = min(case_durations)
    max_duration = max(case_durations)
    print(f"\n5. DURACAO DOS CASOS:")
    print(f"   - Media: {avg_duration/3600:.2f} horas")
    print(f"   - Minima: {min_duration/3600:.2f} horas")
    print(f"   - Maxima: {max_duration/3600:.2f} horas")
```

```
    print("=" * 60)

def discover_variants_pm4py(log):
    """Descobre variantes de processo usando PM4Py"""
    from pm4py.algo.filtering.log.variants import variants_filter

    print("\nVARIANTES DE PROCESSO (PM4Py):")
    print("=" * 60)

    # Obter variantes
    variants = variants_filter.get_variants(log)

    def variant_to_str(variant):
        # variant can be a string, a tuple/list of event dicts, or a
            tuple/list of names
        if isinstance(variant, str):
            return variant
        try:
            parts = []
            for ev in variant:
                if isinstance(ev, dict):
                    parts.append(ev.get("concept:name", str(ev)))
                else:
                    parts.append(str(ev))
            return " -> ".join(parts)
        except Exception:
            return str(variant)

    print(f"Total de variantes unicas: {len(variants)}")
    print(f"Total de casos: {len(log)}\n")

    # Mostrar variantes ordenadas por frequencia
    for i, (variant, cases) in enumerate(sorted(variants.items(), key=
        lambda x: len(x[1]), reverse=True)):
        count = len(cases)
        percentage = (count / len(log)) * 100

        # Simplificar visualizacao da variante
```

```
            variant_str = variant_to_str(variant)

            print(f"Variante␣{i+1}:␣{percentage:.1f}%␣dos␣casos␣({count}␣
                casos)")
            print(f"␣␣Sequencia:␣{variant_str}")
            print(f"␣␣IDs␣dos␣casos:␣{list(cases)[:5]}{'...'␣if␣len(cases)␣
                >␣5␣else␣''}")
            print()

        print("=" * 60)
        return variants

def analyze_conformance_pm4py(log):
    """Analise␣de␣conformidade␣usando␣PM4Py"""
    from pm4py.algo.discovery.alpha import algorithm as alpha_miner
    from pm4py.algo.conformance.tokenreplay import algorithm as
        token_replay

    print("\nANALISE␣DE␣CONFORMIDADE␣(PM4Py):")
    print("=" * 60)

    # 1. Descobrir modelo
    net, initial_marking, final_marking = alpha_miner.apply(log)

    # 2. Executar token replay
    replayed_traces = token_replay.apply(log, net, initial_marking,
        final_marking)

    # 3. Calcular metricas
    total_traces = len(replayed_traces)
    fitting_traces = sum(1 for trace in replayed_traces if trace['
        trace_is_fit'])
    missing_tokens = sum(trace['missing_tokens'] for trace in
        replayed_traces)
    remaining_tokens = sum(trace['remaining_tokens'] for trace in
        replayed_traces)
    produced_tokens = sum(trace['produced_tokens'] for trace in
        replayed_traces)
    consumed_tokens = sum(trace['consumed_tokens'] for trace in
```

```
    replayed_traces)

# Calcular fitness
fitness = 0.5 * (1 - missing_tokens/consumed_tokens if
    consumed_tokens > 0 else 1) + \
            0.5 * (1 - remaining_tokens/produced_tokens if
                produced_tokens > 0 else 1)

print(f"1. METRICAS DE CONFORMIDADE:")
print(f"   - Fitness: {fitness:.3f} (0-1, onde 1 eh perfeito)")
print(f"   - Traces conformes: {fitting_traces}/{total_traces} ({(
    fitting_traces/total_traces)*100:.1f}%)")
print(f"   - Tokens faltando: {missing_tokens}")
print(f"   - Tokens restantes: {remaining_tokens}")

# 4. Detalhar problemas por caso
print(f"\n2. CASOS COM PROBLEMAS:")
problematic_cases = []
for i, trace in enumerate(replayed_traces):
    if not trace['trace_is_fit'] or trace['missing_tokens'] > 0 or
       trace['remaining_tokens'] > 0:
        case_id = log[i].attributes['concept:name'] if i < len(log)
            else f"Case_{i}"
        problematic_cases.append({
            'case': case_id,
            'is_fit': trace['trace_is_fit'],
            'missing': trace['missing_tokens'],
            'remaining': trace['remaining_tokens']
        })

if problematic_cases:
    for case in problematic_cases[:3]:  # Mostrar apenas 3
        status = "OK" if case['is_fit'] else "NOT_OK"
        print(f"   {status} {case['case']}: {case['missing']}
            tokens faltando, {case['remaining']} tokens restantes")
    if len(problematic_cases) > 3:
        print(f"   ... e mais {len(problematic_cases) - 3} casos")
else:
    print("   Todos os casos sao conformes!")
```

```
    print("=" * 60)

    return {
        'fitness': fitness,
        'fitting_traces': fitting_traces,
        'total_traces': total_traces,
        'problematic_cases': problematic_cases
    }

def analyze_bottlenecks_pm4py(log):
    """Identifica␣bottlenecks␣usando␣PM4Py"""
    import importlib
    spec = importlib.util.find_spec("pm4py.statistics.sojourn_time.log"
        )
    if spec is not None:
        try:
            from pm4py.statistics.sojourn_time.log import get as
                sojourn_time
        except Exception:
            sojourn_time = None
    else:
        sojourn_time = None
    # importar workflow_graph se disponivel
    try:
        import importlib
        spec_wf = importlib.util.find_spec("pm4py.algo.analysis.
            workflow_graph")
        if spec_wf is not None:
            from pm4py.algo.analysis.workflow_graph import algorithm as
                wf_graph
        else:
            wf_graph = None
    except Exception:
        wf_graph = None

    print("\nIDENTIFICACAO␣DE␣BOTTLENECKS␣(PM4Py):")
    print("=" * 60)
```

```
# 1. Tempo de espera por atividade
if sojourn_time is not None:
    try:
        sojourn_times = sojourn_time.apply(log)
    except Exception:
        sojourn_times = None
else:
    sojourn_times = None

# Se nao houver funcao disponivel, calcular tempos manualmente a
    partir do log
if sojourn_times is None:
    from collections import defaultdict
    sojourn_times = defaultdict(list)
    try:
        for trace in log:
            # trace is a list of events
            for i in range(len(trace) - 1):
                e_curr = trace[i]
                e_next = trace[i + 1]
                t_curr = e_curr.get("time:timestamp") if isinstance
                    (e_curr, dict) else getattr(e_curr, "time:
                    timestamp", None)
                t_next = e_next.get("time:timestamp") if isinstance
                    (e_next, dict) else getattr(e_next, "time:
                    timestamp", None)
                name_curr = e_curr.get("concept:name") if
                    isinstance(e_curr, dict) else getattr(e_curr, "
                    concept:name", None)
                if t_curr is None or t_next is None or name_curr is
                     None:
                    continue
                try:
                    delta_min = (t_next - t_curr).total_seconds() /
                         60.0
                except Exception:
                    continue
                sojourn_times[name_curr].append(delta_min)
    except Exception:
```

```
            sojourn_times = {}

    print("1. TEMPO DE ESPERA POR ATIVIDADE:")
    bottlenecks = []
    for activity, times in sojourn_times.items():
        if times:
            avg_time = sum(times) / len(times)
            if avg_time > 60:  # Mais de 1 hora
                bottlenecks.append((activity, avg_time, len(times)))

    if bottlenecks:
        for activity, avg_time, count in sorted(bottlenecks, key=lambda
            x: x[1], reverse=True):
            print(f"   - {activity}: {avg_time:.1f} min (media) em {
                count} ocorrencias")
    else:
        print("   Nenhum bottleneck significativo encontrado!")

    # 2. Analise de fluxo de trabalho
    print(f"\n2. ANALISE DE FLUXO DE TRABALHO:")
    if wf_graph is not None:
        try:
            workflow_graph = wf_graph.apply(log)
            # Aqui poderiamos extrair mais metricas do grafo de
                workflow
            print(f"   - Grafo de workflow gerado com {len(
                workflow_graph.nodes())} nos")
        except Exception:
            print("   - Analise de workflow falhou ao aplicar o
                algoritmo")
    else:
        print("   - Analise de workflow nao disponivel (pm4py.algo.
            analysis.workflow_graph ausente)")

    print("=" * 60)

    return bottlenecks

def export_results_pm4py(log, dfg_freq, dfg_perf, variants,
```

```
conformance_results, bottlenecks):
 """Exporta␣resultados␣do␣PM4Py"""
 import json

 print("\nEXPORTANDO␣RESULTADOS␣PM4Py...")

 # 1. Exportar DFG para CSV
 dfg_data = []
 for (from_act, to_act), freq in dfg_freq.items():
     perf = dfg_perf.get((from_act, to_act), 0)
     dfg_data.append({
         'From': from_act,
         'To': to_act,
         'Frequency': freq,
         'Avg_Time_Seconds': perf
     })

 dfg_df = pd.DataFrame(dfg_data)
 dfg_df.to_csv('pm4py_dfg_analysis.csv', index=False, encoding='utf
    -8')

 # 2. Exportar variantes
 # Normalizar variantes para string (lida com varios formatos
    retornados pelo PM4Py)
 def variant_to_str_local(variant):
     if isinstance(variant, str):
         return variant
     try:
         parts = []
         for ev in variant:
             if isinstance(ev, dict):
                 parts.append(ev.get("concept:name", str(ev)))
             else:
                 parts.append(str(ev))
                 return "␣->␣".join(parts)
     except Exception:
         return str(variant)

 variants_data = []
```

```
for variant, cases in variants.items():
    variant_str = variant_to_str_local(variant)
    variants_data.append({
        'Variant': variant_str,
        'Case_Count': len(cases),
        'Percentage': (len(cases) / len(log)) * 100,
        'Cases': list(cases)
    })

variants_df = pd.DataFrame(variants_data)
variants_df.to_csv('pm4py_variants.csv', index=False, encoding='utf
    -8')

# 3. Exportar metricas de conformidade
conformance_data = {
    'fitness': conformance_results['fitness'],
    'fitting_traces': conformance_results['fitting_traces'],
    'total_traces': conformance_results['total_traces'],
    'problematic_cases_count': len(conformance_results['
        problematic_cases'])
}

with open('pm4py_conformance.json', 'w') as f:
    json.dump(conformance_data, f, indent=2)

# 4. Exportar bottlenecks
bottlenecks_data = [{'Activity': a, 'Avg_Time_Min': t, 'Occurrences
    ': c}
                    for a, t, c in bottlenecks]
bottlenecks_df = pd.DataFrame(bottlenecks_data)
bottlenecks_df.to_csv('pm4py_bottlenecks.csv', index=False,
    encoding='utf-8')

print(f"   - pm4py_dfg_analysis.csv: {len(dfg_df)} transicoes")
print(f"   - pm4py_variants.csv: {len(variants_df)} variantes")
print(f"   - pm4py_conformance.json: metricas de conformidade")
print(f"   - pm4py_bottlenecks.csv: {len(bottlenecks_df)}
    bottlenecks")
```

```
# ==================== FUNCAO PRINCIPAL PM4Py ====================
def main_pm4py():
    """Funcao principal usando PM4Py"""

    print("PROCESS MINING COM PM4Py")
    print("=" * 60)

    # 1. Carregar dados
    log_content = """CaseID,Activity,Resource,Timestamp,Status
1001,Ticket Received,System,2024-01-01 09:00:00,Open
1001,Initial Triage,Agent_John,2024-01-01 09:15:00,In Progress
1001,Assign to Specialist,Agent_John,2024-01-01 09:30:00,In Progress
1001,Technical Analysis,Specialist_Maria,2024-01-01 10:00:00,In
    Progress
1001,Solution Provided,Specialist_Maria,2024-01-01 11:30:00,Resolved
1001,Ticket Closed,Agent_John,2024-01-01 12:00:00,Closed
1002,Ticket Received,System,2024-01-01 09:05:00,Open
1002,Initial Triage,Agent_Sarah,2024-01-01 09:20:00,In Progress
1002,Assign to Specialist,Agent_Sarah,2024-01-01 09:35:00,In Progress
1002,Technical Analysis,Specialist_Maria,2024-01-01 10:30:00,In
    Progress
1002,Waiting for User,Specialist_Maria,2024-01-01 11:00:00,Waiting
1002,Technical Analysis,Specialist_Maria,2024-01-01 14:00:00,In
    Progress
1002,Solution Provided,Specialist_Maria,2024-01-01 15:30:00,Resolved
1002,Ticket Closed,Agent_Sarah,2024-01-02 09:00:00,Closed
1003,Ticket Received,System,2024-01-01 09:10:00,Open
1003,Initial Triage,Agent_John,2024-01-01 09:25:00,In Progress
1003,Escalate to Manager,Agent_John,2024-01-01 09:45:00,In Progress
1003,Manager Review,Manager_David,2024-01-01 11:00:00,In Progress
1003,Solution Provided,Manager_David,2024-01-01 12:30:00,Resolved
1003,Ticket Closed,Agent_John,2024-01-01 13:00:00,Closed
1004,Ticket Received,System,2024-01-02 10:00:00,Open
1004,Initial Triage,Agent_Sarah,2024-01-02 10:15:00,In Progress
1004,Assign to Specialist,Agent_Sarah,2024-01-02 10:30:00,In Progress
1004,Technical Analysis,Specialist_Maria,2024-01-02 11:00:00,In
    Progress
1004,Solution Provided,Specialist_Maria,2024-01-02 12:00:00,Resolved
1004,Ticket Closed,Agent_Sarah,2024-01-02 12:30:00,Closed
```

```
1005,Ticket␣Received,System,2024-01-02␣14:00:00,Open
1005,Initial␣Triage,Agent_John,2024-01-02␣14:15:00,In␣Progress
1005,Assign␣to␣Specialist,Agent_John,2024-01-02␣14:30:00,In␣Progress
1005,Technical␣Analysis,Specialist_Carlos,2024-01-02␣15:00:00,In␣
    Progress
1005,Escalate␣to␣Manager,Specialist_Carlos,2024-01-02␣16:00:00,In␣
    Progress
1005,Manager␣Review,Manager_David,2024-01-03␣09:30:00,In␣Progress
1005,Solution␣Provided,Manager_David,2024-01-03␣11:00:00,Resolved
1005,Ticket␣Closed,Agent_John,2024-01-03␣11:30:00,Closed"""

    # Converter para DataFrame
    from io import StringIO
    df = pd.read_csv(StringIO(log_content))
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])

    # 2. Criar log PM4Py
    print("1.␣Criando␣log␣PM4Py...")
    log, df_formatted = create_pm4py_log(df)

    # 3. Descobrir DFG
    print("2.␣Descobrindo␣DFG...")
    dfg_freq, dfg_perf, start_activities, end_activities =
        discover_dfg_pm4py(log)

    # 4. Visualizar DFG
    print("3.␣Visualizando␣DFG...")
    gviz_freq, gviz_perf = visualize_dfg_pm4py(dfg_freq, dfg_perf, log)

    # 5. Estatisticas do processo
    analyze_process_stats_pm4py(log, df_formatted)

    # 6. Descobrir variantes
    variants = discover_variants_pm4py(log)

    # 7. Analise de conformidade
    conformance_results = analyze_conformance_pm4py(log)

    # 8. Identificar bottlenecks
```

```
    bottlenecks = analyze_bottlenecks_pm4py(log)

    # 9. Descobrir modelos de processo (opcional - descomente se quiser
        ver)
    # models = discover_process_model_pm4py(log)

    # 10. Exportar resultados
    export_results_pm4py(log, dfg_freq, dfg_perf, variants,
        conformance_results, bottlenecks)

    print("\n" + "=" * 60)
    print("ANALISE COM PM4Py CONCLUIDA!")
    print("=" * 60)

# ==================== FUNCAO DE COMPARACAO ====================
def compare_pm4py_vs_custom():
    """Compara resultados PM4Py vs implementacao customizada"""

    print("COMPARACAO: PM4Py vs IMPLEMENTACAO CUSTOMIZADA")
    print("=" * 60)

    # Dados
    log_content = """CaseID,Activity,Timestamp
1001,Ticket Received,2024-01-01 09:00:00
1001,Initial Triage,2024-01-01 09:15:00
1001,Assign to Specialist,2024-01-01 09:30:00
1001,Technical Analysis,2024-01-01 10:00:00
1001,Solution Provided,2024-01-01 11:30:00
1001,Ticket Closed,2024-01-01 12:00:00
1002,Ticket Received,2024-01-01 09:05:00
1002,Initial Triage,2024-01-01 09:20:00
1002,Assign to Specialist,2024-01-01 09:35:00
1002,Technical Analysis,2024-01-01 10:30:00
1002,Solution Provided,2024-01-01 11:30:00
1002,Ticket Closed,2024-01-01 12:00:00"""

    df = pd.read_csv(StringIO(log_content))
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

```
    # PM4Py
    print("\nPM4Py:")
    log_pm4py, _ = create_pm4py_log(df)
    dfg_freq_pm4py, _, _, _ = discover_dfg_pm4py(log_pm4py)
    print(f"   - Transicoes encontradas: {len(dfg_freq_pm4py)}")

    # Customizado
    print("\nCustomizado:")
    dfg_custom, total_flows = build_directly_follows_graph(df)
    print(f"   - Transicoes encontradas: {len(dfg_custom)}")
    print(f"   - Total de fluxos: {total_flows}")

    # Comparar
    print("\nCOMPARACAO:")

    # Converter para conjuntos para comparacao
    pm4py_set = set(dfg_freq_pm4py.keys())
    custom_set = set(dfg_custom.keys())

    print(f"   - Transicoes iguais: {len(pm4py_set.intersection(
        custom_set))}")
    print(f"   - Apenas no PM4Py: {len(pm4py_set - custom_set)}")
    print(f"   - Apenas no Custom: {len(custom_set - pm4py_set)}")

    if pm4py_set == custom_set:
        print("   Resultados identicos!")
    else:
        print("   Diferencas encontradas:")
        for trans in pm4py_set - custom_set:
            print(f"      - PM4Py tem: {trans}")
        for trans in custom_set - pm4py_set:
            print(f"      - Custom tem: {trans}")

    print("=" * 60)

# ================== EXECUCAO ==================
if __name__ == "__main__":
    import sys
```

```
print("Selecione a opcao:")
print("1. Analise completa com PM4Py")
print("2. Comparacao PM4Py vs Custom")
print("3. Sair")

choice = input("\nDigite sua escolha (1-3): ")

if choice == "1":
    main_pm4py()
elif choice == "2":
    compare_pm4py_vs_custom()
elif choice == "3":
    sys.exit()
else:
    print("Opcao invalida!")
```

APÊNDICE D – Helpdesk em pm4py com Redes de Petri

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from io import StringIO
import pm4py
from pm4py.objects.petri_net.utils import petri_utils
from pm4py.algo.evaluation.replay_fitness import algorithm as
    replay_fitness
from pm4py.algo.evaluation.precision import algorithm as
    precision_evaluator
from pm4py.algo.evaluation.generalization import algorithm as
    generalization_evaluator
from pm4py.visualization.petri_net import visualizer as pn_visualizer

# DADOS DO HELPDESK
HELPDESK_LOG = """CaseID,Activity,Resource,Timestamp,Status
1001,Ticket␣Received,System,2024-01-01␣09:00:00,Open
1001,Initial␣Triage,Agent_John,2024-01-01␣09:15:00,In␣Progress
1001,Assign␣to␣Specialist,Agent_John,2024-01-01␣09:30:00,In␣Progress
1001,Technical␣Analysis,Specialist_Maria,2024-01-01␣10:00:00,In␣
    Progress
1001,Solution␣Provided,Specialist_Maria,2024-01-01␣11:30:00,Resolved
1001,Ticket␣Closed,Agent_John,2024-01-01␣12:00:00,Closed
1002,Ticket␣Received,System,2024-01-01␣09:05:00,Open
1002,Initial␣Triage,Agent_Sarah,2024-01-01␣09:20:00,In␣Progress
1002,Assign␣to␣Specialist,Agent_Sarah,2024-01-01␣09:35:00,In␣Progress
1002,Technical␣Analysis,Specialist_Maria,2024-01-01␣10:30:00,In␣
    Progress
1002,Waiting␣for␣User,Specialist_Maria,2024-01-01␣11:00:00,Waiting
1002,Technical␣Analysis,Specialist_Maria,2024-01-01␣14:00:00,In␣
    Progress
1002,Solution␣Provided,Specialist_Maria,2024-01-01␣15:30:00,Resolved
1002,Ticket␣Closed,Agent_Sarah,2024-01-02␣09:00:00,Closed
```

```
1003,Ticket␣Received,System,2024-01-01␣09:10:00,Open
1003,Initial␣Triage,Agent_John,2024-01-01␣09:25:00,In␣Progress
1003,Escalate␣to␣Manager,Agent_John,2024-01-01␣09:45:00,In␣Progress
1003,Manager␣Review,Manager_David,2024-01-01␣11:00:00,In␣Progress
1003,Solution␣Provided,Manager_David,2024-01-01␣12:30:00,Resolved
1003,Ticket␣Closed,Agent_John,2024-01-01␣13:00:00,Closed
1004,Ticket␣Received,System,2024-01-02␣10:00:00,Open
1004,Initial␣Triage,Agent_Sarah,2024-01-02␣10:15:00,In␣Progress
1004,Assign␣to␣Specialist,Agent_Sarah,2024-01-02␣10:30:00,In␣Progress
1004,Technical␣Analysis,Specialist_Maria,2024-01-02␣11:00:00,In␣
    Progress
1004,Solution␣Provided,Specialist_Maria,2024-01-02␣12:00:00,Resolved
1004,Ticket␣Closed,Agent_Sarah,2024-01-02␣12:30:00,Closed
1005,Ticket␣Received,System,2024-01-02␣14:00:00,Open
1005,Initial␣Triage,Agent_John,2024-01-02␣14:15:00,In␣Progress
1005,Assign␣to␣Specialist,Agent_John,2024-01-02␣14:30:00,In␣Progress
1005,Technical␣Analysis,Specialist_Carlos,2024-01-02␣15:00:00,In␣
    Progress
1005,Escalate␣to␣Manager,Specialist_Carlos,2024-01-02␣16:00:00,In␣
    Progress
1005,Manager␣Review,Manager_David,2024-01-03␣09:30:00,In␣Progress
1005,Solution␣Provided,Manager_David,2024-01-03␣11:00:00,Resolved
1005,Ticket␣Closed,Agent_John,2024-01-03␣11:30:00,Closed"""

def carregar_dados_helpdesk():
    # Carrega e prepara dados do helpdesk
    df = pd.read_csv(StringIO(HELPDESK_LOG))
    df['Timestamp'] = pd.to_datetime(df['Timestamp'])

    df = df.rename(columns={
        'CaseID': 'case:concept:name',
        'Activity': 'concept:name',
        'Timestamp': 'time:timestamp',
        'Resource': 'org:resource',
        'Status': 'lifecycle:transition'
    })

    log = pm4py.convert_to_event_log(df)
    return log, df
```

```
def descobrir_modelo_petri_alpha(log):
    # Descobre modelo usando Alpha Miner (Rede de Petri classica)
    from pm4py.algo.discovery.alpha import algorithm as alpha_miner

    print("="*70)
    print("DESCOBERTA DO MODELO - ALPHA MINER")
    print("="*70)

    net, initial_marking, final_marking = alpha_miner.apply(log)

    print("ESTATISTICAS DA REDE DE PETRI:")
    print("  Transicoes (atividades): ", len(net.transitions))
    print("  Lugares (estados): ", len(net.places))
    print("  Arcos (relacoes): ", len(net.arcs))
    print("  Marcacao inicial: ", initial_marking)
    print("  Marcacao final: ", final_marking)

    gviz = pn_visualizer.apply(net, initial_marking, final_marking,
                               parameters={pn_visualizer.Variants.
                                   WO_DECORATION.value.Parameters.FORMAT
                                   : "png"})
    pn_visualizer.save(gviz, "helpdesk_petri_alpha.png")

    return net, initial_marking, final_marking

def descobrir_modelo_petri_inductive(log):
    # Descobre modelo usando Inductive Miner (moderno, garante
        soundness)
    from pm4py.algo.discovery.inductive import algorithm as
        inductive_miner

    print("\n" + "="*70)
    print("DESCOBERTA DO MODELO - INDUCTIVE MINER")
    print("="*70)

    net, initial_marking, final_marking = inductive_miner.apply(log)

    print("ESTATISTICAS DA REDE (INDUCTIVE):")
```

```
    print("  Transicoes: ", len(net.transitions))
    print("  Lugares: ", len(net.places))
    print("  Soundness garantida pelo algoritmo")

    gviz = pn_visualizer.apply(net, initial_marking, final_marking)
    pn_visualizer.save(gviz, "helpdesk_petri_inductive.png")

    return net, initial_marking, final_marking

def descobrir_modelo_petri_heuristics(log):
    # Descobre modelo usando Heuristics Miner (robusto a ruidos)
    from pm4py.algo.discovery.heuristics import algorithm as
        heuristics_miner
    from pm4py.visualization.heuristics_net import visualizer as
        hn_visualizer

    print("\n" + "="*70)
    print("DESCOBERTA DO MODELO - HEURISTICS MINER")
    print("="*70)

    heu_net = heuristics_miner.apply_heu(log)

    print("ESTATISTICAS DA HEURISTICS NET:")
    print("  Nos: ", len(heu_net.nodes))
    print("  Arestas: ", len(heu_net.edges))
    print("  Robustez a ruidos: Alta")

    gviz = hn_visualizer.apply(heu_net)
    hn_visualizer.save(gviz, "helpdesk_heuristics_net.png")

    net, initial_marking, final_marking = heuristics_miner.apply(log)

    gviz_petri = pn_visualizer.apply(net, initial_marking,
        final_marking)
    pn_visualizer.save(gviz_petri, "helpdesk_petri_heuristics.png")

    return net, initial_marking, final_marking, heu_net

def analisar_conformidade_petri(log, net, initial_marking,
```

```
final_marking):
 # Analise avancada de conformidade usando Redes de Petri
 from pm4py.algo.conformance.tokenreplay import algorithm as
     token_replay

 print("\n" + "="*70)
 print("ANALISE DE CONFORMIDADE - REDES DE PETRI")
 print("="*70)

 replayed_traces = token_replay.apply(log, net, initial_marking,
     final_marking)

 total_traces = len(replayed_traces)
 fitting_traces = sum(1 for trace in replayed_traces if trace['
     trace_is_fit'])
 missing_tokens = sum(trace['missing_tokens'] for trace in
     replayed_traces)
 remaining_tokens = sum(trace['remaining_tokens'] for trace in
     replayed_traces)
 consumed_tokens = sum(trace['consumed_tokens'] for trace in
     replayed_traces)
 produced_tokens = sum(trace['produced_tokens'] for trace in
     replayed_traces)

 fitness = 0.5 * (1 - missing_tokens/consumed_tokens if
     consumed_tokens > 0 else 1) + \
           0.5 * (1 - remaining_tokens/produced_tokens if
               produced_tokens > 0 else 1)

 print("METRICAS DE CONFORMIDADE:")
 print("  1. Fitness: ", round(fitness, 3), " (0-1, onde 1 e
     perfeito)")
 print("  2. Casos conformes: ", fitting_traces, "/", total_traces,
       " (", round((fitting_traces/total_traces)*100, 1), "%)")
 print("  3. Tokens faltando: ", missing_tokens)
 print("  4. Tokens restantes: ", remaining_tokens)

 try:
     precision = precision_evaluator.apply(log, net, initial_marking
```

```
        , final_marking)
    print("  5. Precision: ", round(precision, 3), " (alta = modelo
        especifico)")
except:
    print("  5. Precision: Nao calculavel")

try:
    generalization = generalization_evaluator.apply(log, net,
        initial_marking, final_marking)
    print("  6. Generalization: ", round(generalization, 3), " (
        alta = boa generalizacao)")
except:
    print("  6. Generalization: Nao calculavel")

print("ANALISE DETALHADA DOS DESVIOS:")
problematic_cases = []

for i, trace in enumerate(replayed_traces):
    if not trace['trace_is_fit'] or trace['missing_tokens'] > 0:
        case_id = log[i].attributes['concept:name'] if i < len(log)
            else f"Case_{i}"
        problematic_cases.append({
            'case': case_id,
            'is_fit': trace['trace_is_fit'],
            'missing': trace['missing_tokens'],
            'remaining': trace['remaining_tokens']
        })

if problematic_cases:
    print("  Total de casos com problemas: ", len(problematic_cases
        ))
    for case in problematic_cases[:3]:
        status = "OK" if case['is_fit'] else "NOT OK"
        print("  ", status, case['case'], ": ", case['missing'], "
            tokens faltando")
else:
    print("  Todos os casos sao perfeitamente conformes!")

return {
```

```
        'fitness': fitness,
        'fitting_traces': fitting_traces,
        'total_traces': total_traces,
        'problematic_cases': problematic_cases,
        'replayed_traces': replayed_traces
    }

def analisar_estrutura_petri(net, initial_marking, final_marking):
    # Analisa propriedades estruturais da Rede de Petri

    print("\n" + "="*70)
    print("ANALISE ESTRUTURAL - REDE DE PETRI")
    print("="*70)

    print("CONSTRUCOES IDENTIFICADAS:")

    sequencias = 0
    escolhas = 0
    loops = 0

    for place in net.places:
        input_arcs = len([arc for arc in net.arcs if arc.target ==
            place])
        output_arcs = len([arc for arc in net.arcs if arc.source ==
            place])

        if input_arcs == 1 and output_arcs == 1:
            sequencias += 1
        elif output_arcs > 1:
            escolhas += 1

    print("  Sequencias: ", sequencias)
    print("  Pontos de escolha (XOR/OR): ", escolhas)
    print("  Loops identificados: ", loops)

    print("TRANSICOES CRITICAS:")

    for transition in net.transitions:
        if transition.label is None:
```

```
            print("  ", transition.name, ": Transicao silenciosa (tau)"
                )
        else:
            print("  ", transition.label, ": Transicao visivel")

    print("VERIFICACAO DE SOUNDNESS:")

    if len(initial_marking) > 0:
        print("  Marcacao inicial presente: ", initial_marking)
    else:
        print("  Sem marcacao inicial")

    if len(final_marking) > 0:
        print("  Marcacao final presente: ", final_marking)
    else:
        print("  Sem marcacao final")

    print("ANALISE DE DEADLOCKS POTENCIAIS:")

    potential_deadlocks = []
    for place in net.places:
        input_transitions = [arc.source for arc in net.arcs if arc.
            target == place]
        output_transitions = [arc.target for arc in net.arcs if arc.
            source == place]

        if len(input_transitions) > 1 and len(output_transitions) == 0:
            potential_deadlocks.append(place.name)

    if potential_deadlocks:
        print("  Lugares com potencial deadlock: ", len(
            potential_deadlocks))
        for place in potential_deadlocks[:3]:
            print("      - ", place)
    else:
        print("  Nenhum deadlock potencial identificado")

    return {
        'sequencias': sequencias,
```

```
        'escolhas': escolhas,
        'loops': loops,
        'potential_deadlocks': potential_deadlocks
    }

def simular_processo_petri(net, initial_marking, final_marking):
    # Simula execucao do processo na Rede de Petri
    from pm4py.algo.simulation.playout.petri_net import algorithm as
        simulator

    print("\n" + "="*70)
    print("SIMULACAO DO PROCESSO - REDE DE PETRI")
    print("="*70)

    simulated_log = simulator.apply(net, initial_marking,
                                    parameters={
                                        simulator.Variants.BASIC_PLAYOUT
                                            .value.Parameters.NO_TRACES:
                                            10
                                    })

    print("RESULTADOS DA SIMULACAO (10 casos):")
    print("  Casos simulados: ", len(simulated_log))

    from pm4py.algo.filtering.log.variants import variants_filter
    variants = variants_filter.get_variants(simulated_log)

    print("  Variantes unicas na simulacao: ", len(variants))

    print("CAMINHOS SIMULADOS:")
    for i, (variant, cases) in enumerate(sorted(variants.items(),
                                                key=lambda x: len(x[1])
                                                    ,
                                                reverse=True)[:3]):
        variant_str = " -> ".join([event['concept:name'] for event in
            variant])
        print("  ", i+1, ". ", variant_str)
        print("      Ocorrencias: ", len(cases))
```

```
    print("COMPARACAO COM LOG REAL:")
    print("  A simulacao explora caminhos possiveis no modelo")
    print("  Permite identificar comportamentos nao observados no log")

    return simulated_log, variants

def analisar_bottlenecks_petri(net, log):
    # Analise avancada de bottlenecks usando teoria de Redes de Petri

    print("\n" + "="*70)
    print("ANALISE DE BOTTLENECKS - TEORIA DE REDES DE PETRI")
    print("="*70)

    sojourn_times = {}
    for trace in log:
        for i in range(len(trace) - 1):
            current = trace[i]
            next_event = trace[i + 1]
            activity = current['concept:name']
            duration = (next_event['time:timestamp'] - current['time:
                timestamp']).total_seconds() / 60
            sojourn_times.setdefault(activity, []).append(duration)

    print("LUGARES DE ESPERA/ESTOQUE:")

    bottlenecks = []
    for place in net.places:
        input_arcs = [arc for arc in net.arcs if arc.target == place]
        output_arcs = [arc for arc in net.arcs if arc.source == place]

        if len(input_arcs) > 1 and len(output_arcs) == 1:
            print("  ", place.name, ": Ponto de sincronizacao (input: "
                ,
                    len(input_arcs), ", output: ", len(output_arcs), ")")

    print("ANALISE DE CAPACIDADE:")

    resource_places = []
    for place in net.places:
```

```
        if "Resource" in place.name or "Agent" in place.name or "
            Specialist" in place.name:
            resource_places.append(place.name)
            print("  ", place.name, ": Representa recurso limitado")

    print("ANALISE DE FLUXO:")

    transition_demand = {}
    for transition in net.transitions:
        if transition.label and transition.label in sojourn_times:
            times = sojourn_times[transition.label]
            if times:
                avg_time = sum(times) / len(times)
                if avg_time > 60:
                    transition_demand[transition.label] = avg_time

    if transition_demand:
        print("  TRANSICOES COM ALTO TEMPO DE PROCESSAMENTO:")
        for activity, avg_time in sorted(transition_demand.items(), key
            =lambda x: x[1], reverse=True):
            print("      ", activity, ": ", round(avg_time, 1), "
                minutos (media)")
    else:
        print("  Nenhuma transicao com tempo excessivo identificada")

    return {
        'sojourn_times': sojourn_times,
        'resource_places': resource_places,
        'high_demand_transitions': transition_demand
    }

def visualizacao_avancada_petri(net, initial_marking, final_marking,
    log):
    # Visualizacoes avancadas da Rede de Petri

    print("\n" + "="*70)
    print("VISUALIZACAO AVANCADA - REDE DE PETRI")
    print("="*70)
```

```
    print("1. REDE DE PETRI COM FREQUENCIAS:")

    parameters = {
        pn_visualizer.Variants.WO_DECORATION.value.Parameters.FORMAT: "
            png",
        pn_visualizer.Variants.WO_DECORATION.value.Parameters.DEBUG:
            False,
        pn_visualizer.Variants.WO_DECORATION.value.Parameters.RANKDIR:
            "LR"
    }

    gviz_decorated = pn_visualizer.apply(net, initial_marking,
        final_marking,
                                         parameters=parameters,
                                         variant=pn_visualizer.Variants
                                             .WO_DECORATION)
    pn_visualizer.save(gviz_decorated, "helpdesk_petri_decorated.png")

    print("2. MATRIZ DE INCIDENCIA DA REDE:")

    places = list(net.places)
    transitions = list(net.transitions)

    print("  Dimensoes: ", len(places), " lugares x ", len(transitions)
        , " transicoes")
    print("  Matriz sparse (muitos zeros)")

    print("3. ESPACO DE ESTADOS (SIMPLIFICADO):")

    print("  Estados possiveis: Exponencial no numero de lugares")
    print("  Para analise completa: Arvore/grafo de alcancabilidade")

    return gviz_decorated

def gerar_relatorio_petri(net, initial_marking, final_marking, log,
                          conformance_results, structural_results,
                          simulation_results, bottleneck_results):
    # Gera relatorio completo da analise com Redes de Petri
```

```
print("\n" + "="*70)
print("RELATORIO␣COMPLETO␣-␣ANALISE␣COM␣REDES␣DE␣PETRI")
print("="*70)

print("RESUMO␣EXECUTIVO:")
print("␣␣Processo␣analisado:␣Helpdesk␣de␣TI")
print("␣␣Casos␣analisados:␣", len(log))
print("␣␣Fitness␣do␣modelo:␣", round(conformance_results['fitness'
    ], 3))
print("␣␣Conformidade:␣", conformance_results['fitting_traces'],
      "/", len(log), "␣casos")

print("INSIGHTS␣PRINCIPAIS:")

if bottleneck_results['high_demand_transitions']:
    print("␣␣1.␣BOTTLENECKS␣IDENTIFICADOS:")
    for activity, avg_time in bottleneck_results['
        high_demand_transitions'].items():
        print("␣␣␣␣␣␣", activity, ":␣", round(avg_time, 1), "␣
            minutos")
else:
    print("␣␣1.␣Nenhum␣bottleneck␣critico")

if structural_results['potential_deadlocks']:
    print("␣␣2.␣POTENCIAIS␣DEADLOCKS:␣",
          len(structural_results['potential_deadlocks']))
else:
    print("␣␣2.␣Nenhum␣deadlock␣potencial")

from pm4py.algo.filtering.log.variants import variants_filter
variants = variants_filter.get_variants(log)
print("␣␣3.␣VARIABILIDADE:␣", len(variants), "␣variantes␣unicas")

print("RECOMENDACOES:")

if conformance_results['fitness'] < 0.9:
    print("␣␣1.␣Melhorar␣conformidade␣do␣processo␣(fitness:␣",
          round(conformance_results['fitness'], 3), ")")
    print("␣␣␣␣␣␣Analisar␣casos␣problematicos:␣",
```

```
            len(conformance_results['problematic_cases']))

    if bottleneck_results['high_demand_transitions']:
        print("  2. Otimizar atividades lentas:")
        for activity, avg_time in bottleneck_results['
            high_demand_transitions'].items():
            if avg_time > 120:
                print("     URGENTE: ", activity, " (", round(avg_time
                    , 0), " min)")
            elif avg_time > 60:
                print("     IMPORTANTE: ", activity, " (", round(
                    avg_time, 0), " min)")

    print("PROXIMOS PASSOS SUGERIDOS:")
    print("  1. Implementar monitoramento em tempo real com o modelo")
    print("  2. Estabelecer KPIs baseados nas metricas identificadas")
    print("  3. Realizar simulacoes de cenarios de melhoria")
    print("  4. Implementar sistema de detecao precoce de desvios")

    print("DADOS EXPORTADOS:")
    print("  1. helpdesk_petri_alpha.png - Modelo Alpha Miner")
    print("  2. helpdesk_petri_inductive.png - Modelo Inductive Miner")
    print("  3. helpdesk_petri_heuristics.png - Modelo Heuristics Miner
        ")
    print("  4. helpdesk_heuristics_net.png - Heuristics Net")

    return {
        'summary': {
            'cases': len(log),
            'fitness': conformance_results['fitness'],
            'variants': len(variants),
            'bottlenecks': len(bottleneck_results['
                high_demand_transitions']),
            'potential_deadlocks': len(structural_results['
                potential_deadlocks'])
        }
    }

def main():
```

```
# Funcao principal da analise com Redes de Petri

print("\n" + "="*70)
print("ANALISE DE PROCESSOS COM REDES DE PETRI - HELPDESK")
print("="*70)
print("Autor: Process Mining Specialist")
print("Data: 2024")
print("="*70)

print("1. CARREGANDO DADOS DO HELPDESK...")
log, df = carregar_dados_helpdesk()
print("   ", len(log), " casos carregados")
print("   ", sum(len(trace) for trace in log), " eventos totais")

print("2. DESCOBRINDO MODELOS COM REDES DE PETRI...")

net_alpha, im_alpha, fm_alpha = descobrir_modelo_petri_alpha(log)
net_ind, im_ind, fm_ind = descobrir_modelo_petri_inductive(log)
net_heu, im_heu, fm_heu, heu_net =
    descobrir_modelo_petri_heuristics(log)

print("3. ANALISANDO CONFORMIDADE...")
conformance_results = analisar_conformidade_petri(log, net_alpha,
    im_alpha, fm_alpha)

print("4. ANALISANDO ESTRUTURA DA REDE...")
structural_results = analisar_estrutura_petri(net_alpha, im_alpha,
    fm_alpha)

print("5. SIMULANDO PROCESSO...")
simulated_log, simulation_variants = simular_processo_petri(
    net_alpha, im_alpha, fm_alpha)

print("6. IDENTIFICANDO BOTTLENECKS...")
bottleneck_results = analisar_bottlenecks_petri(net_alpha, log)

print("7. GERANDO VISUALIZACOES AVANCADAS...")
gviz_decorated = visualizacao_avancada_petri(net_alpha, im_alpha,
    fm_alpha, log)
```

```
    print("8. GERANDO RELATORIO FINAL...")
    relatorio = gerar_relatorio_petri(net_alpha, im_alpha, fm_alpha,
        log,
                                      conformance_results,
                                         structural_results,
                                      {'variants': simulation_variants},
                                          bottleneck_results)

    print("\n" + "="*70)
    print("ANALISE COM REDES DE PETRI CONCLUIDA COM SUCESSO!")
    print("="*70)

    return {
        'log': log,
        'models': {
            'alpha': (net_alpha, im_alpha, fm_alpha),
            'inductive': (net_ind, im_ind, fm_ind),
            'heuristics': (net_heu, im_heu, fm_heu)
        },
        'conformance': conformance_results,
        'structure': structural_results,
        'bottlenecks': bottleneck_results,
        'report': relatorio
    }

if __name__ == "__main__":
    try:
        resultados = main()

        import json
        with open('resultados_petri.json', 'w') as f:
            resultados_serializaveis = {
                'statistics': {
                    'cases': len(resultados['log']),
                    'fitness': resultados['conformance']['fitness'],
                    'problematic_cases': len(resultados['conformance'][
                        'problematic_cases'])
                },
```

```
                'bottlenecks': resultados['bottlenecks']['
                    high_demand_transitions'],
                'report_summary': resultados['report']['summary']
            }
            json.dump(resultados_serializaveis, f, indent=2)

        print("\nArquivos gerados:")
        print("   resultados_petri.json - Dados da analise")
        print("   helpdesk_petri_*.png - Modelos visuais")
        print("\nAnalise pronta para apresentacao e tomada de decisao!"
            )

    except Exception as e:
        print("Erro durante a execucao: ", e)
        print("Solucao: Certifique-se de ter PM4Py instalado:")
        print("  pip install pm4py[all]")
```

ANEXO A – Saída do Script `petri_helpdesk.py`

Identificação do relatório

Título Análise de Processos com Redes de Petri – Helpdesk

Autor Process Mining Specialist

Ano 2024

1. Carregamento de dados

- Casos carregados: 5
- Eventos totais: 34

2. Descoberta de modelos com Redes de Petri

2.1. Descoberta do modelo – Alpha Miner

Estatísticas da Rede de Petri:

Elemento	Valor
Transições (atividades)	9
Lugares (estados)	10
Arcos (relações)	22
Marcação inicial	`[start:1]`
Marcação final	`[end:1]`

2.2. Descoberta do modelo – Inductive Miner

Estatísticas da Rede (Inductive):

Elemento	Valor
Transições	12
Lugares	10
Soundness	Garantida pelo algoritmo

2.3. Descoberta do modelo – Heuristics Miner

Estatísticas da Heuristics Net:

Elemento	Valor
Nós	9
Arestas	N/A
Robustez a ruídos	Alta

3. Análise de conformidade

3.1. Métricas de conformidade (replay com TBR)

Métrica	Resultado
Fitness	0.843 (escala 0–1; 1 = perfeito)
Casos conformes	0/5 (0.0%)
Tokens faltando	5
Tokens restantes	11
Precision	0.558 (quanto maior, mais específico o modelo)
Generalization	0.428 (quanto maior, melhor generalização)

3.2. Análise detalhada dos desvios

- Total de casos com problemas: 5
- Casos reportados (exemplos):
  - `NOT_OK 1001`: 0 tokens faltando
  - `NOT_OK 1002`: 1 token faltando
  - `NOT_OK 1003`: 2 tokens faltando

4. Análise estrutural da Rede de Petri

4.1. Construções identificadas

- Sequências: 4
- Pontos de escolha (XOR/OR): 2
- Loops identificados: 0

4.2. Transições críticas (visíveis)

- Ticket Received
- Ticket Closed
- Solution Provided
- Manager Review
- Assign to Specialist
- Escalate to Manager
- Waiting for User
- Initial Triage
- Technical Analysis

4.3. Verificação de soundness e deadlocks

- Marcação inicial presente: `[start:1]`
- Marcação final presente: `[end:1]`
- Deadlocks potenciais: nenhum deadlock potencial identificado

5. Simulação do processo

5.1. Resultados

- Casos simulados: 10
- Variantes únicas na simulação: 10

5.2. Caminhos simulados (versão compactada para leitura)

Observação: repetições extensas de `Waiting for User` foram suprimidas por brevidade no anexo.

1. `Ticket Received → Waiting for User → Initial Triage → Assign to Specialist → Technical Analysis → Solution Provided → Ticket Closed`
   (com esperas intermediárias repetidas por longos períodos)
2. `Waiting for User → Ticket Received → Initial Triage → Assign to Specialist → Technical Analysis → Solution Provided → Ticket Closed`
   (com esperas intermediárias repetidas por longos períodos)

3. `Waiting for User` → `Ticket Received` → `Initial Triage` → `Assign to Specialist` → `Technical Analysis` → `Solution Provided` → `Ticket Closed`
   (com esperas intermediárias repetidas por longos períodos)

5.3. Comparação com o log real

- A simulação explora caminhos possíveis no modelo.
- Permite identificar comportamentos não observados no log.

6. Identificação de bottlenecks

6.1. Lugares de espera/estoque (pontos de sincronização)

- (`{Manager Review, Technical Analysis}`, `{Solution Provided}`): sincronização (input: 2; output: 1)
- (`{Initial Triage, Technical Analysis}`, `{Escalate to Manager}`): sincronização (input: 2; output: 1)

6.2. Análise de capacidade (recursos limitados)

- (`{Initial Triage}`, `{Assign to Specialist, Escalate to Manager}`): representa recurso limitado
- (`{Assign to Specialist}`, `{Technical Analysis}`): representa recurso limitado

6.3. Transições com alto tempo médio de processamento

Transição	Tempo médio (min)
Escalate to Manager	562.5
Solution Provided	234.0
Waiting for User	180.0
Manager Review	90.0
Technical Analysis	66.0

7. Visualizações avançadas

1. Rede de Petri com frequências
2. Matriz de incidência da rede
   - Dimensões: 10 lugares × 9 transições

- Matriz esparsa (muitos zeros)

3. Espaço de estados (simplificado)
    - Estados possíveis: crescimento exponencial no número de lugares
    - Para análise completa: árvore/grafo de alcançabilidade

8. Relatório final (síntese executiva)

8.1. Resumo executivo

- Processo analisado: Helpdesk de TI
- Casos analisados: 5
- Fitness do modelo: 0.843
- Conformidade: 0/5 casos

8.2. Insights principais

1. Bottlenecks identificados (tempo médio em minutos):
    - Solution Provided: 234.0
    - Manager Review: 90.0
    - Escalate to Manager: 562.5
    - Waiting for User: 180.0
    - Technical Analysis: 66.0
2. Nenhum deadlock potencial identificado
3. Variabilidade: 4 variantes únicas (log real)

8.3. Recomendações

1. Melhorar conformidade do processo (fitness = 0.843): analisar os 5 casos problemáticos.
2. Otimizar atividades lentas:
    - **URGENTE**: Solution Provided (234.0 min)
    - **IMPORTANTE**: Manager Review (90.0 min)
    - **URGENTE**: Escalate to Manager (562.5 min)
    - **URGENTE**: Waiting for User (180.0 min)
    - **IMPORTANTE**: Technical Analysis (66.0 min)

8.4. Próximos passos sugeridos

1. Implementar monitoramento em tempo real com base no modelo
2. Estabelecer KPIs a partir das métricas identificadas
3. Realizar simulações de cenários de melhoria
4. Implementar detecção precoce de desvios

8.5. Dados exportados e arquivos gerados

- `helpdesk_petri_alpha.png` – Modelo Alpha Miner
- `helpdesk_petri_inductive.png` – Modelo Inductive Miner
- `helpdesk_petri_heuristics.png` – Modelo Heuristics Miner
- `helpdesk_heuristics_net.png` – Heuristics Net
- `resultados_petri.json` – Dados da análise
- `helpdesk_petri_*.png` – Modelos visuais

Conclusão

Análise com Redes de Petri concluída com sucesso. A saída acima encontra-se formatada para documentação técnica e suporte à tomada de decisão.