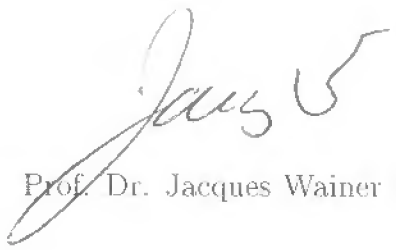
Instituto de Computação
Universidade Estadual de Campinas

Algoritmos de Detecção de Anomalias em Logs de Sistemas Baseados em Processos de Negócios

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Fábio de Lima Bezerra e aprovada pela Banca Examinadora.

Campinas, 21 de junho de 2011.



Prof. Dr. Jacques Wainer (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Bezerra, Fábio de Lima
B469a Algoritmos de detecção de anomalias em logs de sistemas baseados em processos de negócios/Fábio de Lima Bezerra-- Campinas, [S.P. : s.n.], 2011.

Orientador : Jacques Wainer.
Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Anomalias. 2.Sistemas de informação gerencial - Medidas de segurança . I. Jacques, Wainer, 1958-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Anomaly detection algorithms in logs of business process aware systems

Palavras-chave em inglês (Keywords): 1.Anomaly. 2.Management information systems - Safety measures.

Área de concentração: Ciência da Computação

Titulação: Doutor em Ciência da Computação

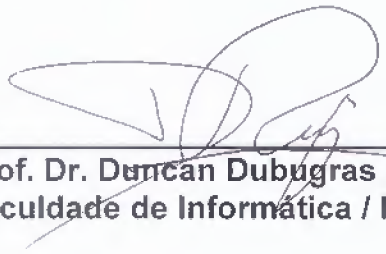
Banca examinadora: Prof. Dr. Jacques Wainer (IC – UNICAMP)
Prof. Dr. Duncan Dubugras Alcoba Ruiz (PUC-RS)
Profa. Dra. Flávia Maria Santoro (UNIRIO)
Profa. Dra. Maria Beatriz Felgar de Toledo (IC – UNICAMP)
Prof. Dr. Edmundo Roberto Mauro Madeia (IC – UNICAMP)

Data da defesa: 13/05/2011

Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 13 de maio de 2011, pela Banca examinadora composta pelos Professores Doutores:

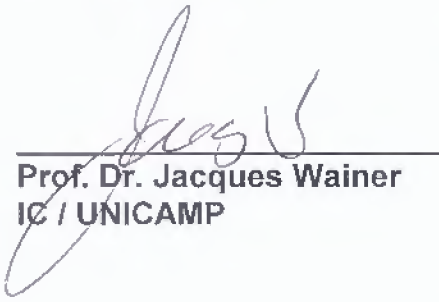


Prof. Dr. Duncan Dubugras Alcoba Ruiz
Faculdade de Informática / PUC-RS

Profa. Dra. Flávia Maria Santoro
Centro de Ciências Exatas e da Terra / UNIRIO

Prof.. Dr. Prof. Dr. Edmundo Roberto Mauro Madeira
IC / UNICAMP

Profa. Dra. Maria Beatriz Felgar de Toledo
IC / UNICAMP



Prof. Dr. Jacques Wainer
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos de Detecção de Anomalias em Logs de Sistemas Baseados em Processos de Negócios

Fábio de Lima Bezerra[1]

13 de Maio de 2011

Banca Examinadora:

- Prof. Dr. Jacques Wainer (Orientador)
- Prof. Dr. Duncan Dubugras Ruiz
  Faculdade de Informática – PUC-RS
- Profa. Dra. Flávia Maria Santoro
  Centro de Ciências Exatas e da Terra – UNIRIO
- Profa. Dra. Maria Beatriz Felgar de Toledo
  Instituto de Computação – UNICAMP
- Prof. Dr. Edmundo Roberto Mauro Madeira
  Instituto de Computação – UNICAMP

[1]Este trabalho foi financiado pela CAPES e CNPq

Resumo

Atualmente há uma variedade de sistemas que apoiam processos de negócio (ex. WfMS, CRM, ERP, SCM, etc.). Muitos desses sistemas possuem uma forte característica de coordenação das atividades dos processos de negócios, garantindo que essas atividades sejam executadas como especificadas no modelo de processo. Entretanto, há domínios com maior necessidade de flexibilidade na execução desses processos, por exemplo, em atendimento hospitalar, cuja conduta pode variar para cada paciente. Essa característica desses domínios demanda o desenvolvimento de sistemas orientados a processos fracamente definidos, ou com execução mais flexível. Nesses domínios, a execução de algumas atividades comuns pode ser violada, ou a execução de uma atividade "incomum" pode ser necessária, ou seja, tais processos são suscetíveis a execuções excepcionais ou mesmo fraudulentas. Assim, o provimento de flexibilidade não pode ser considerado sem melhorar as questões relacionadas à segurança, pois flexibilidade e segurança são requisitos claramente conflitantes. Portanto, é necessário desenvolver mecanismos ou métodos que permitam a conjugação desses dois requisitos em um mesmo sistema, promovendo um balanço entre flexibilidade e segurança.

Esta tese tem por objetivo projetar, implementar e avaliar métodos de detecção de anomalias em logs de sistemas de apoio a processos de negócios, ou seja, o desenvolvimento de métodos utilizados para descobrir quais instâncias de processo podem ser uma execução anômala. Desta forma, através da integração de um método de detecção de anomalias com um sistema de apoio a processos de negócio, tais sistemas poderão oferecer um ambiente de execução flexível, mas capaz de identificar execuções anômalas que podem indicar desde uma execução excepcional, até uma tentativa de fraude. Assim, o estudo de métodos de detecção de eventos anômalos vem preencher um espaço pouco explorado pela comunidade de *process mining*, que tem demonstrado maior interesse em entender o comportamento comum em processos de negócios. Entretanto, apesar desta tese não discutir o significado das instâncias anômalas, os métodos de detecção apresentados aqui são importantes porque permitem selecionar essas instâncias.

Abstract

Nowadays, many business processes are supported by information systems (e.g. WfMS, CRM, ERP, SCM, etc.). Many of these systems have a strong characteristic of coordination of activities defined in the business processes, mainly for ensuring that these activities are performed as specified in the process model. However, there are domains that demand more flexible systems, for example, hospital and health domains, whose behavior can vary for each patient. Such domains of applications require an information system in which the business processes are weakly defined, supporting more flexible and dynamic executions. For example, the execution of some common activities may be violated, or some unusual activity may be enforced for execution. Therefore, in domains of applications in which the systems support a high level of flexibility the business processes are susceptible to exceptional or even fraudulent executions. Thus, the provision of flexibility can not be considered without improving the security issues, since there is clearly a trade-off between flexibility and security requirements. Therefore, it is necessary to develop a mechanism to allow the combination of these two requirements in a system, that is, a mechanism that promotes a balance between flexibility and security.

This thesis aims to design, implement and evaluate methods for detecting anomalies in logs of process-aware information systems, that is, the development of methods to find out which process instances may be an anomalous execution. Thus, when incorporating a method for detecting anomalies in such systems, it would be possible to offer a flexible and safer execution environment, since the system is also able to identify anomalous executions, which could be a simple exception or a harmful fraud attempt. Thus, the study of methods for detecting anomalous events will fill an area largely unexplored by the community of process mining, which has been mainly interested in understanding the common behavior in business processes. Furthermore, although this thesis does not discuss the meaning of an anomalous instance, the methods and algorithms presented here are important because they allow us to identify those instances.

Agradecimentos

Agradeço a **DEUS**, que na sua omnipotência, me deu vida, fortaleza, sabedoria e inteligência para trabalhar nessa tese. Também me forneceu todas as razões para não esquecer de registrar os próximos agradecimentos.

Agradeço ao meu grande orientador, **Prof. Jacques Wainer**, cuja inteligência, humor e sabedoria me impressionam. Obrigado por me deixar trabalhar de casa!

Agradeço às agências de pesquisa que forneceram-me o suporte material necessário para manter-me firme no processo de conclusão desta tese.

Agradeço ao supervisor de meu estágio doutoral na Holanda, o **Prof. Wil van der Aalst**, pela oportunidade de trabalhar com um grupo de pesquisadores muito competentes na área de *Process Mining*.

Agradeço aos meus pais, **Aldenora** (carinhosamente chamada pela minha filha de Vó Lola) e **Claudionor** (Dodô).

Agradeço aos meus irmãos, **Eduardo** e **Diego**.

Agradeço a minha esposa **Erika** pela parcela de "investimento" nesse projeto profissional, pois acredito que nenhum sucesso profissional pagaria o preço de um fracasso pessoal.

Agradeço a minha filha **Ana Beatriz**, que indiretamente me mantém firme no desejo de ser um exemplo.

Agradeço aos meus muitos **familiares**. Com eles entendi que orgulhar-se é motivar.

Agradeço aos meus **amigos** de Campinas, Eindhoven e Belém, que apesar de juntos formarem um conjunto finito, preferi me acovardar em citá-los que correr o risco de esquecer algum nome aqui. Amigos, vocês foram muito importante para o fim dessa tese!

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Um processo de negócio é um conjunto de atividades e recursos (humanos ou materiais) organizados para resolver um problema particular, ou seja, produzir um valor a partir de um processamento de entrada. Há empresas que costumam adotar um conjunto de ferramentas computacionais que automatizam a execução ou gerenciamento de seus processos a fim de melhorar ou mesmo conhecer seus processos de negócios.

Durante a década de 90, em razão das novas tendências na área de gerenciamento, os sistemas de informação orientados a processos (do inglês, ***P**rocess **A**ware **I**nformation **S**ystems*, PAIS) foram amplamente adotados nas organizações [22]. A adoção desses sistemas representava uma mudança dos sistemas orientados a dados, para sistemas orientados a processos, que claramente separam lógica do negócio e aplicações, o que facilita mudanças nos modelos de processo adotados.

Essas ferramentas computacionais, possuem pelo menos dois objetivos: (*i*) orientar e conduzir o processo de negócio; e (*ii*) orientar os usuários a realizarem corretamente os seus trabalhos [25]. Esses sistemas são oferecidos em diferentes formatos e para aplicações variadas, por exemplo: ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), SCM (Supply Chain Management), WfMS (Workflow Management System), entre outros.

1.1 Contextualização e Problema

A necessidade de aderência a requisitos legais foi também responsável pela adoção desses sistemas (PAIS), pois eles apoiam as melhores práticas de governança (ex. COBIT, *Control Objectives for Information and related Technology*), ou seja, melhoram o controle da execução dos processos. Nesse contexto há o exemplo da *Sarbanes-Oxley Act*, que é uma lei federal nos Estados Unidos, criada em resposta aos escândalos contábeis e financeiros da Enron e da WorldCom.

Por outro lado, o controle de processo de negócios de empresas com modelos de processos muito dinâmicos, especialmente sensíveis a competitividade, não deve ser apoiado por sistemas normativos, por exemplo, por um WMS (do inglês, *Workflow Management System*). Essas empresas exigem modelos flexíveis de gerenciamento de seus processos de negócio, pois precisam responder rapidamente a novas estratégias de mercado ou novos modelos de negócio.

Além disso, em certos domínios de aplicação, como desenvolvimento de software e atendimento hospitalar, onde o controle dos processos também não pode ser realizado através de uma ferramenta normativa, pois o modelo operacional do processo não é totalmente conhecido antes de sua execução, portanto, não pode ser representado antes de sua execução. Nesses domínios os participantes do processo necessitam de maior flexibilidade para executar seu trabalho. Por exemplo, no contexto de atendimento hospitalar, o sistema não pode obrigar a execução de uma tarefa específica (ex. administração de um medicamento) em um dado atendimento, pois, mesmo que tal atendimento seja semelhante a outros, essa execução é única e deve ser flexível em relação ao participante (ex. médico ou enfermeiro), permitindo que o mesmo execute a tarefa que julgar mais apropriada ou até conveniente (ex. administrar um medicamento X na ausência de um medicamento Y). Neste exemplo, a flexibilidade é oferecida quando o participante decide qual atividade executar.

Por outro lado, um sistema de informação com maior flexibilidade de execução é mais vulnerável a execuções incorretas ou até fraudulentas, pois os usuários têm liberdade para executar a(s) atividade(s) que julgarem necessária(s) durante uma instância de processo em particular. Portanto, há claramente um cenário conflitante entre os interesses de flexibilidade e de segurança, por exemplo, que dificultem ou evitem a ocorrência de fraudes. Em outras palavras, o sistema deve prover a flexibilidade por razões de competitividade, mas também deve evitar ou identificar o mau uso do sistema. Assim, há claramente uma demanda por sistemas de auditoria, e palavras como BAM (do inglês, *Business Activity Monitoring*), BOM (do inglês, *Business Operations Management*) e BPI (do inglês, *Business Process Intelligence*) ilustram o interesse de empresas fornecedoras de sistemas em desenvolver soluções de monitoramento e análise de processo de negócios[36].

Esta tese tem o objetivo de desenvolver soluções que satisfaçam os interesses conflitantes de flexibilidade e segurança em ambientes ou aplicações apoiadas por sistemas gerenciadores de processos de negócios. Especificamente, este trabalho desenvolverá métodos de detecção de instâncias de processos, também chamadas de *traces*, que caracterizem uma anomalia, ou seja, uma execução irregular, que pode estar vinculada a diferentes semânticas, inclusive a de uma fraude. Então, a integração de uma ferramenta de detecção de anomalias com um sistema de apoio a processos de negócios possibilitará o controle da execução dos processos mesmo em sistemas flexíveis, pois tais ferramentas seriam capa-

zes de identificar as instâncias anômalas ao mesmo tempo em que apoiariam a execução flexível dos processos de negócios.

1.2 Abordagens de Detecção

Um processo de negócio acomoda três perspectivas de informações: (*i*) a perspectiva **organizacional**, que descreve os responsáveis pelas atividades; (*ii*) a perspectiva dos **dados** manipulados pela execução do processo; e (*iii*) a perspectiva do **fluxo de controle**, que descreve as atividades do processo, além das restrições de ordem de execução dessas atividades [44, 48]. A alta oferta de *logs* gerados pelos sistemas de informação e a elevada perspectiva de adoção de sistemas de gerenciamento de processo motivaram o desenvolvimento da área de *process mining*, interessada principalmente em descobrir modelos de processos a partir de um *log*[44, 48, 46, 45].

Entretanto, movimentos da comunidade acadêmica têm demonstrado a necessidade de descobrir também o comportamento não normal ou anômalo, que eventualmente existam nesses *logs*[43, 3, 5, 4, 8, 7, 9, 10, 11]. Esta tese explora duas abordagens de detecção, todas baseadas na avaliação do **fluxo de controle** das instâncias de processo analisadas (*traces*). Assim, quando um modelo de processo possui um caminho que acomode a execução (fluxo) de um *trace* do *log*, dizemos que esse *trace* casa com o modelo, portanto é um *trace* normal, do contrário, é um *trace* anômalo.

A **primeira abordagem**, apresentada no Capítulo 4, considera um *trace* anômalo quando esse *trace*, para ser executado, exige **grandes** modificações no modelo de processo de negócio descoberto por um algoritmo/método de mineração de processos. A **segunda abordagem**, apresentada no Capítulo 5, classifica um *trace* como anômalo se esse *trace* não é instância de um *modelo apropriado*, onde apropriado significa: **(i)** um modelo dinamicamente descoberto por um algoritmo/método de mineração de processos, que **(ii)** é capaz de executar completamente um número mínimo de *traces* do *log* utilizado para encontrar o modelo, e **(iii)** maximiza uma função que representa o balanço entre complexidade (tamanho do modelo) e especificidade (capacidade de executar apenas os *traces* do *log*).

A implementação de cada uma das abordagens exploradas nesta tese possui diferentes variações de composição. Por exemplo, um número variado de algoritmos de mineração de processo pode ser utilizado para gerar os modelos de processos usados na detecção da anomalia. Além das opções de algoritmos de mineração de processo, há também uma variada disponibilidade de ferramentas para avaliação de modelos (ex.: métricas de conformidade). Essas opções de ferramentas de avaliação consideram características como complexidade (ex.: número de vértices e/ou arestas) e especifidade de modelos (ex.: o quanto o modelo é capaz de prever novas instâncias de processos de negócio).

As ferramentas utilizadas para compor a implementação das abordagens, ou estão disponíveis gratuitamente no *Framework ProM*[1], ou foram implementadas. Entre as ferramentas implementadas podemos citar o algoritmo de mineração incremental e a métrica de cálculo de custo de inclusão[51, 3, 5, 8]. Assim, em razão das diferentes variações de composição dos algoritmos de mineração de processos e algoritmos de análise de processos, temos diferentes soluções de detecção de *traces* anômalos. Cada solução foi exaustivamente testada com uma variada coleção de *logs* sinteticamente criados, cujo processo de avaliação e geração dos *logs* é explicado na Seção 1.3.

1.3 Dados para avaliação dos algoritmos

O processo de desenvolvimento dos algoritmos propostos neste trabalho considerou a execução de quatro atividades:

1. A *definição intuitiva ou formal* de *trace* anômalo que apoie o processo de concepção de um algoritmo de detecção.

2. A *concepção de um algoritmo*, ou seja, uma proposição abstrata do que acreditamos, a partir de estudos teóricos, ser uma boa abordagem de detecção dos *traces* anômalos, como anteriormente definidos.

3. O *projeto e implementação do algoritmo* em alguma linguagem de programação, assim temos um componente de software real que pode ser avaliado.

4. Finalmente, a *avaliação* do algoritmo implementado, que para ser avaliado, necessitava de um conjunto de *logs* com *traces* anômalos identificados. Assim, a eficácia dos algoritmos de detecção pode ser medida e é baseada em minimizar o número de falsos positivo (*traces* normais classificados como anômalos) e maximizar o número de verdadeiros positivo (*traces* anômalos classificados como anômalos).

Por estarmos realizando uma pesquisa experimental, acreditamos que a atividade de avaliação sempre é decisiva para imprimir um caráter científico, bem como validar os resultados, especialmente porque os algoritmos propostos serão avaliados com dados sintéticos. Na Seção 1.3.1 apresentamos uma justificativa para utilização de dados sintéticos no processo de avaliação dos algoritmos. Na Seção 1.3.2 apresentamos como os *traces* anômalos são criados, ou seja, que características diferem os *traces* anômalos quando comparados com as instâncias e *traces* normais. Finalmente, na Seção 1.3.3 apresentamos a dinâmica de criação dos *logs* utilizados na avaliação dos algoritmos.

[1] Disponível em `http://prom.sf.net`.

1.3.1 Utilização de Dados Sintéticos

Dois motivos influenciaram o uso de dados sintéticos na avaliação dos algoritmos: (*i*) a indisponibilidade de uma fonte com dados reais; mas principalmente (*ii*) a imprecisão da definição de *trace* anômalo em um *log* real, pois para avaliarmos a eficácia da detecção seria necessário conhecermos o(s) *trace(s)* anômalo(s) no *log* antes da aplicação do algoritmo, o que seria extremamente complicado ou mesmo impossível com dados reais. Quanto ao problema da imprecisão da definição de um *trace* anômalo em um *log* (item *ii* acima), Pandit et al., em [32], descrevem sucintamente o problema de identificar as instâncias anômalas em um conjunto de dados reais. A identificação manual dessas instâncias anômalas é muito dispendiosa e subjetiva, pois é provável que duas pessoas classifiquem de forma diferente – normal ou anômala – a mesma instância. Além disso a definição de anomalia varia de domínio para domínio.

Diferentemente, a utilização de dados sintéticos simplifica a avaliação dos algoritmos, pois ao conhecermos o modelo de processo utilizado para criar o *log*, os *traces* normais são os *traces* que são instância desse modelo, enquanto que os *traces* anômalos são os *traces* que **não** são instância desse modelo (um critério bem objetivo do que seria uma instância de processo anômala). Dessa forma, um *log* é normal caso seja composto apenas pelas instâncias do modelo que gerou esse *log*.

Entendemos que a utilização de dados reais poderia imprimir ao trabalho um valor científico maior, já que os resultados suscitariam menos dúvidas da aplicação dos métodos de detecção em cenários reais. Entretanto, acreditamos que os *logs* sintéticos utilizados neste trabalho são uma boa aproximação de cenários reais, como descreve a Seção 1.3.2. Dessa forma, entendemos que os resultados e as conclusões apresentadas nesta tese são significativos.

1.3.2 Abordagens de Criação dos *Traces* Anômalos

Os *logs* utilizados na avaliação são preenchidos com *traces* normais e *traces* anômalos, do contrário seria muito complicado ou até mesmo impossível medir a acurácia dos algoritmos de detecção em encontrar as instâncias anômalas. Uma vez que a criação do *log* é baseada em um modelo de processo dinamicamente construído e de forma aleatória, o problema seria como construir os *traces* anômalos e adicioná-los ao *log*.

No caso da criação dos *traces* anômalos, utilizamos diferentes estratégias de geração, citadas a seguir:

Duplicação de uma Atividade Aleatória do Trace. Essa abordagem considera a duplicação de uma atividade de um *trace* que é instância do modelo utilizado para criar o *log* normal. Por exemplo, o *trace* $[a - b - c - a]$ é anômalo ao modelo a) da

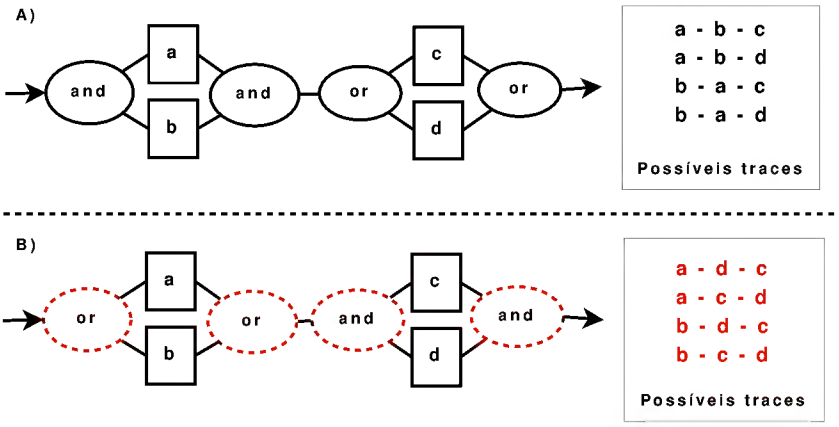


Figura 1.1: Exemplo de criação de *traces* anômalos.

Figura 1.1, e foi criado a partir do *trace* normal $[a - b - c]$, através da duplicação da atividade a.

Remoção de uma Atividade Aleatória do Trace. Essa abordagem considera a remoção de uma atividade de um *trace* que é instância do modelo utilizado para criar o *log* normal. Por exemplo, o *trace* $[a - b]$ é anômalo ao modelo a) da Figura 1.1, e foi criado a partir do *trace* normal $[a - b - c]$, através da remoção da atividade c.

Inclusão de uma Atividade Aleatória do Log. Essa abordagem considera a inclusão de uma atividade no *trace* dentre as atividades que foram executadas em todo o *log*. Por exemplo, o *trace* $[a - b - x - c]$ é anômalo ao modelo a) da Figura 1.1 e foi criado a partir do *trace* normal $[a - b - c]$, através da inclusão da atividade x, assumindo que a atividade x é uma atividade existente no *log*. No exemplo da Figura 1.1, a atividade $x \in \{a, b, c\}$, já que são as únicas atividades que previsíveis pelo modelo de processo e contidas no conjunto de possíveis *traces*. Essa abordagem de geração de *trace* anômalo pode gerar um *trace* semelhante a um *trace* gerado pela abordagem de duplicação. Esse resultado é obtido quando a atividade sorteada para ser incluída no *trace* coincide com uma das atividades do *trace* original, no exemplo com o *trace* $[a - b - c]$, seriam as atividades a, b ou c.

Troca de Blocos Estruturais AND e OR. Essa abordagem de geração de *traces* anômalos é ilustrada na Figura 1.1. Os possíveis traces do modelo A são anômalos (não são instância) ao modelo B, e vice-versa. Diferente das abordagens anteriores, essa abordagem considera primeiramente a modificação de um modelo que existe (no exemplo, do modelo A para o modelo B), enquanto as outras abordagens consideram o *trace* gerado por um modelo, para então obter um *trace* anômalo.

É importante observar que todas as abordagens de geração dos *traces* anômalos apresentadas acima consideram o mesmo conjunto de atividades dos *traces* normais. Acreditamos que essa estratégia de criação dos *traces* anômalos aumenta o rigor da avaliação, pois caso utilizássemos atividades diferentes das utilizadas nos *traces* normais, os *traces* anômalos seriam muito diferentes dos normais, o que possivelmente simplificaria a detecção desses *traces*. Além disso, é razoável acreditar que em cenários reais um fraudador não tentará executar novas atividades, pois maximizaria a possibilidade de identificação da fraude.

Além disso, as abordagens de geração dos *traces* anômalos corroboram para cenários de anomalia cuja semântica esteja relacionada a *traces* que representam um ruído, normalmente gerado pelos aplicativos responsáveis pelo *log*, ou mesmo uma exceção. Por exemplo, a duplicação e a remoção de atividades podem representar as situações em que o sistema de *log*, por alguma falha, registra mais de uma vez um evento (atividade), ou mesmo não o registra.

1.3.3 Criação dos Logs

A criação dos *logs* é automatizada por duas funções. Uma função é responsável por criar um modelo de processo, enquanto outra é responsável por criar os *traces* ou instâncias a partir de um modelo de processo informado. A criação do modelo de processo deve preceder à criação dos *logs* porque o modelo representa a matriz dos *traces* normais. Assim, uma vez que os *traces* normais são conhecidos, os *traces* anômalos são aqueles que são diferentes das instâncias normais, criados como explicado na Seção 1.3.2, e também não são instâncias do modelo utilizado para criar os *traces* normais.

A função responsável pela criação dos modelos é baseada em cinco parâmetros, como segue:

- o tamanho máximo do *trace* (número máximo de atividades do maior *trace*) que o modelo de processo pode criar (instanciar);
- o tamanho mínimo do *trace* (número mínimo de atividades do menor *trace*) que o modelo de processo pode criar (instanciar);
- o número mínimo de *traces* que o modelo pode criar;
- o número máximo de *traces* que o modelo pode criar[2];

[2]Para os modelos que possuam alguma estrutura de *loop*, as instâncias criadas a partir desses modelos representam no máximo duas iterações. Essa restrição evita o problema relatado por Aalst et al. em [44], relacionado a possibilidade do modelo gerar infinitas instâncias ao tentar gerar um *log* completo.

- a quantidade de modelos que devem ser criados que satisfaçam os parâmetros fornecidos acima.

É importante observar que apesar da geração dos modelos ser orientada por parâmetros que nós fornecemos, e os *traces* anômalos adicionados aos *logs* serem identificados, nenhuma dessas informações é conhecida pelos algoritmos de detecção. Portanto, nem os modelos dinamicamente criados para gerar os *traces* normais, nem os *traces* anômalos, criados a partir dos *traces* normais e do modelo de processo, são conhecidos pelos algoritmos de detecção.

Os modelos de processo são criados através da combinação aleatória de blocos de construção (AND, OR, LOOP e atividade), pois são escolhidos por sorteio em cada trecho da construção do modelo. Por exemplo, a Tabela 1.1 ilustra como o algoritmo de geração/construção de modelos constrói o modelo $[a, or([or([], [b])], [c]), d]$, ou seja, os passos da geração do modelo, cujo gráfico equivalente é exibido na Figura 1.2.

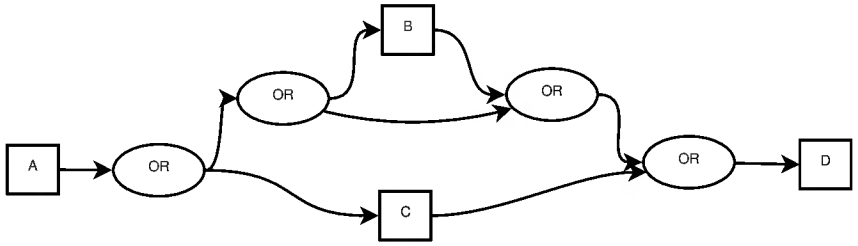


Figura 1.2: Representação gráfica do modelo [a, or([or([], [b])], [c]), d]

Passo (descrição)	**Efeito**
Sorteado adicionar uma atividade	[a]
Sorteado adicionar um bloco OR	[a, or(**[?]**, [?])]
O bloco OR tem dois ramos, que são outros dois modelos	
Sorteado adicionar um bloco OR	[a, or([or(**[?]**, [?])], [?])]
O bloco OR tem dois ramos, que são outros dois modelos	
Sorteado adicionar uma transição vazia	[a, or([or([], **[?]**)], [?])]
Sorteado adicionar uma atividade	[a, or([or([], [b])], **[?]**)]
Sorteado adicionar uma atividade	[a, or([or([], [b])], [c])]
Sorteado adicionar uma atividade	[a, or([or([], [b])], [c]), d]

Tabela 1.1: Exemplo de criação de um modelo de processo

É possível observar na tabela que o modelo é construído através da adição de elementos (atividades ou blocos). Sempre que um bloco é adicionado, é necessário construir seus ramos. No exemplo ilustrado na tabela, o próximo ramo a ser construído é identificado

com um sinal de interogação em negrito. Para o modelo de processo $[a, or([or([], [b])], d)]$, os *traces* com tamanho máximo que podem ser gerados são $[a, b, d]$ e $[a, c, d]$. Além disso, esse modelo também pode gerar no máximo os seguintes *traces*: (i) $[a, b, d]$, (ii) $[a, c, d]$ e $(iii)[a, d]$.

Uma vez que tem-se o modelo de processo, gerado pela função de criação de modelos, um conjunto de *traces* normais e anômalos são gerados. Como a distribuição da frequência dos *traces* contidos no *log* é variável para cada classe de *trace* (normal ou anômalo), a função de criação do *log* inicialmente popula aleatoriamente os blocos OR e LOOP do modelo criado com frequências que indicam a chance de um determinado caminho ser executado, enquanto que para os blocos AND todas as variações possíveis de combinação das atividades do bloco tem a mesma chance de ocorrer, mas uma será sorteada durante a geração do *trace*. A Figura 1.3 é uma variação da Figura 1.2, mas com a definição das probabilidades nos blocos OR. Nessa figura é possível observar a chance que os *traces*, que podem ser criados a partir do modelo sem probabilidades, têm de serem criados em um *log* pelo modelo com probabilidades. Por exemplo, o *trace* $[a, d]$ tem $10,5\%$ de chance de ser adicionado em um *log* normal.

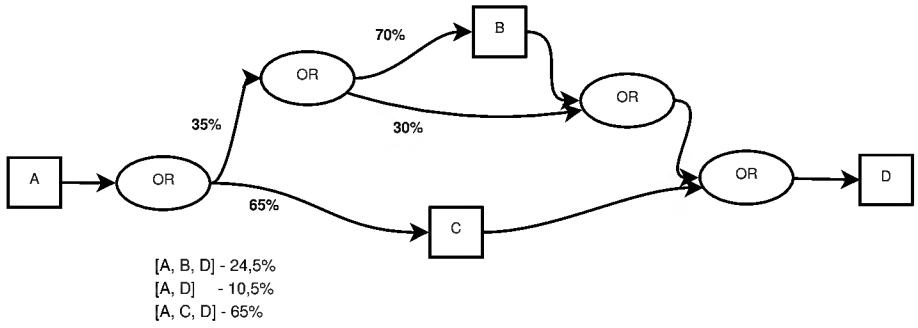


Figura 1.3: Modelo [a, or([or([], [b])], [c]), d] enriquecido com probabilidades.

Assim, os logs utilizados nesta tese para avaliação dos algoritmos combinam os traces normais, com distribuição não uniforme das classes de trace que podem ser instanciadas pelo modelo, e traces anômalos, que são criados a partir das abordagens apresentadas na Seção 1.3.2.

1.4 Roteiro

Esta tese está organizada em seis capítulos, incluindo esta introdução. No Capítulo 2 apresentamos uma revisão bibliográfica de trabalhos relacionados com o problema explorado nesta tese. Optamos por organizar esta revisão em três seções, como segue: na Seção 2.1 apresentamos alguns trabalhos que já exploraram o problema da detecção de eventos

anômalos, mas em outros contextos de aplicação; na Seção 2.2 apresentamos trabalhos relacionados a mineração de processos de negócios (*process mining*), ferramenta muito utilizada pelas abordagens de detecção propostas nesta tese, mas que desde sua concepção tinha como principal objetivo estudar o comportamento normal dos processos, enquanto neste trabalho é utilizada para identificar o comportamento anormal; finalmente, na Seção 2.3 apresentamos trabalhos relacionados a ferramentas e algoritmos de análise de processos de negócios, que também foram assessórias as abordagens de detecção exploradas neste trabalho.

No Capítulo 3 apresentaremos duas definições para *trace* anômalo: **(i)** uma definição utilizada pela abordagem de detecção baseada no grau de modificação de um modelo de processo de negócio, apresentada na Seção 3.3, e **(ii)** uma definição utilizada pela abordagem de detecção baseada na seleção do modelo de processo mais apropriado, apresentada na Seção 3.4. O objetivo desse capítulo é apresentar um referencial teórico para a concepção dos algoritmos e abordagens de detecção de anomalia desenvolvidas nesta tese.

Nos Capítulos 4 e 5, que são os capítulos centrais deste trabalho, apresentamos as duas abordagens de detecção de *traces* anômalos em *logs* de sistemas orientados a processos de negócio desenvolvidas nesta tese. A primeira abordagem de detecção, detalhada no Capítulo 4, classifica um *trace* como anômalo se este *trace*, para ser instância de um modelo de processo, necessita que o modelo acomode/implemente muitas alterações de sua definição. A solução de detecção com melhor desempenho quanto a capacidade de classificar corretamente um *trace* como normal ou anômalo foi a abordagem de *sampling* baseado no algoritmo de mineração incremental. Entretanto, essa solução é inadequada em cenários reais, pois a utilidade do algoritmo de mineração incremental é limitada a *logs* com pouca variação de classe e *traces* de no máximo 10 atividades. Então, um algoritmo semelhante foi concebido, mas baseado em ferramentas de mineração mais robustas. Apresentamos na Seção 4.3 uma análise exaustiva de algoritmos que seguem esse modelo de detecção.

A segunda abordagem de detecção, detalhada no Capítulo 5, considera a busca por um modelo de processo chamado *modelo de processo apropriado*. Então, os *traces* anômalos são os *traces* que não são instância desse modelo, denominado de modelo apropriado. No caso da segunda abordagem de detecção, foi conduzido um estudo com um *log* real, e os resultados são apresentados na Seção 5.4.

Finalmente, no Capítulo 6 apresentamos as conclusões sobre os estudos empíricos realizados com as diferentes variações de soluções de detecção de anomalias, bem como sugestões de trabalhos futuros (Seção 6.2).

Capítulo 2

Trabalhos Correlatos

Apresentamos na Seção 2.1 algumas propostas de algoritmos utilizados para detectar eventos anômalos em diferentes áreas de aplicação. Este capítulo não tem o objetivo de listar todos os trabalhos relacionados à detecção de anomalia, mas sim o de apresentar uma visão geral do quanto investigar e detectar ocorrências de eventos anormais é importante para a área de segurança e *data mining*. No entanto, esta seção também será importante para indicar que no contexto de processos de negócios há pouca contribuição da comunidade científica, o que representa uma grande oportunidade para o desenvolvimento de contribuições inovadoras na área. Por exemplo, a comunidade de *process mining* historicamente dedicou maior interesse em estudar o comportamento comum ou normal dos processos de negócios, ao contrário desta tese, que está interessada em identificar aquelas ocorrências de instância de processo que são incomuns ou anômalas, como denominamos.

No contexto deste trabalho, *process mining* é utilizado como uma ferramenta de apoio à classificação de um *trace* como anômalo ou normal, pois o modelo gerado por um algoritmo de mineração pode apoiar o processo de classificação de um *trace* do *log*. Portanto, entendemos que a eficácia da mineração, medida através da capacidade de construir modelos de processo que descrevem bem um *log*, tem influência na eficácia da classificação dos *traces* como anômalos ou normais. Assim, apresentamos trabalhos relacionados à mineração de processos (*process discovery*) na Seção 2.2.

Na Seção 2.3 apresentamos algumas abordagens utilizadas para avaliar processos de negócios e *logs*, por exemplo, o grau de generalidade (capacidade de prever instâncias não observadas no *log*) e o grau de especificidade (capacidade representar apenas o comportamento observado no *log*). As ferramentas de avaliação são importantes porque são utilizadas pelos métodos de detecção, por exemplo, para medir o grau de conformidade entre um *log* e um modelo de processo.

2.1 Detecção de Eventos Anômalos

O desenvolvimento de métodos de detecção de eventos anômalos tem despertado o interesse da comunidade acadêmica há vários anos, especialmente das comunidades de *data mining* e segurança. Por exemplo, Donoho, em [21], apresenta como as técnicas de mineração de dados podem ser utilizadas para detectar antecipadamente fraudes relacionadas ao uso de informações sigilosas sobre empresas negociadas no mercado de ações. Um trabalho menos recente, apresentado em [23], mostra como as fraudes relacionadas à clonagem de celulares podem ser detectadas. Na área de comércio eletrônico e leilões existem soluções relacionadas a detecção de bandidos ou fraudadores (ex.: ver em [32]). Na área de segurança há soluções relacionadas a detecção de intrusão em redes de computadores (ex.: ver em [28] e [31]). Em [1], o autor descreve um método de detecção de epidemias a partir do *log* de emergência hospitalar. De forma semelhante, em [37] os autores apresentam um método de detecção de epidemias a partir de dados sobre a venda de medicamentos em farmácias.

No contexto mais específico das redes de computadores e internet, há o trabalho de Patcha e Park[33], onde os autores apresentam diferentes técnicas de detecção de intrusão em redes de computadores. Diferente das abordagens mais comuns de detecção de intrusão, normalmente representada como regras conhecidas de ataque, a detecção de anomalia modela o comportamento normal e é capaz prever novas abordagens de ataque.

Chandola et al. em [12], apresentam uma classificação para os diferentes métodos de detecção de anomalia desenvolvidos para diversas áreas de aplicação. Nesta classificação, cada classe ou grupo de métodos de detecção possui uma suposição ou definição comum do que significa um *evento* anômalo e um *evento* normal. Além disso, os autores apresentaram um modelo de técnica de detecção comum a cada classe, onde os métodos de detecção da mesma classe representam uma extensão desse modelo comum de detecção.

Apesar de existirem várias soluções relacionadas à detecção de eventos anômalos em dados, menos atenção tem sido dada pela comunidade acadêmica no contexto de sistemas de apoio ao processo de negócios. A grande maioria dos trabalhos está mais preocupada em investigar o comportamento comum ou normal dos processos, do que entender ou identificar as ocorrências anormais.

No entanto, podemos citar o trabalho de Aalst e Medeiros, em [43], entre as contribuições fortemente relacionadas a esta tese e a área de *process mining* que identificamos na literatura. Nesse trabalho os autores apresentam dois métodos de detecção apoiados pelo α-algoritmo [46]. Os métodos propostos nesse trabalho consideram que um *log* formado por *traces* "normais" é conhecido e então minerado para definir um classificador. Esse classificador é utilizado para analisar o *log* que será auditado, a fim de identificar os *traces* que caracterizam uma ocorrência anômala. Entretanto, há claramente uma limitação, que

é a existência de um *log* "normal" para definir o classificador de *traces* anômalos, pois é muito difícil, ou mesmo impossível, em alguns domínios de aplicação. Outra limitação é a utilização do α-algoritmo, que tem pouca utilidade prática, apesar de ser uma referência teórica para a comunidade de *process mining*. Por exemplo, esse algoritmo de mineração exige que o *log* minerado possua certas propriedades que são complicadas ou impossíveis de serem garantidas em cenários reais. Entre as limitações do α-algoritmo que vale citar: *loops* curtos, *non-free-choice* e dependência implícita entre atividades[46, 16].

2.2 Mineração de Processos

A mineração de processos é uma técnica que visa reconstruir um modelo de processo, com atividades e relações entre atividades, a partir de um *log* gerado por um sistema [44, 20, 46]. Nos últimos 14 anos, a área de *process mining* tem despertado a atenção de vários pesquisadores no mundo. Também denominada *process discovery*, foi concebida inicialmente no contexto de processos de software. Cook e Wolf, em [14], cunharam o termo *process discovery* como uma ferramenta de apoio ao projeto de processos de software, pois a definição de modelos de processo é uma atividade difícil, cara e sujeita a erros, especialmente para processos muito grandes ou complexos. Também nesse trabalho são apresentados três algoritmos de mineração de processos, inspirados no problema da inferência de uma gramática a partir de um conjunto de exemplos de uma linguagem.

O artigo de Agrawal et al., em [2], que é outro trabalho precursor dos trabalhos recentes em *process mining*, apresenta um algoritmo de mineração que gera um modelo de processo que preserva três características: completude, não redundância e minimalidade. Tais características coincidem com a definição de um bom modelo, proposta por Schimm, em [38]. Agrawal et al., em [2], também apresentam uma extensão do seu algoritmo que considera a mineração de *logs* que possuem ruídos gerados pelo registro incorreto de atividades. Essa extensão desconsidera relações de dependência entre atividades cuja frequência da relação no *log* é inferior a um valor indicado, portanto um modelo de processo é gerado sem considerar essas relações, chamadas de relações espúrias.

Mais recentemente, muitos outros algoritmos e problemas da área de *process mining* foram apresentados [44, 46, 45, 38, 25, 16]. Hammori et al., em [25], apresentam a abordagem de mineração de modelos de processos conhecida como interativa, pois consideram uma participação constante do analista responsável pela execução da mineração do *log*, que deve definir um conjunto de parâmetros. Esses parâmetros consideram: *(i)* a definição do tamanho do espaço de busca pelo modelo de processo; *(ii)* a definição do tamanho do modelo de processo que "melhor" descreve o *log*, ou seja, mais específico (maior) ou mais genérico (menor); e *(iii)* a definição da tolerância a ruído, definida através de filtros que excluem relações ou atividades pouco frequentes da mineração.

Dentre os algoritmos de mineração de processo de negócio, o mais difundido ou referenciado pela comunidade de *process mining* é o $\alpha-$algoritmo [44, 46, 45]. A eficácia desse algoritmo foi provada para uma classe de modelos de processos de negócios, SWF-Net (*Structured Workflow Net*), mas possui algumas limitações como a mineração de *loops* curtos (*loops* com uma única atividade), tarefas duplicadas e a relação implícita entre duas atividades.

Algumas extensões para o $\alpha-$algoritmo foram desenvolvidas[52, 15, 22, 18, 53, 54]. Por exemplo, o problema da detecção de tarefas implícitas é resolvido em [54], enquanto que o problema da detecção de tarefas duplicadas é explorado em [18].

Uma discussão sobre a área de *process mining*, suas limitações e uma definição mais crítica do real problema da área é apresentada por Wainer et al., em [51]. Nesse trabalho os autores argumentam que o problema da mineração de processos está mal definido, pois a busca por um modelo que gera todos os *traces* existentes no *log* pode resultar em um número enorme de soluções diferentes. Assim, o problema deveria ser reformulado a fim de acrescentar ao problema da descoberta, a seleção do modelo que "melhor" descreva o *log*. Além disso, nesse artigo os autores propõem um método de mineração de processos incremental que ilustra bem a necessidade de reformulação do problema *process mining*. Diferente da maioria das abordagens anteriores, um modelo de processo é gerado incrementalmente através da junção, *trace* a *trace*, dos *traces* existentes no *log*. Uma descrição mais detalhada desse método será apresentada na Seção 4.1.

Schimm, em [38], apresenta um método de mineração incremental de modelos de processos semelhante ao método apresentado em [51], pois também é baseado em um conjuntos de regras e também gera um modelo de processo bloco estruturado. Esses modelos são chamados de bloco estruturado porque um bloco *split* (AND ou OR) está sempre combinado com um bloco *join*; e um bloco *join*, referente a um bloco *split* mais externo, é aplicado apenas quando todos os blocos *join* mais internos são aplicados, ou seja, os blocos de construção do modelo estão sempre aninhados. Nesse artigo o autor define três propriedades que um modelo de processo deve ter: completude, especificidade e minimalidade. A *completude* significa que o modelo mantém todas as tarefas do *log* e as respectivas relações de dependências. A *especificidade* significa que o modelo não adiciona novas tarefas, nem novas dependências (dependências espúrias). A *minimalidade* significa que o modelo é descrito com o menor número de elementos.

Há também o algoritmo de mineração *Multiphase Miner*, que utiliza EPC (*Event-driven Process Chain*) como linguagem de representação do modelo de processo gerado[50, 49]. Esse algoritmo é executado em duas etapas, por essa razão é chamado de *multiphase*. Na primeira etapa são identificadas as relações binárias de ordem parcial entre as atividades, enquanto que na segunda etapa são identificadas como essas relações de ordem entre as atividades podem ser casadas de modo a utilizar elementos estruturais como

paralelismo e seleção.

Outra técnica de mineração de processos bastante difundida é o *genetic mining* [16, 19]. Essa técnica garante, em teoria, que sempre será gerado um modelo de processo com *fitness* igual a 1, ou seja, em que todos os *traces* do *log* podem ser executados pelo modelo. No entanto, não há garantia de quanto tempo o algoritmo precisará para encontrar esse modelo. O *Genetic mining* é baseado na abordagem de programação de algoritmos genéticos, que são algoritmos que buscam por uma solução (ou indivíduo) utilizando heurísticas similares ao processo de evolução (elitismo, herança e mutação). Uma outra vantagem desse algoritmo de *process mining* é a capacidade de lidar com *logs* incompletos ou com ruídos, que é uma característica típica dos algoritmos genéticos.

Há outros métodos de *process mining* robustos a ruído, ou seja, que consideram a mineração de processos em *logs* com ruídos. No entanto, esses algoritmos são abordagens limitadas ao uso da frequência das relações de dependência entre as atividades no *log*. Assim, trechos infrequentes de um *trace* podem ser desconsiderados do processo de mineração, ou seja, não representados no modelo de processo descoberto [2, 44, 13, 34, 26].

Nesta tese, os algoritmos de detecção propostos utilizaram como parâmetro de entrada uma instância de um algoritmo de mineração (*process discovery*). No caso, cinco algoritmos foram utilizados: (i) o algoritmo de mineração incremental, cuja saída é um modelo bloco estruturado[51, 3]; (ii) o algoritmo *alpha*[46]; duas extensões do algoritmo *alpha*, o (iii) *alpha++*[54] e o (iv) *heuristic miner*[53]; além do algoritmo *(v) multiphase miner*[50].

2.3 Análise de Modelos de Processos

Diferentes métricas e métodos de avaliação de processos foram propostos na literatura, tais como os apresentados em [42, 35, 47, 17, 40, 41]. Aalst, em [42], apresenta dois métodos de análise de processo: um qualitativo, baseado na análise Delta, e outro quantitativo, baseado no teste de conformidade. Rozinat e Aalst, em [35], apresentam diferentes métricas para o teste de conformidade (teste de aderência entre um modelo e um *log*). Medeiros et al., em [17], apresentam as métricas *precision* e *recall* como instrumento de medição da equivalência comportamental entre dois modelos de processo. Por exemplo, tais métricas são utilizadas no algoritmo de mineração *genetic mining* [19], como instrumento de seleção de modelos comportamentalmente mais aproximados.

Um trabalho mais recente, descrito em [40], relata uma nova forma de realizar a auditoria de sistemas, *Auditing 2.0*. Essa nova forma de auditoria é apoiada por ferramentas e técnicas de *process mining*, disponíveis no *ProM*[1]. Dessa forma, os auditores não precisam

[1]Ferramenta de *process mining*, disponível em `http://prom.sf.net`

mais se limitar a analisar apenas um subconjunto limitado dos eventos do *log* (normalmente a auditoria sorteia alguns eventos para analisar), mas o *log* inteiro, pois todo o trabalho de auditoria pode ser automatizado. A análise dos *logs* também pode ajudar a predizer comportamentos do processo (tempo de execução restante) e realizar recomendações (que atividades podem ser evitadas ou executadas para otimizar uma propriedade do processo).

O modelo de processo gerado após a mineração de um *log* depende dos *traces* existentes nesse *log*. Entendemos que essa premissa é importante na definição dos algoritmos de detecção de *traces* anômalos apresentados nesta tese, pois acreditamos que um *log* "contaminado" com *traces* anômalos gera um modelo bastante diferente de um outro modelo quando descoberto sem a presença dos *traces* anômalos. Para medir essa diferença é necessário métricas de avaliação ou análise de modelos de processos. Assim, consideramos o uso das métricas de análise como instrumento de avaliação da conformidade de um modelo com dois *logs*: um *log* que contém um *trace* sob análise, e outro *log* que não contém o *trace* sob análise. Então, uma hipótese adotada nesta tese considera que o grau de conformidade entre um modelo e um *log* é muito menor quando o *log* contém um *trace* anômalo. O Capítulo 4 descreve melhor as métricas de conformidade utilizadas e como essas métricas de conformidade podem ajudar no processo de detecção de *traces* anômalos em *logs*.

Capítulo 3

Definição de Anomalia

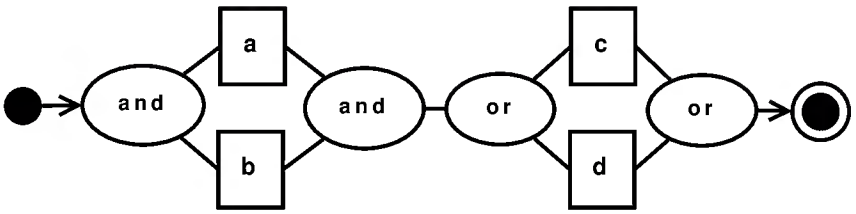
O conceito de anomalia em um *log* pode ser associado a várias semânticas. Por exemplo, uma anomalia pode ser um **ruído**, quando um evento (tarefa) não é registrado ou é registrado em duplicidade, consequência de algum erro no componente de gravação do *log*, ou mesmo um erro transacional da aplicação que usa o componente de gravação no *log*.

Uma anomalia também poderia ser uma exceção, uma imperícia, ou uma tentativa de fraude. A anomalia que tem a semântica de uma **exceção** representa uma execução incomum, mas tolerável pelo negócio, já que em ambientes de negócios flexíveis, não é possível prever todas os caminhos de execução permitidos, portanto recorrentemente há a necessidade de mudança, por exemplo, para acomodar novas estratégias de negócios ou para atender uma necessidade de um cliente muito importante. Nesse caso, identificar a anomalia é importante para conhecer melhor o negócio, conhecer as situações que provocam a exceção, ou mesmo evitar surpresas.

No entanto, as anomalias que tem a semântica de uma tentativa de **fraude** ou uma **imperícia** são incomuns e produzem prejuízos ou resultados indesejáveis para o negócio, sendo imperativo identificá-las.

3.1 Apresentação

A fronteira entre as diferentes semânticas associadas a uma anomalia pode não ser muito clara. Por exemplo, considere o modelo de processo hospitalar apresentado na Figura 3.1. Esse exemplo considera o processo de tratamento de pacientes com insuficiência renal, ou seja, quando os rins param de funcionar. Para substituir a função dos rins, dois tratamentos podem ser aplicados: a hemodiálise ou a diálise [30]. Apesar dos tratamentos serem considerados alternativos (bloco OR entre as atividades c e d), é comum preparar os pacientes para ambos os procedimentos (bloco AND entre as atividades a e b), pois



a - Realizar cirurgia de formação de uma fístula artereovenosa
b - Realizar cirurgia de introdução de cateter peritoneal
c - Executar hemodiálise
d - Executar diálise

Figura 3.1: Exemplo de processo de tratamento de pacientes com insuficiência renal.

ambos os tratamentos podem oferecer riscos ou complicações ao paciente. Assim, um paciente preparado para ambos os tratamentos pode modificar o tratamento quando houver necessidade.

Em um tratamento de pacientes com insuficiência renal, a diálise e a hemodiálise não podem ser executadas ao mesmo tempo, pois caracterizaria uma exceção grave (ou mesmo imperícia) com consequências muito danosas ao paciente, como hipotensão arterial, perda de proteínas e outros nutrientes. No entanto, há raríssimos casos em que o paciente passa pelos dois tratamentos (diálise e hemodiálise); por exemplo, o paciente não responde bem a um dos tratamentos e deve ser submetido imediatamente ao outro tratamento. Portanto, como classificar essa execução incomum como uma **imperícia** (erro médico) ou uma **exceção** que exigiu um tratamento alternativo?

Independente da semântica associada à anomalia, é bastante comum considerá-la como um evento raro ou infrequente. Contudo, classificar um *trace* (ou instância de processo) como anômalo baseando-se apenas em sua frequência no *log* não é simples ou é muito ingênuo, pois é provável que alguns *traces* "normais" também sejam infrequentes, ou seja, alguns caminhos de um modelo de processo de negócio podem ser mais exercitados que outros. Por exemplo, não parece apropriado classificar como anômalo todos os *traces* com frequência no *log* inferior a 3% ou 4%, pois é provável que *traces* normais também ocorram com essas frequências no *log*. Por outro lado, uma abordagem baseada apenas na frequência seria muito ingênua, o que poderia distorcer o número de falsos positivos (instâncias normais classificadas como anômalas) e falsos negativos (instâncias anômalas classificadas como normais).

A Figura 3.2 ilustra o problema de classificar um *trace* como anômalo baseado apenas em sua frequência. Tal figura contém um modelo de processo e quatro *logs* (um com todos os possíveis *traces*, o Log 1, e três *logs* com os subconjuntos dos possíveis *traces*, os Logs 2, 3 e 4).

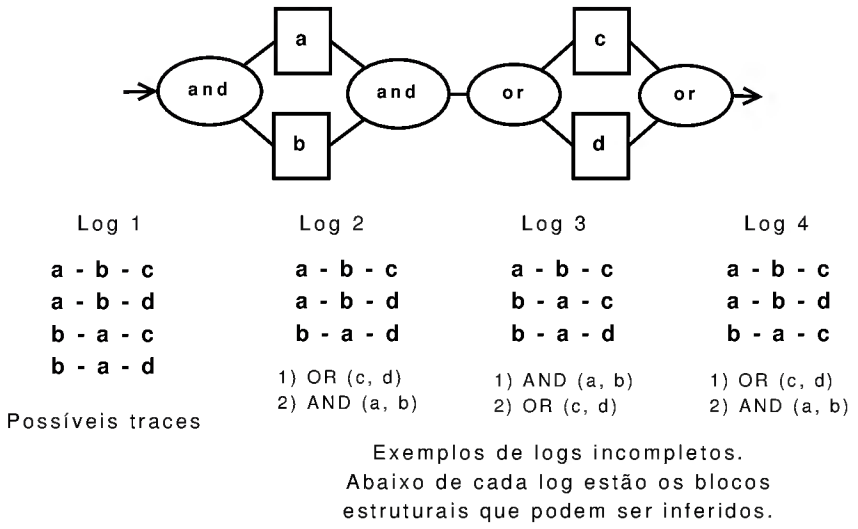


Figura 3.2: Mineração de um modelo com um conjunto incompleto de *traces*.

Na Figura 3.2, um subconjunto dos possíveis *traces* de um modelo pode ser utilizado para reconstruir o mesmo modelo. Por exemplo, no *log* incompleto *Log 2*, os *traces* $[a-b-c]$ e $[a-b-d]$ quando minerados podem gerar o modelo $[a-b-or(c,d)]$ a partir da inclusão do bloco *OR(c, d)*, que indica uma escolha entre as atividades c e d. Em seguida, caso o *trace* $[b-a-d]$ seja adicionado a esse modelo, geraria um novo modelo igual ao original $[and(a,b)-or(c,d)]$, dessa vez através da adição do bloco *AND(a, b)*, que indica que as atividades a e b podem ser executadas em paralelo (a ou b podem ser concluídas antes).

Nesse exemplo, o *trace* $[b-a-c]$, não existente no *log* incompleto (*Log 2*) é também uma instância do modelo gerado, pois existe um caminho no modelo de processo que coincide com o *trace* apresentado, mesmo sem ter sido utilizado para construir o modelo. Portanto, caso o *trace* $[b-a-c]$ pertencesse ao *log* e fosse infrequente, não poderíamos classificá-lo como anômalo. Dessa forma, há claramente a necessidade de considerar outras propriedades, além da frequência, para classificar um *trace* como anômalo.

Este trabalho tem o objetivo de apresentar métodos que ajudem a identificar os *traces* infrequentes que são anomalias. Assim, outras questões, além da frequência, são observadas nos método de detecção de anomalias propostos neste trabalho. Por exemplo, um método de detecção deve considerar a definição de um modelo de processo "normal", que funcionará como um classificador. No entanto, definir esse modelo é muito complicado, pois:

- cada domínio de aplicação exigiria um modelo diferente;
- a definição do modelo normal pode estar desatualizada em um dado instante, resul-

tado de uma evolução natural do modelo normal;

- ou então, a própria instância anômala pode ser, propositalmente, uma aproximação de uma instância normal gerada pelo fraudador.

Portanto, é evidente que os desafios do problema de detecção de anomalia são enormes, já que devem considerar as variadas questões citadas acima. Como apresentado em [12], há várias definições de anomalia, que consideram diferentes pressupostos e dependem da área de aplicação do método de detecção. Então, consideramos que uma definição objetiva de anomalia, mesmo que específica a um domínio ou que satisfaça certas condições e pressupostos, ajudaria na definição de um método ou algoritmo de detecção de anomalias.

As Seções 3.3 e 3.4 apresentam duas definições diferentes para *trace* anômalo, cada uma assumindo pressupostos diferentes. No entanto, há definições preliminares comuns às duas definições de anomalia. Essas definições preliminares serão apresentadas na Seção 3.2. Assim, o objetivo deste capítulo é oferecer um referencial formal para apoiar o desenvolvimento de uma solução objetiva para o problema da detecção de *traces* anômalos em *logs* gerados por Sistemas de Informação orientados a Processos de Negócios.

3.2 Definições Preliminares

O termo *trace*, como inicialmente apresentado na Seção 1.1, será utilizado nesta tese como uma instância de processo, ou seja, como um caminho de execução de um modelo de processo de negócio. Este *trace* representa a ordem em que um conjunto de atividades completou sua execução. Assim, o *trace* $[a\ b\ c\ d\ e]$ indica que a atividade a foi concluída antes da atividade b, que foi concluída antes da atividade c, e assim por diante.

A Definição 1 descreve formalmente o que é um *trace*, enquanto que a Definição 2 descreve formalmente o conceito de *log*, que é o repositório dos eventos (atividades), e por conseguinte dos *traces*, registrados por um sistema de informação.

Definição 1 *Trace.*
Seja A um conjunto de atividades. Um trace t *representa a sequencia de atividades, tal que $t \in A^*$. Ou seja, considerando que A é um alfabeto e A^* denota todas as palavras que podem ser derivadas a partir de A, então o* trace t *é uma palavra baseada neste alfabeto.*

Definição 2 *Log.*
Seja $T \subseteq A^*$ o conjunto dos traces *definido sobre as atividades A. O multiconjunto $L = \{(t', n) \mid t' \in T \ \wedge \ n \in \mathbb{N}^*\}$ é definido como um* log.

Na Definição 2 é importante observar que t' representa uma classe de *trace*, enquanto o valor n é um número natural diferente de zero que indica o número de ocorrências da classe

no *log*, ou seja, a frequência com que a classe t' aparece no *log*. Essa definição baseada em multiconjunto é mais relevante quando considerarmos a utilização de algoritmos de mineração de processos que dependem da frequência dos *traces* no *log*.

No entanto, na prática, o *log* que será submetido a um método de detecção de *traces* anômalos precisa ser filtrado, antes da aplicação do método, para remover aquelas instâncias de processo que são claramente anômalas. Por exemplo, no momento em que um *log* foi importado/extraído para análise, várias instâncias estavam em execução e não foram concluídas, portanto, essas instâncias não devem ser consideradas na detecção das anomalias.

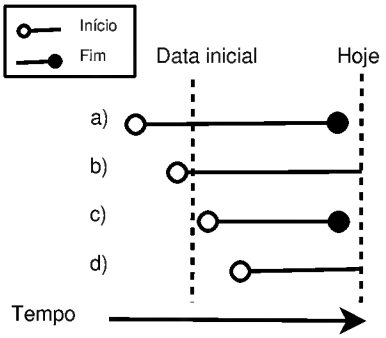


Figura 3.3: Problemas relacionados com um *log* não filtrado.

A Figura 3.3 ilustra os problemas que o analista de domínio precisa resolver antes de aplicar algum método de detecção de *traces* anômalos. Nesse caso, é necessário aplicar alguns filtros no *log* coletado para análise/identificação dos *traces* anômalos. Por exemplo, há quatro *traces* nessa figura, que denominamos a, b, c, e d. Representamos com linhas tracejadas o período de coleta do *log* utilizado para análise, então:

- o *trace* **a** deve ser removido, pois não tem a atividade inicial;
- o *trace* **b** deve ser removido, pois não tem as atividades inicial e final;
- o *trace* **d** deve ser removido, pois não tem a atividade final;
- o *trace* **c** é o único que deve permanecer no *log* filtrado, pois representa um *trace* completo.

A definição formal para *log filtrado* é apresentada a seguir.

Definição 3 *Log Filtrado.*
Sejam:

- *Um* log L *(Definição 2).*

- *Um conjunto A^S de atividades filtradas (scoped), tal que $A^S \subseteq A$.*
- *Uma função* filter(t, A^S) *que remove todas as atividades em um* trace *t que não estão em A^S.*
- *Uma função booleana* complete(t) *que retorna* false *se t é um* trace *incompleto e* true *quando o trace é completo, ou seja, possui as atividades inicial e final (trace c da Figura 3.3).*

Então um log *filtrado $L^S \subseteq L$ é um multiconjunto dos* traces *t baseados nas atividades em A^S, como segue:*

$$L^S = \{filter(t, A^S) \mid t \in L \wedge complete(t)\}$$

Dizemos que um *trace* do *log* tem capacidade de ser executado completamente pelo modelo quando há no modelo um caminho de execução, do início ao fim, igual ao *trace* sob análise. Essa capacidade do *trace* é medida através da função que chamamos de *fitness do trace*, cuja definição formal é apresentada na Definição 4. Para medir o grau de *fitness* do *log* inteiro, que indica quantos *traces* observados no *log* podem ser executados completamente pelo modelo, utilizamos a função *fitness do log*, cuja definição formal é apresentada na Definição 5. Dessa forma, um *fitness do log* de 100% significa que o modelo executa o *log* inteiro.

Definição 4 *Fitness do Trace.*
Seja $T = \{t \mid (t, n) \in L\}$ o conjunto dos traces *ou classes dos traces do* log *L. A função $f_M : T \to \mathbb{B}$ é o* fitness do trace *que indica se um* trace *do* log *L é uma instância do modelo M. Então um* trace *t é instância do modelo M se t pode ser totalmente executado pelo modelo M, como segue:*

$$f_M(t) = \begin{cases} true, & \text{se } t \text{ pode ser executado por } M \\ false, & \text{do contrário} \end{cases}$$

Definição 5 *Fitness do Log.*
É uma função $f : \{(M, L) | M \text{ is a model} \wedge L \text{ is a log}\} \to [0, 1]$ que indica o grau de fitness *entre um modelo M e um log L, ou seja, esta função indica quantos* traces *do* log *L podem ser completamente executados pelo modelo M. Esta função é definida como segue:*

$$f(M, L) = \frac{\sum_{\{(t',n') \in L | f_M(t')\}} n'}{\sum_{(t'',n'') \in L} n''}$$

3.3 Trace Anômalo: Definição I

Esta seção considera a apresentação de uma definição de *trace* anômalo baseada no pressuposto de que um *trace* é anômalo quando demanda elevado grau de modificação em um modelo. De outra forma, para um modelo acomodar um *trace* anômalo a sua estrutura sintática ou definição - representada pelas atividades, blocos de repetição, paralelismo ou seleção, etc. - precisa ser excessivamente aumentada para acomodar o *trace* anômalo.

Um algoritmo ou método de detecção de *traces* anômalos deve considerar que qualquer instância existente no *log* pode ser classificada como um *trace* anômalo, ou seja, todas os *traces* do *log* podem ser uma anomalia. Por outro lado, é muito razoável supor que apenas aquelas instâncias infrequentes no *log* são os *traces candidatos* à classificação como anômalos, como exposto na apresentação inicial na Seção 3.1. Assim, apresentamos na Definição 6 o que denominamos de *Trace Candidato a Anômalo.*

Definição 6 *Trace Candidato a Anômalo.*
Sejam:

- *Um* log filtrado L *(Definição 3);*
- *Um conjunto* $C_L = \{c|(c,n) \in L\}$ *das classes de* traces *existentes no* log L*;*
- *Um valor real* $x \in (0,1)$*;*
- *Uma classe* $c \in C_L$ *dos* traces *do* log L*.*
- $s_L = \sum_{(c,n)\in L} n$ *a quantidade de* traces *no* log L*;*
- $f_c = \frac{n}{S_L}$ *a frequência da classe de* trace c *no* log L*.*

Então t^c *é um* trace *candidato a anômalo se pertence ao conjunto* T_C *de* traces *candidatos a anômalo como segue:*

$$T_C = \{c \in C_L \mid f_c \leq x\}$$

Definição 7 *Trace Anômalo'.*
Sejam:

- *Um* log filtrado L *(Definição 3).*
- *Um conjunto* T_C *de* traces *candidatos a anômalo (Definição 6).*
- *Um* log $L' = L - \{(t^c, n)\}$*, que não contém um* trace *candidato a anômalo* $t^c \in T_C$*.*
- *Um modelo* M *minerado a partir do* log L'*.*

- *A função de fitness do log $f(M, L)$ (Definição 5).*

Então t^a é um trace *anômalo se pertence ao conjunto T_A de* traces *anômalos definido como segue:*

$$T_A = \{t^c \in T_C \mid f(M, L) \ll f(M, L')\}$$

A definição acima foi inspirada pela definição de *outlier*, utilizada no campo da estatística, e que significa um valor numericamente *muito distante* do resto dos dados. Essa definição indica que quando o *fitness* do *log* com o *trace* anômalo é muito menor que o *fitness* do *log* sem o *trace* anômalo, então a estrutura do modelo M precisaria ser mais complexa (blocos estruturais AND, OR, etc.) para acomodar o *trace* anômalo. Portanto, tentar acomodar um *trace* anômalo em um modelo de processo "normal", ou seja, minerado sem o *trace* anômalo, requererá muitas modificações no modelo. Este trabalho desenvolveu várias abordagens de detecção orientadas pela Definição 7, que serão apresentadas em detalhes no Capítulo 4.

3.4 Trace Anômalo: Definição II

Esta seção considera a apresentação de uma definição de *trace* anômalo baseada no pressuposto de que um *trace* é anômalo quando não é instância de um **modelo apropriado**. Portanto, a definição formal de *trace* anômalo depende da definição do que seria um modelo apropriado.

Consideramos que um modelo é apropriado quando satisfaz um valor mínimo da função de *fitness* do *log*(Definição 5), mas maximiza uma função que denominamos *appropriateness* (Definição 8). Nesse caso, o valor mínimo de *fitness* é um parâmetro que seleciona alguns modelos, dentre os vários (possivelmente infinito) que podem ser descobertos a partir do mesmo *log*. Por exemplo, no mínimo é possível termos tantos modelos para um *log* quantos algoritmos de mineração de processo existirem, se considerarmos que esses algoritmos não encontram/geram o mesmo modelo para o *log*.

Para ilustrar melhor o problema de encontrarmos o modelo apropriado, considere a função de *fitness* do *log* (Definição 5), que indica o quanto do *log* pode ser completamente executado pelo modelo. Então um *fitness* de 100% indica que o modelo pode executar por completo todos os *traces* do *log*.

Entretanto, um modelo com 100% de *fitness* ainda não é necessariamente um modelo apropriado, por exemplo, o modelo genérico da Figura 3.4 pode executar por completo qualquer *trace* definido com as atividades {A, B, C, D}. Assim, o modelo genérico nunca será capaz de detectar as instâncias anômalos do *log* baseadas nessas atividades. Um outro exemplo seria o modelo apresentado na Figura 3.5, que para um *log* $L = \{[a, b, c], [a, c, d], [a, c, b]\}$, com três classes de *trace*, seria um modelo com *fitness* de

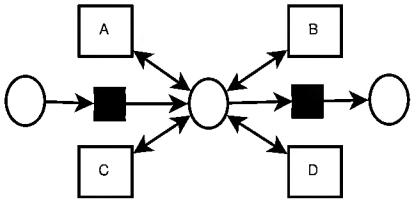


Figura 3.4: Exemplo de modelo simples e muito genérico.

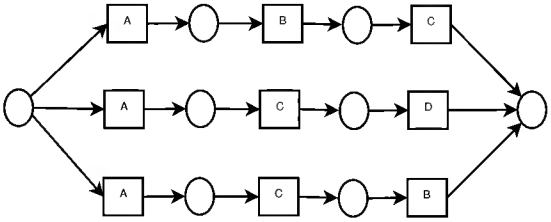


Figura 3.5: Exemplo de modelo complexo e específico.

100%, mas específico (ou exclusivo) para o *log*, pois classificaria como anômalo todas os *traces* não observado no *log*.

O modelo da Figura 3.4 tem a vantagem de ser simples, pois é definido com um número pequeno de construtores (vértices e arestas), também possui a desvantagem de ser muito genérico. Enquanto, o modelo da Figura 3.5 tem a desvantagem de ser muito complexo e muito específico.

Nesse contexto, a função *appropriateness* torna-se fundamental porque ajuda-nos a escolher, dentre os modelos com alto valor de *fitness*, aquele que é mais apropriado, ou seja, aquele que descreve o *log* com um modelo que represente um balanço entre complexidade e especificidade (um modelo simples e específico). **Simples** porque utiliza o menor número de blocos de construção do modelo (AND, OR, LOOP, etc.), mas **específico** porque uma solução como a do modelo genérico nunca será capaz de detectar *traces* anômalos. Na Definição 8 apresentamos formalmente o que é a função *appropriateness*.

Definição 8 *Função Appropriateness.*
É a função $a : \{(M, L) | M$ *é um modelo* $\wedge\ L$ *é um log*$\} \rightarrow [0, 1]$ *que indica quão apropriado é um modelo* M *quando comparado com um* log L. *O valor 0 indica que o modelo não é apropriado, enquanto o valor 1 indica que o modelo é totalmente apropriado.*

Portanto, o termo **apropriado** significa que um modelo simples é preferível a um modelo complexo, e que adicionar muito comportamento ao modelo (generalizá-lo) é indesejável. Assim, o objetivo da função *appropriateness* é representar um balanço entre complexidade estrutural do modelo e a capacidade de executar *traces* não observados no *log*.

Uma vez que temos uma função que calcula o quão apropriado é um modelo quando comparado a um *log*, podemos definir o que é um *trace* anômalo. Assim, apresentamos formalmente na Definição 9 que um *trace* anômalo é um *trace* de um *log* (filtrado), que não é instância do modelo mais apropriado, dentre todos os modelos descobertos a partir desse *log*.

Definição 9 *Trace Anômalo".*
Sejam:

- *O* log filtrado L *(Definição 3).*
- *O grau* $p \in [0,1]$ *que indica o* fitness *mínimo desejável em um modelo quando comparado com um* log.
- *A função de fitness do trace* $f_M(t)$ *(Definição 4).*
- *A função de fitness do log* $f(M,L)$ *(Definição 5).*
- *O modelo apropriado* M^**, onde:*
  - $\exists M \ f(M^*, L) \geq p$*; e*
  - $\forall M' \ f(M', L) \geq p \Rightarrow a(M', L) \leq a(M^*, L)$.

Então t^a *é um* trace *anômalo se pertence ao conjunto* T_A *de* traces *anômalos definido como segue:*

$$T_A = \{t^a \mid (t^a, n) \in L \ \wedge \ \neg f_{M^*}(t^a)\}$$

Capítulo 4

Detecção de Anomalias: Grau de Modificação do Modelo

Neste capítulo apresentaremos três métodos de detecção de *traces* anômalos. Todos foram orientados pela definição apresentada na Seção 3.3 do Capítulo 3. A inspiração para todos os métodos que serão apresentados neste capítulo é o trabalho de Wainer et al. [51]. Nesse trabalho, Wainer et al. argumentam que para um dado *log* sempre é possível encontrar/minerar diferentes modelos que o descrevem. Assim, através desse artigo pudemos entender que o problema da detecção seria o quanto a presença de um *trace* anômalo "perturba" ou influencia no resultado da mineração de um modelo de processo, já que qualquer combinação de *traces* pode gerar um modelo. Dito de outra forma, percebemos que poderíamos detectar um *trace* anômalo ao medir o grau de modificação desse *trace* ao incrementar a definição de um modelo de processo.

Na Seção 4.1 apresentamos o algoritmo de *process mining* incremental proposto por Wainer et al. em [51] e estendido em [3]. Esse algoritmo de mineração de processo e a métrica de custo de inclusão, que será descrita na Seção 4.1.2, foram fundamentais para elaborar as primeiras experiências com o projeto do algoritmo de detecção baseado no *sampling* do *log*, apresentado na Seção 4.1.3, que então serviu de base para definição de outros algoritmos, os algoritmos *Threshold* (Seção 4.1.4) e Iterativo (Seção 4.1.5), que são esforços de tornar a detecção mais eficiente e eficaz.

Nas Seções 4.2.3, 4.2.4 e 4.2.5 apresentamos outros algoritmos que consideram novas estratégias para detectar um *trace* anômalo. No caso da versão estendida dos algoritmos *Threshold* e Iterativo, diferente da abordagem baseada no *sampling*, há uma tolerância para a conformidade entre o modelo e o *log*, então os *traces* anômalos são aqueles em que o grau de conformidade é maior que esse limiar (*threshold*).

4.1 Mining Incremental: Em Direção a Detecção de Anomalia

Em [51], os autores apresentam um método incremental de mineração de processos de negócios. Esse método aplica um conjunto de heurísticas para reescrever um modelo a cada junção com um novo *trace* do *log*, ou seja, o modelo resultante final é definido a partir da mudança incremental de um modelo inicial, que é sequencial e definido a partir da leitura do primeiro *trace*.

As regras (ou heurísticas) utilizadas pelo algoritmo são classificadas como *estruturais* e de *introdução*. As **regras estruturais** são regras que não modificam o modelo, mas são utilizadas para especificar onde as regras de introdução serão aplicadas; enquanto que as **regras de introdução** são as regras que adicionam os blocos estruturais ao modelo – AND Split, AND Join, OR Split e OR Join. A seguir apresentamos a notação, a definição das regras estruturais e a definição das regras de introdução como apresentadas em [51], e adaptada em [3].

4.1.1 Regras de Definição do Modelo

Para definição das regras estruturais e de introdução utilizaremos a notação apresentada a seguir:

- $\alpha, \beta, \gamma, \delta, \ldots$ representam uma sequência de atividades, blocos estruturais (AND, OR, LOOP) ou uma combinação de atividades e blocos estruturais (um submodelo);
- $\alpha\beta$ representa uma concatenação de sequências ou submodelos, ou seja, que α é seguido por β;
- $\alpha + \beta$ representa uma seleção, ou seja, OR-split/OR-join entre os sequências ou submodelos α e β;
- $\alpha \parallel \beta$ representa um paralelismo, ou seja, AND-split/AND-join entre as sequências ou submodelos α e β;
- $\top\alpha$ representa uma iteração da sequência α;
- $\oplus$ é o operador de junção entre um modelo e um *trace*, assim $M \oplus \alpha$ denota a junção entre o modelo M e o *trace* α;
- se α é um modelo, então α' é uma instância de α.

As **regras estruturais** são utilizadas pelo *mining* incremental para definir o local de modificação da estrutura do modelo. Elas são definidas como segue:

S1 $\alpha \oplus \alpha' \Rightarrow \alpha$ Não há necessidade de alterar o modelo quando o *trace* utilizado na junção for uma instância desse modelo.

S2 $\alpha\beta \oplus \gamma\delta \Rightarrow (\alpha \oplus \gamma)(\beta \oplus \delta)$ Segmentação do modelo e do *trace* a fim explorar uma alternativa de junção, ou seja, buscar segmentos do modelo $\alpha\beta$ que podem acomodar uma instância $\gamma\delta$.

S3 $\alpha + \beta \oplus \gamma \Rightarrow (\alpha \oplus \gamma) + \beta$ Indica uma alternativa de junção de um *trace* com um dos ramos de um bloco OR.

S4 $\top\alpha \oplus \gamma \Rightarrow \top(\alpha \oplus \gamma)$ Junção de um *trace* com a definição interna de um bloco LOOP;

S5 $\alpha \parallel \beta \oplus \gamma\delta \Rightarrow (\alpha \oplus \gamma) \parallel (\beta \oplus \delta)$ Indica uma alternativa de junção de um *trace* com os ramos de um bloco AND.

As **regras de introdução** são utilizadas pelo *mining* incremental para aplicar ou introduzir um bloco ou estrutura no modelo. Elas são definidas como segue:

Or-I $\alpha \oplus \beta \Rightarrow \alpha + \beta$ É sempre possível construir um modelo que representa uma seleção entre os *traces* de um *log*, ou seja, um bloco OR entre as classes de *trace* de um *log*. De outra maneira, um bloco OR é definido quando dois *traces*, ou um modelo e um *trace*, não podem ser unidos (mesclados).

And-I $\alpha\beta \oplus \beta'\alpha' \Rightarrow \alpha \parallel \beta$ Regra utilizada para gerar o operador AND. Assim, o *trace* $\beta'\alpha'$ é uma instância invertida do modelo representado pela seqüência $\alpha\beta$, o que sugere um paralelismo entre as α e β.

And2-I $(\alpha \parallel \beta)\gamma \oplus \alpha'\gamma'\beta' \Rightarrow (\alpha\gamma) \parallel \beta$ Regra utilizada para introduzir a seqüência γ em um ramo de um bloco AND, preservando o paralelismo entre as seqüências α e β, além da relação de dependência $\alpha \rightarrow \gamma$.

And3-I $\gamma(\alpha \parallel \beta) \oplus \alpha'\gamma'\beta' \Rightarrow \alpha \parallel (\gamma\beta)$ Regra utilizada para introduzir a seqüência γ em um ramo de um bloco AND, preservando o paralelismo entre as seqüências α e β, além da relação de dependência $\gamma \rightarrow \beta$.

Loop-I $\alpha \oplus \alpha'\alpha' \Rightarrow \top\alpha$ Regra utilizada para gerar um operador de LOOP.

As regras de reescrita apresentadas podem gerar uma grande variedade de modelos, o que compromete sua aplicação em cenários reais. O número de modelos gerados em cada combinação de regras, modelos e *traces* consome muita memória, pela própria natureza recursiva de aplicação das regras. Assim, vale citar que esse problema é consequência de três fatores:

(i) a **diversidade de regras** que podem ser aplicadas para gerar um modelo, pois quanto mais regras tivermos, maior será o número de modelos que podem ser gerados/construídos;

(ii) a **ordem de aplicação das regras**, pois cada sequência de aplicação das regras (derivações do modelo) define novos modelos, que serão entradas diferentes para as próximas sucessivas junções;

(iii) a **ordem de junção dos *traces***, pois dependendo de quais *traces* são juntados, um grupo de regras poderá ser aplicado definindo modelos que não seriam criados se outros *traces* tivessem sido considerado na junção.

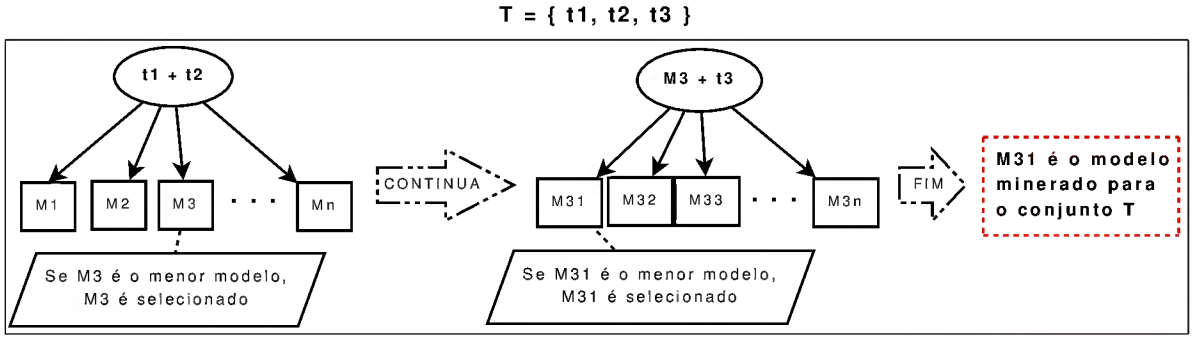


Figura 4.1: Busca gulosa pelo menor modelo.

Para minimizar esses problemas, nós propomos uma busca gulosa pelo modelo de menor tamanho [3]. A Figura 4.1 é um esquema simplificado de como funciona a busca gulosa para um *log* T com três *traces*. Dentre os modelos gerados durante a junção de um modelo com um *trace*, escolhe-se aquele que possui menor tamanho para a próxima iteração do processo de mineração, que termina quando todos os *traces* são selecionados para serem unidos ao modelo. Para o problema da ordem de junção dos *traces*, nós sempre realizamos a junção considerando a ordem lexicográfica dos *traces*, considerando o *trace* como uma palavra.

Veja na Figura 4.2 como o algoritmo de mineração incremental funciona. Durante a mineração, inicialmente um modelo sequencial ($[a - b - c - d]$) de tamanho quatro (quatro vértices) é gerado a partir do primeiro *trace*. Em seguida, o segundo *trace* é adicionado ao modelo ($abcd \oplus abce$) e um novo modelo M de tamanho sete (sete vértices) é gerado ($[a - b - c - or(d, e)]$). Por fim, o terceiro *trace* é adicionado ao modelo M ($[a - b - c - or(d, e)] \oplus acbd$) e o modelo final de tamanho nove (nove vértices) é gerado ($[a - and(b, c) - or(d, e)]$).

Entendemos que o algoritmo de mineração apresentado é um componente dos métodos de detecção dos *traces* anômalos de um *log* que serão apresentados nesta seção.

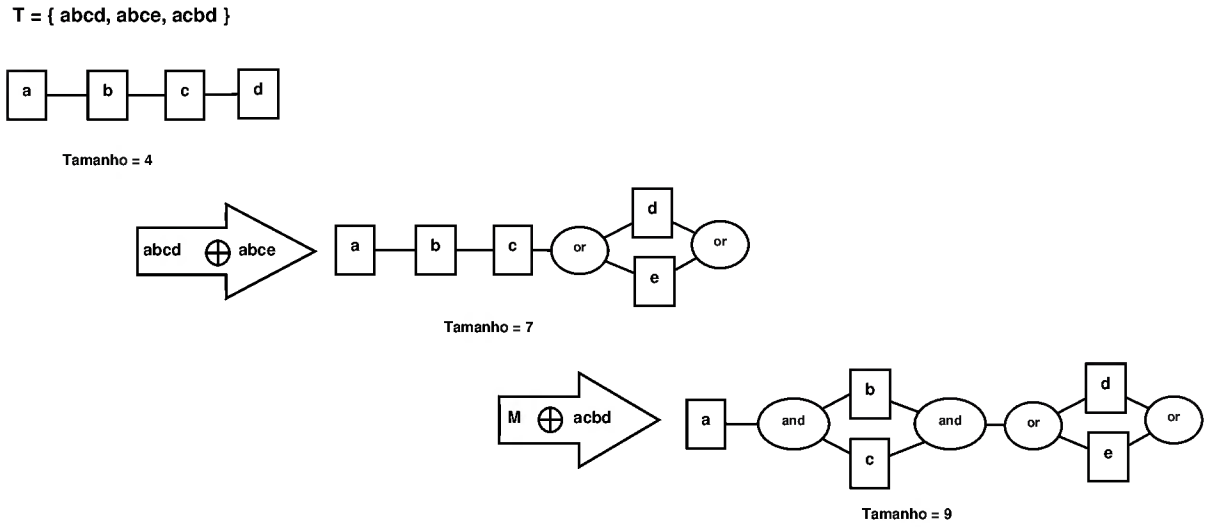


Figura 4.2: Exemplo da mineração de um log com três *traces*.

Outro componente dos métodos de detecção é a métrica de custo de inclusão, que será apresentada na Seção 4.1.2.

4.1.2 Custo de inclusão

Nós chamamos de **custo de inclusão** a métrica que indica o número de alterações realizadas na definição ou estrutura do modelo de processo de negócios após a junção de um *trace* a esse modelo. Essa modificação é realizada a fim de gerar um novo modelo capaz de instanciar o *trace* adicionado. Assim, um valor alto de custo de inclusão indica que para o novo modelo ser capaz de instanciar o *trace* adicionado, muitas alterações estruturais necessitam ser feitas no modelo antigo. Enquanto que um custo de inclusão zero indica que nenhuma modificação é necessária, e que o *trace* adicionado já é uma instância do modelo.

O custo de inclusão considera que o modelo de processo seja representado como um grafo, conforme Definição 10 (ex. Figura 3.2, página 19). O custo de inclusão é calculado a partir da variação do tamanho entre dois modelos, que neste trabalho definimos tamanho como sendo o número de vértices do modelo, conforme Definição 11.

Definição 10 *Modelo de Processo.*
Um modelo de processo é um grafo orientado e sem ciclos $M = (V(M), E(M))$*, tal que os vértices* $V(M)$ *representam as tarefas ou os blocos AND, OR ou LOOP (split e join) de construção do modelo, enquanto que as arestas* $E(M)$ *representam a ordem de execução de seus vértices.*

O grafo que representa o modelo de processo da Definição 10 não possui ciclos mesmo para as estruturas de repetição, pois tais estruturas são representadas como um bloco *LOOP*, semelhante aos blocos AND e OR.

Nós entendemos que a Definição 11 é válida apenas para modelos bloco-estruturados, como os gerados pelo algoritmo de mineração apresentado em [51, 3]. A semântica dos modelos bloco-estruturados, representada pelo número de classes de *traces* que o modelo pode gerar, é função da forma como os blocos semânticos (paralelismo, seleção, iteração e sequência), representados por **vértices** nesses modelos, são combinados. Por outro lado, é importante observar que em outras representações de um modelo, como as redes de *petri*, a semântica do modelo é função dos vértices (*places* ou transições), das arestas e como estas estão ligadas aos vértices, o que exigiria uma outra definição de tamanho de um modelo para essas outras representações (ver Seção 4.2).

Definição 11 *Custo de Inclusão.*
Sejam:

- *Um conjunto T das classes de trace de um log;*
- *Um modelo $M_1 = (V(M_1), E(M_1))$ minerado a partir dos traces em $T - \{t\}$;*
- *Um modelo $M_2 = (V(M_2), E(M_2))$ minerado a partir dos traces em T;*

Então, o custo de inclusão do c do trace t no modelo M_1 é calculado como segue:

$$c = |V(M_2)| - |V(M_1)|$$

Os primeiros projetos de um algoritmo de detecção elaborados nesta tese consideraram o cálculo do custo de inclusão como elemento central do processo. Assim, as abordagens de detecção de anomalia baseadas no custo de inclusão adotaram a seguinte hipótese para identificar os *traces* anômalos no *log*: *traces* que não são anomalias, quando minerados, definem um modelo e tentar realizar a junção de um *trace* anômalo a esse modelo requerirá muitas modificações estruturais. Entretanto, entendemos que a junção de um *trace* não anômalo também pode requerer modificações, mas possivelmente em uma escala menor que as modificações geradas com a junção de um *trace* anômalo.

A Figura 4.3 exemplifica o processo de cálculo do custo de inclusão de um *trace*. Um pré-processamento inicial é realizado a fim de gerar um conjunto com as diferentes classes de *trace* do *log*. Esse pré-processamento é importante porque o resultado do algoritmo de mineração incremental não depende da frequência das classes de *trace* no *log*.

Em seguida, no **primeiro passo**, o *trace-2* é separado/escolhido. No **segundo passo**, os *traces* remanescentes são minerados produzindo um modelo $M_1 = (V(M_1), E(M_1))$ com tamanho $|V(M_1)|$, representado pelo valor X. No **terceiro passo** realiza-se a junção do

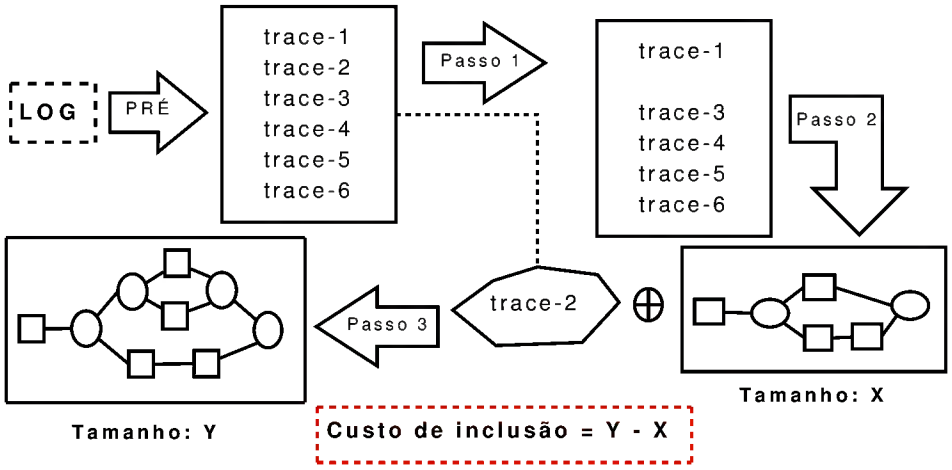


Figura 4.3: Exemplo do cálculo do custo de inclusão.

modelo gerado no segundo passo com o *trace* separado no primeiro passo. Assim, um novo modelo $M_2 = (V(M_2), E(M_2))$ com tamanho $|V(M_2)|$ é gerado, representado pelo valor Y. Finalmente o custo de inclusão do *trace-2* é calculado ($|Y - X|$) conforme Definição 11.

Nas subseções que seguem descrevemos três algoritmos de detecção de *traces* anômalos inspirados no conceito de custo de inclusão (mudança incremental). Na Seção 4.1.3 apresentamos o algoritmo *sampling*, que considera que uma amostra do *log* provavelmente gerará um modelo "normal", que então será usado para classificar um *trace* sob análise. Na Seção 4.1.4 apresentamos o algoritmo *threshold*, que considera que um *trace* para ser anômalo deve exigir mais modificações no modelo que um dado limite. Finalmente, na Seção 4.1.5 apresentamos o algoritmo iterativo, que é uma variação iterativa do algoritmo *threshold*, tal que em cada iteração apenas o *trace* com maior custo de inclusão é classificado como anômalo.

4.1.3 Projeto Inicial do Algoritmo *Sampling*

O algoritmo *Sampling*, inicialmente apresentado em [3], considera que um *trace* é anômalo quando não é instância de um modelo de processo construído com um *sampling* do *log* (amostra). Assim, um *trace* sob análise é anômalo se necessitar que um modelo, gerado a partir de um *sampling* do *log*, sofra modificações estruturais a fim de gerar um novo modelo capaz de instanciar esse *trace* sob análise.

A ideia do *sampling* é obter uma amostra do *log* que seja uma boa representação das instâncias normais, assim podemos assumir que o modelo minerado também é normal. Então se o *trace* sob análise não é instância do modelo normal, tal *trace* é uma instância anômala.

O projeto inicial do algoritmo é ilustrado no Algoritmo 1. Vale observar no algoritmo

que os primeiros passos representam um pré-processamento utilizado para selecionar os *traces* infrequentes, ou seja, separar apenas os *traces* candidatos a anômalo[1]. O objetivo desse pré-processamento é diminuir o tempo de detecção, já que os *traces* frequentes não são analisados.

Em seguida, o algoritmo realiza a classificação de cada *trace* candidato a partir de um modelo de processo minerado com os *traces* que compõem o *sampling* do *log*. O modelo utilizado na classificação do *trace* é resultado da mineração incremental.

Algoritmo 1: Primeiro projeto do algoritmo *Sampling*.

Entrada: Um *log* L.
Entrada: Um valor de *sampling* $s \in (0, 1)$.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T = {t'|(t', n) ∈ L} com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidatos a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  para cada t ∈ T^C faça
7      Defina um conjunto S com um sampling de s% dos traces de L;
8      Crie um modelo M com os traces em S;
9      se t não é instância de M então
10         Adicione t ao conjunto T^A;
```

Um *trace* é anômalo se não é instância do modelo minerado. Por não ser instância do modelo, o custo de incluir o *trace* testado é maior que zero, então assumimos que o *trace* é anômalo, pois exigirá modificações no modelo para acomodá-lo. Dito de outra forma, o conjunto de *traces* observados no *log*, provavelmente composto apenas por *traces* normais, não foi capaz de gerar um modelo que acomodasse o *trace* testado.

Acreditamos que dois componentes desse algoritmo podem influenciar a sua eficácia: (**i**) o tamanho do *sampling*, que é fornecido como parâmetro de entrada; e (**ii**) o algoritmo de mineração utilizado para criar o modelo M, que nessa implementação utiliza o *mining* incremental, que é mais apropriado para calcular a métrica de custo de inclusão apresentada na Seção 4.1.2.

Assim, vale observar para o Algoritmo 1, que sua eficácia pode ser influenciada como segue:

[1] Aqui consideramos que os *traces* candidatos a anômalos são aqueles *traces* com frequência inferior ou igual a 2%, um valor heuristicamente definido.

(a) se o tamanho do *sampling* for grande, aumentamos a chance de um *trace* anômalo, remanescente no *log*, ser selecionado. Dessa forma, o modelo minerado seria um modelo "contaminado", o que poderia aumentar a taxa de falsos negativos (*traces* anômalos não detectados);

(b) por outro lado, se o tamanho do *sampling* for pequeno, diminuímos a chance de um *trace* normal e infrequente ser selecionado. Dessa forma, o modelo minerado seria uma definição incompleta do modelo normal, o que poderia aumentar a taxa de falsos positivos (*traces* normais detectados como anômalos);

(c) se houver mais *traces* anômalos da mesma classe no *log*, aumentam as chances desse *trace* anômalo ser selecionado no *sampling*, aumentando a taxa de falsos negativos (*traces* anômalos não detectados);

(d) se houver *traces* normais infrequentes no *log*, aumentam as chances de um *trace* normal não ser selecionado no *sampling*, aumentando a taxa de falsos positivos (*traces* normais detectados como anômalos).

4.1.4 Projeto Inicial do Algoritmo *Threshold*

Algoritmo 2: Primeiro projeto do algoritmo *Threshold*.

Entrada: Um *log* L.
Entrada: Um valor de *threshold* $x \in \mathbb{N}^*$.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T = {t'|(t', n) ∈ L} com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidato a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  para cada t ∈ T^C faça
7      Defina um conjunto L' = T − {t};
8      Crie um modelo M com os traces do conjunto L';
9      custo ← custoDeInclusao(M,t);
10     se custo ≥ x então
11         Adicione t ao conjunto T^A;
```

O algoritmo *Sampling* utiliza o conceito do custo de inclusão de forma indireta, pois quando um *trace* não é instância do modelo, o custo de incluí-lo no modelo é necessariamente maior do que zero. No entanto, o algoritmo *Sampling* não realiza processamento

nas seguintes atividades: (i) mineração do modelo incrementado com o *trace* sob análise; (ii) cálculo do tamanho dos modelos e (iii) cálculo do custo de inclusão. Por outro lado, a eficácia do algoritmo depende do tamanho do *Sampling* adotado e da frequência dos *traces* no *log*.

Então, com o intuito de tentarmos resolver ou minimizar essas limitações, desenvolvemos uma outra abordagem de detecção de *traces* anômalos, descrita no Algoritmo 2, que chamamos de Algoritmo *Threshold*. Para esse algoritmo, assumimos que os *traces* anômalos devem satisfazer duas condições: (i) eles não são instância do modelo e (ii) possuem um custo de inclusão acima de um limiar, o *threshold*. Para satisfazer a segunda condição é necessário calcular o custo de inclusão do *trace*, representado pela função $custoDeInclusao(M, t)$ no Algoritmo 2.

O algoritmo de *threshold* considera que *traces* normais podem ter custo de inclusão, mas um custo menor que o custo de inclusão de *traces* anômalos[6, 8, 7]. Ou seja, os *traces* normais também podem necessitar de modificações no modelo gerado a partir dos outros *traces* do *log*, mas em uma escala menor que *traces* anômalos.

No entanto, é necessário definir um valor de *threshold* que maximize a eficácia da detecção dos *traces* anômalos, ou seja, aumentando o número de *traces* anômalos detectados como anômalos (*True Positive Rate*), e diminuindo o número de *traces* normais detectados como anômalos (*False Positive Rate*).

No Algoritmo 2, o valor do *threshold* é um parâmetro de entrada. Então, para encontrarmos um valor de *threshold* razoável, decidimos realizar um estudo estatístico com os valores de custo de inclusão dos *traces* de 300 *logs* que continham apenas *traces* "normais", dos quais: 150 *logs* continham 100% dos possíveis *traces* de um modelo de processo, enquanto os 150 *logs* restantes continham 70% dos possíveis *traces* de um modelo.

Os *logs* utilizados foram gerados a partir de 150 modelos de processo randomicamente criados, como descrito na Seção 1.3.3, Página 7. A ideia seria encontrar um valor "comum" de custo de inclusão quando os *logs* tivessem apenas *traces* normais, ou seja, *traces* que são instância do modelo de processo usado para criar o *log*.

Através dessa análise, foi possível conhecer a variação e a frequência do custo de inclusão de um *trace* não anômalo em relação a um modelo construído com os outros *traces* não anômalos do *log*. A seguir detalhamos como os dados analisados foram coletados e organizados.

1. Incialmente geramos, de forma aleatória, um conjunto M com 150 modelos de processos. Todos os modelos gerados podiam instanciar no mínimo 8 e no máximo 15 classes de *trace*. Além disso, esses modelos eram capazes de instanciar *traces* com até seis atividades (tamanho máximo do *trace*), pois *traces* maiores são um problema para o *miner* incremental.

2. Para cada modelo $m_i \in M$:

   (i) Definimos dois conjuntos de *traces*: $P_{100\%}$ com todos os possíveis *traces* de m_i (*loops* limitados a duas iterações) e $P_{70\%} \subset P_{100\%}$ com 70% dos *traces* de $P_{100\%}$, aleatoriamente selecionados. O conjunto $P_{70\%}$ foi gerado para representar uma aproximação de cenários reais de execução, pois raramente todos os caminhos de um modelo de processo são explorados em cenários reais;

   (ii) $\forall t \in P_{100\%}$, adicionamos o custo de inclusão de t em uma lista L_{100}, onde cada custo foi calculado através da diferença de tamanho entre o modelo gerado com os *traces* do conjunto $P_{100\%}$ e o modelo gerado com os *traces* do conjunto $P' = P_{100\%} - \{t\}$ (ver Figura 4.3);

   (iii) $\forall t \in P_{70\%}$, adicionamos o custo de inclusão de t em uma lista L_{70}, onde cada custo foi calculado através da diferença de tamanho entre o modelo gerado com os *traces* do conjunto $P_{70\%}$ e o modelo gerado com os *traces* do conjunto $P' = P_{70\%} - \{t\}$.

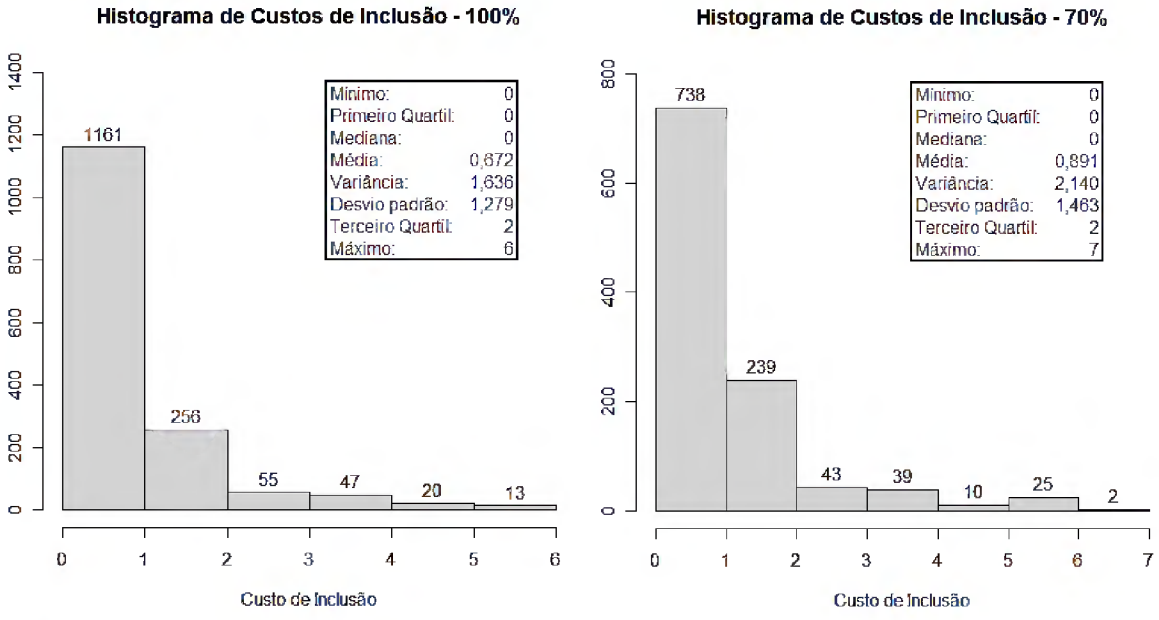


Figura 4.4: Valores em L_{100} e L_{70}.

A Figura 4.4 ilustra a análise dos valores de custo de inclusão coletados nas listas L_{100} e L_{70}, para cada um dos 150 modelos. Como pode ser notado, a junção de um *trace* "normal" raramente produz modificações no modelo. Nos testes em que o *log* continha 100% dos

possíveis *traces*, em mais de 91% dos casos a junção gerava no máximo modificações de custo um. Para os *logs* que continham 70% dos possíveis *traces* esse índice foi de aproximadamente 89%. Além disso, os dois histogramas nos revelam que a quantidade de diferentes *traces* no *log* influencia a variação e a distribuição dos valores de custo de inclusão. Por exemplo, em L_{100} aproximadamente 74% dos *traces* não geraram custo de inclusão, enquanto que em L_{70} esse número foi de aproximadamente 67%. Assim, após essa análise, acreditamos que o valor 2 é um valor razoável para o *threshold*.

No entanto, vale observar que essa análise não nos permite assumir um valor de *threshold* rígido, que se aplique a todos os tipos de *log*, por isso o projeto do algoritmo considera o valor de *threshold* como um parâmetro. No entanto, essa análise é interessante para nos orientar na escolha de um valor padrão, mas que pode ser modificado.

4.1.5 Projeto Inicial do Algoritmo Iterativo

Algoritmo 3: Primeiro projeto do algoritmo Iterativo.

Entrada: Um *log* L.
Entrada: Um valor de *threshold* $x \in \mathbb{N}^*$.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T = {t'|(t', n) ∈ L} com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidato a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  repita
7      C_max ← 0;
8      para cada t ∈ T^C faça
9          Defina um conjunto L' = T − {t} − {T^A};
10         Crie um modelo M com os traces do conjunto L';
11         custo ← custoDeInclusao(M,t);
12         se custo > C_max então
13             C_max ← custo;
14             t_max ← t;
15     se C_max ≥ x então
16         Adicione t_max ao conjunto T^A;
17         T^C = T^C − { t_max };
18 até C_max ≤ x ;
```

A abordagem de detecção iterativa é semelhante ao algoritmo *Threshold*, mas não seleciona de imediato todos os *traces* com custo de inclusão maior que um dado *threshold*. Sua diferença está em utilizar o valor de *threshold* como condição de parada da iteração, que selecionará como anômalo apenas o *trace* com **maior** custo de inclusão em cada iteração.

Uma consequência direta dessa modificação em tornar o algoritmo *Threshold* em um algoritmo Iterativo é o tempo de execução. No melhor caso, quando não houver *traces* anômalos, o tempo de execução do algoritmo iterativo será igual ao tempo de execução do algoritmo *Threshold*. Assim, o número de *traces* anômalos detectados é igual a número de iterações menos um, já que pelo menos uma iteração ocorrerá.

Como pode ser observado no Algoritmo 3, um *trace* anômalo é aquele *trace* com maior custo de inclusão, detectado para cada iteração, além de possuir custo de inclusão maior que a condição de parada, ou seja, o valor de *threshold* informado. Além disso, vale observar que o conjunto de *traces* candidatos a anômalo é diminuído a cada iteração.

A intuição utilizada na definição desse algoritmo é que vários *traces* anômalos poderiam, quando minerados juntos com outros *traces* normais, gerar um modelo que o custo de inclusão de algum *trace* anômalo seja menor que o *threshold* informado, diminuindo a eficácia do algoritmo (*True Postive Rate* - TPR). Então pensamos em selecionar iterativamente cada um dos *traces* anômalos, diminuindo a chance dos *traces* anômalos, quando combinados, "camuflarem" a ocorrência de algum dos *traces* anômalos. Dessa forma, seleciona-se o *trace* anômalo com maior custo de inclusão, um a um, até que nenhum *trace* anômalo seja selecionado.

4.2 Integração com o Framework ProM

Os algoritmos de detecção apresentados nas seções anteriores mostraram-se eficazes, pois foi possível obter acurácia pouco acima dos 90% e detecção de quase 99% dos anômalos (*True Positive Rate*)[8, 7]. No entanto, os resultados da aplicação de experimentos com dados sintéticos demonstraram-nos que tais abordagens são muito limitadas para cenários reais. Observamos que a mineração de um modelo, quando baseada em *logs* com muitas classes de *traces* e/ou com *traces* longos (ex. *traces* baseados em 10 ou mais atividades), provocava um *estouro de memória* durante a execução dos algoritmos. Esse comportamento é explicável e previsível, já que utilizamos um algoritmo de mineração incremental que aplicava diferentes combinações das regras e composições de *traces* para obter um modelo de processo. Ao mesmo tempo, ao identificarmos essas limitações, percebemos a necessidade de explorar outras alternativas, seja explorando outros algoritmos de mineração, seja explorando outras ferramentas de avaliação de modelos de processo.

Para tanto, optamos pela utilização do *Framework ProM* que disponibiliza vários

algoritmos de mineração. Esse *framework* é um ambiente integrado e gratuito de ferramentas de mineração de processo, disponível para download no endereço eletrônico `http://www.processmining.org`. Muito difundido tanto no meio acadêmico com na indústria, o *ProM* mostrou-se uma alternativa robusta para o processamento de *logs* grandes, portanto, uma solução para os problemas da aplicabilidade dos algoritmos de detecção apresentados nas Seções 4.1.3, 4.1.4 e 4.1.5.

Entretanto, os algoritmos de mineração de processo disponíveis no ProM representam os modelos de processo em um formato (ou linguagem) diferente do formato produzido pelo algoritmo de mineração incremental. Muitos dos algoritmos de mineração do ProM não produzem como resultado modelos de processo bloco estruturado, mas redes de *petri*[49]. Por exemplo, os algoritmos α-*algorithm* e α^{++}-*algorithm* geram modelos de processo representados como uma rede de *petri*[46, 54, 29], enquanto os algoritmos *heuristic miner* e *genetic miner* geram modelos de processo chamados de *heuristic net*[52, 16, 47].

No caso das rede de *petri*, elas utilizam dois tipos de vértice, os *places* e as transições, que possuem semântica diferente, além das arestas, que nunca ligam dois vértices do mesmo tipo. Diferente da representação bloco estruturada, que admite o aninhamento de blocos *and* e *or*, as redes de *petri* não permitem representações encadeadas ou aninhadas de blocos. Portanto, não é mais razoável medir o tamanho dos modelos considerando apenas a contagem dos vértices, como a métrica de custo de inclusão apresentada na Seção 4.1.2.

Além disso, os algoritmos disponíveis no ProM não são algoritmos que constroem um modelo de forma iterativa e incremental, ou seja, um modelo de processo não é "adaptado" para acomodar um novo *trace*. Por isso, também é inapropriado chamar de custo de inclusão os efeitos da adição do novo *trace* ao processo de geração do novo modelo, pois o novo modelo minerado não é uma adaptação de um modelo antigo para acomodar um novo *trace*.

Portanto, é necessário realizar adaptações no projeto inicial dos algoritmos de detecção, a fim de suportar as diferenças de representação de modelos de processo apresentadas acima, sem perder a característica central dos métodos de detecção, que consideram anômalo aquele *trace* que demanda elevado grau de modificação no modelo. Assim, a **primeira mudança no projeto inicial** é a utilização dos algoritmos de mineração disponíveis no *ProM*. A **segunda mudança no projeto inicial**, influenciada pela primeira, considera a adoção de outras métricas de avaliação de modelo, também disponíveis no *framework ProM*.

4.2.1 Algoritmos de Mineração

A integração com o *framework* ProM permitiu que utilizássemos diferentes *plug ins* de *process discovery* para mineração de um modelo de processo. Diferente do algoritmo de mineração de processo incremental, utilizado no projeto inicial dos algoritmos de detecção, e que limitava bastante a aplicação prática dos algoritmos de detecção propostos nesta tese, utilizamos os seguintes algoritmos de mineração:

Alpha. é o algoritmo mais difundido na comunidade de *Process Mining* por apresentar uma eficácia formalmente provada para uma classe de *workflows*[44, 48, 46], entretanto possui algumas limitações como a mineração de *loops* curtos (de uma atividade), tarefas duplicadas e a relação implícita entre duas atividades, além de considerar que o *log* utilizado pelo algoritmo não possui ruído e é completo, no sentido de possuir todas as relações de precedência entre as atividades quando estas tiverem que aparecer no modelo minerado; outra característica deste algoritmo é que o modelo gerado é uma rede de *petri*;

Alpha++. é uma extensão do algoritmo anterior que apresenta uma solução para a mineração de relações implícitas entre atividades (*non-free-choice*)[54]; semelhante ao seu algoritmo predecessor, gera modelos de processo como redes de *petri*;

Heuristic. é também uma extensão do algoritmo *alpha*, mas é um algoritmo robusto a ruído pois considera a frequência das relações entre as atividades para representá-las no modelo[52]; outra característica deste algoritmo é o formato de saída, a *heuristic net*;

Multiphase. é um algoritmo de mineração de processos em que a linguagem de representação dos processos é EPC (*Event-driven Process Chain*)[50]; sua execução ocorre em dois passos: inferência de relações binárias entre as atividades de um mesmo trace e junção dessas relações em um modelo, identificando elementos estruturais como *AND*, *OR* e *XOR*.

Outro algoritmo disponível no *framework* ProM é o *genetic process mining*[16]. Entretanto, não é uma solução de mineração de processo apropriada para as abordagens de detecção exploradas nesta tese, mesmo sendo um algoritmo robusto a presença de ruídos no *log* e que representa uma solução para muitos dos problemas de descoberta de modelos de processo relatados pela comunidade de *process mining*. Essa limitação do uso do *genetic process mining* está especialmente relacionada à quantidade de modelos de processo com *fitness* 1 que podem ser gerados pelo algoritmo, além do tempo de processamento para se obter um resultado, que pode ser extremamente elevado, por exemplo, levar dias para encontrar um modelo com *fitness* 1.

4.2.2 Métricas de Conformidade

As métricas de conformidade, apresentadas em [35, 36], têm o propósito de indicar o grau de harmonia entre um modelo de processo e um *log*, ou seja, o quanto um modelo representa corretamente as instâncias de um *log*. Essas métricas são definidas sob duas dimensões: *fitness* e *appropriateness*. A dimensão *fitness* é avaliada por uma métrica de mesmo nome, enquanto que a dimensão *appropriateness* é avaliada por duas métricas, *structural* e *behavioral appropriateness*.

Essas métricas podem assumir um valor real entre 0 e 1, tal que o valor 1 representa 100% de conformidade (conformidade total), enquanto o valor 0 representa 0% de conformidade (nenhuma conformidade). Assim, o valor 1 para a métrica *fitness* entre um modelo e um *log* indica que todos os *traces* do *log* podem ser instâncias do modelo, ou seja, todos os *traces* podem ser executados totalmente pelo modelo. Enquanto o valor 0 indica que nenhum *trace* pode ser instanciado pelo modelo, ou seja, nenhum *trace* do *log* pode ser executado pelo modelo.

Como é possível descobrir vários modelos que podem ter *fitness* com valor 1 para um *log* (ex.: os modelos genérico e específico, Figuras 3.4 e 3.5) as métricas *structural* e *behavioral appropriateness* são utilizadas para indicar a preferência por um modelo, ou seja, complementam a definição de conformidade quando combinada com a métrica *fitness*.

A métrica *structural appropriateness* penaliza modelos grandes e muito específicos, pois são capazes de instanciar apenas os *traces* observados no *log*. Por outro lado, a métrica *behavioral appropriateness* penaliza modelos muito genéricos, pois são modelos que adicionam comportamento extra ao observado no *log*, ou seja, são capazes de instanciar muitos *traces* diferentes dos existentes no *log*.

Considerando o exemplo dos modelos muito genérico ou muito específico, que teriam *fitness* 1, teríamos: (i) o modelo genérico pode prever *traces* com qualquer combinação das atividades existentes no *log*, então seu *behavioral appropriateness* seria próximo de 0; enquanto que (ii) o modelo específico seria complexo demais (um *OR* de cada *trace*) para observar apenas os *traces* existentes no *log*, então seu *structural appropriateness* seria próximo de 0.

Essas métricas consideram que o modelo de processo está representado como uma rede de *petri*. Assim, as métricas de avaliação de modelos (*fitness*, *structural* e *behavioral appropriateness*) são aplicadas sobre redes de *petri*, obtidas a partir de algoritmos como o α-algoritmo[44] e α++[54].

Uma versão mais simplificada de avaliação da complexidade de um modelo foi apresentada em [5], a métrica *size*. Essa métrica foi inspirada no conceito de custo de inclusão[3, 8, 7], discutido na Seção 4.1.2. Ela indica a variação de tamanho entre dois modelos de processo, através da simples contagem de vértices e arestas da *petri net*. No

caso, a variação de elementos estruturais é normalizada pelo tamanho de um modelo, como segue: $1 - |s_1 - s_2|/s_1$. Onde s_1 e s_2 indicam o número de elementos estruturais (vértices e arestas) dos modelos 1 e 2, respectivamente.

4.2.3 Algoritmo *Sampling*

Algoritmo 4: Algoritmo *Sampling*

Entrada: Um *log* L.
Entrada: Um valor de *sampling* $s \in (0, 1)$.
Entrada: Um algoritmo de mineração de processo MP.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidatos a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  para cada t ∈ T^C faça
7      Defina um conjunto S com um sampling de s% dos traces de L;
8      Crie um modelo M com os traces em S usando o algoritmo MP;
9      se t não é instância de M então
10         Adicione t ao conjunto T^A;
```

O Algoritmo 4 é uma adaptação do projeto inicial do algoritmo *sampling* (Algoritmo 1). A principal diferença está relacionada à integração com o *framework* ProM, que ocorre em dois pontos: (i) o algoritmo de *process discovery*, utilizado na Linha 8; e (ii) o teste de instância, através da métrica de avaliação *fitness* na Linha 9.

Nesse algoritmo, para cada *trace* candidato a anômalo, Linha 6, é executado como segue: (i) obtém uma amostra do *log* (Linha 7); (ii) descobre um modelo de processo a partir dessa amostra utilizando o algoritmo de mineração de processo indicado (Linha 8); e (iii) testa se o *trace* candidato é uma instância do modelo de processo descoberto (Linha 9). Os *traces* anômalos são aqueles *traces* que não são instância do modelo de processo descoberto em cada iteração.

4.2.4 Algoritmo *Threshold*

A abordagem de detecção de anomalias baseada na avaliação de conformidade adota a seguinte hipótese para identificar *traces* anômalos no *log*: um *log* que contém uma

Algoritmo 5: Algoritmo *Threshold*

Entrada: Um *log* L.
Entrada: Um valor de *threshold* $x \in (0, 1)$.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidatos a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  para cada t ∈ T^C faça
7      Defina um conjunto L' = {t'|(t', n) ∈ L} − {t};
8      Crie um modelo M com os traces do conjunto L';
9      conformidade ← grauDeConformidade(M,L);
10     se conformidade < x então
11         Adicione t ao conjunto T^A;
```

instância anômala de processo tem menor conformidade que um *log* apenas com instâncias normais. O *threshold* indica o grau de conformidade tolerável para o modelo de processo descoberto com o *log* sem a instância anômala candidata.

Assim, as adaptações estão representadas no Algoritmo 5. Nesse algoritmo, para cada *trace* candidato a anômalo, Linha 6, é executado como segue: (i) um novo *log* é definido a partir do *log* de entrada, mas desconsiderando ocorrências do *trace* candidato (Linha 7); (ii) um modelo de processo é descoberto a partir desse novo *log* (Linha 8); (iii) calcula o valor de conformidade entre o *log* original fornecido para detecção e o modelo de processo descoberto com o *log* novo, que não contém ocorrências do *trace* candidato (Linha 9); então (iv) testa se o grau de conformidade é menor que um valor de *threshold* informado como parâmetro do algoritmo (Linha 10). Os *traces* anômalos são aqueles *traces* que tem um grau de conformidade menor que o *threshold* informado.

A integração com o *framework* ProM ocorre em dois pontos: (i) na Linha 8, durante a mineração de modelo de processo com o algoritmo de *process discovery* informado como entrada da função; e (ii) na Linha 9, durante o cálculo do grau de conformidade, através da métrica de avaliação de modelo informada como entrada da função.

Outra modificação é o operador do teste de classificação do *trace* na Linha 10, que como é para o grau de conformidade, não mais interessa os maiores valores (como utilizado no custo de inclusão), mas os menores valores. Ou seja, o valor de *threshold* expressa um limite inferior de conformidade e não mais um limite superior de custo de inclusão. Portanto, quanto menor a conformidade maior a chance de ser um *trace* anômalo.

4.2.5 Algoritmo Iterativo

Algoritmo 6: Algoritmo Iterativo

Entrada: Um *log* L.
Entrada: Um valor de *threshold* $x \in (0, 1)$.
Saída: Um conjunto T^A de *traces* anômalos.

```
1  Defina um conjunto T com as classes de trace do log L;
2  Defina um conjunto T^C = {} com as classes de trace candidatos a anômalo;
3  para cada t ∈ T faça
4      se Frequência de t no log ≤ 2% então
5          Adicione t ao conjunto T^C;
6  repita
7      C_min ← 1;
8      para cada t ∈ T^C faça
9          Defina um conjunto L' = {t'|(t', n) ∈ L} − {t} − {T^A};
10         Crie um modelo M com os traces do conjunto L';
11         conformidade ← grauDeConformidade(M,L);
12         se conformidade < C_min então
13             C_min ← conformidade;
14             t_min ← t;
15     se C_min < x então
16         Adicione t_min ao conjunto T^A;
17         T^C = T^C − { t_min };
18 até C_min ≥ x ;
```

As mesmas adaptações implementadas no algoritmo de *Threshold* foram aplicadas no algoritmo iterativo, ou seja, (i) a utilização de um algoritmo de mineração do ProM, na Linha 10; e (ii) a avaliação da conformidade do *trace*, na Linha 11, que será testada para um valor de *threshold* informado (provavelmente o mesmo valor utilizado pelo algoritmo *Threshold*).

Outras diferenças, quando comparado com o projeto inicial, estão no operador utilizado no teste de parada do algoritmo (Linha 18) e na seleção do *trace* anômalo (Linha 15). Diferente da abordagem inicial, que procurava pelo *trace* com maior valor de custo de inclusão, agora a busca é pelo *trace* com menor conformidade.

4.3 Estudo Comparativo dos Algoritmos

4.3.1 Criação dos *logs* para avaliação

Tabela 4.1: Parâmeros para criação dos modelos.

Nome	# de modelos	Lag min do *trace*	Lag max do *trace*	# min de *traces*	# max de *traces*
Logs10y	100	5	10	6	20
Logs15y	100	5	15	6	20
Logs15y	100	5	15	6	20

Para avaliação dos algoritmos foram utilizados **1800 *logs***, divididos em três classes diferentes, de acordo com o tamanho máximo dos traces contidos nesses *logs*, no caso, 600 *logs* com *traces* de comprimento máximo de 10 atividades, 600 *logs* com *traces* de comprimento máximo de 15 atividades, e 600 *logs* com com *traces* de comprimento máximo de 20 atividades.

Além da organização baseada no tamanho dos *traces* contidos no *log*, os *logs* foram gerados com as seguintes características: (i) *logs* com uma ou duas classes de *trace* anômalo; e (ii) *logs* com uma, três ou cinco ocorrências de uma classe de *trace* anômalo. Assim, dentre os 600 *logs*, foram gerados 100 *logs*:

- 1A – com uma classe com uma ocorrência;
- 1B – com duas classes com uma ocorrência de cada classe;
- 3A – com uma classe com três ocorrências;
- 3B – com duas classes com três ocorrências de cada classe;
- 5A – com uma classe com cinco ocorrências;
- 5B – com duas classes com cinco ocorrências de cada classe;

Portanto, considerando as propriedades de (i) tamanho máximo de um *trace*, (ii) o número de classes de *trace* anômalo no *log* e (iii) o número de ocorrências da classe de *trace* no *log*, os *logs* utilizados na avaliação dos algoritmos apresentavam 18 diferentes combinações de características. Ou seja, cada grupo dos 100 arquivos de *log* dos 1800 diferentes arquivos de *logs* criados possuem a mesma combinação de propriedade.

Esses *logs* são instâncias dos **300 modelos** de processo criados com a função de criação de modelo apresentada na Seção 1.3.3. A Tabela 4.1 resume os parâmetros utilizados para a criação dos modelos utilizados para criação dos *logs*. A coluna **nome** não é um parâmetro

da função de criação dos modelos, mas aqui serve para ilustrar o prefixo utilizado no nome dos arquivos de *log*, enquanto que as outras cinco colunas são parâmetros da função de criação dos modelos. Para cada modelo de processo foram criados seis diferentes *logs*, cada um com 1000 *traces* normais e mais os *traces* anômalos (1A, 1B, 3A, etc.). Além disso, não *traces* com menos de cinco atividades nos *logs*.

4.3.2 Parametrização dos algoritmos avaliados

Os algoritmos apresentados neste capítulo possuem dois parâmetros, no caso do algoritmo de *sampling*, ou três parâmetros, no caso dos algoritmos iterativo e *threshold*. Essa diversidade de parâmetros torna a execução dos algoritmos bastante flexível. Por outro lado, podemos dizer que cada combinação de parâmetros define um novo detector de anomalias. Portanto, o resultado de uma avaliação dos algoritmos não apenas indicaria qual é o algoritmo mais eficaz, mas também qual é a combinação de parâmetros do algoritmo (detector) mais eficaz.

Considerando o universo dos 1800 *logs* criados para a avaliação dos algoritmos, testar cada uma das opções de algoritmos para todos esses *logs* seria muito custoso. Além disso, em situações reais, o auditor raramente desejará avaliar o resultado da detecção do *log* auditado para cada uma das combinações de parâmetro possível. Por essas razões, optamos por estudar (ou identificar) primeiramente a combinação de parâmetros mais eficaz para cada um dos algoritmos, para então realizarmos a avaliação de cada um dos algoritmos. Por exemplo, na área de inteligência artificial é comum haver uma seleção de um grupo de exemplos (casos ou indivíduos) para a realização do treinamento de uma solução, o chamado grupo de treino. Então, uma vez que a solução está apropriadamente configurada ou treinada, o grupo de indivíduos restante, chamado de grupo de teste, é utilizado para avaliar a eficácia da solução. Nós utilizamos a mesma abordagem, onde o grupo de treino foi o conjunto de *logs* utilizado para identificar os parâmetros ou detector mais eficaz, enquanto que o grupo de teste foi o conjunto de *logs* restante.

Para compor o grupo de treino dessa avaliação, selecionamos aleatoriamente 15% dos *logs*. Assim, dos 600 *logs* com *traces* de mesmo tamanho máximo (10 ou 15 ou 20), aproximadamente 100 foram separados para o grupo de treino, perfazendo um total de quase 300 *logs* dos 1800 criados para esta avaliação.

Após a separação dos *logs*, cada um dos algoritmos de detecção foi aplicado. No caso do algoritmo de *sampling*, são **12 possibilidades** de execução sobre os quase 300 *logs*: quatro opções de algoritmos de mineração (*alpha*, *alpha++*, *multiphase* e *heuristic*) e três fatores de *sampling* (20%, 50% e 70%). O fator de *sampling* é um parâmetro com infinitas possibilidades de valor, já que trata-se de um parâmetro contínuo. Entretanto, optamos por discretizar as opções utilizadas nessa avaliação, como segue: metade (50%), menos

da metade (20%) e mais da metade (70%).

No caso dos algoritmos *threshold* e iterativo, são **60 possibilidades** de execução: quatro opções de algoritmos de mineração (*alpha*, *alpha++*, *multiphase* e *heuristic*); cinco opções de algoritmos de avaliação de modelo (*fitness*, *behavioral*, *structural*, *appropriateness* e *size*); e três valores de *threshold* (0.5, 0.7 e 0.9). O fator de *threshold* é também um parâmetro contínuo, mas novamente optamos por discretizar as opções: quanto maior o valor, menor a tolerância para classificar um *trace* como anômalo.

Considerando as diferentes configurações de execução dos algoritmos, temos 132 detectores ou soluções de detecção de anomalias (12 para o *sampling*, 60 para o *threshold* e 60 para o iterativo).

Na execução dos algoritmos, foram coletadas três métricas de avaliação[24]: (i) *f-measure*, que significa a média harmônica entre *precision* e *recall*; (ii) *recall*, que significa a razão entre o número de *traces* anômalos corretamente classificados e o total de *traces* anômalos existentes no *log* (ou *true positive rate*); e (iii) a acurácia, que significa a proporção de *traces* corretamente classificados como normal e anômalo.

Os resultados foram ordenados em ordem decrescente de *f-measure*, em ordem decrescente de *recall* e em ordem decrescente de acurácia, nessa ordem. Então, definimos para cada um dos algoritmos a seguinte combinação de parâmetros, com os respectivos valores médio de desempenho:

Sampling .

Parâmetros :

Algoritmo de mineração: *Heuristic*;
Fator de *sampling*: 70%.

Desempenho médio :

f-measure: 0,75
Recall: 1
Acurácia: 0,78

Threshold .

Parâmetros :

Algoritmo de mineração: *Alpha*;
Algoritmo de avaliação: *Fitness*;
Fator de *threshold*: 90%.

Desempenho médio :

f-measure: 0,29

Recall: 0,62

Acurácia: 0,35

Iterativo .

Parâmetros :

Algoritmo de mineração: Alpha;

Algoritmo de avaliação: Appropriateness;

Fator de *threshold*: 90%.

Desempenho médio :

f-measure: 0,16

Recall: 0,35

Acurácia: 0,48

4.3.3 Execução e Resultados

Como apresentado na Seção 3.1, uma abordagem de detecção baseada na classificação dos *traces* infrequentes como anômalos seria muito ingênua, pois é provável que *traces* normais também ocorram com baixa frequência no *log*, por exemplo, porque alguns caminhos do *workflow* sejam mais exercitados que outros.

Os três algoritmos de detecção selecionam como candidato a *trace* anômalo as execuções infrequentes. Então, considerando que os *traces* anômalos são infrequentes, um método que classifique todos os candidatos como anômalo garantiria um *recall* de 1. Por outro lado, o valor do *precision* seria baixo, especialmente quando houver muito candidatos e poucos anômalos.

Para obter os dados de desempenho dessa abordagem baseada na frequência (aqui vamos chamar de "algoritmo ingênuo"), realizamos os seguintes cálculos para cada um dos *logs* do grupo de teste: (i) *recall* é constante e vale 1; (ii) a acurácia é a razão entre o número de anômalos e o número de candidatos; (iii) *precision* é igual à acurácia, pois todos os candidatos são classificados como anômalos; e o (iv) *f-measure* é obtido a partir do *recall* e *precision*, como em sua definição[24]. Dessa forma, é possível comparar essa abordagem com os outros três algoritmos de detecção.

Após a seleção do melhor conjunto de parâmetros para execução, como relatado na seção acima, cada um dos algoritmos de detecção foi executado sobre aproximadamente 1500 *logs* pertencentes ao grupo de teste. Após a execução realizamos as mesmas análises realizadas durante a parametrização, então obtivemos as médias apresentadas na Tabela 4.2.

Tabela 4.2: Desempenho médio da execução dos algoritmos com o grupo de teste.

Médias	Sampling	Threshold	Iterativo	"Algoritmo Ingênuo"
Tempo (ms)	13595,513	15265,109	46873,175	n.a.
F-Measure	0,753	0,322	0,156	0,474
Recall	0,996	0,648	0,285	1,000
Acurácia	0,787	0,379	0,508	0,351

Tabela 4.3: Resultado do *Tukey HSD Test*. Todos os algoritmos.

Algoritmo	Diferença das Médias	Limite Inferior	Limite Superior
Sampling-Iterativo	0.5969499	0.5728652	0.6210347
Sampling-Threshold	0.4310013	0.4069166	0.4550860
Sampling-Ingênuo	0.2788142	0.2547295	0.3028990
Ingênuo-Iterativo	0.3181357	0.2940510	0.3422204
Ingênuo-Threshold	0.1521871	0.1281024	0.1762718
Threshold-Iterativo	0.1659486	0.1418639	0.1900333

Os dados apresentados na Tabela 4.2 indicam que o Algoritmo *Sampling* tem um desempenho melhor que os outros algoritmos, tanto quanto à eficácia na classificação dos *traces* anômalos, como quanto o tempo de execução – exceção ao tempo de execução do "algoritmo ingênuo", cujo desempenho não se aplica para comparação, pois os resultados de eficácia apresentados não foram obtidos através da execução de um algoritmo. No entanto, a fim de obtermos evidências mais fortes dessa diferença de desempenho entre as quatro abordagens, aplicamos o teste *One-way ANOVA* sobre os dados de execução dos três algoritmos, mais os dados do "algoritmo ingênuo".

One-way ANOVA (análise de variância de um único fator) é uma técnica de teste de hipótese para comparar as médias de três ou mais populações[27]. Assim, a hipótese nula nessa análise é que não há diferença entre as médias dos *F-Measure* obtidos pelos quatro algoritmos. No entanto, após o teste, obtemos um $p\text{-}value{=}2.2e-16$, que indica que há fortes evidências para rejeitarmos a hipótese nula com pelo menos 95% de confiança. Portanto, existe ao menos uma diferença entre as quatro médias.

No entanto, com o teste ANOVA não podemos avaliar onde está essa diferença ou, nesse caso, qual algoritmo é mais eficaz que outro. Por essa razão, aplicamos o *Tukey HSD Test*, que é um teste *post hoc* utilizado para indicar onde está a diferença identificada pelo teste ANOVA[39]. O resultado desse teste é apresentado na Tabela 4.3, que descreve par-a-par qual método é melhor que outro, a diferença entre as médias dos métodos analisados e os limites inferior e superior para essas médias, considerando um nível de confiança de 95%.

A partir desse resultado podemos afirmar que o Algoritmo *Sampling* foi o melhor

Tabela 4.4: Resultado do *Tukey HSD Test*. Número de classes. Todos os algoritmos.

Algoritmo	Dif. da Média	Lim. Inferior	Lim. Superior	Classes
Sampling-Ingênuo	0.3057219	0.27134092	0.3401029	Uma
Sampling-Iterativo	0.5881417	0.55376070	0.6225227	Uma
Sampling-Threshold	0.4620588	0.42767781	0.4964398	Uma
Ingênuo-Iterativo	0.2824198	0.24803878	0.3168008	Uma
Ingênuo-Threshold	0.1563369	0.12195589	0.1907179	Uma
Threshold-Iterativo	0.1260829	0.09170188	0.1604639	Uma
Sampling-Ingênuo	0.2526753	0.2207457	0.2846049	Duas
Sampling-Iterativo	0.6055065	0.5735769	0.6374361	Duas
Sampling-Threshold	0.4008312	0.3689016	0.4327608	Duas
Ingênuo-Iterativo	0.3528312	0.3209016	0.3847608	Duas
Ingênuo-Threshold	0.1481558	0.1162262	0.1800855	Duas
Threshold-Iterativo	0.2046753	0.1727457	0.2366049	Duas

método de detecção, com eficácia melhor que todos os outros métodos, pois possui mais acurácia média, estatisticamente superior as médias de acurácia dos outros métodos de detecção. Entretanto, a abordagem do "algoritmo ingênuo" mostrou-se mais eficaz que os algoritmos *Threshold* e Iterativo. Além disso, nenhum par de métodos possui eficácia equivalente, pois o intervalo da diferença entre as médias (observada pelos limites inferior e superior) não contém o valor 0.

Além dessa análise mais geral, também investigamos se havia diferença de eficácia entre os métodos quando certas propriedades dos *logs* eram observadas, entre elas: (i) o número de classes de *traces* anômalos (uma ou duas classes); (ii) o número de ocorrências de *trace* anômalo da mesma classe (uma, três ou cinco ocorrências); e o (iii) tamanho máximo dos *traces* no *log* (10, 15 ou 20 atividades). Assim, após executarmos o teste *One-way ANOVA* sobre os resultados da execução sobre *logs* com características especiais, notamos que também há diferença de eficácia entre os métodos de detecção. Em todos os testes o *p-value* obtido foi o mesmo, $2.2e-16$.

Uma vez que identificamos que havia diferença de eficácia entre métodos quando processado sobre os *logs* com características especiais, também aplicamos o teste *post hoc* para indicar onde essa diferença ocorria. Notamos que a diferença de eficácia par-a-par manteve-se entre os métodos, novamente com destaque para o Algoritmo *Sampling*, com melhor eficácia entre as quatro abordagens de detecção, enquanto o Algoritmo Iterativo teve pior desempenho. Apresentamos nas Tabelas 4.4, 4.5 e 4.6 os resultados obtidos com a execução dos testes *post hoc* para os *logs* com diferenças no número de classe, no tamanho dos *traces* e no número de ocorrências respectivamente.

Porque o Algoritmo *Sampling* teve melhor nível de eficácia dentre todos os algoritmos

Tabela 4.5: Resultado do *Tukey HSD Test*. Número de atividades. Todos os algoritmos.

Algoritmo	Dif. da Média	Lim. Inferior	Lim. Superior	Atividades
Sampling-Ingênuo	0.3527539	0.3175857	0.3879221	20
Sampling-Iterativo	0.7117773	0.6766091	0.7469456	20
Sampling-Threshold	0.5557031	0.5205349	0.5908714	20
Ingênuo-Iterativo	0.3590234	0.3238552	0.3941917	20
Ingênuo-Threshold	0.2029492	0.1677810	0.2381175	20
Threshold-Iterativo	0.1560742	0.1209060	0.1912425	20
Sampling-Ingênuo	0.2932213	0.2536242	0.3328185	15
Sampling-Iterativo	0.6384387	0.5988416	0.6780358	15
Sampling-Threshold	0.4366403	0.3970432	0.4762374	15
Ingênuo-Iterativo	0.3452174	0.3056203	0.3848145	15
Ingênuo-Threshold	0.1434190	0.1038219	0.1830161	15
Threshold-Iterativo	0.2017984	0.1622013	0.2413955	15
Sampling-Ingênuo	0.18852	0.14416777	0.2328722	10
Sampling-Iterativo	0.43738	0.39302777	0.4817322	10
Sampling-Threshold	0.29760	0.25324777	0.3419522	10
Ingênuo-Iterativo	0.24886	0.20450777	0.2932122	10
Ingênuo-Threshold	0.10908	0.06472777	0.1534322	10
Threshold-Iterativo	0.13978	0.09542777	0.1841322	10

avaliados, decidimos analisar se certas características no *log* influenciariam sua eficácia. Semelhante à análise que realizamos entre os quatro algoritmos, avaliamos o efeito do número de classes de *traces* anômalos, do número de ocorrências de *trace* anômalo com a mesma classe e do tamanho máximo dos *traces* no *log*.

Além dessas três características, que foram avaliadas usando o teste *One-way* ANOVA, avaliamos se o número de candidatos a *traces* anômalo influenciaria a eficácia (*F-Measure*) do Algoritmo *Sampling*. Para tanto, aplicamos a regressão linear, pois as variáveis analisadas são numéricas, o que permitiria representar a relação entre elas por uma reta[2]. Nesta análise, a regressão linear foi utilizada para sabermos quando o *F-Measure* aumenta, se quando o número de *traces* candidatos é maior ou quando ele é menor. A seguir apresentamos uma síntese dos resultados obtidos com as análises:

Número de classes . O número de classes de *traces* anômalos no *log* é uma característica que influencia a eficácia do Algoritmo *Sampling*. Após a análise de variância das médias dos dois grupos (uma ou duas classes) de resultados, obtivemos um *p*-

[2]A regressão linear tenta encontrar uma reta em que a soma dos quadrados entre o valor esperado e o valor observado seja mínima, assim é possível prever o valor de y (nesse caso, *F-Measure*) para um dado valor x (nesse caso, número de candidatos)[27]

Tabela 4.6: Resultado do *Tukey HSD Test*. Número de ocorrências. Todos os algoritmos.

Algoritmo	Dif. da Média	Lim. Inferior	Lim. Superior	Ocorrências
Sampling-Ingênuo	0.27264	0.2307260	0.3145540	Uma
Sampling-Iterativo	0.59378	0.5518660	0.6356940	Uma
Sampling-Threshold	0.42582	0.3839060	0.4677340	Uma
Ingênuo-Iterativo	0.32114	0.2792260	0.3630540	Uma
Ingênuo-Threshold	0.15318	0.1112660	0.1950940	Uma
Threshold-Iterativo	0.16796	0.1260460	0.2098740	Uma
Sampling-Ingênuo	0.2881731	0.2472531	0.3290930	Três
Sampling-Iterativo	0.6025577	0.5616378	0.6434776	Três
Sampling-Threshold	0.4304808	0.3895608	0.4714007	Três
Ingênuo-Iterativo	0.3143846	0.2734647	0.3553045	Três
Ingênuo-Threshold	0.1423077	0.1013878	0.1832276	Três
Threshold-Iterativo	0.1720769	0.1311570	0.2129969	Três
Sampling-Ingênuo	0.2752410	0.2327524	0.3177295	Cinco
Sampling-Iterativo	0.5942771	0.5517886	0.6367656	Cinco
Sampling-Threshold	0.4367470	0.3942585	0.4792355	Cinco
Ingênuo-Iterativo	0.3190361	0.2765476	0.3615247	Cinco
Ingênuo-Threshold	0.1615060	0.1190175	0.2039946	Cinco
Threshold-Iterativo	0.1575301	0.1150416	0.2000187	Cinco

value$=2.511e-12$. Portanto, temos evidência suficientemente forte para rejeitarmos a hipótese de que não há diferença entre as médias dos dois grupos. Apresentamos na Tabela 4.7 o resultado do teste *post hoc*, com nível de confiança de 95%, que utilizamos para conhecermos o intervalo da diferença entre as médias. Observe que o valor 0 **não** está incluído no intervalo, indicando que as médias **não** são iguais, com confiança de 95%.

Tabela 4.7: Diferença entre as médias. Número de classes. Algoritmo *Sampling*.

Classes	Dif. das Médias	Lim. Inferior	Lim. Superior
Duas - Uma	0.09113216	0.06581548	0.1164488

Número de ocorrências de *traces* da mesma classe . Esta característica não influencia a eficácia do Algoritmo *Sampling*. Após a análise de variância das médias dos três grupos (uma, três ou cinco ocorrências) de resultados, obtivemos um *p-value*$=0.8523$. Portanto, **não** temos evidências suficientes para rejeitarmos a hipótese de que não há diferença em pelo menos uma das médias dos três grupos. Apresentamos na Tabela 4.8 o resultado do teste *post hoc*, com nível de confiança

de 95%, que utilizamos para conhecermos par-a-par o intervalo da diferença entre as médias. Observe que em todas as linhas o valor 0 está incluído no intervalo, indicando que as médias podem ser iguais.

Tabela 4.8: Diferença entre as médias. Número de ocorrências. Algoritmo *Sampling*.

Ocorrências	Dif. das Médias	Lim. Inferior	Lim. Superior
Cinco - Uma	0.004230843	-0.03372612	0.04218781
Três - Uma	0.009039231	-0.02851330	0.04659177
Três - Cinco	0.004808387	-0.03278257	0.04239935

Número máximo de atividades no *traces* . Esta característica não influencia a eficácia do Algoritmo *Sampling*. Após a análise de variância das médias dos três grupos (10, 15 ou 20 atividades) de resultados, obtivemos um *p-value*=0.254. Portanto, **não** temos evidências suficientes para rejeitarmos a hipótese de que não há diferença em pelo menos uma das médias dos três grupos. Apresentamos na Tabela 4.9 o resultado do teste *post hoc*, com nível de confiança de 95%, que utilizamos para conhecermos par-a-par o intervalo da diferença entre as médias. Observe que em todas as linhas o valor 0 está incluído no intervalo, indicando que as médias podem ser iguais.

Tabela 4.9: Diferença entre as médias. Número de atividades. Algoritmo *Sampling*.

Atividades	Dif. das Médias	Lim. Inferior	Lim. Superior
20 - 10	0.019816719	-0.01784924	0.05748268
15 - 10	0.025381581	-0.01239456	0.06315772
15 - 20	0.005564862	-0.03198795	0.04311767

Número de *traces* candidatos . O número de *traces* candidato a anômalo é uma característica do *log* que influencia a eficácia do Algoritmo *Sampling*. Após a aplicação da regressão linear sobre o conjunto de pares ordenados (candidatos, *F-Measure*), obtivemos uma reta de regressão com as seguintes características:

- o coeficiente de inclinação obtido foi de -0.037387, com intervalo entre -0.039238 e -0.035536, com mais de 99% de nível de confiança e valor de *p-value*$=2.2e-16$;
- o fato do intervalo de valores para o coeficiente de inclinação não conter 0 e o valor do *p-value* ser extremamente baixo são evidências suficientemente fortes para rejeitarmos a hipótese nula de que o coeficiente de inclinação é 0 (valor que indicaria que não há relação entre as variáveis número de *traces* candidatos e *F-Measure*);

- como o valor do coeficiente é negativo, podemos afirmar que a eficácia do Algoritmo *Sampling* é melhor quando houver menos *traces* candidatos.

Capítulo 5

Detecção de Anomalias: Seleção do Modelo mais Apropriado

Este capítulo trata da segunda abordagem de detecção explorada neste trabalho (ver Seção 3.4), que é a abordagem baseada na seleção de um modelo de processo, dentre os vários que podem ser descobertos a partir do *log*, aquele que seja mais apropriado para classificar os *traces* como anômalos ou normais. Para tanto, este capítulo foi organizado da seguinte forma: na Seção 5.1 descrevemos de uma forma geral como a abordagem de detecção baseada na seleção do modelo mais apropriado funciona; na Seção 5.2 apresentamos como o *Framework ProM* pode ser utilizado para apoiar em cada uma das etapas da abordagem; na Seção 5.3 apresentamos um estudo de caso do método com dados reais obtidos do governo holandês; as considerações finais da abordagem são apresentadas na Seção 5.4.

5.1 Visão Geral

A Figura 5.1 representa um esquema geral da abordagem de detecção, que é organizada em cinco passos:

Preparação do *log*. Trata-se da fase, dependente de domínio, responsável por aplicar filtros no *log* a fim de remover os *traces* incompletos e claramente anômalos ou as atividades que não são importantes para a análise (ver Seção 3.2);

Mineração dos modelos. Trata-se da fase que representa a descoberta de processos que descrevam o *log* filtrado;

Separação dos modelos. Trata-se da fase em que os modelos que satisfazem o requisito mínimo de *fitness*, valor p% (ver Definição 9, Página 26), são separados;

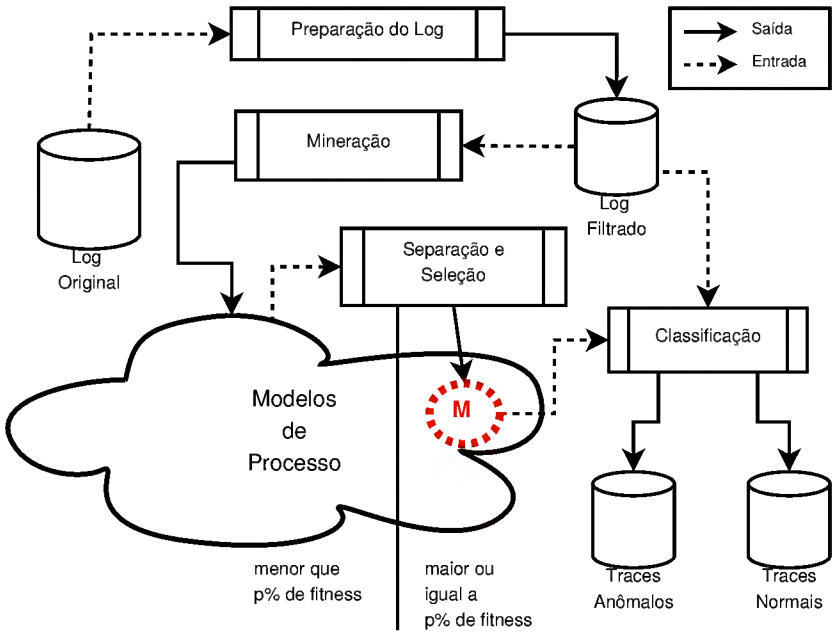


Figura 5.1: Visão geral da abordagem.

Seleção do modelo. Trata-se da fase que, após a separação dos modelos, escolhe aquele modelo mais apropriado para a fase seguinte, de classificação dos *traces*;

Classificação dos *traces*. Trata-se da última fase, quando ocorre a classificação dos *traces* do *log* em dois grupos: os *traces* normais e os *traces* anômalos.

5.2 Aplicação do ProM

O *Framework ProM* é um ambiente extensível para *process mining* formado por vários *plug ins*[49]. Como é implementado com a linguagem Java, é um ambiente independente de plataforma, além de ser uma solução *open-source*. É um ambiente de mineração que aceita diferentes formatos de entrada de dados, além de gerar modelos e novos dados em diferentes formatos.

Por ser um ambiente extensível, novos *plug ins* de *process mining* podem ser desenvolvidos e baseados nos *plug ins* do próprio ambiente, que atualmente abriga mais de 280 *plug ins*.

As ferramentas de *process mining* disponíveis no ProM apóiam a análise de processos em três diferentes perspectivas:

Perspectiva de Processo. Responsável pela mineração do fluxo de controle do processo.

Perspectiva Organizacional. Responsável pela mineração dos participantes (pessoas, papéis, relação de poder, etc.) da execução do processo.

Perspectiva de Caso ou Dados. Responsável pela mineração dos dados e informações manipuladas pelo processo.

As abordagens de detecção de anomalias desenvolvidas neste trabalho estão limitadas à análise do fluxo de controle de um *trace*, portanto, estamos interessados nos *plug ins* do *Framework ProM* que lidam com a perspectiva de processo. Nesta seção, mostramos como o *Framework ProM* pode ser utilizado para apoiar a detecção de *traces* anômalos a partir da seleção do modelo mais apropriado (ver Seção 5.2.3). O objetivo é apontar quais recursos do ProM podem ser utilizados diretamente em cada uma das etapas, bem como indicar limitações do uso direto do ProM, sugerindo outras formas de operação das ferramentas disponíveis.

5.2.1 Etapa 1: Preparação do *Log*

O primeiro passo do processo de detecção de anomalias está preocupado com a remoção de *traces* e atividades do *log* que não são interessantes para análise ou que podem ser claramente classificados como anomalias, por exemplo, a presença de um *trace* incompleto. Esta etapa é responsável por gerar o que denominamos de *log filtrado*(ver Definição 3).

O *Framework ProM* tem várias ferramentas para filtrar o *log* que podem ser aplicadas nesta etapa. Por exemplo, é possível indicar quais são as atividades de início e fim dos *traces* do *log*, então o *log filtrado* conteria apenas os *traces* começam e terminam com as atividades selecionadas.

O ProM também fornece ferramentas para avaliar a frequência das atividades no *log*. Dessa forma, é possível realizar uma análise baseada apenas em *traces* frequentes ou remover do *log* os *traces* infrequentes. Além dos filtros básicos, o ambiente do ProM também fornece uma ferramenta de análise chamada *LTL Checker*, que pode ser usada para filtrar os *traces* que satisfaçam determinadas propriedades, representadas através de uma linguagem declarativa de definição de restrições. Por exemplo, com o *LTL Checker* é possível filtrar os *traces* que satisfaçam uma determinada relação causal entre duas atividades ou mais atividades.

5.2.2 Etapas 2 e 3: Mineração e Separação dos Modelos de Processo

As duas próximas etapas da detecção de anomalias são a mineração dos modelos de processo e separação dos modelos que satisfazem a um determinado critério. A etapa de

mineração lida com a construção automática de um modelo de processo a partir do *log* informado, enquanto que a etapa de separação está relacionada com a seleção dos modelos que satisfazem a restrição de *fitness* mínimo (o valor do parâmetro p na Definição 9).

A etapa de mineração é facilmente suportada pelo *Framework ProM*, que possui diversos algoritmos de mineração/descoberta de processo disponíveis, por exemplo, o $\alpha - algorithm$[44, 46], o algoritmo $\alpha^{++} - algorithm$[54], o *multi-phase miner*[50], o *heuristic miner*[53], o *genetic miner*[16, 19], entre outros.

A função de *fitness do log* (Definição 5), apesar de diretamente disponível no *Framework ProM* através da métrica *PM* (*Parsing Measure*), contida no *plug in* chamado *control-flow benchmark*, é limitada a modelos de processo que são representados como uma *heuristic net*[53, 16, 19] (um modelo específico de representação de processos). Portanto, a aplicação dessa métrica é restrita, especialmente no contexto deste trabalho, que a maioria dos algoritmos de mineração geram *petri nets* (outro modelo de representação de processos) e não há no *Framework ProM* nenhum *plug in* que converta uma *petri net* em uma *heuristic net*.

Outra limitação do ProM é no teste de conformidade entre um *trace* e um modelo, como apresentado na Definição 4, pois essa métrica não é suportada diretamente pelo ProM. No entanto, o ambiente fornece um teste de conformidade de granularidade mais fina, pois a função de *fitness* disponível no *plug in conformance checker*[49] considera a perspectiva de atividade, indicando, por exemplo, que algumas atividades do *trace* não podem ser executadas.

Para contornarmos essas limitações, sem deixar de explorar toda a riqueza de ferramentas disponível no *Framework ProM*, adotamos o *plug in conformance checker* da seguinte forma, conforme já definido no Capítulo 3 (Definições 4 e 5):

Fitness do *trace*. Um *trace* com *fitness* de valor 1 (100%) é um *trace* que pode ser executado completamente pelo modelo, portanto, é instância do modelo. *Traces* com *fitness* menor do que 1, não são instância do modelo.

Fitness do *log*. É a razão entre o número de *traces* com *fitness* de valor 1 (100%) e a quantidade de *traces* no *log*. Operacionalmente, através da interface do *plug in*, podemos solicitar a seleção apenas dos *traces* que são instâncias do modelo, então é possível visualizar a porcentagem de *traces* selecionados.

5.2.3 Etapa 4: Seleção do modelo mais apropriado

A seleção do modelo mais apropriado é a quarta etapa da detecção de anomalias, onde o termo *modelo mais apropriado* significa um modelo que seja o mais simples e não genérico, dentre os modelos disponíveis para seleção. Para tanto, a seleção deve considerar

a escolha do modelo que possui maior valor para a função *appropriateness* (ver Seção 3.4, Definição 8).

Porém, não há no *Framework ProM* nenhuma função que diretamente satisfaça essa definição. Então, semelhante à função de avaliação do *fitness* do *trace* e do *log*, foi necessário explorar alternativas no ProM que pudessem ajudar na elaboração de uma função mais objetiva.

Há no ambiente ProM funções de avaliação de modelo de processo que medem o grau de complexidade e o grau de generalidade de um modelo (capacidade de prever *traces* não observados no *log*). Essas funções estão disponíveis tanto no *plug in* chamado *conformance checker*, como no *plug in* chamado *control-flow benchmark*, citados na Seção 5.2.2. Então decidimos utilizar essas funções para compor uma definição objetiva da função *appropriateness*, apresentada no Capítulo 3 (Definição 8). Por isso, apresentamos na Equação 5.1 uma definição dessa função, cujo valor é obtido utilizando as funções disponíveis no ProM como segue:

- utilizando a métrica chamada *structural appropriateness*, que avalia o grau de complexidade de um modelo, representada aqui através da função $f_S(M)$, onde M é um modelo;
- utilizando a métrica chamada *behavioral appropriateness*, que avalia o grau de generalidade de um modelo, representada aqui através da função $f_B(M, L)$, onde M é um modelo e L é um *log*;
- finalmente, como as funções possuem o mesmo contra-domínio ($[0, 1]$), nós definimos a *função appropriateness* como sendo o balanço entre adequação estrutural (métrica *structural appropriateness*) e comportamental (métrica *behavioral appropriateness*), como segue:

$$a(M, L) = \frac{f_S(M) + f_B(M, L)}{2} \tag{5.1}$$

5.2.4 Etapa 5: Classificação dos *Traces*

A partir da seleção do modelo mais apropriado, a última etapa da detecção de anomalias é a classificação dos *traces*. Para tanto, utilizamos novamente o *plug in* que testa o grau de conformidade de um *trace* com um modelo, o *conformance checker*.

Com a interface de operação deste *plug in* podemos selecionar apenas os *traces* que são instâncias do modelo (*fitting traces*). Como estamos interessados na seleção dos *traces* que são anômalos, é possível realizar na interface de operação do *plug in* a operação *inverter seleção*. Finalmente podemos visualizar a porcentagem de *traces* selecionados.

5.3 Estudo de Caso

Nesta seção apresentamos uma aplicação real da abordagem de detecção de anomalias, diretamente apoiada pelo framework ProM. O *log* utilizado neste exemplo é de um sistema de informação municipal da Holanda. O processo representado no *log* é referente ao suporte que os municípios oferecem a pessoas que necessitam de cadeiras de rodas, patinete para portadores de deficiência, adaptação nas casas (ex. elevador), ajuda familiar, etc.

A seguir, listamos as propriedades do *log* utilizado neste exemplo:

- é de informações referente ao período de janeiro de 2007 a agosto de 2008;
- contém informações de 876 *traces*;
- possui 5497 atividades, somando-se as atividades de cada *trace*;
- há dez diferentes classes de atividade;
- o tamanho do *trace* mais curto foi de uma atividade;
- o tamanho do *trace* mais longo foi de 12 atividades;
- o tamanho médio dos *traces* foi de seis atividades.

Então, considerando que vários (talvez infinito) modelos podem ser descobertos a partir do *log* e considerando a ausência no ProM de uma ferramenta que construa todos os modelos candidatos a análise/seleção do mais apropriado, optamos por explorar a descoberta dos modelos de uma forma semi-automática. Por essa razão, nas seções seguintes, descrevemos os parâmetros e operações utilizadas para gerar o modelo que satisfizesse os critérios de seleção do modelo mais apropriado, dentre os modelos explorados na análise.

5.3.1 Etapa 1: Preparação do *Log*

Essa preparação do *log* claramente é dependente do negócio, mas uma análise da frequência dos *traces* e das atividades ajuda muito na escolha dos filtros que podem ser aplicados. Então, para o *log* utilizado, aplicamos a análise de frequência das atividades que iniciam e terminam um *trace*. Apresentamos na Tabela 5.1 as informações de frequência para o *log* utilizado nesta análise, cujos valores foram obtidos com o framework ProM.

Essa análise da frequência é importante, pois, como argumentado na Seção 3.4, dependendo do período utilizado para obter o *log*, alguns *traces* podem ter iniciado ou começado com atividades intermediárias.

Por essa razão, após a análise da frequência e apoio dos usuários do sistema, aplicamos os seguintes filtros no *log*:

- definir "Request registration" como a única atividade inicial, já que ela é muito predominante no *log* (ver Tabela 5.1);
- definir "Final Phase" como a única atividade final, já que ela é muito predominante também (ver Tabela 5.1);

Tabela 5.1: Frequência das atividades que iniciam e terminam os *traces* do *log*

Frequência das atividades iniciais		Frequência das atividades finais	
Atividade	Frequência	Atividade	Frequência
Request registration	96,12%	Final Phase	94,52%
Reporting & Decision	3,43%	Reporting & Decision	2,06%
Private research	0,34%	Request registration	1,03%
Research	0,11%	Left filing	0,91%
		Keys and decide	0,69%
		Accounting	0,34%
		Waiting recovery	0,23%
		Research	0,11%
		Return	0,11%

Após a aplicação dos filtros nós obtivemos um *log filtrado* com as seguintes características:

- 796 *traces*;
- 5191 atividades, somando-se as atividades de cada *trace*;
- o tamanho do *trace* mais curto foi de cinco atividades;
- o tamanho do *trace* mais longo foi de 12 atividades;
- o tamanho médio dos *traces* foi de seis atividades;

5.3.2 Etapas 2, 3 e 4: Mineração, Separação e Seleção do Modelo

Apesar do título desta seção sugerir que vários modelos foram minerados por diferentes algoritmos, de fato um modelo de processo foi interativamente minerado até que satisfizesse a propriedade de ser o modelo mais apropriado, dentre os modelos explorados. Sem perda de valor, o nome atribuído à seção tem o objetivo didático de manter um alinhamento/correspondência com as etapas da abordagem de detecção de anomalia propostas neste capítulo.

Na etapa de mineração, utilizamos o algoritmo de mineração *heuristic miner*[53], por ser um algoritmo robusto a *logs* com ruídos e exceções, já que considera a frequência das relações entre as atividades para encontrar um modelo de processo. Este algoritmo está

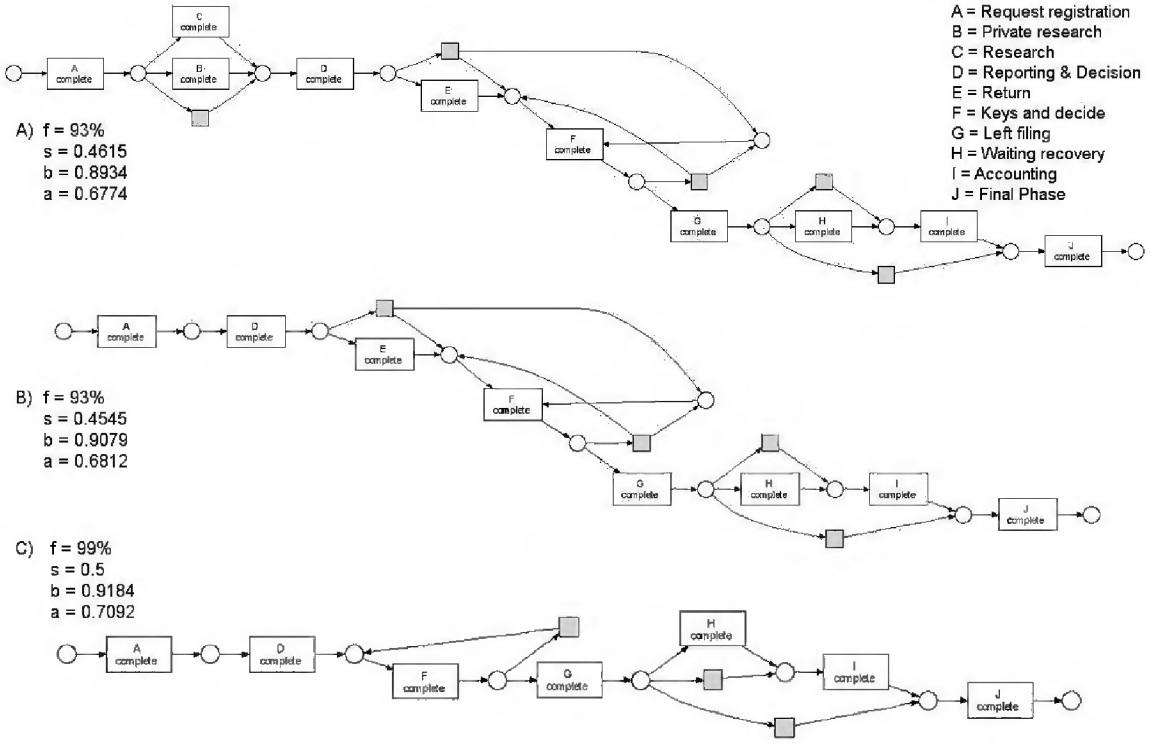


Figura 5.2: Modelos de processo (*Petri net*), após aplicação de filtros.

disponível no ProM através de um *plug in* de mesmo nome. Além disso, nós assumimos para o parâmetro p, *fitness* mínimo, um valor de 80%. Esse valor representa o mínimo de conformidade que esperamos que os modelos explorados no exemplo possuam. Por seu um parâmetro, valores mais ou menos restritivos de conformidade poderiam ser utilizados.

Apresentamos na Figura 5.2 os três modelos de processos que foram iterativamente minerados a partir do *log filtrado*. Para cada modelo, apresentamos as seguintes propriedades: **f** para *fitness*, **s** para *structural appropriateness*, **b** para *behavioral appropriateness*, e **a** for *appropriateness*.

O primeiro modelo de processo encontrado foi a *petri net* A. Entretanto, analisando esse modelo e a frequência das atividades no *log*, que apresentamos na Tabela 5.2, podemos perceber que as duas atividades mais infrequentes, "Private research" (B) e "Research" (C), adicionam, desnecessariamente, complexidade ao modelo, mesmo sendo atividades muito infrequente, quando comparadas com as outras. Então, aplicamos um segundo filtro ao *log* removendo essas atividades, cujo *log* resultante foi utilizado para gerar o segundo modelo, a *petri net* B.

A *petri net* B, apresentada na Figura 5.2, é um modelo que satisfaz o critério de *fitness* e também é mais apropriado que a *petri net* A. No entanto, novamente analisando a frequência das atividades no *log*, podemos notar que a atividade "Return" (E), mesmo

Tabela 5.2: Frequência das Atividades no *Log*.

Atividade	Frequência (relativa)
Keys and decide	16,41%
Reporting & Decision	15,43%
Left filing	15,39%
Request registration	15,33%
Final Phase	15,33%
Accounting	11,42%
Waiting recovery	9,59%
Return	1,00%
Private research	0,04%
Research	0,04%

sendo significativamente mais frequente que as atividades já removidas, ainda acrescenta complexidade ao modelo desnecessariamente. Quando comparada a frequência de "Return" com as outras atividades, "Return" ainda é muito infrequente.

Por essa razão, a atividade "Return" foi também removida do *log* e a *petri net* C foi gerada, que além de ser um modelo mais apropriado que os outros modelos, possui um *fitness* superior. Então a *petri net* C foi selecionada como o modelo mais apropriado, servindo como classificador dos *traces* do *log*.

Novamente vale observar que todo essa processo de mineração, separação e seleção dos modelos não foi executado de forma automatizada e nenhuma busca exaustiva de novos modelos foi realizada. Porém, a inspeção manual indica que o modelo selecionado é de fato o mais apropriado entre os modelos explorados.

5.3.3 Etapa 5: Classificação dos *Traces*

Por fim, nós obtivemos os *traces* que são instância do modelo apropriado e os *traces* que não são instância desse modelo (*Petri net* C na Figura 5.2). Como na análise utilizada consideramos que o valor do atributo p era de 80% (mínimo *fitness* do *log*), o máximo de *traces* anômalos que esperávamos encontrar no *log* era de 20%.

No entanto, o modelo mais apropriado após a análise tem *fitness* de 99%. Então, foram detectados apenas seis *traces* anômalos de um total de 796 *traces* existentes no *log* filtrado.

5.4 Considerações Finais

A abordagem de detecção de anomalia proposta neste capítulo considera a busca de um modelo que seja *apropriado* – que satisfaça um valor mínimo de *fitness*, mas que maximize a função *appropriateness* (Seção 3.4, Definição 8). No entanto, nenhum mecanismo auto-

mático que suporte o modelo de detecção proposto neste capítulo foi desenvolvido, mas o estudo de caso com dados reais apresentado na Seção 5.3 representa uma boa aproximação de como um componente automático poderia funcionar.

Além disso, vale ressaltar que este modelo de detecção parece mais um problema de mineração de processo, do que, diretamente, um problema de detecção de anomalias no *log*. Por exemplo, o *genetic process mining* também realiza uma busca por um modelo, mas sua função de seleção do modelo de processo é diferente da função *appropriateness* descrita na Definição 8.

Capítulo 6

Conclusões

Modelos mais recentes de gestão, que incluem a adoção de práticas rigorosas de governança corporativa, estimularam a implantação de PAIS (*Process Aware Information Systems*), a fim de automatizar e controlar seus processos de negócio[22]. Uma das vantagens da adoção de tais sistemas é a possibilidade de rastrear desvios operacionais (por exemplo, escândalos financeiros relacionados com a má gestão).

No entanto, o controle fornecido por sistemas normativos pode comprometer a flexibilidade necessária para as empresas serem ágeis e competitivas no mercado. Por essa razão, é necessário desenvolver soluções capazes de balancear esses requisitos conflitantes: flexibilidade e segurança.

Esta tese apresentou dois modelos de detecção de *traces* anômalos, que quando integrados a um PAIS, podem ajudar na implantação de sistemas de informação mais flexíveis e seguros. Por exemplo, a detecção de *traces* anômalos pode sugerir uma investigação, no caso de fraude, ou uma adaptação do modelo de processo, no caso de uma exceção. Na Seção 6.1 citamos esses dois modelos, destacando as contribuições da tese, bem como apresentamos algumas considerações finais sobre os resultados dessas contribuições. Finalmente, na Seção 6.2 apresentamos algumas sugestões de trabalhos futuros.

6.1 Contribuições

O primeiro modelo de detecção, apresentado no Capítulo 4, considera como anômala as instâncias do *log* que têm menor conformidade com o modelo de processo descoberto, ou seja, são as instâncias do *log* que necessitam que o modelo de processo descoberto sofra mais modificações para acomodar essas instâncias (*fitness=1*). Para esse modelo de detecção foram desenvolvidos três algoritmos: *Sampling*, *Threshold* e Iterativo.

Apresentamos na Seção 4.3 uma avaliação rigorosa da eficácia desses algoritmos, realizada com 1800 *logs* sintéticos. Essa avaliação também incluiu na comparação uma

abordagem de detecção ingênua, que classificava automaticamente como anômalo todos os *traces* candidatos (*traces* com frequência $\leq 2\%$). Após a avaliação notamos que o Algoritmo *Sampling* é o melhor entre os algoritmos propostos, com eficácia média de: *F-Measure*$= 0,753$; Acurácia$= 0,787$; e *Recall*$= 0,996$. Os parâmetros de execução do Algoritmo *Sampling* são um algoritmo de *process discovery* e um fator de *sampling*. No caso, os parâmetros utilizados na avaliação foram o *Heuristic Miner* como algoritmo de *process discovery*, com fator de *sampling* de 70%.

Além disso, percebemos que a eficácia do Algoritmo *Sampling* é melhor quando o número de classes de *trace* anômalo é maior no *log* e quando o número de *traces* candidatos a anômalo é menor. Esse comportamento pode estar associado aos seguintes fatores:

- algoritmos de *process discovery* do *framework* ProM raramente encontram modelos de processo que descrevem completamente todos os *traces* utilizados em sua construção; assim, raramente os modelos construídos a partir do *sampling* do *log* executarão completamente um *trace* candidato sob análise, pois dificilmente faria parte do *sampling* utilizado para criar o modelo, diminuindo ainda mais a chance do *trace* ser completamente executado pelo modelo descoberto;

- relacionado ao fator acima, como os modelos raramente executam completamente até mesmo os *traces* utilizados em sua construção, provavelmente a maioria dos *traces* candidatos serão classificados como anômalos, especialmente os *traces* anômalos, que são *traces* com características estruturais bem diferentes dos *traces* normais – fato que explicaria a elevada taxa de *Recall*$= 0,996$ e o aumento da eficácia do algoritmo quando aumenta o número de classes de *trace* anômalo no *log*;

- também relacionado ao primeiro fator, quando o *log* possui mais candidatos, significa que o *log* tem mais *traces* normais candidatos, pelo menos para os *logs* utilizados na avaliação; assim, provavelmente mais *traces* normais serão classificados como anômalo, diminuindo a acurácia e o valor do *precision*, que é utilizado para calcular o *F-Measure*.

O segundo modelo de detecção, apresentado no Capítulo 5, considera como anômala as instâncias do *log* que não podem ser completamente executadas pelo modelo de processo mais apropriado (modelo que maximiza uma função e que satisfaz certas propriedades). Nenhum algoritmo foi desenvolvido para esse modelo, mas desenvolvemos um estudo de caso com dados reais obtidos de um serviço municipal de Eindhoven, na Holanda. Por essa razão não realizamos uma estudo mais objetivo que comparasse os dois modelos de detecção.

6.2 Trabalhos Futuros

Process Mining é uma especialidade da área de mineração de dados preocupada em descobrir informações sobre processos de negócios. Essa descoberta de informação pode estar contextualizada em três perspectivas: fluxo, caso e organizacional. A perspectiva de *fluxo* diz respeito à descoberta das atividades que fazem parte do modelo de processo, além da ordem de execução dessas atividades. O perspectiva de *caso* diz respeito a descoberta dos dados gerados e manipulados pelo processo, enquanto a perspectiva *organizacional* diz respeito a descoberta dos papéis e recursos responsáveis pela execução do processo.

Nos últimos anos vários algoritmos de descoberta de processo (*process discovery*) foram desenvolvidos [2, 14, 20, 44, 50, 46, 45, 49, 51, 25, 4]. No entanto, até mesmo pela variedade de opções de algoritmos disponíveis, não há uma definição precisa do que é um bom modelo de processo que descreva o *log* utilizado para construí-lo. Por essa razão, alguns trabalhos questionam essa imprecisão na definição do problema de *process discovery*[51] e propõem esforços em abordagens alternativas, por exemplo, através do *probabilistic process mining*[4].

6.2.1 Process Mining: Fluxo, Caso e Organizacional

Esta tese explorou o problema da detecção de anomalia como sendo um problema de análise do fluxo de execução dos *traces* do *log*. Apesar de ser um abordagem simples, válida e razoável, também pode ser uma estratégia de detecção limitada, especialmente em cenários muito dinâmicos, onde os *traces* são únicos ou ocorrem com baixa frequência (ex. atendimento de emergência dos hospitais).

Por essa razão, acreditamos que a anomalia pode ser identificada através da análise de outras perspectivas do *log*, como a perspectiva de dados e a perspectiva organizacional. Por exemplo, a fraude pode seguir um fluxo normal, mas com atividades que realizem a produção de dados anômalos, como elevadas quantias de dinheiro ou muitas transações fora do valor médio das operações; ou então, atividades que são executadas por papéis ou usuários não autorizados, como na violação do "princípio dos quatro olhos". Por exemplo, um algoritmo que considere as várias perspectivas pode classificar o *trace* como anômalo ao considerar a análise de fluxo de controle, mas sob a perspectiva organizacional ou de dados (caso), esse *trace* pode ser normal.

Portanto, acreditamos em pesquisas que explorem isoladamente ou em conjunto as perspectiva de dados e organizacional, que adicionariam maior precisão ao processo de detecção das instâncias anômalas no *log*. Assim, podemos prever que o desenvolvimento de abordagens que combinem essas diferentes perspectivas de *process mining* exigirão heurísticas de detecção mais refinadas, um desafio para aqueles interessados em colaborar com a área.

O ambiente *ProM* possui um conjunto de ferramentas de mineração que não foram avaliadas. Como acreditamos que a eficácia da detecção depende da eficácia da descoberta (ou seleção) de um modelo de processo a partir de uma ferramenta de mineração, consideramos como trabalho futuro a avaliação de outras ferramentas de mineração existentes nesse ambiente.

6.2.2 *Process Discovery*

Todos os algoritmos de detecção de anomalias apresentados nesta tese consideram *process discovery* como um elemento essencial para a eficácia da detecção, pois ajuda na descoberta de um modelo de processo que será utilizado como um instrumento auxiliar na classificação do *trace* como normal ou anômalo.

Dos algoritmos de *process discovery* disponíveis, o único que consegue garantidamente encontrar um modelo de processo com *fitness* - que é a proporção de *traces* completamente instanciáveis pelo modelo - igual a 100% é o *genetic process mining*[16]. Entretanto, seu tempo de execução é proibitivo, já que seu processamento pode durar horas, dias ou meses para encontrar um modelo com 100% de *fitness*. Assim, trabalhos futuros poderiam explorar a criação de algoritmos de *process discovery* mais eficientes e ao mesmo tempo eficazes.

Nesse contexto, o modelo de processo probabilístico[4] pode ser uma boa alternativa, não apenas como um instrumento de descoberta de processo, mas também como um detector de anomalias. Nesse caso, o modelo de processo descoberto não necessariamente representa todas as instâncias observadas no *log*. Então, as instâncias que não são executadas pelo modelo seriam classificadas como anômalas.

Outra possibilidade é desenvolver uma extensão do *genetic process mining* de modo a apoiar o processo de descoberta e seleção do "melhor" modelo de processo. A versão original do *genetic process mining*, como apresentada em [16, 19], utiliza um algoritmo genético para buscar um indivíduo (modelo de processo) com melhor *fitness*. Diferentemente, o modelo de detecção de anomalias apresentado no Capítulo 5 considera a seleção do modelo mais apropriado como forma de classificar os *traces* do *log* como normal ou anômalo. Para tanto, assume a existência de um modelo com um grau mínimo de *fitness*, mas que seja o modelo mais apropriado (onde apropriado é uma métrica objetivamente definida). Assim, o *genetic process mining* poderia ser modificado da seguinte forma: (i) buscar indivíduos que satisfazem o grau mínimo de *fitness*, mas (ii) preferir aquele que seja mais apropriado (Definição 8).

6.2.3 Métricas de avaliação

Os algoritmos de detecção *threshold* e iterativo utilizam uma métrica de avaliação do modelo para medir o grau de conformidade entre um *log* e um modelo de processo. Essas métricas de avaliação são utilizadas desde o projeto inicial, que considera o cálculo do custo de inclusão, até o projeto adaptado, que utiliza métricas como *fitness* e *behavioral appropriateness*. Assim, semelhante ao componente de *process discovery*, o componente de avaliação de modelos também é importante para melhorar a eficácia dos métodos de detecção.

Portanto, trabalhos futuros podem considerar o desenvolvimento de extensões das métricas existentes ou mesmo o desenvolvimento de novas métricas poderá ajudar o processo de detecção de anomalias. Essa necessidade é bem evidente se considerarmos a definição de modelo apropriado apresentada na Seção 3.4 (Definição 8). Apesar de ser uma definição bastante razoável, pois representa um balanço entre a preferência por modelos compactos e específicos, essa definição é preliminar. Portanto, uma abordagem mais refinada pode ser desenvolvida para avaliar o grau de adequação de um modelo para um *log*.

6.2.4 Áreas de interesse

Este projeto possui uma identidade com a área de *data mining*, pois propõe novos métodos ou algoritmos para descoberta de informação em um conjunto de dados. No entanto, considerando a perspectiva da sua aplicação prática em segurança de sistemas de apoio a processos de negócios, esse projeto está relacionado com as áreas de *Segurança* e *BPM* (*Business Process Management*). Tal multidisciplinaridade pode ser vista tanto como um "problema", como uma oportunidade. Um "problema" porque exige diferentes formatos de apresentação das idéias, um para cada comunidade. Por outro lado, é uma oportunidade, pois podemos produzir e colaborar com diferentes grupos de pesquisadores.

Considerando a relevância prática deste projeto, acreditamos que a capacidade de detectar eventos anômalos em *logs* gerados por sistemas de informação é fundamental ao apoio de processos de negócios flexíveis. Assim, através da detecção de eventos anômalos é possível identificar tentativas de fraude, ou como as exceções são executadas, apoiando a modelagem ou re-engenharia dos processos organizacionais. A detecção de eventos anômalos também pode apoiar o processo de mineração, eliminando ruídos no *log* através de um pré-processamento.

Referências Bibliográficas

[1] Deepak K. Agarwal. An empirical bayes approach to detect anomalies in dynamic multidimensional arrays. In *ICDM*, pages 26–33, 2005.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *EDBT '98: Proceedings of the 6th International Conference on Extending Database Technology*, pages 469–483, London, UK, 1998. Springer-Verlag.

[3] Fábio Bezerra and Jacques Wainer. Selecting anomalous traces from workflow event logs. In Alberto Barbosa Raposo and Flávia Maria Santoro, editors, *III Simpósio Brasileiro de Sistemas Colaborativos*, pages 98–106, Natal, RN, Brazil, November 2006. SBC.

[4] Fábio Bezerra and Jacques Wainer. First steps towards probabilistic process mining. In *XXII Simpósio Brasileiro de Banco de Dados - Sessão de Pôsteres*, João Pessoa, PB, Brazil, October 2007. SBC.

[5] Fábio Bezerra and Jacques Wainer. Towards detecting fraudulent executions in business process aware systems. In *WfPM 2007 - Workshop on Workflows and Process Management*, Timisoara, Romania, September 2007. In conjunction with SYNASC 2007.

[6] Fábio Bezerra and Jacques Wainer. Um método de detecção de anomalias em logs de processos de negócios. In Maria Beatriz Felgar de Toledo and Edmundo Mauro Madeira, editors, *I Brazilian Workshop on Business Process Management*, Gramado, RS, Brazil, 2007. SBC. In Conjunction with Webmedia 2007.

[7] Fábio Bezerra and Jacques Wainer. Anomaly detection algorithms in business process logs. In *10th International Conference on Enterprise Information Systems*, pages 11–18, Barcelona, Spain, June 2008.

[8] Fábio Bezerra and Jacques Wainer. Anomaly detection algorithms in logs of process aware systems. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 951–952, New York, NY, USA, 2008. ACM.

[9] Fábio Bezerra and Jacques Wainer. Auditing workflow logs for fraud detection. In *Proceedings of the KDD 2008 Workshop on Data Mining for Business Applications*, Las Vegas, NV, 2008. ACM. In conjunction with KDD'08.

[10] Fábio Bezerra and Jacques Wainer. Fraud detection in process aware systems. In *II Brazilian Workshop on Business Process Management*, 2008.

[11] Fábio Bezerra, Jacques Wainer, and W. M. P. van der Aalst. Anomaly detection using process mining. In *Enterprise, Business-Process and Information Systems Modeling*, volume 29, pages 149–161, Amsterdam, The Netherlands, April 2009. Springer Berlin Heidelberg. 10th International Workshop on Business Process Modeling, Development and Support.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):1–58, 2009.

[13] Jonathan E. Cook, Zhidian Du, Chongbing Liu, and Alexander L. Wolf. Discovering models of behavior for concurrent workflows. *Computers in Industry*, 53(3):297–319, 2004.

[14] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, Vol. 7(3):p. 215–249, 1998.

[15] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the a algorithm to mine short loops. In *BETA Working Paper Series*, Eindhoven, 2004. Eindhoven University of Technology.

[16] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining: A basic approach and its challenges. In *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 203–215, Nancy, France, September 2006. ISBN 978-3-540-32595-6.

[17] A.K. Alves de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Quantifying process equivalence based on observed behavior. *Data & Knowledge Engineering*, 64(1):55–74, January 2008.

[18] A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters. Process mining for ubiquitous mobile systems: An overview and a concrete algorithm. In Luciano Baresi, Schahram Dustdar, Harald Gall, and Maristella Matera, editors, *Ubiquitous Mobile Information and Collaboration Systems*, volume 3272 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin - Heidelberg, 2005.

[19] Ana Karla Alves de Medeiros. *Genetic Process Mining.* PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2006. ISBN 978-90-386-0785-6.

[20] Ana Karla Alves de Medeiros, Wil M. P. van der Aalst, and A. Weijters. Workflow mining: Current status and future directions. In R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems*, volume 2888 of *LNCS*, 2003.

[21] Steve Donoho. Early detection of insider trading in option markets. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, New York, NY, USA, 2004. ACM Press.

[22] Marlon Dumas, Wil van der Aalst, and Arthur ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology.* Wiley, 2005. ISBN 13 978-0-471-66306-5.

[23] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, Vol. 1:p. 291–316, 1997.

[24] A. Göker and J. Davies. *Information Retrieval: Searching in the 21st Century*, chapter 8, pages 159–179. Wiley, 2009.

[25] Markus Hammori, Joachim Herbst, and Niko Kleiner. Interactive workflow mining - requirements, concepts and implementation. *Data & Knowledge Engineering*, 56(1):41–63, January 2006.

[26] Joachim Herbst and Dimitris Karagiannis. Workflow mining with inwolve. *Computers in Industry*, 53(3):245–264, 2004.

[27] Ron Larson and Elizabeth Farber. *Elementary Statistics: Picturing the World.* Prentice Hall, second edition, March 2003. ISBN-13: 978-0130655950.

[28] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, 2001.

[29] Jiafei Li, Dayou Liu, and Bo Yang. Process mining: Extending a algorithm to mine duplicate tasks in process logs. In Kevin Chen-Chuan Chang, Wei Wang, Lei Chen, Clarence A. Ellis, Ching-Hsien Hsu, Ah Chung Tsoi, and Haixun Wang, editors, *Advances in Web and Network Technologies, and Information Management*, August 2007.

[30] Manual merck de saúde para a família. Internet, Dezembro 2007. http://www.manualmerck.net/.

[31] Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, New York, NY, USA, 2003. ACM Press.

[32] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 201–210, New York, NY, USA, 2007. ACM Press.

[33] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448 – 3470, 2007.

[34] Shlomit S. Pinter and Mati Golani. Discovering workflow models from activities' lifespans. *Computers in Industry*, 53(3):283–296, 2004.

[35] A. Rozinat and Wil M. P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Business Process Management Workshops*, pages 163–176, 2005.

[36] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, March 2008. ISSN 0306-4379.

[37] Robin Sabhnani, Daniel Neill, and Andrew Moore. Detecting anomalous patterns in pharmacy retail data. In *Proceedings of the KDD 2005 Workshop on Data Mining Methods for Anomaly Detection*, August 2005.

[38] Guido Schimm. Mining exact models of concurrent workflows. *Comput. Ind.*, Vol. 53(3):p. 265–281, 2004.

[39] W.J. Steinberg. *Statistics Alive!*, chapter 26 - Tukey HSD Test, pages 318–324. SAGE Publications, second edition, 2010.

[40] Wil M.P. van Aalst, Kees M. van Hee, Jan Martijn van Werf, and Marc Verdonk. Auditing 2.0: Using process mining to support tomorrow's auditor. *Computer*, 43:90–93, 2010.

[41] Wil van der Aalst, Kees van Hee, Jan Martijn van der Werf, Akhil Kumar, and Marc Verdonk. Conceptual model for online auditing. *Decision Support Systems*, 50(3):636 – 647, 2011. On quantitative methods for detection of financial fraud.

[42] Wil M. P. van der Aalst. Business alignment: using process mining as a tool for delta analysis and conformance testing. *Requirements Eng.*, 10(3):198–211, August 2005.

[43] Wil M. P. van der Aalst and Ana Karla A. de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121(4):3–21, February 2005.

[44] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, November 2003.

[45] Wil M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, April 2004.

[46] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004.

[47] W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Process equivalence in the context of genetic mining. Research Note BPM-06-15, BPM Center Report, 2006.

[48] Wil M.P. van der Aalst Minseok Song. Mining social networks: Uncovering interaction patterns in business processes. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management: Second International Conference*, volume 3080 of *LNCS*, pages pp. 244 – 260, 2004.

[49] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer Berlin / Heidelberg, 2005. ISBN978-3-540-26301-2.

[50] B.F. van Dongen and W.M.P. van der Aalst. Multi-phase process mining: Building instance graphs. In *Conceptual Modeling - ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer Berlin / Heidelberg, 2004.

[51] J. Wainer, K. Kim, and C. A. Ellis. A workflow mining method through model rewriting. In H. Fuks, S. Lukosch, and A. C. Salgado, editors, *Groupware: Design, Implementation, and Use: 11th International Workshop*, volume 3706, pages p. 184–191, Porto de Galinhas, Brazil, Setembro 2005. CRIWG 2005.

[52] A.J.M.M. Weijters and W.M.P van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[53] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros;. Process mining with the heuristicsminer algorithm. Technical report, Beta Research School for Operations Management and Logistics, 2006. ISBN 90-386-0813-6 / 978-90-386-0813-6.

[54] Lijie Wen, Jianmin Wang, and Jiaguang Sun. Detecting implicit dependencies between tasks from event logs. In Xiaofang Zhou, Jianzhong Li, Heng Shen, Masaru Kitsuregawa, and Yanchun Zhang, editors, *Frontiers of WWW Research and Development - APWeb 2006*, volume 3841 of *Lecture Notes in Computer Science*, pages 591–603. Springer Berlin - Heidelberg, 2006. 10.1007/11610113_52.