Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

XChangeMining: Mineração de dados em documentos XML

Samuel Mendes Vieira

JUIZ DE FORA
NOV, 2014

XChangeMining: Mineração de dados em documentos XML

Samuel Mendes Vieira

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Alessandreia Marta de Oliveira

JUIZ DE FORA

NOV, 2014

XCHANGEMINING: MINERAÇÃO DE DADOS EM DOCUMENTOS XML

Samuel Mendes Vieira

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Alessandreia Marta de Oliveira
M. Sc. (UFRJ)

Luciana Conceição Dias Campos
D. Sc. (PUC - RJ)

Victor Stroele de Andrade Menezes
D. Sc. (UFRJ)

JUIZ DE FORA

19 DE NOV, 2014

Resumo

Documentos XML estão cada vez mais sendo utilizados no armazenamento e troca de informações nos dias de hoje. Isso gera um grande volume de dados ao longo do tempo devido à característica dinâmica deste tipo de documento, que possui grande flexibilidade na representação dos dados. Para aproveitar esses dados e obter informações úteis sobre a evolução das versões dos documentos, torna-se necessária a adaptação de técnicas existentes para realizar análises mais profundas e complexas. As regras de associação da mineração de dados vêm sendo muito empregadas na descoberta de informações em grandes bases de dados. Diferentes abordagens vêm sendo propostas neste sentido e algumas são apresentadas neste trabalho. Além disso, esta monografia apresenta uma abordagem para realizar a mineração de regras de associação a partir do histórico de mudanças de documentos XML, denominada XChangeMining. A partir de um número de versões de um documento XML, as diferenças entre as versões são detectadas e transformadas em uma representação para o algoritmo Apriori, para que sejam mineradas as regras de associação. Esta regras são úteis, como mostrado neste trabalho, no apoio à ferramenta *XChange* para a criação de regras de inferência e na interpretação das mudanças ocorridas.

Palavras-chave: Documentos XML, mineração de dados, regras de associação

Agradecimentos

Aos meus pais Sônia e Marcos, pelo amor incondicional e pela dedicação empregada no meu crescimento pessoal e profissional. À minha irmã, Verônica, pelos bons exemplos aos quais me baseio.

A todos os meus amigos, por me darem forças e me ajudarem neste trabalho.

Aos meus colegas do Grupo de Educação Tutorial da Computação (GETComp UFJF) e laboratório de pesquisa, em especial a Carlos Oliveira, Eduardo Soares, Fernando Marques, João Paulo Ferreira, Márcio Júnior, Matheus Bigogno, Matheus Marques, Lenita Ambrósio e Ludmila Yung pelas revisões e sugestões no trabalho.

À professora Alessandreia, pela motivação, orientação, confiança no meu trabalho e, principalmente, paciência nos momentos de dificuldade, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Aos professores Victor Menezes e Luciana Campos pela avaliação deste trabalho.

"A paz é a única forma de nos sentirmos realmente humanos".

Albert Einstein

Sumário

Lista de Figuras

Lista de Abreviações

API	*Application Programming Interface*
ARFF	*Attribute-Relation File Format*
DOM	*Document Object Model*
DTD	*Document Type Definition*
FLEX	*Frequent Lexicographic Patterns*
KDD	*Knowledge Discovery in Database*
SAX	*Simple API for XML*
UFJF	Universidade Federal de Juiz de Fora
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*
XSLT	*eXtensible Stylesheet Language for Transformation*

1 Introdução

1.1 Contextualização

Nos últimos anos, tem-se visto um grande crescimento no volume de dados nas áreas que empregam o uso da tecnologia da informação. Este crescimento é impulsionado pelo grande tráfego de informações na Internet e pelo desenvolvimento da tecnologia em si, permitindo, por exemplo, que as empresas armazenem grandes quantidades de informações referentes ao seus negócios. Essa grande quantidade de dados torna a tarefa de extração de informações muito mais complexa e as chances de se perder informações preciosas se tornam ainda maiores (LAN, 2009).

O uso de documentos semiestruturados, principalmente documentos XML, tem sido cada vez maior na representação de dados, seja para armazenamento ou como meio de comunicação entre aplicações. Eles apresentam uma estrutura hierárquica e marcadores que são definidos pelo próprio usuário e, desta forma, permitem uma flexibilidade considerável na representação de dados, justificando seu grande emprego. Tal flexibilidade torna os documentos XML passíveis de alterações frequentes, tornando necessário entender e monitorar a evolução desses documentos (BRAY et al., 2008).

Para acompanhar essa evolução e extrair informações que não estão claramente disponíveis para o usuário, técnicas de mineração de dados têm sido empregadas. Mais especificamente, as regras de associação estão sendo cada vez mais utilizadas (MORADI, 2013). Elas permitem que os dados disponíveis sejam combinados, formando uma relação que pode ser interpretada de diversas formas, de acordo com o contexto (AGRAWAL et al., 1993a).

Muitos trabalhos têm sido desenvolvidos para realizar a mineração em documentos XML. Abordagens como (BRAGA et al., 2002; ZHANG et al., 2006; DING, 2006) empregam o uso de mineração de dados em documentos XML, mas não contemplam a característica evolutiva desses documentos, analisando apenas uma única versão da base. Outros trabalhos, como (ZHAO et al., 2004a; CHEN et al., 2004; RUSU et al., 2006),

observam exatamente essa característica evolutiva do documento XML e realizam a mineração de dados sobre as diferenças entre as versões do documento, buscando extrair informações úteis de acordo com as mudanças ocorridas.

1.2 Justificativa

A flexibilidade e a portabilidade são características que vêm fazendo com que, nos últimos anos, documentos XML sejam utilizados como um padrão para representação, intercâmbio e manipulação de dados em aplicações nas mais diversas áreas (W3C, 2014). O crescimento na utilização desses documentos levou a um aumento significativo no volume de dados de aplicações no mundo todo. Com isso, torna-se necessário um maior emprego de técnicas e ferramentas que possam auxiliar a manutenção e recuperação de informações importantes nessa grande massa de dados. Como os documentos XML permitem uma estruturação própria, é necessário um tratamento diferenciado no que se refere à aplicação de determinadas técnicas.

As soluções encontradas na literatura, para realizar a extração de informações, procuram adaptar técnicas, como as regras de associação da mineração de dados, às características específicas dos documentos XML, o que vem evoluindo nos últimos anos (MORADI, 2013). Porém, devido a grande dificuldade desta tarefa, os trabalhos encontrados ainda possuem algumas limitações, como a falta de uma ferramenta usual para os usuários, a complexidade dos algoritmos utilizados, limitações computacionais de memória e processamento, entre outros. Isso faz com que o tema ainda seja muito pesquisado a fim de encontrar melhores soluções.

1.3 Objetivos

Diante do interesse na detecção de informações interessantes através da análise das mudanças ocorridas em documentos XML, este trabalho tem por finalidade realizar um levantamento bibliográfico que permite analisar algumas das principais propostas encontradas na literatura.

A partir dessa análise, cumpre-se o objetivo principal desta monografia, o desen-

volvimento do XChangeMining. Trata-se de um módulo do *XChange*, ferramenta utilizada para apoiar a evolução de documentos XML baseada em inferência. Este módulo permite extrair informações úteis, a partir do histórico de mudanças de versões de documentos XML, e auxiliar na construção de regras *Prolog* para a realização de inferências no *XChange*. Esta extração ocorre através da aplicação da mineração de dados, mais especificamente a mineração de regras de associação, permitindo acompanhar e entender a evolução dos documentos.

1.4 Organização do trabalho

O presente trabalho está organizado em 5 capítulos, incluindo esta introdução. No capítulo 2 são apresentados alguns conceitos fundamentais para o entendimento do trabalho referentes às técnicas de mineração de dados, algoritmos de extração de regras de associação, definições sobre documentos XML e detecção de diferenças nesse tipo de documento.

No capítulo 3 são apresentados alguns trabalhos relacionados à mineração de dados em documentos XML, das abordagens pioneiras às mais recentes. Ao final deste capítulo tem-se a descrição da abordagem proposta por esta monografia e um comparativo com as abordagens citadas.

O capítulo 4 mostra um exemplo de utilização da abordagem proposta nesta monografia, explicando também como foi criada a base de dados utilizada nos exemplos e uma breve análise dos resultados obtidos.

Por fim, o capítulo 5 apresenta as considerações finais sobre a pesquisa e algumas sugestões de trabalhos futuros.

2 Fundamentação teórica

Nos últimos anos, tem-se visto um elevado crescimento no uso e armazenamento de dados por empresas dos mais variados setores, como provedores de Internet, que registram as ações de seus usuários, empresas na área da saúde, que mantém bancos de dados sobre seres vivos, doenças, medicamentos e informações genéticas, ou empresas em geral, que registram suas movimentações, históricos e funcionários.

No geral, essas informações costumam ser armazenadas nos mais diversos tipos de formato, como banco de dados convencionais, de vídeos, de áudios ou de arquivos. À medida que as tecnologias se desenvolvem, cada vez mais é possível manter todas essas informações de forma que possam ser recuperadas e utilizadas de alguma forma útil. Um problema visto nesta utilização dos dados de forma proveitosa está em como conseguir recuperar, interpretar e analisar toda essa grande massa de dados de forma prática e viável.

Entre outras formas de interpretação e análise de dados, como análises estatísticas, análises matemáticas e pesquisas diretas, existe uma de extrema eficiência para se descobrir conhecimentos não claramente observáveis com as técnicas comuns citadas, o KDD (*Knowledge Discovery in Database* - Descoberta de Conhecimentos em Bases de Dados).

Este capítulo apresenta como é realizado esse processo de descoberta de conhecimentos, como a mineração de dados e as regras de associação são utilizadas para obter esse conhecimento, o algoritmo Apriori, escolhido para realizar a mineração, e conceitos relativos à documentos XML, detecção de diferenças entre versões de documentos XML e o algoritmo desenvolvido neste trabalho para a realização do *diff*.

2.1 KDD e mineração de dados

Não é unânime a definição exata dos termos KDD e mineração de dados. Para alguns autores, tais como (HAN AND FU, 1994; BRAGA et al., 2003), os dois termos são sinônimos. Para outros, como (FAYYAD et al., 1996), o KDD diz respeito ao processo de descoberta

de conhecimento, enquanto a mineração de dados é uma das atividades do processo. Este trabalho segue a definição de HAN AND FU (1994), considerando o termo mineração de dados para todo o processo.

Pela definição utilizada por FAYYAD et al. (1996), KDD é um processo não trivial de identificação de novos padrões válidos, úteis e compreensíveis. Já a mineração de dados é um passo na descoberta de conhecimento que consiste na realização da análise dos dados e na aplicação de algoritmos de descoberta que, sob certas limitações computacionais, produzem um conjunto de padrões de certos dados.

A Figura 2.1 mostra uma representação do processo de KDD em que a mineração de dados se apresenta como uma fase do processo.

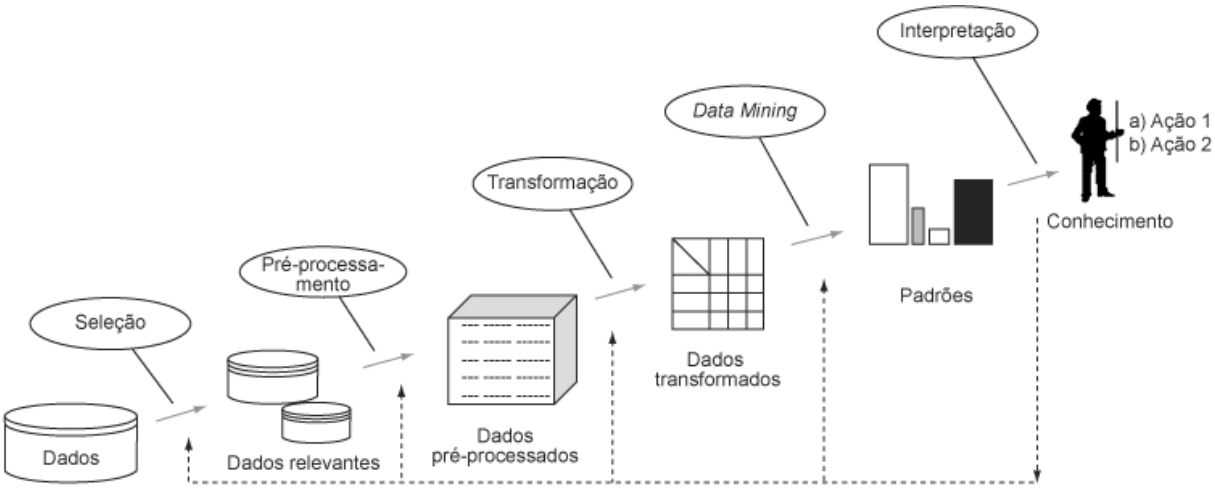


Figura 2.1: Etapas do processo KDD (FAYYAD et al., 1996).

As informações extraídas das bases de dados pela mineração de dados, diferente do conhecimento extraído nos processos comuns, estão ocultas no meio da grande quantidade de dados armazenados. O que torna a mineração diferente das consultas normais é a exploração e a inferência de informações.

O processo de mineração envolve o uso de ferramentas de análise de dados e a integração de técnicas de diversas disciplinas, como matemática, estatística, aprendizado de máquina, reconhecimento de padrões, redes neurais e recuperação da informação.

A mineração de dados pode ser usada em diferentes tipos de bases de dados, como bases relacionais, transacionais, orientadas a objetos e *data warehouses*, ou em diferentes tipos de repositórios de informações, como bases espaciais, de séries temporais, de textos

ou mídias e a própria *Web* (HAN AND FU, 1994).

2.1.1 As fases do *Knowledge Discovery in Database*

Como apresentado na Figura 2.1, o processo de descoberta de conhecimento, conforme definido por FAYYAD et al. (1996), pode ser visto como um ciclo em que o dado percorre até se transformar em uma informação útil. Esse ciclo é dividido em fases, as quais são brevemente descritas a seguir.

A fase de seleção tem por finalidade extrair subconjuntos de dados úteis para a mineração, identificando informações importantes e ajustando o banco de dados de uma forma adequada aos algoritmos de mineração de dados.

A fase de pré-processamento é uma das mais importantes de todo o processo, além de ser a que mais consome tempo. Grandes bases de dados geralmente contém alguns erros no armazenamento dos dados e essa fase ajuda a identificar alguns desses erros, como informações incorretas e dados incompletos.

Na fase de transformação, os dados são adequados às formas compatíveis com a entrada esperada dos algoritmos de mineração, no qual são utilizadas técnicas de *suavization* (alisamento), *generalization* (generalização), *normalization* (normalização) e *aggregation* (agregação). Nessa fase também pode ser feita uma redução da dimensão da base, a fim de facilitar e simplificar o processo de mineração.

Depois da limpeza dos dados brutos, eles se tornam a entrada para os algoritmos na fase de mineração de dados, fase esta que produz saídas na forma de padrões utilizando métodos inteligentes de acordo com o tipo de informação que se queira extrair. Algoritmos de *visualization* (visualização), *classification* (classificação), *clustering* (agrupamento), *regression* (regressão) ou *association* (associação) são utilizados para diferentes tipos de problemas. Somente uma certa parte dos dados minerados, dos padrões retornados, representam algum conhecimento.

A partir desses resultados, algumas medidas funcionais são aplicadas na fase de interpretação e avaliação para separar os padrões interessantes ao contexto aplicado. Por exemplo, o *support* (suporte) e *confidence* (confiança), utilizados como medidas na obtenção de regras de associação e que são definidas mais à frente neste capítulo.

2.1.2 Tarefas realizadas pela mineração de dados

Para saber qual tarefa a ser utilizada, deve-se ter um bom domínio da aplicação, qual tipo de informação se deseja obter e o conhecimento das tarefas existentes para saber qual aplicar. AGRAWAL et al. (1993a) e FAYYAD et al. (1996) descreveram as seguintes tarefas de mineração de dados: *classification* (classificação), *prediction* (predição), *description* (descrição), *estimation/regression* (estimativa/regressão), *clustering* (agrupamento) e *association* (associação).

A classificação é um dos objetivos mais comuns quando se utiliza mineração de dados. Sua função é identificar a qual classe pertence um determinado registro. Na classificação, o sistema analisa o conjunto de dados fornecidos ao algoritmo e nesse conjunto de dados todos os registros já possuem uma indicação informando a qual classe ele pertence. Isso faz com que o algoritmo aprenda como classificar um novo registro informado. É chamado de aprendizado supervisionado.

Essa tarefa de classificação pode ser usada, por exemplo, para identificar quando uma pessoa pode ser uma ameaça para a segurança, diagnosticar onde uma determinada doença pode estar presente ou a qual turma um determinado aluno pode ser melhor incluído.

A predição é parecida com a classificação, porém ela tem como objetivo descobrir o valor futuro de um determinado atributo. Exemplos de aplicação desta tarefa são: predizer o valor de uma ação da bolsa de valores em um determinado mês adiante, prever o aumento de tráfego de uma determinada cidade, ou ainda em esportes utilizando estatísticas passadas.

A descrição tem o objetivo de descrever os padrões e tendências revelados pelos dados passados ao algoritmo. Ela fornece uma possível interpretação para os resultados obtidos e é muito utilizada em conjunto com outras técnicas, como a análise exploratória de dados para comprovar a influência de certas variáveis.

A estimativa, também similar à classificação, é utilizada quando o registro é identificado por um valor numérico ao invés de uma categoria. Neste caso, pode ser estimado o valor de uma determinada variável observando os valores das demais.

Por exemplo, um conjunto de registros de consumidores que possui os valores

gastos em um mês de acordo com seus hábitos. O sistema ao analisar os dados é capaz de extrair informações do tipo: qual será o valor gasto por um novo consumidor. Outro exemplo é o sistema estimar qual a pressão arterial ideal de um paciente baseado na sua idade, peso e outras informações.

O agrupamento busca identificar e aproximar registros similares. Um *cluster* (agrupamento) é definido como uma coleção de registros similares entre si. A diferença entre o agrupamento e a classificação é que esta requer que os registros sejam previamente categorizados. Por não necessitar dessa categorização, o agrupamento é considerado como um aprendizado não-supervisionado. Não faz parte dos objetivos dessa tarefa classificar, estimar ou predizer alguma coisa. Ela apenas identifica os grupos de dados similares.

O agrupamento pode ser aplicado em diversas situações como: taxonomia de plantas, processamento de imagens, classificação de documentos da *Web*, pesquisas geográficas, reconhecimento de padrões, pesquisa de mercado, entre outras (HAN AND FU, 1994). Ela é muito utilizada com outras tarefas.

A associação é a tarefa utilizada neste trabalho e tem por objetivo identificar quais atributos, dentro da base de dados fornecida ao algoritmo, estão relacionados. É uma das tarefas mais conhecidas por causa dos bons resultados obtidos através dela. Um dos principais exemplos de utilização dessa tarefa é o *market basket* (cesta de compras), onde é identificado quais produtos são comprados juntos pelos consumidores. Por exemplo, inferir que muitas pessoas que compram fralda, também compram cerveja. Essa tarefa é mais discutida no tópico a seguir.

2.2 Regras de Associação

A mineração de regras de associação é uma importante técnica de mineração de dados. Sua tarefa é determinar quais registros numa base de dados ocorrem juntos. Ela foi introduzida por AGRAWAL et al. (1993a), demonstrada com um exemplo que se tornou o mais conhecido da área, o *market basket*. Ele mostrava que em 90% das transações, realizadas em um determinado supermercado, quem comprava fralda também comprava cerveja.

Para melhor entender o processo de extração de regras de associação, é apresen-

tado um exemplo utilizando a ideia de um supermercado. Pensando em descobrir quais produtos os clientes tem o costume de comprar ao mesmo tempo quando vão ao supermercado, pode-se descobrir informações muito úteis no que diz respeito ao planejamento de vendas a fim de criar promoções e gerar propagandas.

Cada compra realizada por um cliente é tratada como uma transação e cada transação possui os produtos adquiridos por tal cliente. O termo *conjunto de itens* é utilizado para o conjunto de produtos comprados por um cliente (os produtos de uma transação). A Tabela 2.1 mostra alguns exemplos de produtos vendidos no supermercado e a Tabela 2.2 mostra algumas transações realizadas pelos clientes, que foram utilizadas no exemplo.

Tabela 2.1: Produtos

Produtos
Uva
Cerveja
Leite
Pão
Fralda
Suco
Biscoito
Amendoim
Iogurte

Tabela 2.2: Transações

Transação	Produtos
1	Cerveja, Pão, Amendoim, Leite, Fralda
2	Leite, Fralda, Biscoito, Iogurte, Pão
3	Pão, Biscoito, Fralda
4	Cerveja, Fralda
5	Leite, Pão, Biscoito
6	Uva, Amendoim, Iogurte, Pão, Biscoito, Leite
7	Suco, Pão, Biscoito
8	Suco, Cerveja, Leite, Amendoim, Pão, Biscoito
9	Fralda, Leite, Cerveja, Uva

A Tabela 2.3 apresenta as frequências em que alguns conjuntos de itens frequentes aparecem na base de dados de transações.

Essas frequências definem uma das métricas utilizadas nos algoritmos de mi-

Tabela 2.3: Conjunto de itens frequentes

Conjunto de itens	Suporte
{Pão, Biscoito}	0,66
{Leite, Pão}	0,55
{Cerveja, Fralda}	0,33
{Leite, Biscoito}	0,33
{Pão, Fralda, Biscoito}	0,33
{Cerveja, Amendoim}	0,22
{Pão}	0,77
{Biscoito}	0,66
{Uva}	0,22
...	...

neração de regras de associação, o suporte. O suporte é o percentual de transações onde o conjunto de itens aparece. A Tabela 2.3 apresentada mostra os suportes para alguns conjuntos de itens, mas além desses ocorrem muitos outros, porém com suportes menores.

Algumas medidas devem ser consideradas de acordo com o que o usuário determinar. Caso ele defina que um conjunto de itens que apareça em pelo menos 60% de todas as transações seja considerado frequente, apenas o conjunto de itens {Pão, Biscoito} é considerado. A partir desse resultado, o usuário decide se aumenta ou diminui o valor que ele considera aceitável.

2.2.1 Formalização

Esta seção apresenta conceitos relacionados a mineração de regras de associação, baseados em AGRAWAL et al. (1993b), que definiram o problema formalmente da seguinte maneira.

Dado um conjunto $I = \{i_1, i_2, i_3, ..., i_m\}$ de itens (como o conjunto de produtos do supermercado), e seja D um conjunto de transações (como o conjunto de vendas realizadas no supermercado), onde cada transação, T, é um conjunto de itens ($T \subseteq I$). Cada transação possui um identificador, TID. A Tabela 2.2 é um exemplo do conjunto de transações. Um conjunto de itens é um subconjunto não vazio de I, e uma transação T suporta um conjunto de itens S se S estiver contido em T, $S \subseteq T$.

Uma regra de associação é uma implicação da forma $X \rightarrow Y$, onde $X \subset I$, $S \subset T$ (X e Y são conjuntos de itens e seus elementos estão contidos no conjunto de itens) e $X \cap Y = \emptyset$ (ou seja, X e Y não podem ter o mesmo item).

Trazendo essa formalização para o exemplo do supermercado,

$$\{Pão\} \rightarrow \{Biscoito\}$$

é uma regra de associação que pode ser compreendida como: se alguém compra *Pão* então compra *Biscoito*. Isso é diferente de

$$\{Biscoito\} \rightarrow \{Pão\}$$

Não necessariamente quem compra *Biscoito* também compra *Pão*.

O suporte, como já comentado, é o percentual de transações onde o conjunto de itens aparece. Esta métrica pode ser definida da seguinte forma:

$$sup(X \rightarrow Y) = \frac{número\ de\ transacões\ em\ que\ X\ e\ Y\ aparecem}{número\ total\ de\ transacões}$$

Além do suporte, existem outras métricas que podem ser utilizadas, por exemplo, a confiança, denotado por $conf(\{Pão\} \rightarrow \{Biscoito\})$. Essa medida é o percentual de transações que suportam $X \cup Y$ entre todas as transações que suportam X:

$$conf(X \rightarrow Y) = \frac{número\ de\ transacões\ que\ suportam\ (X \cup Y)}{número\ de\ transacões\ que\ suportam\ X}$$

Por exemplo, a regra $\{Pão\} \rightarrow \{Biscoito\}$ tem o seguinte valor para confiança:

- número de transações que suportam $\{Pão\} \cup \{Biscoito\} = 6$ (as transações $2, 3, 5, 6, 7, 8$)
- número de transações que suportam $\{Pão\} = 7$ (as transações $1, 2, 3, 5, 6, 7, 8$)
- $conf(\{Pão\} \rightarrow \{Biscoito\}) = \frac{6}{7} = 0{,}86 = 86\%$ de confiança.

Ter uma confiança relativamente alta não é suficiente para dizer que uma regra é boa. Por exemplo, a regra $\{Uva\} \rightarrow \{Leite\}$ tem $conf(\{Uva\} \rightarrow \{Leite\}) = 100\%$, mas $sup(\{Uva\} \rightarrow \{Leite\}) = 22\%$. Isso significa que, apesar da regra dizer que toda vez que

Uva é comprada, *Leite* também é comprado, o fato de os dois produtos aparecem juntos em apenas 20% das compras demostra que essa regra não é tão interessante assim, neste contexto.

Uma regra é considerada interessante se $conf(r) \geq c$ e $sup(r) \geq s$, onde c e s são valores mínimos de confiança e suporte fornecidos pelo usuário, respectivamente. Logo, o bom senso e experiência do usuário diz qual a qualidade desejada das regras mineradas.

Uma outra medida utilizada na mineração de regras de associação é o *lift*. Esta métrica indica o quanto mais frequente torna-se Y quando X ocorre. Ela pode ser calculada da seguinte maneira:

$$lift(X \rightarrow Y) = \frac{conf(X \rightarrow Y)}{sup(X \rightarrow Y)}$$

No exemplo da regra $\{Pão\} \rightarrow \{Biscoito\}$, o *lift* é $\frac{0,86}{0,66} = 1,30$. O resultado indica que os clientes que compram *Pão* tem uma chance 1,3 vezes maior de comprar *Biscoito*.

A medida *leverage*, também chamada de *rules interest*, mede a diferença entre a frequência com que X e Y aparecem juntos na mesma regra e a frequência que seria esperada se X e Y fossem independentes. Pode ser calculada com a fórmula:

$$leverage(X \rightarrow Y) = sup(X \rightarrow Y) - sup(X) * sup(Y)$$

No exemplo da regra $\{Pão\} \rightarrow \{Biscoito\}$, o *leverage* é 0,66 - (0,77 * 0,66) = 0,15.

A *conviction* tem como função medir a independência de X relativamente a Y e identificar as regras que tem 100% de confiança (normalmente pouco interessantes), quando X e Y são independentes. Sua fórmula é dada por:

$$conviction(X \rightarrow Y) = \frac{1 - sup(Y)}{1 - conf(X \rightarrow Y)}$$

No exemplo da regra $\{Pão\} \rightarrow \{Biscoito\}$, $\frac{(1-0,66)}{(1-0,86)} = 2,43$.

2.2.2 Algoritmos

Esta seção apresenta brevemente alguns algoritmos de regras de associação, e, com maiores detalhes, o algoritmo Apriori.

Segundo AGRAWAL et al. (1994), o problema de descobrir todas as regras de associação pode ser decomposto em dois subproblemas:

- encontrar todos os conjuntos de itens, no conjunto de transações, que aceitam o suporte mínimo fornecido pelo usuário.
- usar o conjunto de itens frequentes para gerar as regras desejadas. Apenas as regras que possuem grau de confiança mínimo devem ser selecionadas.

Para resolver o problema de mineração, o algoritmo Apriori foi o pioneiro e seu trabalho destaca a utilização da propriedade Apriori, também chamada "Antimonotonia da relação de inclusão entre os conjuntos de itens", para redução do espaço de busca dos conjuntos de itens frequentes. Essa propriedade determina que "todo subconjunto não vazio de um conjunto de itens frequente também é frequente".

2.2.3 Apriori

O algoritmo Apriori foi introduzido por AGRAWAL et al. (1994), e é um dos algoritmos mais conhecidos e usados na mineração de regras de associação. É um algoritmo iterativo que busca por conjuntos de itens de tamanho k a partir dos conjuntos *k-1* da iteração anterior.

O funcionamento do Apriori consiste na geração de um conjunto A_n, que contém os conjuntos de itens candidatos, e a avaliação desse conjunto. A avaliação é realizada com a contagem do suporte de cada conjunto de itens que deve ser maior que o suporte mínimo estabelecido pelo usuário inicialmente. Todos os conjuntos de itens aprovados são adicionados a um conjunto B_n de itens frequentes e este realimenta a iteração do algoritmo.

O Apriori pode ser dividido em dois passos. O seu primeiro passo é dividido em duas fases principais, que são chamadas de fase de junção e poda. Na fase de junção, a união dos conjuntos B_{n-1} geram o conjunto A_n. Essa união ocorrerá se os conjuntos

B_{n-1} compartilharem (n-1) itens. Na fase de poda, são calculados todos os suportes dos conjuntos de itens candidatos de A_n. Utilizando o exemplo do supermercado já apresentado, o primeiro passo do algoritmo é apresentado na Figura 2.2. Na primeira iteração, cada produto individualmente se torna um conjunto de itens de tamanho um, realizado pela fase de junção, e em seguida são eliminados, pela fase de poda, os conjuntos de itens que não satisfaçam o suporte mínimo, definido aqui em 50%.

Em seguida, na segunda iteração, os conjuntos de itens aptos da iteração anterior são unidos pela fase de junção para gerar os conjuntos de itens de tamanho dois. Na fase de poda, estes conjuntos de itens passam pela avaliação de suporte mínimo e os que estiverem aptos alimentam a próxima iteração do algoritmo. Deste exemplo, apenas os conjuntos de itens $\{Pão, Leite\}$ e $\{Pão, Biscoito\}$ satisfazem o suporte mínimo de 50%.

O segundo passo do algoritmo é destinado a gerar as regras de associação. Inicialmente, o algoritmo encontra todos os subconjuntos não-vazios f de cada conjunto de itens frequente F. No exemplo, o conjunto de itens frequente $\{Pão, Leite\}$ possui os seguintes subconjuntos: $\{Pão\}$, $\{Leite\}$ e $\{Pão, Leite\}$. Já o conjunto de itens frequente $\{Pão, Biscoito\}$ possui os seguintes subconjuntos: $\{Pão\}$, $\{Biscoito\}$ e $\{Pão, Biscoito\}$. Depois de encontrar os subconjuntos não vazios, uma combinação de regras é determinada, com o formato $\{f\} \rightarrow \{F - f\}$, e suas respectivas confianças calculadas, mantendo as regras que satisfaçam a confiança mínima estipulada pelo usuário. Os subconjuntos do exemplo, com suas confianças calculadas, são apresentados na Tabela 2.4.

Tabela 2.4: Regras

Conjunto de itens	Regra	Confiança
{Pão, Biscoito}	$\{Pão\} \rightarrow \{Biscoito\}$	0,85
{Pão, Biscoito}	$\{Biscoito\} \rightarrow \{Pão\}$	1,00
{Pão, Leite}	$\{Leite\} \rightarrow \{Pão\}$	0,83
{Pão, Leite}	$\{Pão\} \rightarrow \{Leite\}$	0,71
...	...	...

Neste exemplo, caso o usuário estabelecesse uma confiança mínima de 80%, apenas as regras $\{Pão\} \rightarrow \{Biscoito\}$, $\{Biscoito\} \rightarrow \{Pão\}$ e $\{Leite\} \rightarrow \{Pão\}$ seriam mineradas. O algoritmo Apriori é apresentado em alto nível na listagem 2.1.

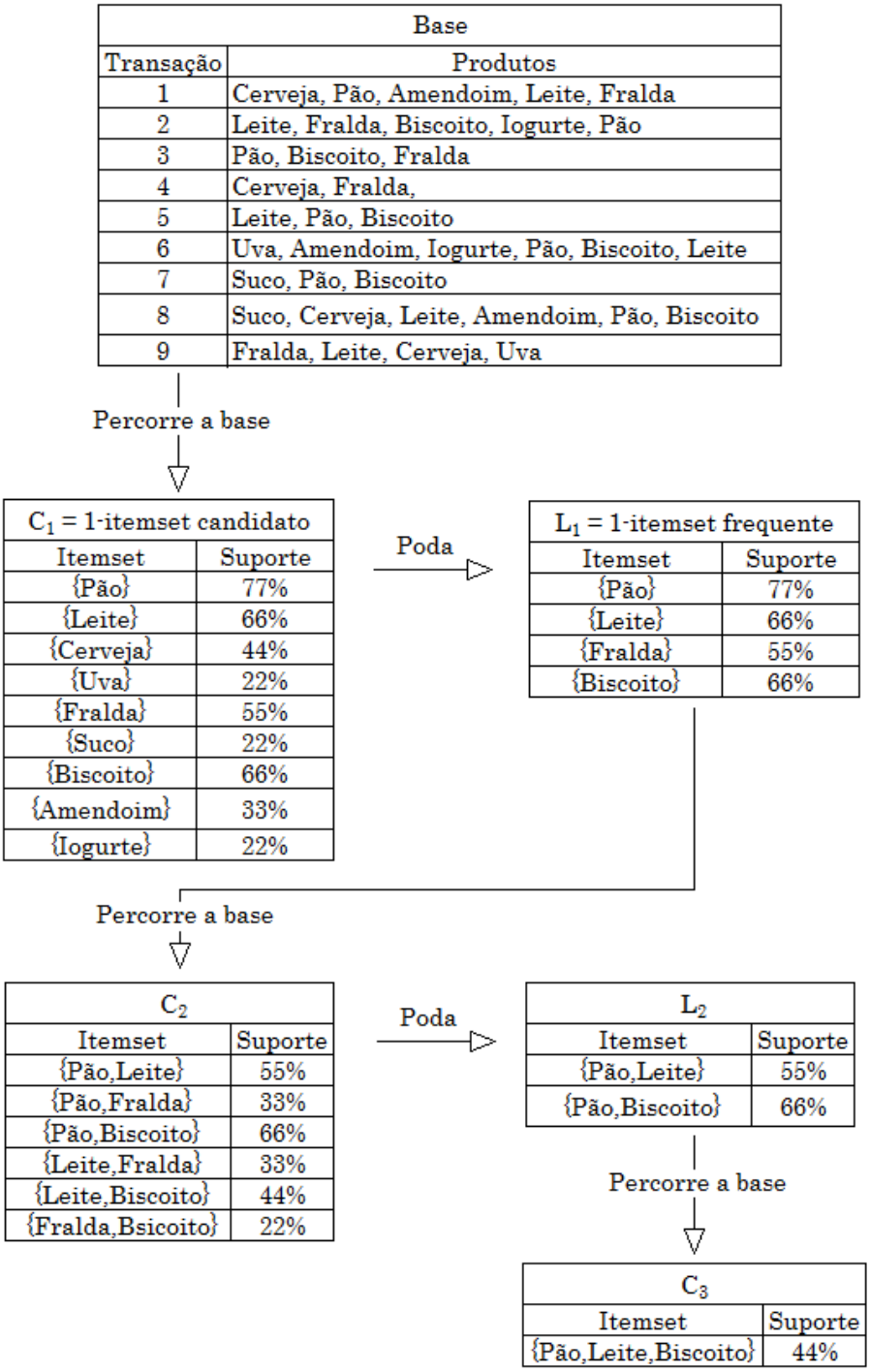


Figura 2.2: Primeiro passo do algoritmo.

Algoritmo 1: Algoritmo Apriori

Entrada: $F_k \leftarrow Conjunto\ de\ itens\ frequentes$

Saída: $F \leftarrow Todos\ os\ itens\ frequentes$

início

  para $k = 2\ ;\ F_{k-1}\ \neq vazio\ ;\ k++$ **faça**

    $C_k \leftarrow aprioriGen(F_{k-1})$ **para todo** $transacao\ t \in T$ **faça**

      $C_t \leftarrow subconjunto(C_k\ ,\ t)$ **para todo** $candidato\ c \in C_t$ **faça**
        $c.contagem++$

      fim para todo

      $F_k \leftarrow c \in C_k \mid c.contagem \geq MinSup$

    fim para todo

  fim para

fim

2.2.4 *Frequent-Pattern Growth*

O algoritmo FP-Growth, introduzido por HAN et al. (2004), adota uma estratégia de divisão e conquista no seu desenvolvimento. Ele computa os itens frequentes e os representa como uma base de dados comprimida em uma estrutura chamada árvore de padrões frequentes (*FP-Tree*), sendo a mineração realizada sobre essa árvore. Essa abordagem permite que o conjunto de transações seja percorrido apenas uma vez, além de não requerer a geração de conjuntos de itens candidatos. Os itens subsequentes são computados como no algoritmo Apriori, representados em uma tabela chamada *header table*.

Cada registro nesta tabela contém o item frequente e uma ligação para o nó da árvore que possui o mesmo nome do item. Seguindo essa ligação, pode-se encontrar todos os nós na árvore com mesmo nome de item. Cada nó na árvore contém o nome do item, a contagem do suporte e uma ligação para um nó com o mesmo nome de item.

São duas fases para a criação da árvore. Na primeira, o conjunto de transações é escaneado e os conjuntos de itens de tamanho 1 são criados, contando o suporte para cada item frequente. Na segunda, o conjunto é ordenado com base no suporte, de forma decrescente, e a tabela é criada. A árvore é criada com raiz vazia e, para cada transação, são realizados os seguintes procedimentos: seleciona e ordena os itens frequentes da transação, na ordem da tabela; chama a função recursivamente para construir a árvore, passando a lista ordenada e a raiz da árvore; se o primeiro item da lista já estiver na árvore, o suporte

é incrementado, senão, cria um novo nó para este item; o processo é repetido até a lista ficar vazia.

Com as duas estruturas criadas, para cada item na tabela, uma lista de ligações é criada. Esta lista liga o item na tabela com a mesma referência na árvore. A lista se chama *padrões base condicionais*. A cada padrão base é atribuído um suporte. Se este valor for menor que o valor de suporte estipulado pelo usuário, ele é ignorado. Para cada padrão base, uma *FP-Tree* condicional é criada. A mineração é feita sobre as árvores até não haver mais itens.

2.2.5 *Frequent Lexicographic Patterns*

O algoritmo FLEX (*Frequent Lexicographic Patterns*) foi inicialmente criado para bases relacionais (MUSTAPHA et al., 2003). Ele pode ser descrito usando uma árvore lexicográfica, supondo que uma ordenação alfabética exista entre os itens no banco de dados. A árvore lexicográfica é usada como uma representação dos padrões frequentes com respeito a este ordenamento.

Para gerar o primeiro nível da FLEX, a base de dados precisa ser percorrida uma vez e os itens que satisfazem o suporte mínimo são parte da construção do primeiro nível da árvore de acordo com a sua lexicografia. O próximo nível da árvore FLEX é gerado enquanto são verificadas as possíveis extensões de padrões frequentes de um nó. As extensões de padrões frequentes fazem a interseção dos possíveis itens frequentes com um determinado nó. Se os suportes de possíveis extensões de padrões frequentes excederem o limite do suporte fornecido pelo usuário, então ele é adicionado ao segundo nível da árvore FLEX. Assim, a criação de um nó é resultado de testar a extensão dos padrões frequentes contra o suporte mínimo.

2.3 Evolução de documentos XML

Neste tópico, são apresentados conceitos relacionados a documentos XML, sua utilização e a mineração de informações nesse tipo de documento, objetivo desse trabalho.

2.3.1 Fundamentos de XML

XML (*eXtensible Markup Language*) é um padrão para representação e descrição de dados semiestruturados em documentos, criado pela W3C (*World Wide Web Consortium*) (W3C, 2014). É considerada uma linguagem de marcação, isto é, seu conteúdo é definido por uma ou várias marcas, mais conhecidas como *tags*. Essas *tags* são definidas pelo próprio usuário criador do documento, permitindo uma grande flexibilidade de representação.

O XML é construído como uma estrutura em árvore, onde os nós são os elementos, atributos e valores, e as arestas são as relações entre elementos, subelementos e valores. Essa estrutura em árvore dá ao XML a característica de hierarquia, fator importante para uso das aplicações que utilizam e manipulam esses arquivos.

O formato XML é amplamente utilizado em diversas aplicações para realizar troca de informações, armazenamento de dados, configuração de recursos, e por causa disso se tornou uma abordagem muito utilizada para uso na Internet (BRAGANHOLO et al., 2004).

A Figura 2.3 mostra um exemplo de documento XML.

```
<transações>
    <transação id="1">
        <cliente>João</cliente>
        <produtos>
            <item>Cerveja</item>
            <item>Pão</item>
            <item>Amendoim</item>
            <item>Leite</item>
            <item>Fralda</item>
        </produtos>
        <valor>100,00</valor>
    </transação>
    <transação id="2">
        <cliente>Maria</cliente>
        <produtos>
            <item>Leite</item>
            <item>Fralda</item>
            <item>Biscoito</item>
            <item>Iogurte</item>
            <item>Pão</item>
        </produtos>
        <valor>80,00</valor>
    </transação>
</transações>
```

Figura 2.3: Exemplo de documento XML.

Apesar de muito úteis, documentos XML também apresentam alguns problemas que devem ser considerados antes de sua escolha. Para o armazenamento de dados, como uma base de todos os funcionários de uma empresa, o documento pode se tornar extremamente grande, tornando sua manipulação complexa e computacionalmente custosa. Nesses casos, a utilização do XML deve ser muito bem analisada, buscando soluções que amenizem o problema, como computadores mais potentes e técnicas de manipulação ou algoritmos específicos.

Além disso, um documento XML pode representar informações que não se mantém ao longo do tempo, ou seja, informações que sofrem variações diversas, como atualizações cadastrais em bases de funcionários ou mudanças de parâmetros em arquivos de configuração. Esse dinamismo do XML também requer tratamentos específicos que lidem com essa característica.

2.3.2 *Diff* em documentos XML

Existem diversas abordagens que apresentam algoritmos para a detecção de diferenças entre documentos XML (WANG et al., 2003; COBENA et al., 2002). Estes algoritmos têm por finalidade, a partir de duas versões de um documento XML, gerar um documento que contém todas as alterações ocorridas entre essas duas versões. Este documento é comumente chamado de delta, e além das alterações, possui diversas operações que permitem a reconstrução de uma versão do documento XML a partir de outra. Este processo é muito útil para o gerenciamento de mudanças de documentos, já que permite observar claramente as mudanças ocorridas entre as versões e retornar essas versões a estados passados para que possam ser recuperadas ou analisadas.

As operações que compõem o delta são definidas como *insert*, quando uma nova informação é incluída na segunda versão; *delete*, quando uma informação é removida da segunda versão e *update*, quando uma informação é alterada da primeira pra segunda versão. Além destas três operações básicas, alguns algoritmos adotam as operações *move*, quando informações simplesmente são movidas para outra parte do documento e *copy*, quando uma informação é copiada para outras partes do documento (LINDHOLM, 2001).

Diferente das abordagens de detecção de diferenças convencionais, que analisam

as alterações em documentos linha por linha, as mudanças em documentos XML devem ser observadas a partir de cada registro ou elemento do XML. Isso se deve ao fato que a estrutura do XML permite que os registros, ou elementos, podem mudar de posição no documento sem que de fato isto represente uma alteração real do conteúdo que ele representa. Por este fato, existem diversos algoritmos de detecção de mudanças em documentos XML que variam na forma como é tratada a identificação dos registros e elementos nas versões analisadas (LINDHOLM, 2001).

Alguns algoritmos utilizam a própria identificação definida no esquema do XML (*XML Schema* ou DTD - *Document Type Definition*). Este esquema define como deve ser a estrutura do XML, como devem ser preenchidos os seus valores e, em alguns casos, qual elemento representa de forma única cada registro. A partir desta identificação única, estes algoritmos conseguem facilmente encontrar os registros correspondentes em cada versão analisada e então detectar as diferenças entre esses registros (OSO, 2014).

Como nem sempre essa identificação é definida, existem outros algoritmos que buscam outras formas de encontrar os registros correspondentes. Algumas abordagens desenvolvem técnicas para aplicar uma identificação única aos registros, de acordo com seu conteúdo, e outras abordagens utilizam técnicas de similaridade. Estas, através de algoritmos de similaridade e algumas métricas, analisam a semelhança entre os registros das duas versões do XML.

2.3.3 Algoritmo de *diff* proposto

Para este trabalho, definiu-se um algoritmo para realizar a detecção de diferenças entre documentos XML, necessárias para a fase de mineração. A criação de um novo algoritmo se deve ao fato de que as ferramentas disponíveis e que satisfazem os critérios necessários não suportam a escalabilidade exigida quando o tamanho dos documentos XML analisados são muito grandes.

O algoritmo criado considera a natureza não-ordenada dos dados contidos no documento XML. Dessa forma, trocas de posições de elementos de mesmo nível não representam uma mudança relevante. Isso faz com que apenas a relação pai e filho entre as *tags* do XML seja observada, impedindo que as operações *move* e *copy* sejam detectadas;

ou seja, apenas as três operações básicas, *insert*, *delete* e *update* são suportadas.

Os algoritmos de detecção de diferenças, pela estrutura de árvore que o XML possui, normalmente são desenvolvidos usando a representação DOM (*Document Object Model*). Trata-se de uma especificação do consórcio W3C (*World Wide Web Consortium*) que padroniza a forma de representar documentos XML na memória RAM de um computador, para que sejam facilmente acessados e manipulados. A desvantagem dessa representação é o consumo de memória e para se trabalhar com documentos muito grandes torna-se inviável (DOM, 2014).

Como alternativa, existe o conceito de acesso serial à documentos XML, o SAX (*Simple API for XML*). Orientado a eventos, ele percorre o documento do início ao fim de forma contínua e sem retrocessos. Para cada evento, como o início ou fim de uma *tag* que representa um elemento ou o início de um texto, um sinal com uma identificação é disparado e o evento pode ser tratado como desejado. Esse método não armazena nenhuma informação na memória, permitindo trabalhar com documentos XML sem limitação de tamanho (SAX, 2014).

Uma diferença entre o algoritmo criado e os encontrados na literatura, é o fato de não manter os deltas consolidados. Como o objetivo é a realização da tarefa de mineração, e não o gerenciamento de versões, foi definido que o mais importante é o desempenho e a escalabilidade da ferramenta. Desta forma, o algoritmo não necessita usar espaço de memória para manter estruturas extras durante sua execução e as diferenças detectadas são diretamente processadas pela fase de mineração.

Para iniciar o processo, o algoritmo recebe como parâmetros duas versões de um documento XML, a chave que identifica unicamente cada registro e uma lista que conterá os resultados. Esta lista pode estar preenchida com resultados de versões anteriores, sendo os novos resultados apenas adicionados à essa lista.

O primeiro passo é a transformação dos documentos em uma lista. Esta lista é composta pelos registros do documento XML, identificado por sua chave, e uma lista de elementos desse registro. Para criá-la, o documento é percorrido, utilizando o SAX através de uma API (*Application Programming Interface*) desenvolvida em Java. Para cada evento que indica o início de um novo registro, uma lista é criada para armazenar os

elementos deste registro. E para cada evento que indica o início de um elemento, o nome e o texto contidos no elemento são armazenados na lista.

No segundo passo, a lista de registros do primeiro documento é percorrida. Em cada iteração, se a chave do registro possuir um correspondente na lista de registros do segundo documento, o algoritmo analisa se os elementos desse registro foram alterados na segunda versão.

Nesta análise, cada elemento que tenha sido alterado é incluído em uma lista temporária, que no final é adicionada à lista de resultados. Nesta lista final, cada lista de elementos representa um registro, mas sem que sejam identificados. Isso acontece porque, para a tarefa de mineração, não é necessária qualquer identificação de registros, importando apenas a informação quantitativa de elementos alterados. Uma apresentação em alto nível do algoritmo é mostrada na listagem 2.2.

Algoritmo 2: Algoritmo de detecção de diferenças

Entrada: (DOC_1, DOC_2, chave, listaDiferencas)

Saída: (listaDiferencas)

início

Transforma DOC_1 em $XHash_1$

Transforma DOC_2 em $XHash_2$

para todo *(L_1 : registro em $XHash_1$)* **faça**

if *($XHash_2$ contem um registro com a chave de L_1)* **then**

L_2 = registro de $XHash_2$

$diffElemento[]$;

para todo *(cada elemento do registro)* **faça**

if *(elemento L_1 != elemento L_2* **then**

$diffElemento[]$ = elemento L_1

end if

fim para todo

if *($diffElemento[]$!= vazio)* **then**

listaDiferencas = $diffElemento[]$

end if

end if

fim para todo

fim

2.3.4 Mineração de regras de associação em documentos XML

A mineração de documentos XML não é algo tão trivial de ser implementada. Muitos trabalhos apresentam técnicas e soluções diferentes para tal tarefa e uma análise detalhada é necessária para verificar sua adequação às características do problema em questão.

Muitos trabalhos tratam a mineração de dados de versões de documentos XML considerando sua natureza estática, isto é, suas informações não mudam com o passar do tempo. Isso permite uma implementação mais simplificada por não precisar verificar as diferenças entre versões de um mesmo documento. Algumas propostas (AGRAWAL et al., 1994; BRAGA et al., 2003; FENG et al., 2003) apresentam formas simplificadas para a

mineração de um documento XML, transformando a estrutura em árvore do XML em um objeto melhor manipulável na memória. Essa abordagem acaba por limitar o tamanho do XML de acordo com a memória disponível no computador utilizado, um fator contra esse tipo de implementação.

Outros trabalhos apresentam soluções para mineração de documentos XML com natureza dinâmica, ou seja, que variam seu conteúdo ao longo do tempo. Algumas abordagens (CHEN et al., 2004; RUSU et al., 2006) demonstram soluções com base em um documento gerado por ferramentas de detecção de diferenças entre arquivos, chamado de delta consolidado, e o histórico desses deltas consolidados são manipulados para alimentar o algoritmo de mineração utilizado. Todos esses trabalhos utilizam o algoritmo Apriori para a solução. Porém, nenhum dos artigos são precisos na forma de implementação, passando apenas a ideia em alto nível da implementação de suas soluções.

Com base nesses artigos, a mineração de regras de associação de documentos XML é desenvolvida a partir da ideia de histórico de mudanças, adaptada às necessidades deste trabalho. Por não ser necessário manter o histórico, os resultados da detecção de diferenças, gerados pelo algoritmo criado, são imediatamente consumidos pelo algoritmo de mineração de regras de associação escolhido, uma implementação do Apriori dentro da ferramenta *Weka*. Todos os pontos da implementação são explicados mais detalhadamente no capítulo seguinte.

O *Weka* (*Waikato Environment for Knowledge Analysis*) é um software gratuito, sob a licença GNU *General Public License*, e de código aberto escrito em Java. Foi desenvolvido na Universidade de Waikato, na Nova Zelândia, e hoje é considerada uma das ferramentas de mineração de dados mais utilizada por estudantes e professores de universidades. Ela abrange quase todos os tipos de tarefas de mineração de dados e apresenta variadas soluções para uma mesma tarefa. Possui interface gráfica para utilizá-la individualmente por qualquer pessoa e uma API (*Application Programming Interface*) para incorporá-la a qualquer projeto de software, principalmente os desenvolvidos na linguagem de programação Java (WEKA, 2014). A parte de extração de regras de associação do *Weka* é utilizada neste trabalho e todos os detalhes da implementação são apresentados no capítulo seguinte.

2.4 Considerações Finais

Este capítulo descreveu os conceitos básicos sobre mineração de dados, extração de regras de associação e documentos XML. Estes conceitos são importantes para o desenvolvimento deste trabalho.

No capítulo seguinte, são apresentados trabalhos relacionados ao tema e os passos realizados para o desenvolvimento de uma aplicação para mineração de regras de associação de documentos XML.

3 Revisão da literatura

A mineração de dados em documentos XML tem sido motivo de pesquisas há alguns anos e diversas abordagens foram apresentadas para resolver este problema. Neste capítulo, são apresentados alguns destes trabalhos com suas particularidades e limitações.

Essas abordagens variam em quais algoritmos são utilizados para realizar a mineração e o tipo de documento XML que se está trabalhando. Algumas abordagens são apresentadas por serem pioneiras no assunto, outras são apresentadas por tratarem de documentos XML estáticos, outras por tratarem documentos XML dinâmicos, além de abordagens que variam os algoritmos empregados para a tarefa de mineração em si.

Para melhor dividir este assunto, as abordagens foram classificadas em mineração de documentos XML estáticos e mineração de documentos XML dinâmicos. Com isso, o conteúdo deste capítulo é apresentado da seguinte forma: na seção 3.1 são apresentados os trabalhos sobre documentos estáticos e na seção 3.2 são apresentados os trabalhos sobre documentos dinâmicos. Na seção 3.3 é apresentado o relacionamento entre as abordagens citadas. Na seção 3.4 é apresentada a proposta definida neste trabalho e na seção 3.5 um comparativo entre as abordagens. Por fim, a seção 3.5 apresenta as considerações finais deste capítulo.

3.1 Mineração de documentos XML estáticos

Documentos XML estáticos são aqueles que mantêm suas informações constantes ao longo de sua existência, ou seja, não é habitual a modificação de seu conteúdo.

O trabalho pioneiro neste tipo de documento foi o proposto por BRAGA et al. (2002). Ele propõe a criação de um operador para descrever regras de associação. Este operador se baseia no uso de linguagens de consulta XML, neste caso o *XQuery*, que é um padrão proposto pelo W3C (*World Wide Web Consortium*) (XQUERY, 2014). Além disso, o operador explora plenamente expressões *XPath*, um dos elementos mais importantes do *XQuery*, tanto em termos de facilidade de expressão quanto de computabilidade

(XPATH, 2014).

Este operador permite expressar complexas tarefas de mineração de forma intuitiva e leva em consideração tanto a estrutura quanto o conteúdo do documento XML. A sintaxe do operador, chamado *XMine Rule*, consiste de seis cláusulas, seguindo o formato da Figura 3.1, apresentada a seguir.

```
XMINE RULE
IN Doc-Name (, Doc-Name)*
FOR ROOT IN XPathExpr (, FOR Variable IN XPathExpr)*
LET BODY := (XPathExpr,)+
    HEAD := XPathExpr (,XPathExpr)*
    (, Variable := XPathExpr)*
[WHERE XQuery-Where-Clause]
[GROUP Variable IN Target BY Criteria ]
EXTRACTING RULES WITH
   SUPPORT = <real> AND CONFIDENCE = <real>
   [AND XQuery-Where-Clause]
[RETURN XQuery-Return-Clause]
```

Figura 3.1: Declaração do XMine Rules (BRAGA et al., 2002).

A cláusula *IN* especifica os documentos de origem. *FOR/LET* definem as variáveis obrigatórias *ROOT*, *HEAD* e *BODY*. Dependendo da necessidade do usuário outras variáveis podem ser adicionadas. As variáveis dessa seção são vinculadas a um conjunto de fragmentos, definidos por uma lista de expressões *XPath* separadas por vírgula, definida na figura como *XPathExpr*.

A cláusula *WHERE* é opcional e expressa condições para filtrar os fragmentos identificados em *ROOT*. As palavras chaves *HEAD* e *BODY* podem ser usadas nesta cláusula para referenciar os conjuntos de fragmentos correspondentes. *GROUP* também é opcional e permite a reestruturação dos dados de origem definindo a tarefa de mineração de forma independente da estrutura real dos documentos de origem.

A cláusula *EXTRACTING RULES WITH* especifica um conjunto de restrições sobre as regras de associação geradas. *SUPPORT* e *CONFIDENCE* são obrigatórias e especificam o suporte e confiança mínimos, expressos como um número real. Restrições adicionais podem ser expressas como mostrado na figura em *XQuery-Where-Clause*. As palavras chaves *HEAD* e *BODY* também podem ser utilizadas para referenciar a parte

esquerda e direita das regras. A cláusula *RETURN* é opcional e define a estrutura das regras que se pretende minerar.

A implementação em si é dividida em 3 fases. Na fase de pré-processamento, o operador é processado na ordem que os operadores são definidos. A análise das expressões *XPath*, contidas em *ROOT*, *HEAD* e *BODY*, pode ser realizada por qualquer interpretador *XPath*, como o *Xalan*, implementado em Java (XALAN, 2014). Em seguida, a cláusula *WHERE* é interpretada para filtrar os fragmentos em *ROOT*. Com este processamento, é gerada uma representação do problema de mineração como uma relação binária.

Na fase de mineração, regras de associação binárias são mineradas a partir da representação binária, explorando as restrições contidas na cláusula *EXTRACTING RULES WITH*. Estas restrições estão limitadas ao número de itens da cabeça e corpo da regra, o suporte mínimo e a confiança mínima. Para realizar a tarefa de mineração, esta abordagem utiliza o algoritmo Apriori. Na fase de pós-processamento, as regras extraídas são mapeadas na representação XML de saída.

Neste trabalho os autores não apresentaram uma avaliação detalhada dos testes realizados. Apenas alguns comentários foram tecidos, focando na não preocupação da eficiência da abordagem. Apesar disso, foi observado que os testes realizados foram muito bons devido ao excelente desempenho do interpretador *XPath* utilizado, o *Xalan*. Além disso, observaram que as fases de pré-processamento e pós-processamento foram razoavelmente mais rápidas que a fase de mineração. Este trabalho serviu de base para quase todos os demais trabalhos sobre mineração de documentos XML mostrados nesta monografia.

Outro trabalho, apresentado por ZHANG et al. (2006), propõe a criação de um *framework* para mineração de regras de associação a partir de documentos XML estáticos. É uma abordagem que desenvolve uma adaptação do algoritmo Apriori e realiza a mineração sobre o conteúdo do documento XML e não sobre sua estrutura.

Os dados do documento inicialmente são pré-processados para serem transformados em uma árvore XML indexada (IX-Tree) ou numa base de dados multi-relacional (Multi-DB), dependendo do tamanho do documento e da memória do computador utilizado. Conceitos relevantes para a tarefa de mineração de regras de associação são gene-

ralizados, se necessário, a um grau apropriado, gerando regras ainda mais significativas.

O *framework* é dividido nas seguintes partes: pré-processamento, geração de meta padrões generalizados e geração de regras de associação dos meta padrões generalizados.

Na fase de pré-processamento os dados do documento XML são extraídos e transformados na IX-Tree ou no Multi-DB. Essa escolha é baseada no tamanho do documento XML e principalmente na memória disponível no computador. O IX-Tree é usado quando o tamanho for suficiente para caber na memória, enquanto o Multi-DB é usado nos demais casos.

Em seguida, a tarefa de mineração emitida pelo usuário é analisada e devidamente formulada pelo sistema. Essa formulação envolve a especificação do formato da regra (antecedente e subsequente), a fonte de dados a partir da qual as regras são mineradas, a expressão condicional que determina o alcance dos dados utilizados e os valores mínimos de suporte e confiança.

Os conceitos na mineração de regras de associação são as noções conceituais envolvidas no antecedente e subsequente. Por exemplo, na tarefa de encontrar regras de associação entre o nível de graduação de uma pessoa e o tipo de emprego que ela está envolvida, os conceitos são *graduação* e *emprego*. Estes conceitos são utilizados para gerar os meta padrões generalizados. Por exemplo, o conceito de *suco* pode ser generalizado como *bebida*, e a tarefa de encontrar regras de associação entre *carne* e *suco*, nos costumes de compras de um supermercado, pode ser generalizado em encontrar regras de associação entre *carne* e *bebida*. Duas restrições são usadas para especificar o grau mínimo e máximo de generalização.

As regras de associação dos meta padrões generalizados, que atendem ao suporte e confiança mínimos, são então minerados usando o algoritmo Apriori .

A geração das primeiras regras de associação depende do tipo de conceito a ser extraído. As regras podem ter conceitos simples, onde antecedente e subsequente da regra possuem o mesmo domínio, ou múltiplos, onde antecedente e subsequente da regra podem ter domínios diferentes.

A generalização dos dados é necessária devido ao espalhamento dos dados no documento XML. Normalmente o espalhamento ocorre devido a três fatores: (1) grande

número de instâncias distintas de conceitos; (2) grande número de valores sem conceitos definidos na regra de associação; (3) grande número de conceitos, definidos nos múltiplos conceitos na tarefa de mineração de regras. Esses fatores diminuem o suporte das regras mineradas.

Um esquema para generalização de conceitos em diferentes níveis é especificado por uma tabela, criada pelo usuário, que servirá para guiar o processo de generalização antes da mineração. São definidas também métricas para mensurar o grau de generalização dos simples e múltiplos conceitos. Um exemplo dessa tabela é apresentado a seguir na Figura 3.2. A primeira coluna indica o conceito original e as demais colunas os graus de generalização do conceito.

Original concept	1st level generalized concept	2nd level generalized concept	…	m^{th} level generalized concept
C_{10}	C_{11}	C_{12}	…	C_{1m}
C_{20}	C_{21}	C_{22}	…	C_{2m}
C_{30}	C_{31}	C_{32}	…	C_{3m}
…	…	…	…	…
C_{n0}	C_{n1}	C_{n2}	…	C_{nm}

Figura 3.2: Tabela de referência de generalização (ZHANG et al., 2006).

Para terminar essa fase, é utilizado um algoritmo para encontrar todos os graus das generalizações dos meta padrões, satisfazendo um grau mínimo e máximo estipulado pelo usuário.

Depois de obtidos os meta padrões generalizados, o documento é generalizado para serem gerados os conjuntos de regras de associação, utilizando o algoritmo Apriori com base no suporte e confiança mínimos fornecidos pelo usuário.

Para apresentar os resultados experimentais, os testes foram realizados comparando o *framework* a uma implementação do operador *XMine Rules*, proposto por BRAGA et al. (2002). Para a geração dos documentos XML, foi utilizado a ferramenta *IBM XML Generator* para gerar diversos documentos de variados tamanhos.

Na fase de construção das bases, foi observado que, para a mesma quantidade de dados, a IX-Tree é construída de forma mais rápida que a Multi-DB, justificado pelo fato de que a IX-Tree é trabalhada diretamente na memória RAM. Na fase de mineração,

observou-se que o *framework* apresentado é mais rápido que o *XMine Rules* à medida que a quantidade de dados aumenta. Isto ocorre pois, diferente do método *XMine Rules*, os dados não são recuperados diretamente do documento XML, mas das bases construídas antes da mineração.

Outra abordagem para a mineração de documentos XML estáticos foi apresentada por DING (2006). Foi demonstrada a implementação de dois algoritmos para a extração de regras de associação, o Apriori e o FP-Growth. Estas implementações foram comparadas entre si e entre uma implementação baseada em *XQuery*.

A implementação dos algoritmos foi realizada em Java. Para isto, foi utilizado as bibliotecas DOM (*Document Object Model*) e SAX (*Simple API for XML*) do Java. A biblioteca DOM é um *parser* (transformador) que permite representar um documento XML na forma de uma árvore de objetos em que cada nó representa um componente do documento. Esta árvore compreende todo o documento XML e fica armazenada na memória RAM do computador, o que torna limitado seu uso (DOM, 2014).

Já a biblioteca SAX, diferentemente da DOM, não carrega o documento na memória do computador. Ao invés disso, seu funcionamento ocorre através de um fluxo que percorre o documento XML apenas uma vez, mas a medida que o documento é lido, eventos são disparados de acordo com o elemento observado (o nome de uma *tag* ou o valor de uma atributo, por exemplo) (SAX, 2014).

Na abordagem proposta, a biblioteca DOM é utilizada para representar o documento XML na implementação do Apriori e a biblioteca SAX para o FP-Growth, que percorre o documento uma única vez para a construção da FP-Tree. Como o Apriori exige que o documento seja consultado mais de uma vez, o uso do DOM é mais apropriado por carregar todo o documento na memória RAM e com isso possibilitar um acesso mais rápido aos dados.

Para os dois algoritmos, assumiu-se que o documento foi pré-processado para diminuir os níveis entre as *tags*, ficando cada transação com apenas um nível. Neste nível estão todos os itens que compõem aquela transação. Cada *tag* que representa uma transação possui um atributo para identificar unicamente esta transação. Para realizar este pré-processamento, é sugerido o uso de folhas de estilo, como o XSLT (*eXtensible*

Stylesheet Language for Transformation), que permitem definir como o documento XML deve ser visualizado (XSLT, 2014).

Depois de carregados os documentos na memória, um arquivo de configuração para ser lido pelo Java é criado com os dados do documento XML, como o nome do documento, o nome das *tags* que representam as transações e o nome das *tags* que representam os itens. Em seguida, a base representada pelo DOM ou SAX é repassada ao algoritmo para que os conjuntos de itens frequentes sejam extraídos. Depois que os algoritmos terminam suas tarefas, as regras mineradas são exportadas para um novo documento XML.

Para a realização dos experimentos, as bases foram geradas aleatoriamente a partir de um conjunto de 30 itens sendo que em cada transação havia no máximo 20 itens. Os resultados mostram que a implementação Java do Apriori é muito melhor que a implementação *XQuery* do Apriori, em valores de suporte até 0,4. O FP-Growth é mais rápido que o Apriori até um valor de suporte de 0,4. Os gráficos somente representam resultados para um conjunto de 100 transações e não indica o que ocorre a partir do suporte 0,4, mas nesse ponto os algorítmos se igualam.

A última proposta referente a documentos XML estáticos apresentada neste trabalho foi desenvolvida por ABAZEED et al. (2009). Esta abordagem foca nos dados obtidos a partir da *Web*, que usa o XML como um dos principais padrões para a representação e troca de informações na Internet.

O trabalho faz uma distinção entre duas formas de realizar a mineração de dados a partir de documentos XML. Na primeira abordagem, denominada mineração indireta, o documento XML precisa passar por uma fase de pré-processamento para ser manipulado e minerado. O documento é transformado em outra estrutura para ser repassado ao algoritmo de mineração, além de passar por uma fase de pós-processamento para transformar a saída do algoritmo em uma estrutura desejada. Já na segunda abordagem, denominada como mineração direta, o documento pode ser utilizado diretamente como entrada para o algoritmo. Isso pode ser feito, segundo os exemplos citados na abordagem, utilizando linguagens de consulta XML, como o *XQuery*, ou usando *parsers* (transformadores), como as bibliotecas DOM e SAX da linguagem Java.

Nesta proposta, foi desenvolvida a implementação de uma versão modificada do

algoritmo FLEX (MFLEX), usando as bibliotecas DOM e SAX da linguagem Java, dividida em seis classes.

A classe principal solicita ao usuário o documento XML e o suporte mínimo, e um objeto representando o documento é enviado à classe *Parser*. A classe *Parser* realiza a leitura do XML (usando DOM ou SAX) e mapeia as transações para um objeto *LinkedHashMap* (uma combinação de tabela *hash* com lista encadeada). Essa escolha é feita devido à ordem de iteração e facilidade de recuperação de uma informação.

A classe *RootDependentComparator* gera os subsequentes níveis da árvore FLEX a partir da estrutura inicial mencionada anteriormente. Em seguida, essa estrutura é utilizada para gerar a árvore de saída, com o auxílio de outras duas classes, contendo os itens e seus identificadores de transação associados. Depois os itens são convertidos em um vetor para a contagem de itens semelhantes interseccionados. Por fim, a classe *ResultTree* mostra o resultado.

Os experimentos foram realizados com bases geradas pelos próprios autores. Foram divididas em seis bases com 1000 ou 10000 transações e 5, 10 ou 20 itens por transação. Foi realizada uma comparação entre duas implementações do algoritmo MFLEX, uma usando o SAX e outra o DOM. A implementação usando o SAX supera a implementação usando o DOM, já que o SAX trabalha o XML como um fluxo e a cada transação um evento é acionado para trabalhá-la no FLEX.

3.2 Mineração de documentos XML dinâmicos

Documentos XML dinâmicos são aqueles que alteram suas informações ao longo de sua existência, ou seja, sua estrutura ou seu conteúdo podem ser modificados, ter novas informações incluídas ou algumas removidas.

O trabalho pioneiro neste tipo de documento foi o proposto por CHEN et al. (2004). Ele propõe a mineração de regras de associação a partir do histórico de mudanças de um documento XML, denominado delta estrutural. A abordagem proposta foi chamada de *XML Structural Delta Association Rule* (XSD-AR) e considera a frequência e o grau de mudanças na estrutura dos documentos. Para realizar a tarefa de mineração o algoritmo desenvolvido se baseia no FP-Growth, com a inclusão de estratégias de otimização. Além

disso, o documento XML é representado como uma árvore seguindo as especificações da biblioteca DOM, do Java.

A mineração realizada nesta abordagem foca na estrutura do documento XML e não apenas no seu conteúdo. Isto permite observar se quando uma subestrutura muda, alguma outra parte também muda com uma certa probabilidade. O histórico de mudanças mantém as alterações estruturais ocorridas ao longo do tempo, por isso sendo chamado de delta estrutural.

O trabalho define as seguintes operações de mudanças: inserção, que cria um novo nó em determinada parte do XML, e deleção, que remove determinado nó. Definem também as métricas utilizadas no algoritmo proposto, que são: grau de mudança, para cada subárvore, e frequência de mudança, para um conjunto de subárvores.

O grau de mudança é calculado com base na quantidade de operações simples que uma subestrutura precisou para se modificar de uma versão para outra e no conjunto de nós únicos das árvores que representam os documentos. A frequência de mudança é calculada com base no histórico de versões de um documento XML, nos deltas gerados a cada par de versões, no conjunto de subárvores que constituem os deltas e nos seus respectivos graus de mudança.

Outra métrica definida neste trabalho, o peso, mede o quão significativamente uma subárvore geralmente muda num conjunto. Ela é calculada comparando o grau das subestruturas com um valor especificado pelo usuário. A métrica confiança reflete a probabilidade condicional que significativas mudanças de um conjunto de subárvores levam a mudanças significativas de outro conjunto de subárvores.

Um padrão de subárvores frequentes é um conceito que define um subconjunto de subárvores que estão no conjunto de subárvores que mudam juntas com frequência (normalmente mudando significativamente).

O algoritmo de mineração desenvolvido, baseado no FP-Growth, é chamado de *Weighted-FP-Growth*. O processo de mineração como um todo é dividido em três fases: a construção de uma base de deltas estruturais, a descoberta dos padrões de subárvores frequentes e a derivação das regras de associação a partir do conjunto de padrões frequentes. O trabalho apresentado foca apenas na descoberta dos padrões.

Para manter uma informação importante sobre as subárvores, uma estrutura de dado é definida, considerando que uma subárvore de um delta pode ter dois estados: com grau de mudança menor ou maior que o definido pelo usuário. É utilizado um par de identificadores para representar os dois estados.

A diferença crítica entre o algoritmo original e a versão adaptada da proposta é como se calculam a frequência de mudança e o peso dos padrões, e como é construída a *Weighted-FPtree* condicional.

Quando uma subárvore muda significativamente no delta, a aresta na *Weighted-FPtree* é conectada ao nó filho nomeada como positiva, e caso contrário nomeada como negativa. Então o número de nós é reduzido porque qualquer nó na *Weighted-FPtree* otimizada nunca terá mais que um nó filho representando a mesma subárvore, com a qual ela muda de forma significativa.

Os experimentos foram conduzidos variando os limites para frequência de mudança e peso. Comparando o algoritmo original com sua versão otimizada, foi observado que as duas versões ficam mais eficientes à medida que os limites aumentam. A eficiência das duas versões são muito semelhantes e insensíveis às variações mínimas de peso. A escalabilidade foi avaliada variando o número de deltas e observou-se que a versão otimizada é sensivelmente mais eficiente à medida que o número de nós aumenta.

O trabalho apresentado por ZHAO et al. (2004a) aborda o problema da descoberta de estruturas que mudam frequentemente de acordo com determinados padrões. Esta abordagem leva em consideração a natureza dinâmica dos documentos XML e sua estrutura não ordenada.

A proposta se concentra na mineração de estruturas dinâmicas baseada em padrões a partir do histórico de mudanças de um documento XML. Essas estruturas são subárvores, ligadas ou não, que possuem um padrão de alteração de suas estruturas. Por exemplo, uma turma que ao longo de um ano recebeu várias inscrições. A cada nova versão, alunos são inseridos no nó *turma A* e essa frequente inserção pode indicar um momento de popularidade dessa turma. As estruturas mineradas podem ser úteis na indexação, extração de significado semântico e detecções de mudanças de documentos XML.

Para realizar a mineração, a abordagem propõem um algoritmo de três fases,

antecedidos pela criação de um modelo, denominado SMH-Tree (*Structure History Tree*), para guardar o histórico de mudanças estruturais. Este modelo é uma extensão baseada na biblioteca DOM com algumas propriedades de histórico, tornando possível comprimir o histórico de mudanças do XML. Dada uma sequência de documentos XML, o SMH-Tree é inicializado com as estruturas da primeira versão e após isso o algoritmo itera por todas as outras versões extraindo as mudanças estruturais, realizada utilizando o algoritmo para detecção de mudanças SX-Diff, e realizando o mapeamento na SMH-Tree. O SX-Diff é uma modificação do X-Diff que somente gera as estruturas que foram modificadas entre duas versões diferentes de um documento.

Na primeira fase do algoritmo, as estruturas são mapeadas na SMH-Tree de acordo com regras de mapeamento do algoritmo. Para avaliar o comportamento das estruturas, é proposto um conjunto de métricas dinâmicas. A primeira é chamada SMF (*Structure Mutation Factor* - Fator de Mutação da Estrutura), que mensura o quão significativa foi a mudança de uma estrutura entre uma versão e outra. A segunda é chamada VMF (*Version Mutation Factor* - Fator de Mutação da Versão), que mensura o quão frequente uma estrutura mudou. Na segunda fase, tendo a SMH-Tree como entrada, todas as regras dinâmicas baseadas em padrões válidas são descobertas. Os valores dos parâmetros são calculados, comparados com os predefinidos e as estruturas válidas são retornadas. Para estes cálculos, são usados valores definidos pelo usuário para VMF, *strength* (força) e *tolerance* (tolerância).

Strength é uma métrica usada para refletir como as alterações dos valores SMF de uma versão para outra influenciam no aumento do padrão. É o número de mudanças que acompanham o aumento do padrão de mudança em função do número total de vezes que uma estrutura muda. O valor de força é definido entre 0 e 1. Uma vez que é possível que algumas das alterações possam não corresponder com o aumento do padrão, a *tolerance* é definida para restringir outros padrões de alteração. *Tolerance* é uma métrica usada para limitar a significância máxima da diminuição dos padrões no histórico. A terceira fase não é explicada no trabalho, por se tratar apenas de uma representação visual das estruturas.

Os experimentos foram realizados usando conjuntos de dados a partir da DBLP,

uma base de dados com informações bibliográficas de jornais científicos, e SIGMOD, outra base de dados de diversos artigos. Usando um conjunto de dados SIGMOD, foram geradas 20 versões de documentos XML, com média de 468Kb e 1120 nós, para mostrar a eficiência do algoritmo e observar o tempo de execução em diferentes fases do processo.

Os resultados mostraram que as três restrições fornecidas influenciam na performace do algoritmo. Quanto mais os valores de VMF e *tolerance* aumentam, menor o tempo de execução. Ocorre o oposto com o *strengh*. Foi observado também que a fase de detecção de diferenças com o SX-Diff é 50% do tempo total de execução do algoritmo, e que o tamanho da SMH-Tree é cerca de 40% do tamanho original da sequência de documentos XML.

Os mesmos pesquisadores propõem outra abordagem (ZHAO et al., 2004b) que é baseada na identificação de estruturas que se alteram frequentemente através de uma abordagem denominada mineração de delta estrutural. O objetivo é extrair informações através de sequências de mudanças estruturais no documento XML. Aqui também é levada em consideração a natureza dinâmica dos documentos XML e sua estrutura não ordenada.

Para realizar a mineração, a abordagem propõem um algoritmo com três fases principais. A primeira fase é a criação de um modelo, denominado H-DOM (*Historical Document Object Model*), para guardar o histórico de mudanças estruturais. Este modelo é uma extensão da biblioteca DOM com algumas propriedades de histórico, tornando possível comprimir o histórico de mudanças do XML. Dada uma sequência de documentos XML, o H-DOM é inicializado com as estruturas da primeira versão e após isso o algoritmo itera por todas as outras versões extraindo as mudanças estruturais utilizando o SX-Diff e mapeando elas no H-DOM. O SX-Diff é uma modificação do X-Diff que somente gera as estruturas que foram modificadas entre duas versões diferentes de um documento. Esta fase se assemelha muito à criação do modelo SMH-Tree da proposta anterior.

Na segunda fase, tendo o H-DOM como entrada, todas as estruturas que mudam com frequência são descobertas. As métricas são calculadas com base nos valores obtidos do H-DOM, comparadas com os valores predefinidos pelo usuário para as métricas *Structure Dynamic*, *Version Dynamic* e *Degree of Dynamic*, e as estruturas são então retornadas. Na terceira fase (visualização) as estruturas que mudam com frequência são

apresentadas ao usuário como uma representação em árvore que podem ser facilmente interpretadas. Não é detalhada esta fase por ser considerada trivial.

Os resultados dos experimentos mostraram que o algoritmo extrai com sucesso todas as estruturas frequentes. Também mostraram que o H-DOM é muito compacto, ficando com 50% do tamanho original da sequência de documentos XML. Foi observado que com técnicas de otimização de espaço (memória) é possível fazer o H-DOM mais compacto em torno de 20% e consequentemente fazer o algoritmo ser mais escalável, tornando-o capaz de processar uma sequência de 450Mb de documentos XML.

A última abordagem apresentada neste trabalho, e mais próxima da abordagem proposta, foi desenvolvida por RUSU et al. (2006) e tem como objetivo a mineração de regras de associação a partir de documentos XML. Essa proposta foca na mineração de documentos XML dinâmicos, permitindo que as regras extraídas também possam trazer informações de como as mudanças ocorridas em diferentes partes dos documentos podem estar relacionadas.

A abordagem propõe a construção de um algoritmo genérico para a extração das regras de associação baseado no algoritmo Apriori, que foi redesenhado para suportar documentos XML. O processo é desenvolvido em cinco fases, sendo uma fase de preparação e as demais de mineração propriamente dita.

Antes de iniciar o processo, o delta consolidado precisa ser obtido. Este delta contém o histórico de mudanças que ocorreram no documento XML original. Ele é gerado por algoritmos de detecção de diferenças entre documentos XML, e neste trabalho foi utilizado o algoritmo X-Diff (WANG et al., 2003). Este algoritmo recebe duas versões de um documento XML como entrada para gerar o delta consolidado.

A partir do delta consolidado, tem-se início a fase de preparação, que consiste na criação de um documento construído com base nas modificações contidas no delta. Este documento armazena os elementos modificados e a quantidade de transações encontradas em todo o processo, que é utilizada no cálculo do suporte nas fases seguintes.

Na primeira fase, um novo documento XML é construído, contendo os pares *elemento-modificação*, o número de modificações deste par e o suporte associado a ele. A cada novo par encontrado, se já existente, sua quantidade é incrementada e o novo

suporte é calculado. Na segunda fase, o conjunto de itens de tamanho 1, que satisfaça o suporte mínimo estipulado, é extraído do documento gerado na primeira fase.

Na terceira fase, como no Apriori original, os demais conjuntos de itens de tamanho k são construídos a partir do conjunto de itens de tamanho 1 da fase anterior, até que todos os conjuntos de itens, que satisfaçam o suporte mínimo, sejam encontrados. Na quarta e última fase, com base nos conjuntos de itens extraídos na fase anterior, as regras de associação são determinadas e seus valores de confiança calculados.

O trabalho foi avaliado utilizando-se o delta consolidado obtido a partir de diversas versões de um mesmo documento XML. Foram utilizados arquivos de 25, 63, e 127 KB, da base de dados SIGMOD. Para obter as várias versões, foi utilizado um algoritmo de simulação de mudanças. São fornecidas as porcentagens de mudanças desejadas, entre atualizações, inclusões e deleções, e os documentos são então alterados aleatoriamente. Para cada nova versão, um novo delta consolidado é calculado.

Para a realização dos teste, o algoritmo foi implementado na linguagem *Visual Basic* e diferentes medições de tempo de execução foram realizadas para diferentes valores de suporte mínimo. De acordo com o trabalho, a fase de preparação e a primeira fase são as mais custosas de todo o processo, servindo como proposta de futuras melhorias.

3.3 Relacionamento entre as abordagens

A Figura 3.3 apresenta o relacionamento entre as abordagens apresentadas.

Como mostra a figura, a abordagem de BRAGA et al. (2002), pioneira na mineração de documentos XML estáticos, é a mais referenciada dentre os trabalhos selecionados. Ela serviu como base para que CHEN et al. (2004) estendesse a ideia de mineração de documentos XML, para que a tarefa fosse realizada sobre documentos XML dinâmicos.

Em seguida, ZHAO et al. (2004a,b) se baseia na abordagem de CHEN et al. (2004) para desenvolver seu trabalho de mineração de padrões frequentes. Apesar de não minerar diretamente regras de associação, ele usa a ideia de trabalhar com documentos XML dinâmicos, com base no histórico de versões, como utilizado por CHEN et al. (2004). Por fim, para documentos XML dinâmicos, RUSU et al. (2006) se baseia no trabalho de ZHAO et al. (2004a), mas focando na mineração de regras de associação e se diferenciando

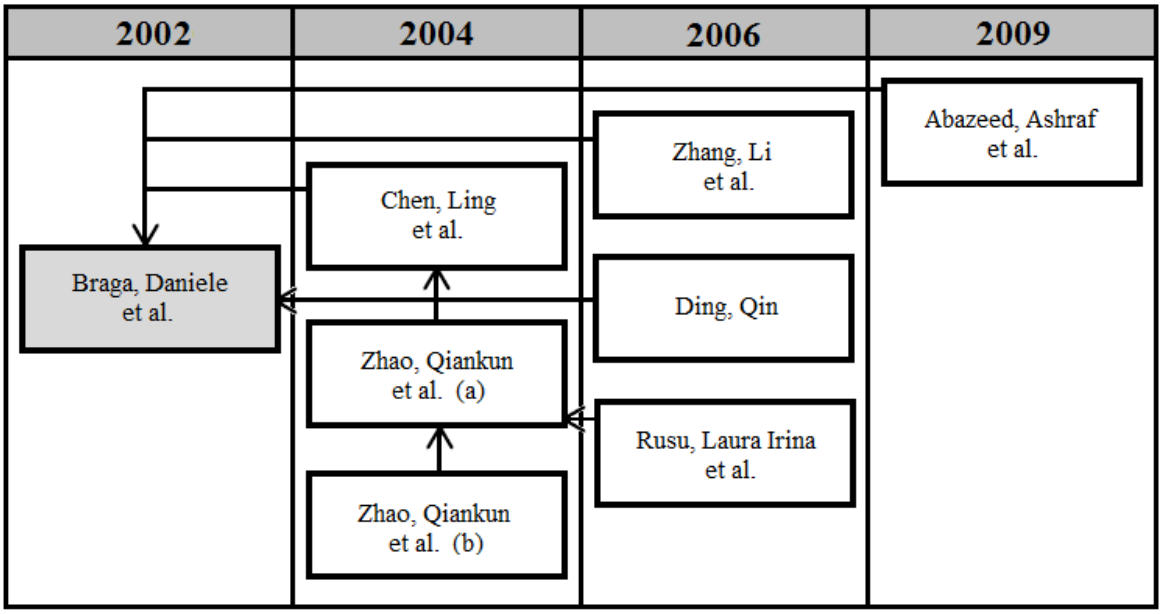


Figura 3.3: Relacionamento entre as abordagens.

do trabalho de CHEN et al. (2004) por desenvolver seu algoritmo com base no Apriori.

Os demais trabalhos, apresentados por ZHANG et al. (2006), DING (2006) e ABAZEED et al. (2009), se baseiam diretamente na abordagem de BRAGA et al. (2002), focando em documentos XML estáticos, cada um com uma característica diferente que são novamente apresentadas no comparativo entre as abordagens no final deste capítulo.

3.4 XChangeMining: Abordagem proposta

A proposta deste trabalho é realizar a mineração de regras de associação, a partir do histórico de mudanças ocorridas na evolução de documentos XML. A tarefa de associação foi escolhida por buscar dados que ocorrem juntos numa base de dados. Para este trabalho, espera-se encontrar elementos de um registro que com certa frequência alteram-se juntos. Com isso, o XChangeMining permite observar relações nas mudanças ocorridas que não estão claras para o usuário.

O fluxo de execução da abordagem de mineração tem início com a escolha das versões dos documentos XML que são utilizadas na mineração (duas versões ou mais). Em seguida, são definidos os elementos do documento a serem considerados. Neste momento o usuário pode remover alguns elementos que considere desnecessários. O próximo passo é a

detecção das diferenças entre as versões dos documentos, que em seguida são manipuladas para formar o documento ARFF (*Attribute-Relation File Format* - Formato de Arquivo Atributo-Relação). Este documento serve como entrada para o algoritmo de mineração, que vai processá-lo, desconsiderando os elementos removidos pelo usuário e com base nas métricas fornecidas, e retorna as regras mineradas.

O pseudocódigo da abordagem proposta pode ser visualizado na listagem 3.1. O processo tem início com o recebimento de seis parâmetros: as versões escolhidas do documento XML, os elementos que são considerados na mineração, a chave que identifica unicamente cada registro do documento, a métrica escolhida para ser usada pelo Apriori e os valores mínimos para o suporte e a métrica escolhida. Para cada par de versões, uma lista de diferenças entre elas é gerada a partir do algoritmo criado neste trabalho. O processo de detecção de diferenças continua até que todas as versões sejam analisadas e as novas diferenças são adicionadas na lista.

Algoritmo 3: Pseudoódigo da abordagem proposta

```
Entrada: lDoc : Lista de documentos, Elementos : Lista de elementos,
         chave, metrica, suporteMin, metricaMin
Saída: lRegras : lista de regras de associacao
início
    listaDiff
    docARFF
    para todo (doc1, doc2 : par de documentos em lDoc) faça
        listaDiff = Diff(doc1, doc2, chave, listaDiff)
    fim para todo
    para todo elemento : Elementos faça
        escrever o elemento no cabecalho de docARFF
    fim para todo
    para todo item : listaDiff faça
        para todo elemento : Elementos faça
            if ( item esta contido em elemento ) then
                escrever 'y' na linha de dados de docARFF
            end if
            else
                escrever '?' na linha de dados de docARFF
            end if
        fim para todo
    fim para todo
    regras = WekaApriori( docARFF, metrica, suporteMin, metricaMin )
    para todo regra : regras faça
        lRegras = elementos de regras
    fim para todo
    retorna lRegras
fim
```

Com a lista de diferenças defininda, o próximo passo do processo é a transformação dessa lista no ARFF, para servir como entrada para o algoritmo de mineração. Um exemplo deste documento é apresentado a seguir na Figura 3.4.

O ARFF é constituído por duas seções: um cabeçalho e os dados propriamente ditos. O cabeçalho é composto pela declaração, mapeada como *@relation*, que define o nome da relação a que se refere a base, e pelos atributos, mapeados como *@attribute*. Cada atributo possui um nome e o tipo de dado a que se refere, e a ordem que são definidos é importante pois são levadas em consideração na seção de dados. Os tipos dos atributos podem ser: numéricos, cadeia de caracteres, datas e valores fixos.

```
@relation root
@attribute 'siape' {y}
@attribute 'firstname' {y}
@attribute 'lastname' {y}
@attribute 'ssn' {y}
@attribute 'phone' {y}
@attribute 'birthdate' {y}
@attribute 'maritalstatus' {y}
@attribute 'removed' {y}
@attribute 'datelastremoved' {y}
@attribute 'jobschema' {y}
@attribute 'title' {y}
@attribute 'level' {y}
@attribute 'class' {y}
@attribute 'departament' {y}
@attribute 'university' {y}
@attribute 'graduateprogram' {y}
@attribute 'numbercourse' {y}
@attribute 'numberadvisors' {y}
@attribute 'classificationpaper' {y}
@data
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,y,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?,?
?,?,?,?,y,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,y,?,?,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?,?,?,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?,?,?,?,?,?,?
?,?,?,y,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y
?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,y,?,?
```

Figura 3.4: Exemplo de ARFF.

A seção de dados, mapeada como *@data*, contém a declaração de todos os registros utlizados como base de dados. Cada linha desta seção representa um registro e cada atributo é separado por uma vírgula. Valores vazios são representados pelo caractere '?'. Estes atributos devem estar na mesma ordem em que foram definidos no cabeçalho (WEKA, 2014).

Na abordagem proposta, algumas definições foram consideradas para simplificar o mapeamento dos dados referentes às diferenças detectadas entre as versões dos documentos XML. Em primeiro lugar, todos os atributos foram definidos como do tipo *valores fixos*, através do caractere '*y*'. A segunda definição corresponde à consideração de que todos os atributos de um determinado registro, que não tiveram valores alterados, são considerados como valores vazios, representados, na seção de dados, pelo caractere '?'.

Essas escolhas foram definidas para que o número de informações passadas ao algoritmo sejam mínimas. Como a mineração pretende encontrar informações de quais atributos se alteram entre as versões dos documentos XML, e não sobre o conteúdo dos atributos, as definições escolhidas não trazem prejuízo à qualidade dos resultados.

Com o ARFF definido, este é repassado para o algoritmo para realizar a mineração. Na proposta apresentada, utiliza-se o projeto *Weka*, que permite realizar diversas tarefas de mineração de dados. Este projeto fornece uma biblioteca versão *jar* (*Java Archive Files*), para que possa ser facilmente utilizada em projetos Java, linguagem escolhida para o desenvolvimento da proposta. Depois de analisar o ARFF e realizar a tarefa de mineração solicitada, o *Weka* retorna uma lista com todas as regras de associação mineradas.

Esta lista é então interpretada, tranformando as informações mineradas em uma lista contendo apenas os elementos que compõem a regra encontrada, e então retornadas ao usuário.

3.5 Comparativo entre as abordagens

Nesta seção são apresentadas algumas características comuns entre todas as abordagens citadas neste trabalho, a fim de melhor observar os pontos mais importantes de cada uma e fornecer uma análise comparativa mais direta. A Tabela 3.1 apresenta essas carac-

terísticas.

Tabela 3.1: Comparativo entre as abordagens

Abordagem	Tipo de XML	Árvore	Análise	Algoritmo	Mineração
Braga (2002)	estático	ordenada	conteúdo	Apriori	regras de associação
Chen (2004)	dinâmico	não-ordenada	estrutura	FP-Growth	regras de associação
Zhao (2004)	dinâmico	não-ordenada	estrutura	Criado pelos autores	padrões frequentes
Rusu (2006)	dinâmico	não-ordenada	estrutura	Apriori	regras de associação
Zhang (2006)	estático	ordenada	conteúdo	Apriori	regras de associação
Ding (2006)	estático	ordenada	estrutura	Apriori e FP-Growth	regras de associação
Abazeed (2009)	estático	ordenada	estrutura	FLEX	regras de associação
XChangeMining	dinâmico	não-ordenada	conteúdo	Apriori	regras de associação

As abordagens de BRAGA et al. (2002), ZHANG et al. (2006), DING (2006) e ABAZEED et al. (2009), apesar de apresentarem boas ideias na transformação dos documentos XML e na adaptação de alguns algoritmos, não podem ser diretamente utilizadas, pois se limitam a minerar apenas documentos XML estáticos, e no contexto deste trabalho deve-se trabalhar com documentos XML dinâmicos e seu histórico de mudanças.

Dentre os trabalhos que levam em consideração a natureza dinâmica dos documentos XML, o proposto por ZHAO et al. (2004a) não foca na mineração de regras de associação, mas sim na mineraçde padrões frequentes, contrariando a proposta deste trabalho.

Os trabalhos de CHEN et al. (2004) e RUSU et al. (2006) são os mais próximos da abordagem proposta, mas não abrangem completamente o escopo ideal por minerarem estruturas inteiras do documento XML e não apenas seu conteúdo.

Com isso, a abordagem deste trabalho se mostra mais adequada ao objetivo proposto por considerar, ao mesmo tempo:

- a utilização de documentos XML que evoluem com o tempo;
- sua característica como árvore não-ordenada, o que permite não considerar a troca

de posições dos elementos como uma alteração entre as versões;

- a utilização de um algoritmo de detecção de diferenças que não se limite pelo tamanho do documento;
- o foco da análise no conteúdo do documento e não na sua estrutura;
- o uso de um algoritmo eficiente e já consolidado como ferramenta de mineração.

Além disso, os trabalhos que consideram a evolução dos documentos XML exigem uma alta escalabilidade no tratamento de grandes quantidades de dados. As três abordagens dinâmicas, CHEN et al. (2004), ZHAO et al. (2004a) e RUSU et al. (2006), utilizam o DOM como forma de representação dos documentos XML para que possam ser mais facilmente trabalhados, o que traz uma limitação quanto a quantidade de dados que possam ser analisados. No trabalho proposto, o uso do SAX permite que o tamanho das bases não se torne um limitador para o XChangeMining.

3.6 Considerações finais

Conforme visto neste capítulo, diversas abordagens ao longos dos anos trataram a mineração de dados sobre documentos XML de várias formas. A maioria das abordagens observadas se propuseram a trabalhar apenas com documentos XML estáticos e as que focaram em documentos XML dinâmicos pouco evoluíram no desenvolvimento de uma ferramenta concreta e abrangente. Desta forma, foi possível analisar e aproveitar as melhores características das abordagens levantadas e definir a abordagem proposta por esse trabalho que se mostra mais completa.

O capítulo seguinte apresenta um exemplo prático de utilização da abordagem definida, o XChangeMining, como foi desenvolvida a base de dados utilizada nos exemplos e uma breve análise dos resultados obitidos.

4 Exemplo de utilização

Depois de terem sido apresentados os conceitos relevantes sobre a mineração de dados e sua aplicação em documentos XML, bem como os trabalhos relacionados ao tema, este capítulo apresenta um exemplo prático de utilização do XChangeMining. Este módulo foi desenvolvido para auxiliar o *XChange* na criação de regras *Prolog*, utilizadas para a realização de inferências na ferramenta.

Assim, o capítulo está dividido da seguinte forma: a seção 4.1 descreve como foi gerada a base de dados empregada. A seção 4.2 apresenta um exemplo de utilização do XChangeMining, passo a passo. A seção 4.3 faz uma breve análise dos tempos gastos nas etapas do processo, além da qualidade dos resultados apresentados. Por fim, a seção 4.5 conclui o capítulo.

4.1 Geração da base de dados

Para apoiar a realização do exemplo de utilização, foi gerada uma base de dados sintética, de forma aleatória, além de versões modificadas a partir da versão original, para simular a evolução de documentos XML. O processo de geração da base será definido a seguir.

A base foi criada no contexto de um cadastro de professores universitários, onde pode ser observado, entre outras informações, os requisitos necessários para o credenciamento e descredenciamento de professores em programas de pós-graduação. As informações são descobertas pela mineração de dados com base nas alterações entre as versões do documento XML. Esses requisitos foram adaptados do regulamento de seleção de professores da Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina[1].

[1] http://ppgcc.posgrad.ufsc.br/

4.1.1 Definições

Foram escolhidos vinte e nove elementos para representar cada professor do documento XML. Parte desses elementos não sofreram alterações entre as modificações geradas a cada versão e, para os que sofreram, foram definidos os percentuais de alterações que deveriam ocorrer entre cada versão. Uma lista com a descrição dos elementos e os percentuais definidos de alterações pode ser visualizados na Tabela 4.1.

Tabela 4.1: Lista de elementos

Atributos	Descrição	Percentual
siape	Código único de identificação	
firstName	Nome do professor	
lastName	Sobrenome do professor	
ssn	CPF ou outro documento de identificação	1%
phone	Telefone	0,5%
email	Email	0,5%
birthdate	Data de aniversário	
sex	Sexo	
fatherName	Nome do pai	
motherName	Nome da mãe	
hiredate	Data de adimissão	1%
maritalStatus	Estado civil	1%
bloodGroup	Grupo sanguíneo	
placeBirth	Cidade de nascimento	
nationality	Nacionalidade	
medicalRemoved	Se já foi afastado	1%
dateLastRemoved	Data do último afastamento	1%
jobSchema	Regime de trabalho	0,5%
title	Titulação	1%
level	Nível do cargo	1,5%
class	Nome do cargo	0,8%
salary	Salário	1,5%
departament	Departamento	0,5%
university	Universidade	0,4%
graduateProgram	Programa de Pós-Graduação que participa	0,2%
numberCourse	Número de disciplinas ministradas	3%
numberAdvisors	Número de orientandos	4%
classificationPaper	Grupo de classificação dos artigos publicados	3,5%

Para compor o documento XML, foi estipulada a criação de uma base inicial de vinte mil professores e a geração de dez versões modificadas, uma a uma, a partir da original. A cada nova versão, quarenta novos professores foram incluídos e dez foram excluídos, além das modificações de cada elemento de acordo com os percentuais estabelecidos. As inclusões são feitas a partir de um documento menor, de dois mil professores,

criado apenas para apoiar a etapa de inclusão. Um resumo das quantidades definidas pode ser visualizado na Tabela 4.2.

Tabela 4.2: Definições da base gerada

Total de professores (versão base)	20.000
Número de elementos (por registro)	29
Número de versões	10
Número de inclusões (à cada versão)	40
Número de exclusões (à cada versão)	10

Para o contexto aplicado, e com base nos elementos escolhidos, foram definidas algumas regras que podem ser utilizadas para inferir algum sentido semântico nas mudanças ocorridas, e que se espera que sejam encontradas com a mineração a partir do XChangeMining. A lista de regras e os elementos que às compõem podem ser observados na Tabela 4.3.

Tabela 4.3: Regras definidas

Transferência		
universityBefore	!=	universityAfter

Credenciamento como permanente		
titleAfter	==	doutor
classificationPaperAfter	==	grupo1
jobSchemaAfter	==	integral
numberAdvisorAfter	<=	8

Credenciamento como visitante		
titleAfter	==	doutor
classificationPaperAfter	==	grupo1
jobSchemaAfter	==	parcial
universityAfter	!=	UFSC

Descredenciamento de permanentes efetivos		
classificationPaperAfter	==	grupo4

Promoção		
salaryBefore	!=	salaryAfter
classBefore	!=	classAfter
titleBefore	!=	titleAfter
levelBefore	!=	levelAfter

Recredenciamento de permanentes efetivos		
classificationPaperAfter	==	grupo1
numberAdvisorAfter	>=	1
numberCourseAfter	>=	1

Descredenciamento de permanentes fora UFSC		
titleAfter	!=	doutor
classificationPaperAfter	!=	grupo1
numberAdvisorAfter	==	0

4.1.2 Geração da versão inicial

Para apoiar a criação da versão inicial do documento XML, foi utilizada uma ferramenta de geração de dados sintéticos chamada *GenerateData* (GENERATEDATA, 2014). Ela

permite a criação de grandes bases de dados sintéticos, com o envolvimento de diversos atributos de características reais como nomes, endereços, números de telefones, e-mails, entre outros.

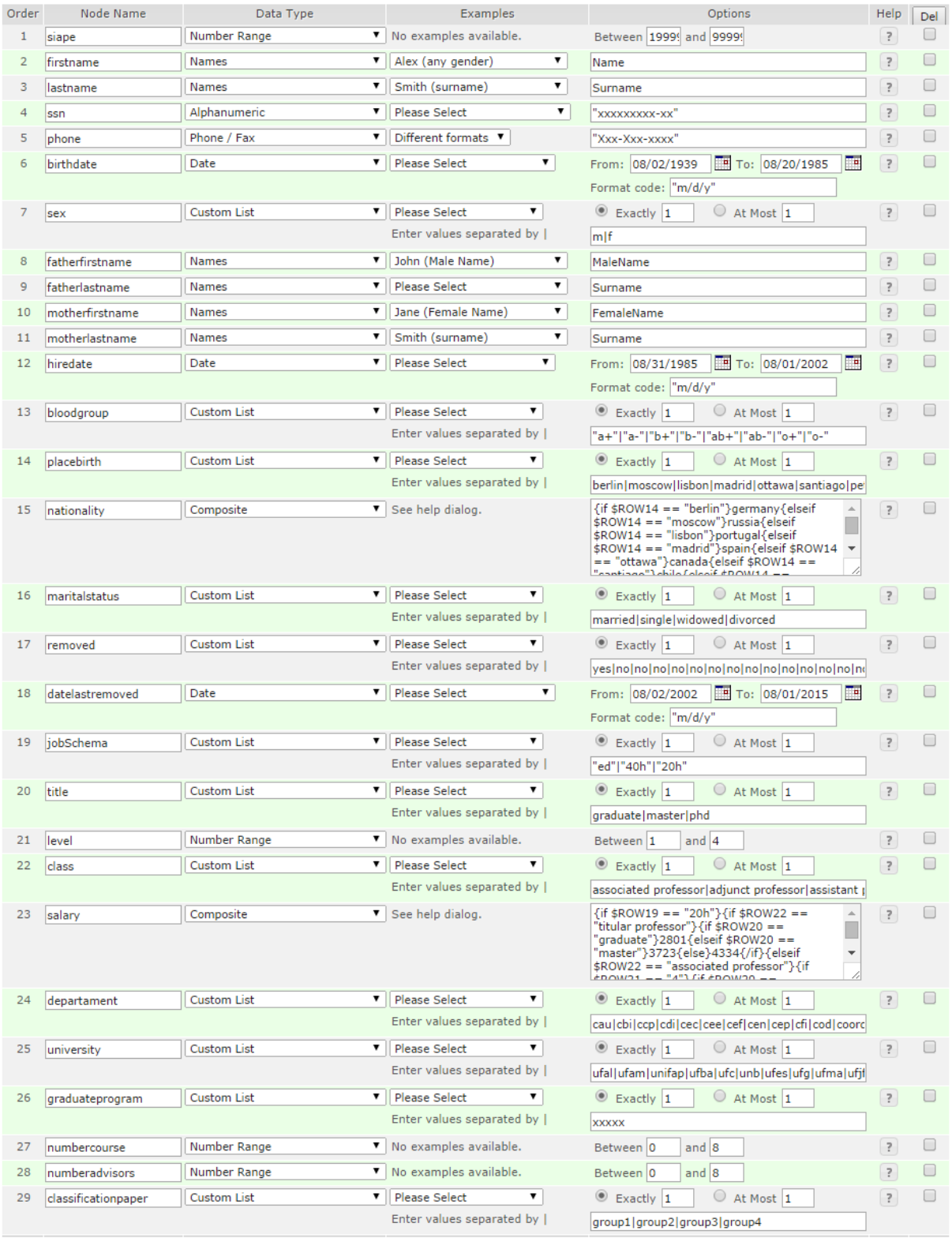
Esta ferramenta foi desenvolvida na linguagem PHP e pode ser utilizada por qualquer pessoa de forma gratuita, bastando configurar um equipamento que possa executar projetos PHP. Neste trabalho foi utilizado o servidor de aplicação Apache 2.0 e o banco de dados MySQL, sugeridos pela ferramenta (PHP, 2014; MYSQL, 2014).

Neste exemplo de utilização, os vinte e nove elementos definidos foram representados utilizando diversas opções de preenchimento de valores disponíveis na ferramenta. Algumas dessas opções vasculham uma base de dados própria da ferramenta para definir quais valores são escolhidos. Outras opções são baseadas em valores numéricos, personalizáveis pelo usuário, e outras são baseadas em listas de valores que o usuário pode fornecer à ferramenta. A Figura 4.1 mostra como foram definidos os elementos.

Os elementos *firstname*, *lastname*, *fatherfirstname*, *fatherlastname*, *motherfirstname* e *motherlastname* são definidos como do tipo *Names* e são preenchidos com valores da base de nomes e sobrenomes da ferramenta. Os elementos *birthdate*, *hiredate* e *datelastremoved* são definidos como do tipo *Date* e escolhidos com base em intervalos de datas, para facilitar o controle da coerência entre datas. O campo *phone*, definido como *Phone/Fax* e *ssn*, como *Alphanumeric*, possuem uma máscara que representa o formato que o valor a ser preenchido deve possuir. O elemento *siape* é definido como do tipo *Number Range*, bem como *level*, *numbercourse* e *numberadvisors*. Neste tipo de campo, os valores são sorteados com base no intervalo definido pelo usuário.

Um tipo muito útil para a composição deste trabalho é o chamado *Custom List*, associado aos elementos *sex*, *bloodgroup*, *placebirth*, *maritalstatus*, *removed*, *jobSchema*, *title*, *class*, *departament*, *university*, *graduateprogram* e *classificationpaper*. Ele permite definir uma lista de valores a serem considerados no sorteio daquele campo. Para aumentar a chance de um valor ser escolhido, basta repetir o valor quantas vezes supor necessário, e para definir valores em branco, basta incluir um valor vazio na lista.

Para cada um dos elementos citados, foi feita uma pesquisa para encontrar valores reais a serem considerados. Foram escolhidas sessenta e quatro universidades brasileiras



Order	Node Name	Data Type	Examples	Options	Help	Del
1	siape	Number Range	No examples available.	Between 1999 and 9999	?	
2	firstname	Names	Alex (any gender)	Name	?	
3	lastname	Names	Smith (surname)	Surname	?	
4	ssn	Alphanumeric	Please Select	"xxxxxxxxx-xx"	?	
5	phone	Phone / Fax	Different formats	"Xxx-Xxx-xxxx"	?	
6	birthdate	Date	Please Select	From: 08/02/1939 To: 08/20/1985 Format code: "m/d/y"	?	
7	sex	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 m|f	?	
8	fatherfirstname	Names	John (Male Name)	MaleName	?	
9	fatherlastname	Names	Please Select	Surname	?	
10	motherfirstname	Names	Jane (Female Name)	FemaleName	?	
11	motherlastname	Names	Smith (surname)	Surname	?	
12	hiredate	Date	Please Select	From: 08/31/1985 To: 08/01/2002 Format code: "m/d/y"	?	
13	bloodgroup	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 "a+"|"a-"|"b+"|"b-"|"ab+"|"ab-"|"o+"|"o-"	?	
14	placebirth	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 berlin|moscow|lisbon|madrid|ottawa|santiago|pe	?	
15	nationality	Composite	See help dialog.	{if $ROW14 == "berlin"}germany{elseif $ROW14 == "moscow"}russia{elseif $ROW14 == "lisbon"}portugal{elseif $ROW14 == "madrid"}spain{elseif $ROW14 == "ottawa"}canada{elseif $ROW14 ==	?	
16	maritalstatus	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 married|single|widowed|divorced	?	
17	removed	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 yes|no|no|no|no|no|no|no|no|no|no|no|no|n	?	
18	datelastremoved	Date	Please Select	From: 08/02/2002 To: 08/01/2015 Format code: "m/d/y"	?	
19	jobSchema	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 "ed"|"40h"|"20h"	?	
20	title	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 graduate|master|phd	?	
21	level	Number Range	No examples available.	Between 1 and 4	?	
22	class	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 associated professor|adjunct professor|assistant	?	
23	salary	Composite	See help dialog.	{if $ROW19 == "20h"}{if $ROW22 == "titular professor"}{if $ROW20 == "graduate"}2801{elseif $ROW20 == "master"}3723{else}4334{/if}{elseif $ROW22 == "associated professor"}{if	?	
24	departament	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 cau|cbi|ccp|cdi|cec|cee|cef|cen|cep|cfi|cod|coorc	?	
25	university	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 ufal|ufam|unifap|ufba|ufc|unb|ufes|ufg|ufma|ufjf	?	
26	graduateprogram	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 xxxxx	?	
27	numbercourse	Number Range	No examples available.	Between 0 and 8	?	
28	numberadvisors	Number Range	No examples available.	Between 0 and 8	?	
29	classificationpaper	Custom List	Please Select Enter values separated by |	Exactly 1 At Most 1 group1|group2|group3|group4	?	

Figura 4.1: Preenchimento dos campos na ferramenta *GenerateData*.

para compor a base, e alguns programas de pós-graduação das respectivas universidades. As cidades-sede dessas universidades foram aproveitadas para compor a base de cidades natais dos professores, adicionadas de dez cidades internacionais para representar professores estrangeiros. Uma pesquisa foi feita também com relação aos cargos e salários dos professores. Foram definidos quatro classes de professores, quatro variações de níveis para estas classes, três titulações e três jornadas de trabalhos para compor uma tabela de salários coerente. Estes dados foram obtidos e adaptados a partir do plano de carreira da Universidade Tecnológica Federal do Paraná[2].

Estes salários foram preenchidos com base em outro tipo de dado muito interessante da ferramenta, o *Composite*. Ele permite usar referências a outros campos já preenchidos para decidir qual o valor a ser utilizado. Os elementos que utilizam este tipo de dado são *nationality*, que se baseia na cidade sorteada para preencher o campo com o país associado, e *salary*, que através de uma sequência de testes, analisa os campos *class*, *level*, *title* e *jobschema* para preencher o salário correto do professor.

Definidos todos os campos, a próxima etapa é estipular quantos registros devem ser criados e qual o tipo de saída pretendida. A ferramenta permite gerar bases prontas nos formatos CSV, Excel, HTML, JSON, LDIF, diretamente em algumas linguagens, SQL e, o caso aplicado a esta abordagem, XML. Neste caso, são definidos ainda um nome para a raiz do documento, neste caso *professors*, e o nome da *tag* que representa cada registro, definida como *professor*. É possível também realizar uma exportação direta no tipo de arquivo escolhido ou na própria tela.

A ferramenta possui uma versão *online*, que permite a geração de apenas cem registros. Para maiores quantidades de registros, a versão *off-line* deve ser instalada pelo usuário.

4.1.3 Correções e geração das versões

Depois de constituída a base inicial, uma análise sobre ela é realizada para corrigir alguns valores que possam estar incoerentes dentro do contexto. Para isso, foi desenvolvido um analisador que corrige possíveis erros referentes às datas, como uma data de nascimento

[2]http://www.utfpr.edu.br/servidores/pagamento/tabela-de-remuneracao-dos-docentes

maior que uma data de adimissão, universidades e seus programas de pós-graduação, que podem não estar corretamente associados, cargos e salários dos professores, entre outras.

Com os dados corrigidos, a versão original é dividida entre a versão inicial, que gera as demais versões, e uma base de inclusões, onde são buscados os professores a serem adicionados a cada nova versão. Em seguida, são definidos quais registros terão algum elemento alterado, de acordo com os percentuais de alterações proposta. Um pequeno algoritmo de aleatoriedade foi construído para apoiar a escolha desses registros, considerando o tamanho da base e a quantidade de alterações desejada. A cada nova versão, esta quantidade de elementos é modificada, quarenta professores são adicionados e dez são removidos, como mencionado anteriormente.

Com estes passos, obtém-se uma base de dados relativamente grande e coerente para apoiar os testes realizados neste trabalho.

4.2 Exemplo prático

Para exemplificar o funcionamento do XChangeMining, a seguir é apresentado um exemplo de utilização empregando a base sintética de professores desenvolvida neste trabalho. Primeiro é apresentado o *XChange* e como são criadas as regras *Prolog* e em seguida a utilização do XChangeMining como abordagem auxiliar na criação dessas regras.

4.2.1 *XChange*

O *XChange* é um sistema que, de um modo geral, visa a compreensão de mudanças ocorridas em diferentes versões de um documento XML. Esta abordagem utiliza o conceito de *diff* semântico em que, a partir do processamento de duas ou mais versões de um documento, é possível compreender as mudanças ocorridas entre as versões, bem como inferir um significado para as mesmas. A inferência é feita através de um motor de inferência *Prolog* que aplica um conjunto de regras, escritas pelo usuário, aos fatos derivados das versões do documento (MARTINS et al., 2013).

Nesta ferramenta, a inferência é feita através de um atributo chave de contexto que o documento XML representa. A chave de contexto deve ser um elemento do docu-

mento que identifica o registro em todas as instâncias do mesmo, independentemente da versão. Portanto, espera-se que o usuário, conhecedor do domínio de negócio em questão, escolha um elemento que possa ser utilizado para este fim.

Em GARCIA (2012) foi proposto um módulo de construção de regras para a inferência em documentos XML. Este trabalho teve como objetivo principal tornar a criação da regra-base um processo semiautomático e fazer com que a construção destas regras seja uma tarefa mais simples, sem a necessidade de conhecer a linguagem *Prolog*.

A utilização do *XChange* para a criação de regras *Prolog* tem início com a construção da base de informações, formada por fatos *Prolog*. Nesta etapa, o usuário deve selecionar duas ou mais versões de um documento XML sobre os quais se deseja inferir as regras semânticas. Para a construção dos fatos, a ferramenta submete os documentos à uma biblioteca, que transforma elementos em predicados e o conteúdo em constantes (MARTINS et al., 2013). A construção dos fatos pode ser feita por chave de contexto ou por similaridade. Neste trabalho, o foco está na abordagem baseada em chave de contexto.

A segunda etapa consiste da análise do contexto de negócio e a construção de regras de inferência. Estas regras estabelecem o que pode ser inferido dos documentos escolhidos. Nesta etapa, o usuário deve carregar o documento contendo as regras, ou criá-las através da interface de construção de regras, apresentada na Figura 4.2.

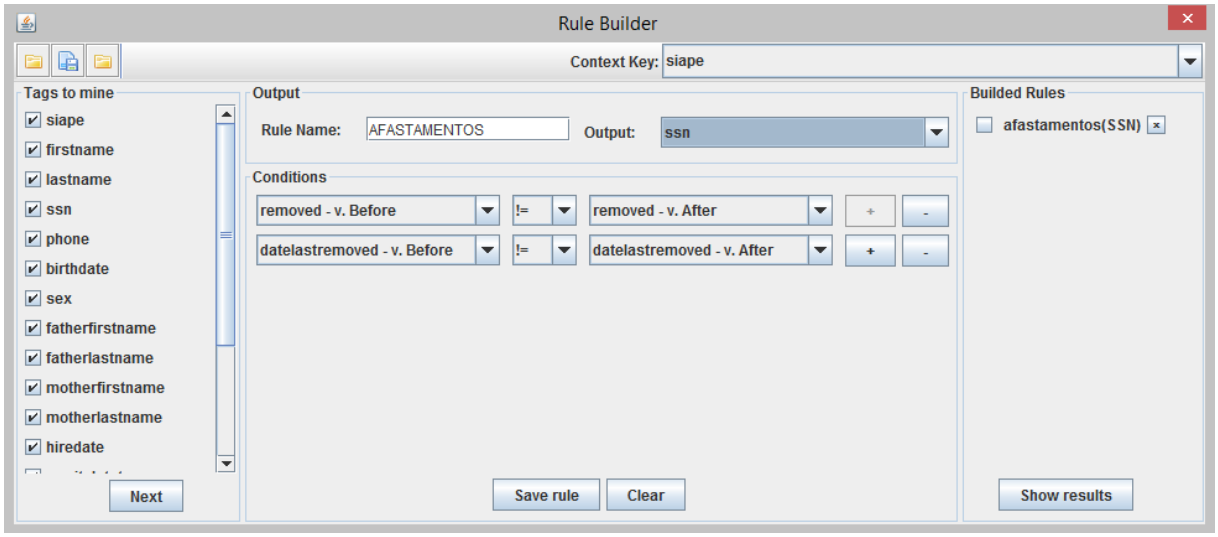


Figura 4.2: Criação de regras no *XChange*.

A última etapa consiste em aplicar as regras criadas anteriormente na base de fatos, fornecida pelo tradutor de XML em *Prolog*. Esta etapa utiliza a biblioteca *tuProlog* (DENTI et al., 2001), para criar um motor de inferência. A inicialização deste motor de inferência é feita atribuindo a este a teoria composta pelos fatos e pelo conjunto de regras. A consulta é realizada através dos nomes das regras selecionadas. Em cada consulta é gerado um conjunto de respostas que atendem a condição presente na regra. Em seguida, as repostas são apresentadas ao usuário no formato de texto, contendo o nome dado à regra e os elementos que as atendem. A Figura 4.3 mostra como os resultados são exibidos.

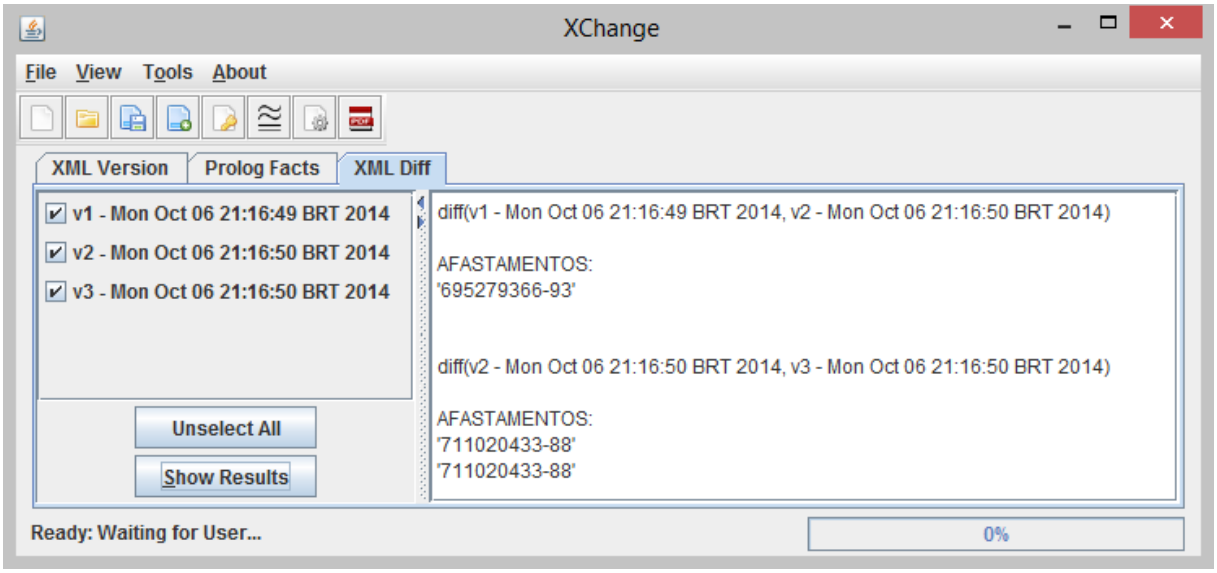


Figura 4.3: Resultados da inferência no *XChange*.

4.2.2 XChangeMining

A definição de regras pelo usuário especialista pode se tornar uma tarefa bastante árdua, principalmente quando se trabalha com bases que possuem muitos elementos. Para apoiar a geração das regras de inferência, o módulo de mineração de regras de associação desenvolvido neste trabalho, XChangeMining, foi adicionado no *XChange*, mais especificamente na tela de criação de regras do *XChange*, e sua execução segue o processo a seguir.

Antes da criação de regras, as dez versões geradas para este trabalho, explicadas na seção anterior, foram carregadas no *XChange*. Para carregá-las, o usuário deve clicar no botão '*Add XML*' e selecionar a versão desejada. Ela é exibida na tela e cada nova versão deve ser adicionada da mesma forma. Apenas duas versões ficam exibidas na tela,

mas podem ser alternadas na caixa de seleção de documentos. Esta caixa e o botão para adicionar versões, bem como as duas primeiras versões carregadas, podem ser visualizadas na Figura 4.4

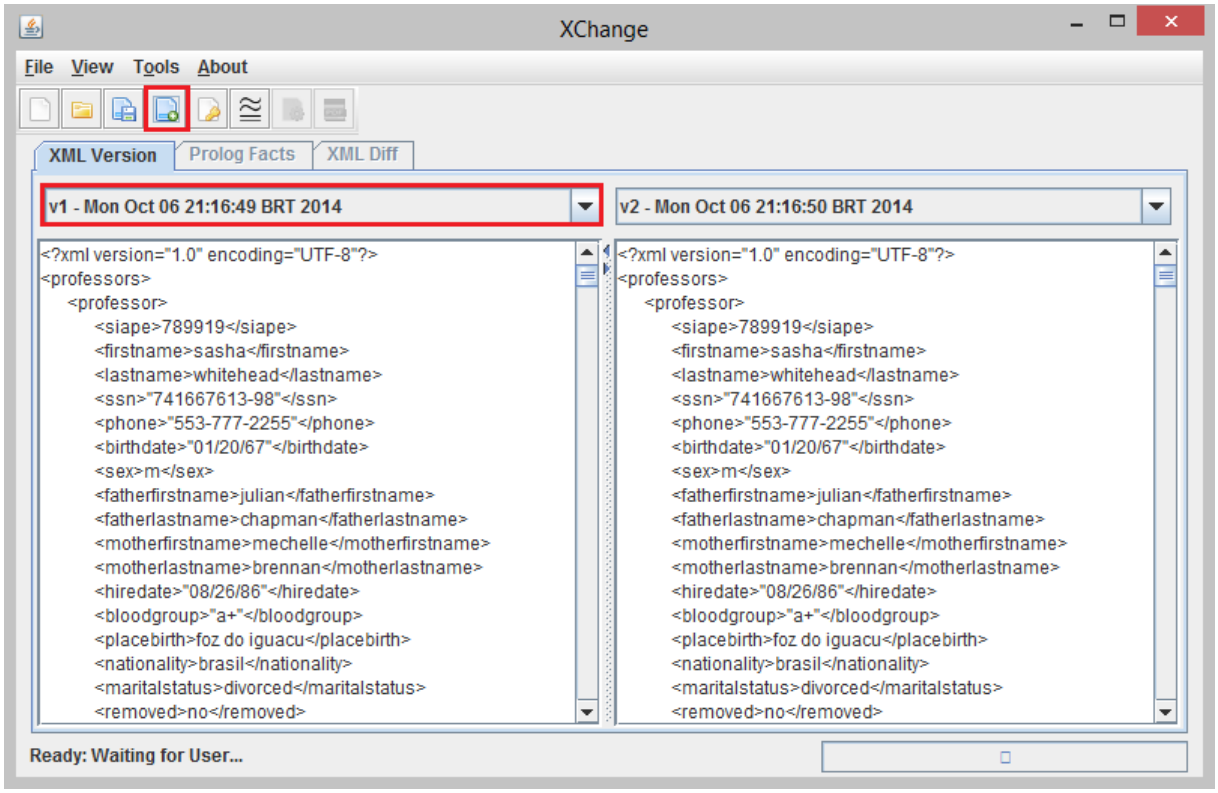


Figura 4.4: Carregamento de versões no *XChange*.

Carregadas as versões, o usuário deve dar início à criação da base de fatos *Prolog*, clicando no botão '*Context Key*'. Após a confirmação do *XChange* de que os fatos foram gerados, o usuário, através do botão '*Manager*', inicia o gerenciador de regras do *XChange*, como mostrado anteriormente na Figura 4.2.

Na parte superior da tela, o usuário define qual o elemento deve ser considerado como chave de contexto. No lado esquerdo da tela, encontra-se a parte destinada à mineração. Inicialmente, são listados todos os elementos que compõem cada registro e o usuário pode definir quais destes elementos devem ser levados em consideração na mineração de dados.

Depois de escolhidos os elementos, devem ser definidas quais versões do documento são consideradas para a realização da mineração, como visto na parte esquerda da Figura 4.2. Escolhidas as versões, o usuário aciona o botão de mineração, e uma nova

janela é apresentada para que sejam fornecidos os valores das métricas a serem utilizadas. Esta janela apresenta um campo para adicionar o valor mínimo de suporte, um campo para escolher qual a segunda métrica a ser utilizada na mineração, entre *confidence*, *lift*, *leverage* e *conviction*, e um campo para o valor mínimo da métrica escolhida. Esta tela pode ser visualizada na Figura 4.5.

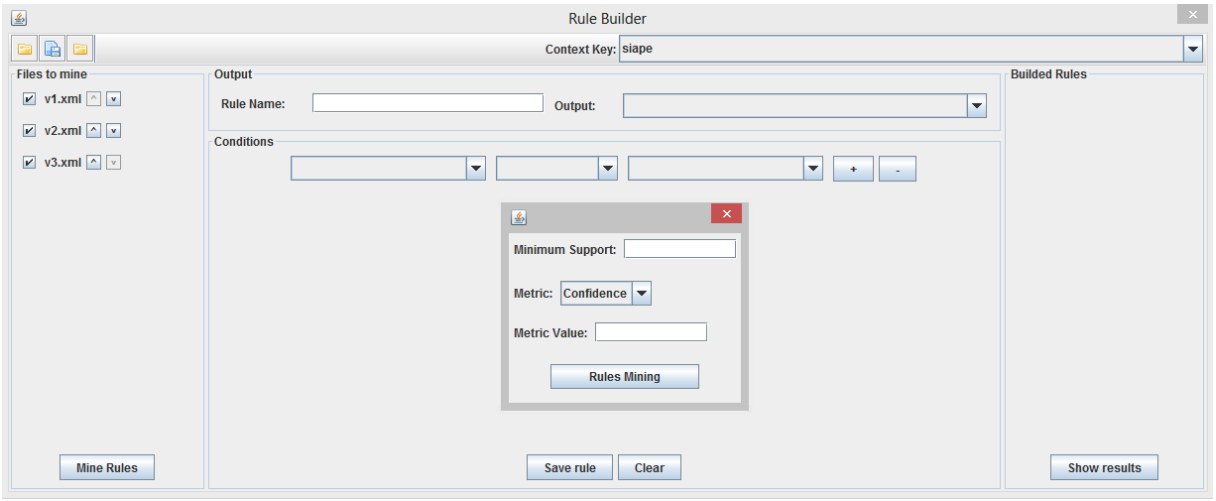


Figura 4.5: Definição de versões e métricas para a mineração.

Ao clicar no botão '*Rules Mining*', o processo de mineração tem início e, quando concluído, retorna uma lista de regras descobertas que são apresentadas ao usuário como mostrado na Figura 4.6. O que é mostrado para o usuário são apenas os elementos que compõem as regras, sem especificar o formato das regras extraídas.

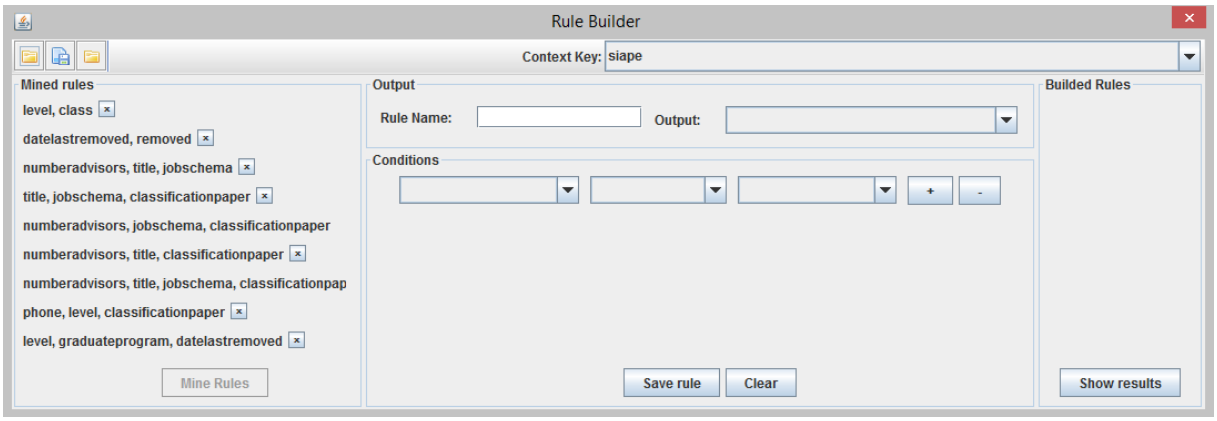


Figura 4.6: Listas de regras mineradas.

A partir da lista de regras encontradas, o usuário pode avaliar quais ele julga úteis para serem utilizadas como base na criação de uma nova regra *Prolog*. Para isto,

ele clica sobre o item que representa a regra escolhida e os campos referentes a esta regra são carregados no painel ao lado para ser personalizada pelo usuário, conforme pode ser visto na Figura 4.7.

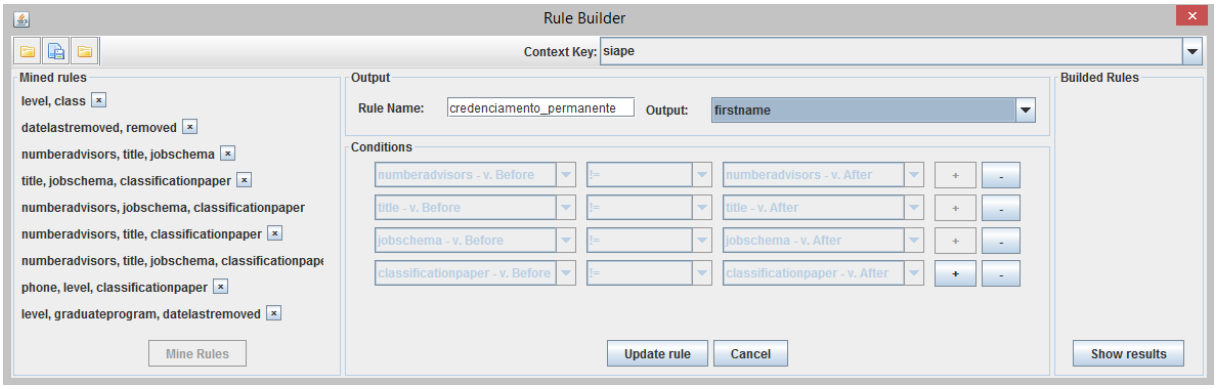


Figura 4.7: Criação de regras *Prolog*.

Cada nova regra criada é apresentada em uma lista na parte direita da tela e no fim do processo as regras podem ser aplicadas na base *Prolog* para que os resultados inferidos sejam apresentados.

4.3 Análise dos resultados

Para a realização do estudo de caso, desde a geração da base de dados até a demonstração do exemplo aplicado, foi utilizado um computador com um processador Intel Core 2 Duo 2.2 GHz e memória RAM de 3 GBs.

A criação da base inicial, feita pela ferramenta *GenerateData*, levou 200 segundos para sua conclusão, considerando a criação de vinte e dois mil registros com vinte e nove elementos cada um. Isso gerou uma versão com aproximadamente 22MB de tamanho. A correção dos valores incoerentes da base durou 158 segundos e a divisão entre a versão inicial e a base auxiliar para inclusão de novos professores geraram documentos com tamanhos de 20MB e 2MB, respectivamente. O processo de criação das dez versões modificadas do documento original levou cerca de 350 segundos no total, com uma média de 30 segundo para cada versão criada.

Com a base em mãos, as dez versões foram carregadas no *XChange*, levando cerca de 300 segundos para carregar todas as versões, uma média de 30 segundos por versão.

Para gerar a base de fatos *Prolog*, por chave de contexto, foram gastos 800 segundos. Já na etapa de criação de regras *Prolog*, depois de definidos os elementos, as versões e as métricas a serem consideradas, o processo de mineração de regras de associação demorou cerca de 30 segundos para apresentar os resultados na tela.

Os primeiros resultados obtidos apresentaram todas as regras com o elemento *salary* como parte delas. Isso ocorreu devido, além da aleatoriedade da base, às associações entre mudança de *title*, *class* e *level*, que sempre implicam em mudança de *salary*. Dessa forma, todas as regras que tinham a presença de um desses elementos incluíam também o elemento *salary*. Para contornar o problema, na tela de criação de regras o elemento *salary* foi desmarcado para que não fosse considerado na mineração.

Observando os resultados apresentados e as regras anteriormente definidas como usuais e pretendidas na mineração, observa-se que a ferramenta cumpriu o trabalho de encontrar as regras mais complexas, que poderiam não estar suficientemente claras ao usuário do *XChange*, seja pela complexidade das regras, seja pela grande quantidade de dados empregada no uso da ferramenta. A Tabela 4.4 faz uma apresentação das regras definidas anteriormente na Tabela 4.3 e dos elementos que compõem as regras que foram mineradas pelo XChangeMining.

Tabela 4.4: Lista de regras

Regras	Elementos das regras extraídas
Transferência	university
Promoção	class, level, title
Credenciamento como permanente	title, classificationpaper, jobschema, numberadvisors
Credenciamento como visitante	title, classificationpaper, jobschema, university
Recredenciamento automático de permanentes	classificationpaper, numberadvisors, numbercourse
Descredenciamento de permanentes fora UFSC	Não encontrada
Descredenciamento de permanentes efetivos	Não encontrada

4.4 Considerações finais

Este capítulo mostrou uma utilização prática para a abordagem desenvolvida neste trabalho, além de apresentar os tempos gastos relativos à execução da tarefa de mineração aplicada a uma base relativamente grande. Neste ponto, observa-se um bom desempenho

do método escolhido, tanto em relação ao tempo, como apresentado, como em relação ao uso de memória, já que a ferramenta de mineração não faz uso de estruturas que mantém informações na memória.

5 Conclusões

O uso de documentos XML aumentou muito nos últimos anos, principalmente no armazenamento e transporte de informações. Um problema encontrado na utilização desse tipo de documento é acompanhar a evolução que acontece com o passar do tempo, onde muitas alterações podem ocorrer em suas diversas versões. Além disso, por possuir uma estrutura flexível, torna-se mais difícil a utilização de ferramentas para acompanhar e gerenciar esse tipo de documento.

Uma das técnicas para realizar a análise desses dados, a mineração de dados, vem sendo utilizada para extrair informações importantes e que nem sempre estão claras para o usuário. Como visto no capítulo 3, as abordagens pioneiras nesse assunto buscavam trazer técnicas de mineração de dados, comumente usadas em bases de dados relacionais, para aplicação direta sobre documentos XML. Muitos desses trabalhos conseguiram realizar a mineração sobre documentos XML que permanecem estáticos durante seu ciclo de vida, mas não contemplaram sua característica evolutiva. Outros trabalhos observaram essa característica, mas acabaram apresentando alguma particularidade que os tornam incompletos para os objetivos pretendidos.

Assim, diante da análise do estado atual da arte e depois de apresentados os conceitos sobre os temas aplicados, este trabalho propôs a criação do XChangeMining. Esta abordagem, junto com o *XChange*, tem como objetivo extrair informações úteis, a partir do histórico de mudanças de versões de documentos XML, e auxiliar na construção de regras *Prolog* para a realização de inferências no *XChange*.

Para demonstrar a utilidade do XChangeMining, foi realizado um exemplo de utilização a fim de apoiar a criação de regras *Prolog* utilizadas pelo *XChange*, que visa analisar de forma semântica as diferenças entre documentos XML. Neste exemplo, também foi desenvolvida uma base sintética para apoiar os testes realizados, além de apresentado os resultados esperados e obtidos e a eficácia da ferramenta desenvolvida.

Contudo, a abordagem apresenta ainda algumas limitações que podem servir como propostas de trabalhos futuros. Os dados passados ao algoritmo de mineração

dizem respeito apenas à alteração ou não dos elementos entre as versões analisadas e não quais são esses valores. Repassar ao algoritmo essas dados de forma completa, com todos os valores que compõem o documento XML, pode trazer resultados mais interessantes, por exemplo, para que sejam extraídas informações sobre aumento ou diminuição de valores.

Outra forma de aproveitar melhor as informações contidas no documento XML é analisando não somente o documento, mas também o XML *Schema*, linguagem oficial de definição de esquemas para documentos XML. Ele prevê quais elementos são encontrados nos documentos, a ordem em que estes elementos podem aparecer, a hierarquização destes elementos, o tipo de dados do conteúdo destes elementos, o elemento que identifica unicamente cada registro, entre outros (XML SCHEMA, 2014).

Uma melhoria interessante seria a criação de um algoritmo de diferenciação mais otimizado e que não dependesse da indicação de qual atributo representa unicamente cada registro do documento XML. Esta informação pode ser aproveitada do XML *Schema*, já que uma das suas características é trazer a informação de qual elemento identifica unicamente cada registro no documento XML.

O algoritmo de diferenciação construído não mantém um histórico das alterações ocorridas, sendo executado a cada nova tentativa de mineração. Para diminuir o tempo da mineração, manter o histórico de mudanças pode ser útil, diminuindo o tempo total de execução.

Por fim, a realização de testes com bases reais pode mostrar ainda mais a utilidade da abordagem desenvolvida. E ainda, uma comparação com outros algoritmos de mineração de regras de associação, como os exemplificados neste trabalho (HAN et al., 2004; MUSTAPHA et al., 2003).

Referências Bibliográficas

ABAZEED, A.; MAMAT, A.; NASIR, M. ; IBRAHIM, H. **Mining association rules from structured XML data**. In: International Conference on Electrical Engineering and Informatics, 2009. ICEEI'09., volume 2, p. 376–379. IEEE, 2009.

AGRAWAL, R.; IMIELINSKI, T. ; SWAMI, A. Database mining: A performance perspective. **Knowledge and Data Engineering, IEEE Transactions on**, v.5, n.6, p. 914–925, 1993.

AGRAWAL, R.; IMIELIŃSKI, T. ; SWAMI, A. **Mining association rules between sets of items in large databases**. In: ACM SIGMOD Record, volume 22, p. 207–216. ACM, 1993.

AGRAWAL, R.; SRIKANT, R. ; OTHERS. **Fast algorithms for mining association rules**. In: Very Large Data Bases, VLDB, volume 1215, p. 487–499, 1994.

BRAGA, D.; CAMPI, A.; KLEMETTINEN, M. ; LANZI, P. **Mining association rules from XML data**. In: Data Warehousing and Knowledge Discovery, p. 21–30. Springer, 2002.

BRAGRA, D.; CAMPI, A.; CERI, S.; KLEMETTINEN, M. ; LANZI, P. **Discovering interesting information in XML data with association rules**. In: ACM Symposium on Applied Computing, p. 450–454. ACM, 2003.

BRAGANHOLO, V. P.; DAVIDSON, S. B. ; HEUSER, C. A. **From XML view updates to relational view updates: old solutions to a new problem**. In: Conference on Very Large Data Bases, VLDB, volume 30, p. 276–287. VLDB Endowment, 2004.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E. ; YERGEAU, F. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. World Wide Web Consortium, Recommendation REC-xml-20081126, 2008.

CHEN, L.; BHOWMICK, S. S. ; CHIA, L.-T. **Mining association rules from structural deltas of historical XML documents**. In: Advances in Knowledge Discovery and Data Mining, p. 452–457. Springer, 2004.

COBENA, G.; ABITEBOUL, S. ; MARIAN, A. **Detecting changes in XML documents**. In: International Conference on Data Engineering, 2002., 2002.

DENTI, E.; OMICINI, A. ; RICCI, A. **TuProlog: A light-weight prolog for internet applications and infrastructures**. In: Practical Aspects of Declarative Languages. 2001.

DING, Q.; SUNDARRAJ, G. **Association rule mining from XML data**. In: The 2006 International Conference on Data Mining, p. 144–152, 2006.

Document Object Model API. http://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Document.html, acessado em 01/11/2014.

FAYYAD, U.; PIATETSKY-SHAPIRO, G. ; SMYTH, P. From data mining to knowledge discovery in databases. **AI magazine**, v.17, n.3, p. 37, 1996.

FENG, L.; DILLON, T.; WEIGAND, H. ; CHANG, E. **An XML-enabled association rule framework**. In: Database and Expert Systems Applications, p. 88–97. Springer, 2003.

GARCIA, P. A. **EDX: Uma abordagem de apoio ao controle de mudanças de documentos XML**, Trabalho de Conclusão de Curso - Universidade Federal de Juiz de Fora, 2012.

GenerateData. http://www.generatedata.com/, acessado em 01/11/2014.

HAN, J.; FU, Y. **Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases.** In: Knowledge-Discovery in Databases Workshop, p. 157–168, 1994.

HAN, J.; PEI, J.; YIN, Y. ; MAO, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. **Data mining and knowledge discovery**, v.8, n.1, p. 53–87, 2004.

LAN, L.; QIAO-MEI, R. **Research of web mining technology based on XML**. In: International Conference on Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09., volume 2, p. 653–656. IEEE, 2009.

LINDHOLM, T. **A 3-way merging algorithm for synchronizing ordered trees - the 3DM merging and differencing tool for XML**. 2001. Dissertação de Mestrado - Helsinki University of Technology.

MARTINS, G.; JUNIOR, C. L.; OLIVEIRA, A.; MURTA, L. ; BRAGANHOLO, V. XChange: Compreensão de mudanças em documentos XML demos e aplicações. **Simpósio Brasileiro de Banco de Dados (SBBD)**, 2013.

MORADI, M.; KEYVANPOUR, M. R. An analytical review of XML association rules mining. **Artificial Intelligence Review**, p. 1–24, 2013.

MUSTAPHA, N.; SULAIMAN, M. N.; OTHMAN, M. ; SELAMAT, M. H. Fast discovery of long patterns for association rules. **International Journal of Computer Mathematics**, v.80, n.8, p. 967–976, 2003.

MySQL. http://www.mysql.com/, acessado em 01/11/2014.

OSO Corporation - Project Merge, acessado em 01/11/2014.

PHP. http://www.php.net/, acessado em 01/11/2014.

RUSU, L. I.; RAHAYU, W. ; TANIAR, D. **Mining changes from versions of dynamic XML documents**. In: Knowledge Discovery from XML Documents, p. 3–12. Springer, 2006.

Simple API for XML. http://docs.oracle.com/javase/7/docs/api/org/xml/sax/package-summary.html, acessado em 01/11/2014.

World Wide Web Consortium. http://www.w3c.br/, acessado em 01/11/2014.

WANG, Y.; DEWITT, D. J. ; CAI, J.-Y. **X-Diff: An effective change detection algorithm for XML documents**. In: Data Engineering, 2003., p. 519–530. IEEE, 2003.

Weka 3 Machine Learning Software in Java. http://www.cs.waikato.ac.nz/ml/weka, acessado em 01/11/2014.

The Apache XML Project. http://xml.apache.org/xalan-j/, acessado em 01/11/2014.

W3Schools. http://www.w3schools.com/schema/, acessado em 01/11/2014.

XPath 3.0. http://www.w3.org/TR/2014/REC-xpath-30-20140408/, acessado em 01/11/2014.

XQuery 3.0. http://www.w3.org/TR/2014/REC-xquery-30-20140408/, acessado em 01/11/2014.

XSLT 3.0. http://www.w3.org/TR/xslt-30/, acessado em 01/11/2014.

ZHAO, Q.; BHOWMICK, S. S. ; MADRIA, S. **Discovering pattern-based dynamic structures from versions of unordered XML documents**. In: Data Warehousing and Knowledge Discovery, p. 77–86. Springer, 2004.

ZHAO, Q.; BHOWMICK, S. S.; MOHANIA, M. ; KAMBAYASHI, Y. **Discovering frequently changing structures from historical structural deltas of unordered XML**. In: Proceedings ACM International Conference on Information and Knowledge Management, p. 188–197. ACM, 2004.

ZHANG, J.; LIU, H.; LING, T. W.; BRUCKNER, R. M. ; TJOA, A. M. A framework for efficient association rule mining in XML data. **Journal of Database Management (JDM)**, v.17, n.3, p. 19–40, 2006.