Universidade Federal de Uberlândia

Faculdade de Computação

Programa de Pós-Graduação em Ciência da Computação

VisTree: Uma Linguagem Visual para Análise de Padrões Arborescentes e para Especificação de Restrições em um Ambiente de Mineração de Árvores

Crícia Zilda Felício

Uberlândia - MG

MARÇO 2008

CRÍCIA ZILDA FELÍCIO

VISTREE: UMA LINGUAGEM VISUAL PARA ANÁLISE DE PADRÕES ARBORESCENTES E PARA ESPECIFICAÇÃO DE RESTRIÇÕES EM UM AMBIENTE DE MINERAÇÃO DE ÁRVORES

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientadora: Profª. Dra. Sandra de Amo

Uberlândia
2008

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**VisTree: Uma Linguagem Visual para Análise de Padrões Arborescentes e para Especificação de Restrições em um Ambiente de Mineração de Árvores**" por **Crícia Zilda Felício** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 25 de Março de 2008

Orientadora: _____

Profª. Drª. Sandra de Amo

Universidade Federal de Uberlândia UFU/MG

Banca Examinadora: _____

Profª. Drª. Rita Maria da Silva Julia

Universidade Federal de Uberlândia UFU/MG

_____

Prof. Dr. Mauro Biajiz

Universidade Federal de São Carlos UFSCar/SP

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Data: Março, 2008

Autor: **Crícia Zilda Felício**

Título: **VisTree: Uma Linguagem Visual para Análise de Padrões Arborescentes e para Especificação de Restrições em um Ambiente de Mineração de Árvores**

Faculdade: **Faculdade de Computação**

Grau: **Mestrado**

_____

Autor

Dedicatória

Aos meus pais Túlio e Silvia

Aos meus sobrinhos Guilherme, Andreza, Jean, Eduardo, Cilton Jr, Juliana, Lucas, Crisley, Thais, Pedro e Vinícius

Agradecimentos

A Deus por todo apoio espiritual que tive durante o decorrer desse trabalho.

A minha orientadora Sandra Aparecida de Amo pela oportunidade oferecida, pela confiança e orientação a mim dedicados.

A Luiza Rangel, e aos alunos de iniciação científica Gabriel Coutinho e Tarcísio Gotto que contribuiram de forma efetiva para o sucesso desse trabalho. Ao Tarcísio Gotto ainda um agradecimento especial, por ter ido além do seu papel, mostrando-se um amigo dedicado com quem pude contar com importante colaboração durante a finalização desse trabalho.

Aos meus pais Túlio e Silvia que me deram suporte e incentivo para chegar até aqui, e que foram a razão maior para as minhas conquistas.

Aos meus irmãos Cinthia, Cilton, Silvio e Túlio Jr pelo exemplo, carinho e companherismo de sempre.

A Henrique que me acompanhou durante parte desse caminho e que talvez tenha sido a pessoa que mais me motivou a percorrê-lo.

Aos meus amigos do Mestrado em Computação: Ricardo, Elaine, Daniel, Junior, Mariângela, Jean, Marcos, Juliano, Tauller e Felipe com os quais eu convivi quase que diariamente durante esse periodo e que sempre me apoiaram, ora me fazendo rir, ora conversando, ora ajudando.

A Klérisson Paixão, pelo simples fato de fazer parte da minha vida nesse momento.

A minha amiga Ana Elisa, pela amizade de sempre, suporte importante em qualquer momento da vida.

Aos meus amigos, ajudantes de última hora, Italo Thiago e Pablo Hernandes, pelo auxílio nas correções da dissertação e pelo apoio nos momentos mais difícies.

"Sede como os pássaros que, ao pousarem um instante sobre ramos muito leves, sentem-nos ceder, mas cantam! Eles sabem que possuem asas."

(Victor Hugo)

Resumo

A mineração de padrões freqüentes em dados representados por estruturas mais complexas como árvores e grafos vêm crescendo muito nos últimos tempos. Entre as razões para esse crescimento está o fato do padrão arborescente ou em forma de grafo possuir mais informações do que os padrões seqüenciais, e na possibilidade de aplicação desse tipo de mineração em várias áreas como *XML Mining*, *Web Mining* e Bioinformática. Um problema que ocorre na mineração de padrões em geral é a grande quantidade de padrões gerados; sendo que muitos deles nem são do interesse do usuário. A diminuição da quantidade de padrões gerados pode ser feita restringido o tipo de padrão produzido através de especificações do usuário. Mesmo incorporando restrições no processo de mineração, a quantidade de padrões arborescentes minerados é grande, o que torna necessário uma ferramenta de análise dos padrões, possibilitando ao usuário especificar consultas para extrair da massa de padrões minerados aqueles que satisfazem os critérios de seleção da consulta.

A mineração de padrões com restrição, visa obter como resultado de um processo de mineração apenas os padrões de real interesse do usuário. Uma restrição sobre padrões será representada de acordo com a estrutura dos mesmos. Para a mineração de padrões seqüencias uma forma de representá-la seria através de expressões regulares, para a mineração de padrões arborescentes, os autômatos de árvore. O uso de restrições resolve o problema da geração de uma grande quantidade de padrões, mas o mecanismo usado para representar a restrição ainda se constitui em um outro problema que seria a dificuldade de um usuário em fazer a entrada da restrição utilizando esse mecanismo.

As consultas sobre padrões freqüentes são feitas de acordo com as características dos dados. Uma forma de extrair padrões específicos em dados estruturados como árvores é armazenar os padrões freqüentes em um documento XML e efetuar uma consulta usando uma das linguagens de consulta a documentos XML. Dentre as linguagens de consulta XML, a linguagem XQuery é muito utilizada, principalmente pelo fato de ser similar semanticamente a SQL (linguaguem de consulta a banco de dados). A consulta aos padrões freqüentes poderia então ser feita utilizando

essa linguagem, mas para isso o usuário teria que conhecer e ser capaz de expressar sua consulta através dela.

Nesse trabalho é apresentada a linguagem visual VisTree, que consiste em uma ferramenta visual a ser utilizada tanto numa fase de Pré-processamento para a especificação das preferências do usuário no que se refere ao formato dos padrões arborescentes que lhe interessa, quanto numa fase de pós-processamento para a análise dos padrões minerados. A sintaxe da VisTree se baseia na sintaxe de um fragmento simples da linguagem Tree Pattern [Miklau and Suciu 2004, Chen et al. 2003], na qual a linguagem XPath 1.0 [Clark and Derose 1999, Olteanu et al. 2002] também se baseou. Entretanto, a semântica de VisTree difere da semântica destas linguagens no sentido de que consultas de VisTree retornam conjuntos de padrões arborescentes. A VisTree utiliza a linguagem XQuery [Chamberlin 2003, Katz et al. 2003] como mecanismo de processamento de consultas: as consultas visuais especificadas em VisTree são mapeadas em consultas da XQuery e suas respostas adaptadas para se adequarem ao formato retornado por VisTree.

Um sistema completo de mineração de padrões arborescentes foi desenvolvido para testar e validar o uso da linguagem VisTree em contextos específicos de aplicações. O sistema foi construído de forma modular para que novas aplicações possam ser incorporadas de maneira simples. A aplicação de mineração de árvores com restrição nas áreas de *XML Mining* e *Web Mining* foi feita através de um estudo de caso. Nas duas aplicações, o sistema utiliza a linguagem VisTree nos módulos que fazem a tarefa de Pré-Processamento (entrada da restrição) e de Análise de Padrões (entrada da consulta).

Palavras chave: *Datamining*, Mineração de Árvores, Mineração de Árvores com Restrição, *Web Mining*, *XML Mining*

Abstract

The frequent pattern mining in data represented by more complex structures like trees and graphs are growing lately. Among the reasons for this improvement is the fact that the tree and graph patterns has more information than sequential patterns, besides there is the possibility of usage of this type of mining in several areas like XML Mining, Web Mining and Bioinformatic. A problem that occurs in mining patterns in general is the great amount of patterns generated. Being some of them not interesting for users. The decrease in the quantity of patterns generated can be done restricting the patterns types produced through the user constraint. Even incorporating constraints in the mining process, the quantity of tree pattern mined is large, what make necessary one tool for pattern analysis, possibiliting the user specify queries to extract in the mass of mined patterns that satisfy the criteria of the selection in the query.

The pattern mining with constraint, aim to obtain as a result of the process of mining only the patterns with the real interest for the user. The constraint about patterns will be represented related to the structure of them. One form to represent the sequential pattern mining would be through regular expressions, for the tree pattern mining, the tree automata. The use of constraints solve the problem to generate a large amout of patterns, but the mechanism used to represent the constraint is still constituted in another problem that would be the difficult for a user do the input of constraint using this mechanism.

The queries about frequent patterns are made according to the characteristics of the data. One way to extract specific patterns in data structured like trees is to store the specific patterns in a XML file and make queries using one of the query languages for XML files. Among the XML query languages, the XQuery language is very used, mainly by the fact that it's similar in semantic to SQL, the query language for databases. The frequently patterns queries could be made using this language, but, for this the user would have to know and be capable to express queries through it.

In this research it will be presented the visual language VisTree that consists of visual tool to be used in a phase of preprocess for specification the user preferences that involves

the format of the tree pattern that are interested to him, as in a phase of postprocess to analyze the mined patterns. The VisTree sintaxe is based on in a fragment of the Tree Pattern language[Chen et al. 2003, Che and Liu 2005], the core of XPath 1.0 [Clark and Derose 1999, Olteanu et al. 2002]. However, the semantic of VisTree differs from the semantic of these languages in the sense that VisTree queries return the sets of tree patterns. VisTree uses a XQuery language [Chamberlin 2003, Katz et al. 2003] like query process mechanism: the visual queries specified in VisTree are mapped in XQuery queries and theirs responses are adapted to fit the format returned by VisTree. VisTree works like a XQuery front-end.

A complete system of mining tree pattern was developed to test and validate the use of VisTree language in specific contexts of applications. The system was made in a modular form, in a way to allow that new applications could be incorporated in a simple way. This research show the application of tree mining with constraint in the areas of XML Mining and Web Mining through study case. In both applications, the system use the VisTree language in the preprocess modules (constraint input) and analysis of patterns (query input).

Keywords: Datamining, Tree mining, Constraint-based tree mining, Web Mining, XML Mining

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto Geral

O surgimento da área de mineração de dados foi motivado pelo crescimento constante de informações armazenadas em meios eletrônicos. Ter uma grande quantidade de informações produz um problema, que é a dificuldade de encontrar nesses dados o que é realmente interessante. Por essa razão, a mineração de dados busca extrair informações "preciosas" de dentro de um grande conjunto de dados. A mineração de padrões freqüentes é uma forma de mineração de dados que tem sido bastante estudada, motivada pelo crescente interesse e aplicabilidade em diferentes áreas, onde a estrutura dos dados desempenha um papel relevante nas técnicas de mineração utilizadas.

As técnicas clássicas de mineração de padrões freqüentes envolvem a mineração de regras de associação [Agrawal et al. 1993, Agrawal and Srikant 1994], apresentada nos trabalhos de Srikant e Agrawal, e de seqüências [Agrawal and Srikant 1995, Srikant and Agrawal 1996] contemplada por trabalhos desses mesmo autores. Atualmente a mineração de padrões tem sido aplicada a dados que possuem uma estrutura mais complexa como árvores [Zaki 2002, Termier et al. 2002, Asai et al. 2003] e grafos [Yan and Han 2002, Washio and Motoda 2003], possibilitando a obtenção de padrões com mais informações. A mineração de árvores possui

aplicação em diversas áreas e dentre elas destaca-se a mineração de documentos XML (*XML Mining*), mineração de *logs* de acesso na Mineração Web (*Web Mining*) e mineração de estruturas de RNA na Bioinformática. A possibilidade de aplicar a mineração de padrões arborescentes em diversas áreas justifica o número crescente de pesquisas e desenvolvimento de técnicas nesse contexto.

Os documentos XML são usados para armazenar diferentes tipos de informações como páginas Web, mensagens Web, tabelas de banco de dados relacionais, *logs* de sistema, transações financeiras, entre outros. O XML é reconhecido pelo W3C[1] como sendo a linguagem padrão para troca e tráfego de dados na Web. Na área de *XML Mining*, algumas das possibilidades de aplicações são: Mineração de Estruturas dos Documentos [Papakonstantinou and Vianu 2000], Mineração de Padrões freqüentes [Asai et al. 2002] e Mineração de Padrões de Consultas freqüentes [Yang et al. 2003]. O contexto do trabalho aqui apresentado se encontra na mineração de árvores em documentos XML para a descoberta de padrões freqüentes. A aplicação desse tipo de mineração pode ser feita da seguinte forma: dada uma coleção de documentos XML, que podem ser representados naturalmente como árvores, seria do interesse de alguns usuários saber quais sub-documentos (sub-árvores) frequentemente aparecem entre os documentos XML.

A mineração de dados na Web ou *Web Mining*, segundo [Kosala and Blockeel 2000], engloba três áreas distintas: Mineração do Conteúdo da Web, Mineração da Estrutura da Web e Mineração do Uso da Web. A Mineração do Conteúdo da Web consiste na extração de informações interessantes sobre o conteúdo, dados e documentos da Web. Na Mineração da Estrutura da Web a aquisição de informações é feita através da topologia, organização e estrutura do *website*. Já na Mineração do Uso da Web a descoberta de informações interessantes é feita através dos *logs* de acesso web, que representa o comportamento do usuário durante a navegação por um *website*. Dentre essa subáreas foi escolhida a Mineração do Uso da Web para ser tratada nesse trabalho, pois o conjunto de acessos feitos por um usuário em um *website* pode ser representado por uma árvore de navegação. A Mineração do Uso da Web é utilizada com a finalidade de obter padrões freqüentes de navegação dos usuários. Através da análise desses padrões é possível determinar como o site é usado, e utilizar essas informações para efetuar mudanças na estrutura do *website*, ou para inserir *links* relevantes em determinadas páginas.

[1] World Wide Web Consortium, responsável pela padronização na Web

Na área de Bioinformática, as pesquisas genéticas feitas nos últimos anos contribuíram para a geração de um enorme banco de dados com informações sobre DNA, RNA, aminoácidos, proteínas, etc. Alguns desse dados, como o RNA pode ser representado por estruturas arborescentes, o que faz com que a mineração de árvore nessa área também seja bastante utilizada. Um exemplo de pesquisa que vem sendo realizada com essa aplicação é a descoberta de estruturas arborescentes de RNA, que são comparadas com moléculas de RNA conhecidas para encontrar informações que levem ao um melhor entendimento do funcionamento dessas estruturas [Shapiro and Zhang 1990, Chevalet and Michot 1992].

Um problema comum na mineração de árvores é a grande quantidade de padrões retornados como resultado do processo de mineração, pois a mineração é feita na maioria dos algoritmos considerando apenas a freqüência em que o padrão ocorre. Essa medida não leva em conta as características dos padrões, o que faz com que além da grande quantidade de padrões minerados tem-se que muitos deles não são do interesse de usuário. A solução para esse problema seria fornecer meios para que o usuário informe o tipo de padrão que ele está interessado em minerar.

Recentemente, foi desenvolvido na Universidade Federal de Uberlândia, Faculdade de Computação, um projeto de mestrado com o propósito de criar um algoritmo de mineração de padrões arborescentes que incorporasse restrição ao processo de mineração. O algoritmo CobMiner (*Constraint based Miner*) [Silva 2007], foi o primeiro algoritmo de mineração de padrões arborescentes com restrição a ser criado. Esse algoritmo utiliza como mecanismo de restrição dos padrões durante a fase de geração, os autômatos de árvore [Neven 2002, Murata et al. 2005].

A incorporação de restrições no processo de mineração obtêm faz com que a quantidade de padrões minerados seja reduzida, mas mesmo com esse ganho pode se ter ainda uma grande quantidade de padrões, o que dificulta a análise do usuário. Para resolver essa questão, faz-se necessária a utilização de ferramentas de análise de padrões. Essas ferramentas possibilitam ao usuário especificar consultas para extrair da massa de padrões minerados aqueles que satisfazem os critérios de seleção da consulta.

Neste trabalho, será apresentada a linguagem visual VisTree que consiste em uma ferramenta visual a ser utilizada tanto numa fase de pré-processamento para a especificação das preferências do usuário no que se refere ao formato dos padrões arborescentes que lhe interessa, quanto numa fase de pós-processamento para a análise dos padrões minerados. Um sistema de

mineração de padrões arborescentes foi desenvolvido para testar e validar o uso da linguagem VisTree em contextos específicos de aplicações. O sistema *CobMiner* faz uso do algoritmo *CobMiner* de mineração árvores com restrição. A aplicação de mineração de árvores nas áreas de *XML Mining* e *Web Mining*, com o uso de restrição, foi analisada através de um estudo de caso. O trabalho teve até o presente momento a publicação do artigo [de Amo and Felício 2007].

1.2 Motivação

Considerando-se o problema da geração de grande quantidade de padrões, o alto custo computacional decorrente desse fato e a insatisfação do usuário com os resultados obtidos têm-se uma forte motivação para a utilização da mineração de dados com restrição. No contexto de mineração de árvores com restrição, foi citado na seçao anterior a existência do algoritmo CobMiner [de Amo et al. 2007], que utiliza como mecanismo de restrição autômatos de árvore. Na utilização desse algoritmo, o problema seria como fazer de uma forma simples e intuitiva a entrada da restrição, já que este espera como entrada um autômato de árvore. Pedir ao usuário que forneça o autômato de árvore referente a sua restrição restringiria o uso de um sistema com esse algoritmo para poucas pessoas que possam saber o que é um autômato de árvore.

Na fase de pós-processamento o usuário poderia estar interessado em selecionar padrões específicos entre os padrões que foram retornados do processo de mineração. Para isso seria necessário que ele fizesse consultas aos padrões de árvore freqüentes procurando por aqueles que quer analisar. Nesse caso o problema se encontra na forma de especificar uma consulta a esses padrões. Uma solução seria armazená-los em um documento XML de forma que fiquem mais organizados. Dessa forma a consulta aos padrões poderia ser feita através de qualquer linguagem de consulta XML, como por exemplo XQuery. Mas assim como o problema da restrição o uso desse recurso ficaria restrito apenas a usuários que conheçam essa linguagem.

A motivação maior para esse trabalho é então oferecer uma forma de representar visualmente um molde para padrões de árvore, onde possam ser representadas tanto a estrutura dos padrões desejados, como condições sobre os nós que eles deveriam apresentar. A linguagem visual VisTree foi então criada com essa finalidade.

A linguagem possui características que permitem sua utilização tanto para a especificação

de restrições quanto para a consulta de padrões de árvore, atuando na fase de pré-processamento e pós-processamento da mineração de padrões arborescentes.

O desenvolvimento de um sistema com uma interface e que contempla todas as fases do processo de mineração, incluindo a utilização do algoritmo de mineração de árvores com restrição CobMiner [de Amo et al. 2007] e duas aplicações em dados reais, foram realizados para que a linguagem VisTree fosse testada e validada.

1.3 Objetivo do Trabalho

O trabalho desenvolvido tem como foco principal a criação de uma linguagem que expressa, de uma forma visual, os tipos de padrões de interesse em uma base de dados de árvore. Além do desenvolvimento de um sistema onde a linguagem VisTree pudesse ser usada em aplicações específicas.

Dentre os objetivos gerais desse trabalho temos:

- Criação e desenvolvimento da linguagem visual VisTree para a especificação de classes de padrões arborescentes, que pode ser usada tanto na fase de pré-processamento na especificação de restrições quanto na fase de análise de padrões (pós-processamento) para especificação de consultas aos padrões minerados.

- Desenvolvimento de um algoritmo para efetuar a conversão de uma expressão da linguagem VisTree, que representa à restrição, em um autômato de árvore.

- Desenvolvimento de um algoritmo para efetuar a conversão de uma expressão da linguagem VisTree, contendo a consulta de entrada do usuário, em uma consulta XQuery.

- Realização de dois estudos de casos no ambiente de *Web Mining* e *XML Mining*, onde pode-se ver na aplicação em dados reais a utilização da linguagem VisTree para os fins propostos.

- Desenvolvimento de algoritmos de pré-processamento dos dados reais: Documentos XML e *logs* de Navegação Web.

- Implementação de um ambiente para a mineração de árvores com restrição composto por todos algoritmos desenvolvidos e por uma interface de entrada e saída de dados. O sistema faz uso do algoritmo CobMiner de mineração de árvores com restrição, desenvolvido em [de Amo et al. 2007, Silva 2007].

1.4 Estrutura da Dissertação

A dissertação aqui apresentada se encontra organizada da seguinte maneira: o capítulo 2 introduz alguns conceitos preliminares que serão usados no restante do trabalho. O capítulo 3 descreve o estado da arte abordando os principais trabalhos relacionados a pré-processamento, mineração de estruturas, mineração de estruturas com restrição, aplicações de mineração de árvores, análise de padrões e interface.

No capítulo 4 é descrita a linguagem VisTree de especificação de classes de padrões arborescentes, sua sintaxe e semântica, comparação com as linguagens de consulta XPath e XQuery, Tree Pattern e autômato de árvore. A utilização da linguagem para a entrada de restrições e consulta também é tratada nesse capítulo.

Na sequência o Sistema CobMiner é detalhado no capítulo 5, onde são apresentados os módulos que compõem o sistema e suas características. O capítulo 6 relata os dois estudos de casos propostos, um para *XML Mining* e outro para *Web Mining*. E por fim, no capítulo 7, tem-se a conclusão do trabalho desenvolvido e os trabalhos futuros previstos.

Capítulo 2

Preliminares

Nesse capítulo serão definidos alguns conceitos relacionados aos assuntos que serão tratados nos próximos capítulos. A seção 2.1 descreve as definições das estruturas de dados: grafos e árvores. Ainda relacionado à árvores, a seção contempla também os conceitos de árvore etiquetada ou de *label*, ancestral e descendente, representação de árvores com *string*, subárvores, padrões de árvore e suporte de um padrão. A segunda seção 2.2 define dois mecanismos de restrição, autômatos finito deterministico e autômato de árvore, utilizados na mineração de padrões seqüencias e arborescentes respectivamente. A última seção dará uma visão geral sobre três linguagens de consulta XML, XPath, Tree Pattern e XQuery.

2.1 Grafos e Árvores

Definição 2.1.1 (Grafo dirigido) Um *grafo dirigido* é uma estrutura $G = (V, E)$, onde V é um conjunto de vértices e E um subconjunto de $V \times V$, isto é, um subconjunto de pares ordenados de vértices. Os elementos de E são chamados *arestas* do grafo.

Definição 2.1.2 (Grafo cíclico) Um grafo é dito *cíclico* se existe uma sequência de arestas (caminho) (v_0, v_1), (v_1, v_2),...,(v_{n-1}, v_0) ligando os vértices $v_0, v_1, ..., v_{n-1}$, começando e terminando em v_0. Um tal caminho fechado é chamado de *ciclo*.

Definição 2.1.3 (Grafo acíclico) Um grafo é *acíclico* se não possui ciclos.

Definição 2.1.4 (Grafo conexo) *Um grafo é conexo se quaisquer dois vértices v e u podem ser ligados por um caminho $(v, v_1), (v_1, v_2), ..., (v_{n-1}, u)$.*

Exemplo 2.1.1 *Na Figura 2.1, são apresentados exemplos de grafos acíclicos, cíclicos e de grafos desconexos. Os grafos (a) e (b) na Figura 2.1 são conexos.*

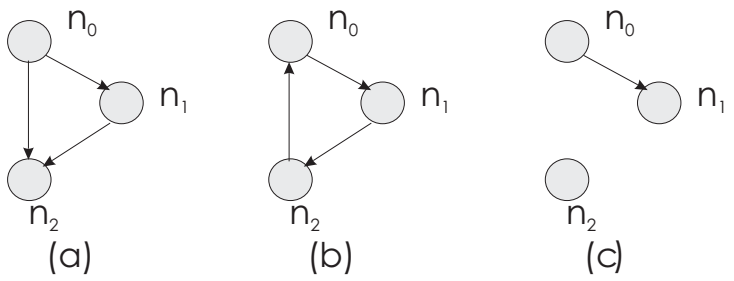

Figura 2.1: (a) Grafo acíclico, (b) Grafo cíclico, (c) Grafo desconexo

Definição 2.1.5 (Árvore) Uma *árvore* é um grafo $G = (V, E)$ acíclico conexo tal que :

1. existe um elemento especial único r com a propriedade de que não existe $v \in V$ tal que $(v, r) \in E$. Isto é, não existe *seta* chegando em r. Este elemento único é chamado de *raiz*.

2. dado v_1 em $V - \{r\}$, então existe um único vértice $v \in V$ tal que $(v, v_1) \in E$. v é dito *pai de* v_1 e v_1 é dito *filho de* v. Se v_1 e v_2 têm o mesmo pai, são ditos *irmãos*.

Numa árvore, os vértices v que possuem filhos são chamados de *nós internos*. Os que não possuem filhos são chamados de *nós-folha* ou simplesmente *folhas*. O tamanho de uma árvore T, denotado por $\mid T \mid$, é definido como sendo o número de nós da árvore.

Tendo em vista as aplicações em *Web Mining* e *XML Mining* que são usualmente baseadas em algoritmos de mineração de árvores, esse estudo será restringindo a árvores *ordenadas* e *etiquetadas*.

Definição 2.1.6 (Árvore Ordenada) Uma árvore é dita *ordenada* se para cada nó v, existe uma ordem no conjunto de seus filhos (denotado por filhos(v)).

Definição 2.1.7 (Árvore Etiquetada) Uma árvore é etiquetada se existe uma função $l : V \rightarrow L$, onde L é um conjunto de etiquetas (*labels*). Cada nó da árvore é identificado pelo seu nome v e seu *label* ou etiqueta $l(v)$.

Em *Web Mining*, os *labels* correspondem às páginas visitadas. Em *XML Mining*, os *labels* correspondem aos elementos e atributos (*tags*) dos documentos.

Exemplo 2.1.2 *Na Figura 2.2, tem-se a ilustração de uma árvore etiquetada de tamanho 5, ou seja, possui 5 nós, e um grafo acíclico e conexo que não é árvore . Os elementos no interior dos círculos representando os nós da árvore são as etiquetas. Os nós são denotados por* $n_0, n_1, ..., n_n$.

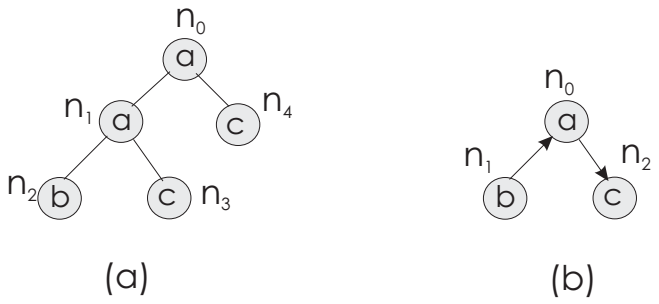

Figura 2.2: (a) Árvore de tamanho 5, (b) Grafo acíclico conexo que não é árvore

De agora em diante, as arestas em uma árvore serão representadas por um segmento de reta sem flecha. A direção da flecha está implícita na posição dos nós na figura 2.2.

Definição 2.1.8 (Ancestrais e Descendentes) Seja T uma árvore com raiz r e x um nó de T. Seja $(r, v_1), (v_1, v_2), ..., (v_n, x)$ o caminho único ligando r a x em T. Os nós $r, v_1, v_2, ..., v_n$ são chamados de *ancestrais* de x. Por outro lado, para cada folha y, considere o caminho único $(x, x_1), (x_1, x_2), ..., (x_m, y)$ ligando x a y em T. Os elementos $x_1, ..., x_m, y$ são chamados de *descendentes* de x. Dois nós v e u que não estão relacionados pela relação de descendência (v não é descendente de u nem u é descendente de v), mas que têm um ancestral em comum são chamados de *primos*.

Definição 2.1.9 (Distância de um nó a seu ancestral) Seja x um nó e y um ancestral de x numa árvore T. Então existe um único caminho $(y, v_1), (v_1, v_2), ...(v_{k-1}, x)$ ligando x a y. O número k é chamado de *distância de x a seu ancestral y*.

Exemplo 2.1.3 *Por exemplo, considere a árvore da Figura 2.2 (a). O nó $n1$ é pai dos nós $n2$ e $n3$. Os nós $n2$ e $n3$ são primos (no caso, eles são ditos irmãos, pois possuem o mesmo pai). Seu menor ancestral comum é o nó $n1$. Os nós $n2$ e $n4$ são primos. Seu menor ancestral comum é o nó $n0$. A distância entre $n0$ e $n2$ é 2. O nó $n2$ é descendente de $n0$ e de $n1$, que são seus ancestrais.*

Representação de Árvores por *Strings*. Existem diversas maneiras de se representar uma árvore etiquetada, entre elas a representação por *matriz de adjacência*, a representação por *lista de adjacências* e a representação por meio de uma tripla (label,ponteiro para o primeiro filho, ponteiro para o primeiro irmão). Nesse trabalho, o algoritmo de mineração de padrões arborescentes CobMiner utiliza a representação por *Strings* que é a mais econômica no que se refere a espaço de armazenamento. Para uma árvore de tamanho n, a representação via matriz de adjacência requer $n(f + 1)$ espaços (onde f = máximo *fanout* (número de arestas saindo de um nó); a representação via lista de adjacência requer $4n - 2$ espaços; a representação via triplas (label,ponteiro filho, ponteiro irmão) requer $3n$ espaços. A representação via *string* requer $2n - 1$ espaços .

Definição 2.1.10 (Representação de Árvore por *String*) Seja T uma árvore. A representação por *string* de T, denotada por $\mathcal{T}$, é definida da seguinte maneira: percorre-se T a partir do nó raiz, em profundidade, da esquerda para a direita, fazendo a enumeração dos nós. O nó raiz é n_0, seu primeiro filho é n_1, o primeiro filho de n_1 é n_2 e assim por diante. Seja f a seguinte função:

f: $\mathbf{N} \rightarrow L \cup -1^n.L$

onde:$-1^n.L$ denota o conjunto de *Strings* formados por um bloco de -1 seguido de um *label* $l \in L$.

A função f é definida da seguinte maneira:

$f(0) = l(0)$,

$f(i)$ = $l(n_i)$ se n_{i-1} é pai de n_i;

$f(i)$ = $(-1)^k l(n_i)$ se n_i é primo de n_{i-1} e k = distância de n_{i-1} ao primeiro ancestral comum entre n_i e n_{i-1}.

$f(i)$ = $(-1)^k$ se i = tamanho da árvore e k = distância de n_i até a raiz.

Então, a representação em string de T é dada pela sequência $f(0)f(1)...f(p)$, onde p é o tamanho da árvore T.

Exemplo 2.1.4 *Por exemplo, a árvore da Figura 2.2(a) é representada pelo string* $aab - 1c - 1 - 1c - 1$, *pois:*

$f(0) = l(n_0) = a$

$f(1) = l(n_1) = a$, *já que* n_0 *é pai de* n_1,

$f(2) = l(n_2) = b$, *já que* n_1 *é pai de* n_2,

$f(3) = -1l(n_3) = -1c$, *já que* n_2 *e* n_3 *são primos (irmãos) e* $k = 1$ *é a distância de* n_2 *até* n_1 *que é o primeiro ancestral comum entre* n_2 *e* n_3

$f(4) = -1 - 1c$, *já que* n_4 *e* n_3 *são primos e* $k = 2$ *é a distância de* n_3 *até* n_0, *o primeiro ancestral comum entre* n_3 *e* n_4.

$f(5) = -1$, *já que* 5 *é o tamanho da árvore e* $k = 1$ *é a distância de* n_4 *até a raiz.*

Definição 2.1.11 (Subárvores) Sejam duas árvores T e S tal que, $T = (\{t_0, ..., t_n\}, E_t)$ e $S = (\{s_0, ..., s_k\}, E_s)$. Diz-se que S é *subárvore de* T, ou que S *está contida em* T, (denotado por $S \preceq T$) se existe uma função *M*: $\{0, 1, ..., k\} \rightarrow \{0, 1, ..., n\}$ tal que:

- $l(s_i) = l(t_{M(i)})$
- se s_i é pai de s_j então $t_{M(i)}$ é ancestral de $t_{M(j)}$.

A sequência $(M(0), ..., M(k))$ é chamada de *match-label* de S com relação a T. Repare que cada *match-label* corresponde a uma *ocorrência* de S em T, podendo haver diversas destas ocorrências.

Uma subárvore de tamanho k é chamada de k-subárvore.

Exemplo 2.1.5 *Considere as árvores T e S ilustradas na Figura 2.1.5. A sequência S é subárvore de T. Existem três ocorrências de S em T. Os match-labels correspondentes são: (0,4), (0,3) e (1,3).*

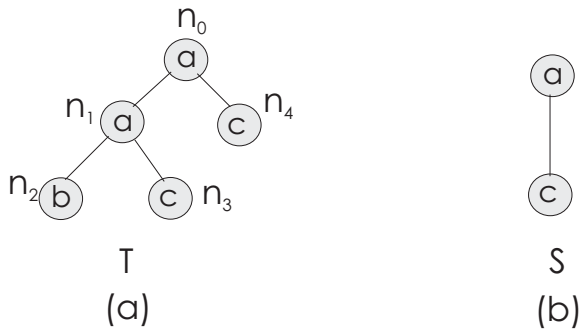

Figura 2.3: Uma árvore e uma subárvore

Definição 2.1.12 (Padrão Arborescente) Seja L um conjunto de *labels*. Um *padrão arborescente* sobre L é uma árvore etiquetada por elementos de L. Um padrão arborescente σ é *suportado* por uma árvore T se e somente se $\sigma \preceq T$.

Definição 2.1.13 (Suporte) Seja **T** um conjunto de árvores ordenadas e etiquetadas sobre um conjunto de *labels* L, e σ um padrão arborescente sobre L. O *suporte* de σ com relação a **T** é a porcentagem de árvores de **T** que suportam σ. Este número será denotado por $\text{sup}_{\mathbf{T}}(\sigma)$. Um padrão arborescente σ é dito *freqüente* com relação a **T** e um nível mínimo de suporte α se $\text{sup}_{\mathbf{T}}(\sigma) \geq \alpha$.

Exemplo 2.1.6 *Considere o banco de dados **T** composto das árvores T_1, T_2, T_3, como mostra a Figura 2.4. Considere também o padrão S ilustrado na mesma figura. Este padrão é suportado pelas árvores T_1 e T_2, mas não é suportado pela árvore T_3. Se o nível mínimo de suporte é 50%, o padrão S é freqüente em **T**, já que seu suporte é 2/3 = 0,66* $\geq 0,5$.

Definição 2.1.14 (Profundidade de uma Árovre) Sejam $T = (\{t_0, ..., t_n\}, E_t)$ uma árvore de *labels* e n o número de nós de T. A profundidade de um nó da árvore T denotada por $Prof(t_i)$ é o comprimento da raiz até o nó t_i. O nó raiz tem profundidade 0. A profundidade da árvore T denotada por $Prof(T)$ é a máxima profundidade do conjunto dos nós de T, expressa por $Prof(T) = max\{P(T_1), ..., P(T_n)\}$.

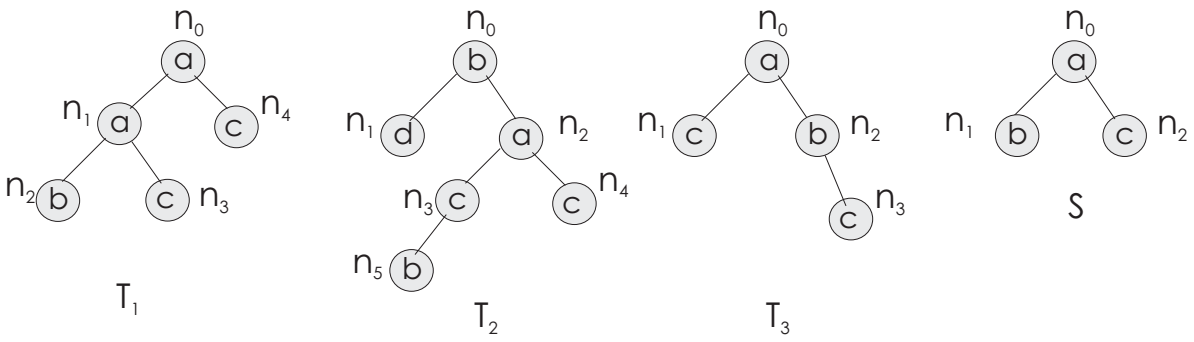

Figura 2.4: Um conjunto de árvores um padrão arborescente S

Exemplo 2.1.7 *Na figura 2.4, a árvore T_2 tem como nó de maior profundidade o nó n_5, portanto $Prof(T_2) = 3$.*

2.2 Autômatos

Essa seção irá definir e exemplificar dois mecanimos de representação de restrições. Os autômatos finitos determinísticos são usados por algoritmos de mineração de padrões sequênciais para gerar padrões válidos segundo uma expressão regular [Garofalakis et al. 1999]. Enquanto que na mineração de padrões arborescentes, o algoritmo CobMiner (algoritmo de mineração de árvores utilizado nesse trabalho) [Silva 2007, de Amo et al. 2007], faz a validação de padrões através de autômatos de árvore, mais precisamente os automâtos de árvore locais que serão descritos a seguir.

2.2.1 Autômatos Finito Determinístico

Definição 2.2.1 (Autômato Finito Determinístico) Um autômato finito deterministico é uma tupla $\mathcal{A}_{\mathcal{R}} = (Q, \Sigma, \delta, q_0, F)$, onde Q é um conjunto finito de estados, Σ é o alfabeto, conjunto finito de símbolos $\delta : Q \times \Sigma$ é a função de transição, $q_0 \subseteq Q$ é o estado inicial e $F \subseteq Q$ é o conjunto de estados finais do autômato. De forma informal, um autômato finito deterministico é uma máquina de estados finito com: um estado inicial q_0 e um ou mais estados aceitos bem definidos, e transições deterministicas entre os estados dos símbolos de um dado alfabeto. Um autômato finito deterministico $\mathcal{A}_{\mathcal{R}}$ pode ser construído a partir de uma expressão regular.

Exemplo 2.2.1 *Na figura 2.5, tem-se o autômato finito deterministico* A_{R1}=$(Q, \Sigma, \delta, q_0, F)$ *que foi construído a partir da expressão regular* $\mathcal{R}$=$a^*(e\ e|e\ i\ o|o\ o)$. *O conjunto de estados* Q=q_0, q_1, q_2, q_3*, sendo que o estado inicial do autômato é* q_0*, e o final* F=q_3*. As funções de transição de* A_{R1} *são:* $\delta(q_0, a) = q_0, \delta(q_0, e) = q_1, \delta(q_0, o) = q_2, \delta(q_1, i) = q_2, \delta(q_1, e) = q_3, \delta(q_2, o) = q_3$.

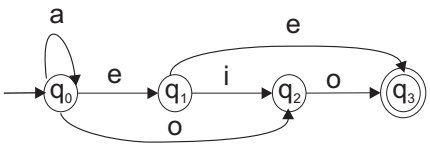

Figura 2.5: Autômato finito deterministico associado a uma expressão regular

2.2.2 Autômatos de Árvore

Enquanto autômatos finitos são projetados para aceitar palavras (seqüências) o autômato de árvore é usado para aceitar árvores cujos nós são *labels* sobre um alfabeto L. Dessa forma, o autômato de árvore pode naturalmente ser pensado como um mecanismo para caracterizar um conjunto de árvores.

Definição 2.2.2 (Autômato de Árvore) Um autômato de árvore, é uma tupla $\mathcal{A}$ = (Q, Σ, q_0, δ) onde Q é um conjunto finito de estados, Σ é o alfabeto (conjunto finito de símbolos), $q_0 \subseteq Q$ é o estado inicial, e $\delta : Q \times \Sigma \rightarrow 2^Q$ é uma função de transição que associa a cada par $(q, a) \in Q \times \Sigma$ uma expressão regular sobre Q (que é um conjunto de *strings* de estados satisfazendo uma expressão regular).

Uma árvore T é aceita pelo autômato de árvore $\mathcal{A}$ se T é percorrida por $\mathcal{A}$. Essa validação de $\mathcal{A}$ pode ser feita sobre os nós da árvore na direção *bottomup* ou *top-down*. Numa visão da validação *top-down*, o estado inicial é associado à raiz e novos estados são associados aos nós internos de T pela função δ, de acordo com seus *labels* e estados de seus nós pais. A árvore é aceita por $\mathcal{A}$ se a função de transição aplicada aos nós folhas de T produzir a *string* vazia.

Exemplo 2.2.2 *Considere o autômato de árvore* $\mathcal{A}$ = (Q, Σ, δ, q_0)*, tal que* $Q = \{q_0, q_1, q_2\}$*, onde* q_0 *é o estado inicial,* $\Sigma = \{a, b, d\}$*,* $\delta(q_0, a) = q_1 q_2$*,* $\delta(q_1, b) = \delta(q_2, d) = \epsilon$*. A figura 2.6*

mostra o exemplo de uma árvore que é aceita por $\mathcal{A}$. Inicialmente, $\mathcal{A}$ associa o estado q_0 à raiz de T. A função δ aplicada à raiz, $\delta(q_0, a)$, associa aos seus nós filhos os estados q_1 e q_2. A aplicação de δ aos labels das folhas produz a string vazia.

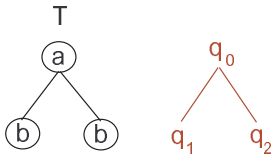

Figura 2.6: Uma árvore e seu percurso de validação pelo autômato $\mathcal{A}$.

Definição 2.2.3 (Gramática de Árvore) Uma gramática de árvore regular é uma tupla $G = (N, T, S, P)$, onde N é o conjunto finito de nós não terminais, T é o conjunto finito de nós terminais, $S \subset N$ é o conjunto de estados iniciais, e P é o conjunto finito de regras do tipo $X \rightarrow a\ r$, onde $X \in N$, $a \in T$ e r é uma expressão regular sobre N.

Exemplo 2.2.3 *No exemplo 2.2.2 a gramática de árvore regular $G_{\mathcal{A}}$ que corresponde ao autômato $\mathcal{A}$, é $G_{\mathcal{A}}$ == (N, T, S, P), tal que $N = \{q_0, q_1, q_2\}$, $T = \{a, b, d\}$, $S = \{q_0\}$ e $P = \{q_0 \rightarrow a\ (q_1 q_2),\ q_1 \rightarrow b\ \epsilon,\ q_2 \rightarrow d\ \epsilon\}$.*

Pode haver entre os nós de uma gramática de árvore regular uma competição. Isso ocorre quando existe dois diferentes nós não terminais q_i e q_j disputando um mesmo nó terminal t.

Definição 2.2.4 (Competição entre nós) Dois nós não terminais de uma gramática regular de árvore q_i e q_j competem entre si quando: $q_i \neq q_j$ e existe um nó terminal t, tal que existe uma regra de produção que mapeia q_i em t e existe uma regra de produção que mapeia q_j em t.

Definição 2.2.5 (Gramática Regular de Árvore) Uma gramática regular de árvore é dita *local* quando não existe competição entre seus nós não terminais.

Definição 2.2.6 (Autômato de Árvore Local) Um autômato de árvore local é um autômato de árvore definido por uma gramática de árvore local.

A gramática $G_{\mathcal{A}}$ é um exemplo de gramática de árvore local. No exemplo 2.2.4 é mostrada uma gramática onde há a competição entre os nós.

Exemplo 2.2.4 *Considere a seguinte gramática de árvore regular $G_{\mathcal{B}}$, onde $N = \{q_0, q_1, q_2, q_3\}$, $T = \{a, b, d\}$, $S = \{q_0\}$ e $P = \{q_0 \rightarrow a\ (q_1q_2 + q_3q_2),\ q_1 \rightarrow b\ \epsilon,\ q_2 \rightarrow d\ \epsilon,\ q_3 \rightarrow b\ (q_2)\}$. Os estados q_1 e q_3 estão competindo o label b.*

As restrições do algoritmo CobMiner mostrado na seção 3.4.2 são restrições locais representadas por um automâto de árvore local.

2.3 Linguagens de Consulta XML

Os dados XML são representados por árvores, e consultas sobre esses dados são intuitivamente representadas por uma árvore de consulta. A idéia é procurar por padrões em uma base de dados que se encaixem ou satisfaça a árvore de consulta, e retornem um conjunto com os padrões encontrado [Amer-Yahia et al. 2002]. A linguagem VisTree, que será definida no capítulo 4, faz a especificação visual de classes de padrões arborescentes e pode ser usada para expressar uma árvore de consulta. No entanto, para fazer uma consulta aos dados XML é necessário converter as expressões da linguagem VisTree em uma expressão de uma linguagem de consulta XML. Nessa seção será dada uma visão geral sobre as linguagens XPath, Tree Pattern e XQuery.

2.3.1 XPATH

A linguagem XPath (*XML Path Language*) é usada para encontrar informações e navegar através de um documento XML. A linguagem XPath é formada por expressões de caminho que fazem a seleção de nós ou conjuntos de nós do documento XML. Elas se assemelham a um sistema de arquivo de um computador.

Os documentos XML são tratados pela linguagem como uma árvore de nós, a raiz da árvore é chamada de nó documento ou nó raiz. A relação entre os nós de um documento em XPath pode ser de filho, pai, irmão, ancestral ou descendente.

A seleção de nós pelas expressões de caminho pode ser feita com o uso de caminhos ou passos. A expressão de caminho traça um caminho através da árvore do documento XML, identificando todos os nós que serão retornados pela expressão. O resultado da avaliação de uma expressão poderá ser uma sequência de nós, um nó único, ou valores simples como *strings*, números, etc.

Os símbolos mais usados nas expressões de caminho são:

- $nodename$: Seleciona todos os nós filhos do nó corrente;
- /: Seleciona os nós a partir do nó raiz;
- //: Seleciona todos os nós do documento que são descendentes do nó corrente, não importando o local onde ele aparece;
- . : Seleciona o nó corrente;
- .. : Seleciona o pai do nó corrente;
- @ :Seleciona os atributos;

Além desses operadores, as expressões XPath podem conter ainda predicados ou chamadas de funções. Predicados em XPath são chamados expressões de filtro e são usados nas expressões de caminho para encontrar nós específicos ou um nó que contêm um valor específico; eles são inseridos em uma expressão entre colchetes ([]).

Exemplo 2.3.1 *Um exemplo de uso de predicados é a seleção de elementos de um nó pelo índice, que indica a posição em que ele aparece no documento. A consulta a seguir, feita no arquivo* books.xml*(figura 2.7), seleciona o primeiro elemento book, filho do elemento bookstore:*

Expressão XPATH=/bookstore/book[0]

Para seleção de nós desconhecidos são usados os curingas:

- $*$: casa com qualquer nó do tipo elemento;
- $@*$: casa com qualquer nó do tipo atributo;
- node(): casa com qualquer nó de qualquer tipo;

Um eixo em XPath define um conjunto de nós relativo ao nó corrente. Os eixos definidos nessa linguagem são:

- ancestor: Seleciona todos os ancestrais (pai, avô, etc.) do nó corrente

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!– Edited with XML Spy v2007 (http://www.altova.com) –>
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

Figura 2.7: Arquivo Books.xml

- ancestor-or-self: Seleciona todos os ancestrais (pai, avô, etc.) do nó corrente e o próprio nó corrente;
- attribute: Seleciona todos atributos do nó corrente;
- child: Seleciona todos os filhos do nó corrente;
- descendant: Seleciona todos descendentes (filho, neto, etc.) do nó corrente;
- descendant-or-self: Seleciona todos descendentes (filho, neto, etc.) do nó corrente e o próprio nó corrente;
- following: Seleciona tudo no documento após a tag de fechamento do nó corrente;
- following-sibling: Seleciona todos irmãos após o nó corrente;

- namespace: Seleciona todos nós namespace do nó corrente;
- parent: Seleciona os pais do nó corrente;
- preceding: Seleciona tudo no documento que é anterior a tag de início do nó corrente;
- preceding-sibling: Seleciona todos irmãos anteriores ao nó corrente;
- self: Seleciona o nó corrente;

Em uma expressão de caminho, um caminho local pode ser absoluto ou relativo. Se o caminho local é absoluto, o conjunto de nós atual é o nó raiz. Se o caminho local é relativo, o conjunto de nós atual consiste do nó onde a expressão começa a ser usada. Um caminho local absoluto começa com uma barra (/) e um caminho local relativo não. Em ambos os casos o caminho local consiste de um ou mais níveis de localização, cada um separado por uma barra. Os passos são avaliados em ordem, um após o outro, da esquerda para direita. Cada passo é avaliado segundo os nós no conjunto de nós atuais.

Exemplo 2.3.2 *Nesse exemplo tem-se um caminho absoluto e um caminho relativo referente ao documento XML* books.xml*(figura 2.7). O caminho absoluto parte da raiz do documento, nó* bookstore *e o caminho relativo parte do nó* book.

Caminho Absoluto*: /bookstore/book/title*

Caminho Relativo*: book/title*

Cada passo é avaliado contra os nós do conjunto de nós corrente. Um passo consiste de:

- eixos: Define a relação na árvore entre os nós selecionados e o nó corrente;
- nó-teste: Identifica um nó com um eixo;
- zero ou mais predicados: Para refinar a seleção do conjunto de nós;

A sintaxe para um passo de localização é:

$eixo :: n - teste[predicado]$

O conjunto de operadores da linguagem XPath são:

- $|$: Concatena dois caminhos, dois conjuntos de nós;
- $+$: Adição;
- $-$: Subtração;
- $*$: Multiplicação;
- div: Divisão;
- $=$: Igualdade;
- $!=$: Desigualdade;
- $<$: Menor que;
- $<=$: Menor igual que;
- $>$: Maior que;
- $>=$: Maior igual que;
- or: OU;
- and: E;
- Mod: Modulo (resto da divisão);

2.3.2 Tree Pattern

Na maioria das linguagens de consulta XML, incluindo XQuery [Chamberlin 2003], a consulta é executada pela associação de váriaveis a nós de interesse. A *tree pattern query* $TP(Q)$ seria uma abstração para especificar as associações entre variáveis e nós, ela é uma árvore T cujos nós possuem variávieis como rótulos, juntamente com uma fórmula booleana F especificando restrições nos nós e suas propriedades, incluindo suas *tags*, atributos e conteúdos [Chen et al. 2003]. A árvore possui dois tipos de bordas: pai-filho (pc) representada por uma borda simples, e ancestral-descendente (ad) representada por uma borda dupla. As consultas

XPath também podem ser representadas visualmente através de *tree patterns* como pode ser visto no trabalho apresentado pelos autores Miklau e Suciu [Miklau and Suciu 2004].

A semântica de uma TPQ é interpretada pela noção de um *match* de padrões, ou seja, um mapeamento de nós do padrão para nós em uma base de dados XML tal que a fórmula associada com o padrão bem como a relação estrutural entre os nós do padrão são satisfeitas. Visto como uma consulta, a resposta a uma TPQ é o conjunto de todos os nós correspondentes a *matches* válidos.

Exemplo 2.3.3 *A figura 2.8 mostra um exemplo de uma TPQ que possui quatro nós; o nó P possui uma relação de ancestral-descendente com o nó S e uma relação pai-filho como o nó I. A tree pattern faria o match com árvores do documento com um nó pessoa que tenha como descendente um nó estado com valor $\neq$ de "GO", e um nó filho perfil com idade $>$ 30.*

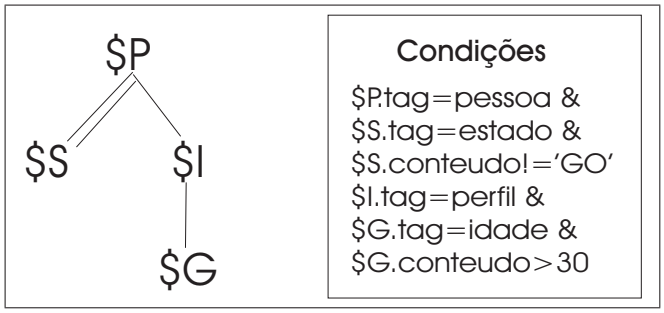

Figura 2.8: Exemplo de uma TPQ(*Tree Pattern Query*) e a fórmula F associada a ela

2.3.3 XQuery

A linguagem de consulta XQuery é derivada de outra linguagem de consulta XML chamada Quilt, que por sua vez possui características de outras linguagens. A linguagem *Quilt* foi influenciada pela semelhança funcional com a linguagem OQL (*Object Query Language*), pela sintaxe baseada em palavras chaves da linguagem de consulta a banco de dados SQL (*Structured Query Language*) e por outras linguagens de consulta a documentos XML anteriores incluindo XPath, XQL, XML-QL e Lorel [Chamberlin 2003]. A linguagem XQuery é compatível com os padrões Web existentes incluindo Schema, XSLT, XPath e XML.

A versão 1.0 da linguagem XQuery é uma extensão da versão 2.0 da linguagem XPath. Isto

significa que qualquer expressão que é sintaticamente válida e executada com sucesso nas duas linguagens, irão retornar o mesmo resultado.

Em um documento XML, um item pode ser um nó ou um valor atômico. Um valor atômico é uma instância de um dos tipos de dados construídos tais como *strings*, inteiros, decimais e data. Um nó pode ser de um dos sete tipos existentes que inclui elementos, atributo, texto, documento, comentário, processamento de instrução e *namespace*.

O modelo de dados da linguagem XQuery é baseado na noção de sequência; onde uma sequência é uma coleção ordenada de zero ou mais itens. Um item é um nó ou um valor atômico. Um nó pode ter outros nós como filhos, formando então uma ou mais hierarquias de nós. Nós possuem um identificador único o que faz com que dois nós sejam distiguidos um do outro mesmo que seus nomes e valores sejam os mesmos; mas valores atômicos não possuem identificador. O documento possui uma ordem total na qual cada nó aparece antes de seu filho. Essa ordem corresponde a ordem que os nós devem aparecer se a hierarquia for representada no formato XML.

A linguagem XQuery consiste de vários tipos de expressões. Dentre as expressões da linguagem têm-se as expressões de caminho, os construtores de elemento, as chamadas de funções, as expressões lógicas e aritméticas, as expressões condicionais, as expressões quantificadoras, as expressões em seqüências e as expressões em tipos. As expressões de caminho presentes na sintaxe da linguagem XQuery possuem uma notação bastante similar à utilizada pela linguagem XPath.

Geralmente, as consultas expressas através da XQuery utilizam as expressões de caminho para identificar o início de uma sub-árvore do documento para em seguida, extrair a informação requerida, utilizando as outras expressões da linguagem. Porém, as consultas podem ser expressas utilizando apenas as expressões de caminho herdadas da linguagem XPath. A diferença de uso de expressões de caminho de uma linguagem para outra está na possibilidade de incluir variáveis para representar o caminho presente na linguagem XQuery. As variáveis são associadas a um caminho por outras estruturas da linguagem XQuery e a expressão de caminho terá sua avaliação dependente do valor atribuído à variável.

Consultas XML podem além de buscar elementos em um documento ter a necessidade de gerar novos elementos. Os construtores de elementos permitem que o resultado de uma consulta

seja uma árvore diferente da árvore fornecida inicialmente. Isso é possível através da criação de novos elementos e modificação da ordem daqueles originalmente fornecidos.

Os construtores de elementos utilizam a mesma sintaxe dos documentos XML para gerar os novos elementos. Desse modo, para criar um novo elemento deve-se inserir um marcador para representá-lo e especificar o delimitador de início e fim, além do conteúdo, caso seja necessário. A utilização mais comum dos construtores de elementos é na produção do resultado de uma consulta. Por isso, é comum aparecer construtores de elementos após a cláusula *RETURN* das expressões FLWOR [Chamberlin 2003]. As expressões FLWOR da linguagem XQuery são formadas pelas cláusulas $For...\ Let...\ Where...\ Orderby...\ Return...$ onde:

- FOR/LET: Associam valores às variáveis;
- WHERE: Filtra o resultado vindo das cláusulas FOR/LET;
- ORDER BY: Ordena os resultados por uma ou mais especificação de ordenação;
- RETURN: Gera a saída da consulta ;

As expressões FLWOR se assemelham às consulta $Select...\ From...\ Where...$ da linguagem de consulta a banco de dados SQL. A descrição do funcionamento de uma expressão FLOWR poderia ser resumida à: a expressão associa valores a uma ou mais variáveis e a partir dos valores associados ela usa essas variáveis para construir o resultado.

Detalhando o funcionamento de cada uma das cláusulas tem-se que a cláusula For é utilizada para executar uma iteração sobre a variável associada. Dessa forma, para cada passo executado na iteração da cláusula *For*, a variável é associada a um novo valor. A variável declarada na cláusula *For* pode ser associada a uma expressão de caminho que retorne uma seqüência de nós, o que faz com que durante o processamento da consulta, o valor de cada nó seja atribuído individualmente a variável da cláusula *For*. A cláusula *Let* também é utilizada para associar variáveis, mas não executa nenhum tipo de iteração. Assim, caso uma seqüência de nós seja atribuída a uma variável da cláusula *Let* essa variável será uma lista com a seqüência de nós completa.

Exemplo 2.3.4 *Dado as consultas a seguir, uma com a cláusula For e a outra com a claúsula Let; feitas sobre o documento $Books.xml$ (Apêndice A):*

$for\ \$root\ indoc('books.xml')/bookstore/book$

$return < resultado > \$root < /resultado >$

$let\ \$root = doc('books.xml')/bookstore/book$

$return < resultado > \$root < /resultado >$

A figura 2.9 mostra como seria o resultado de cada uma das consultas. Para a consulta com For é construído uma *tag* $resultado$ para cada um dos nós $book$ presentes no arquivo. Já a consulta feita com o Let teve como saída da consulta apenas uma tag $resultado$ delimitando todos os nós $book$ do documento.

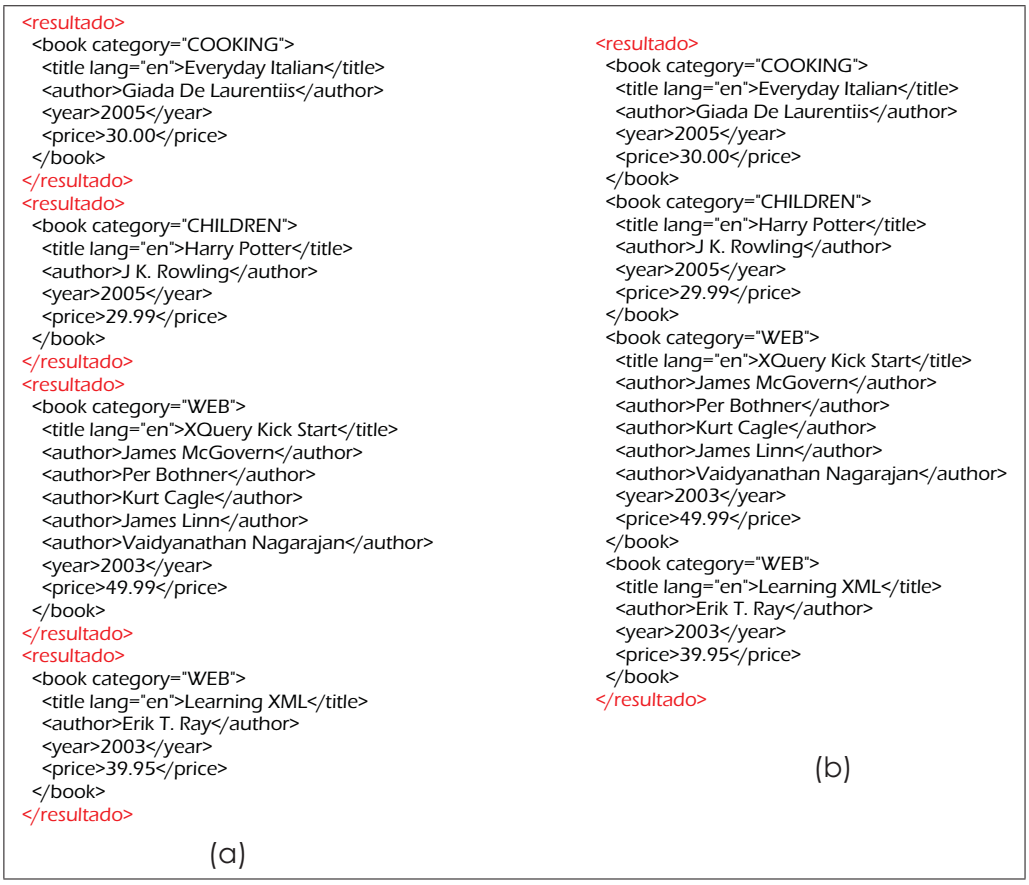

Figura 2.9: (a)Resultado da consulta com a cláusula for;(b)Resultado da consulta com a cláusula let

A cláusula $Where$ é usada para filtrar conteúdos de variáveis associadas pelas cláusulas For e Let, sendo seu uso opcional. A condição expressa na cláusula $Where$ é analisada para deter-

minar se a variável associada deve fazer parte do resultado da consulta. As especificações de ordenação do resultado da consulta são precedidas pela cláusula $Order\ by$ e são denominadas *orderspecs* [Chamberlin 2003], para cada linha do documento resultado elas são avaliadas e a construção do resultado é feita segundo suas condições. Por fim, a cláusula $Return$ determinará qual será o resultado apresentado ao usuário. Normalmente, são utilizadas as variáveis que foram associadas pelas cláusulas For e Let e filtradas pela cláusula $Where$. Além disso, é comum utilizar construtores de elementos para apresentar os resultados através da cláusula $Return$.

Capítulo 3

Estado da Arte

3.1 Introdução

O aumento na quantidade de informações disponíveis e a busca por formas de acessá-las de uma maneira mais eficiente tem contribuído para o crescimento do desenvolvimento de técnicas de mineração de dados. Entre essas técnicas destaca-se a mineração de padrões freqüentes, que possibilita a descoberta de informações escondidas em um grande volume de dados. A estrutura de um padrão varia entre estruturas mais simples como as seqüências à estruturas mais complexas e que possuem mais informações como árvores e grafos.

Um problema comum à mineração de padrões é o grande número de padrões produzidos. Esse problema motivou o desenvolvimento de técnicas que incorporam restrições ao processo de mineração. Como mecanismo de restrição, os algoritmos utilizam estruturas adequadas às características estruturais do padrão. Os autômatos finitos determinísticos são usados para restringir os padrões seqüenciais produzidos e os autômatos de árvore restrigem os padrões arborescentes.

Nesse capítulo serão abordados assuntos que possuem relação com o trabalho que está sendo apresentado. O objetivo da construção desse capítulo foi a introdução de conceitos relacionados ao trabalho desenvolvido e a revisão bibliográfica de alguns trabalhos principais relacionados

a ele. Um esquema comum que representa as etapas da descoberta do conhecimento pode ser visto na figura 3.1. Esse esquema também pode ser aplicado a mineração de padrões, dividindo-a em 3 fases distintas que são: a fase de Pré-processamento, a de Mineração de Dados e a de Análise de Padrões.

Seguindo a ordem das etapas, o capítulo inicia com uma seção sobre a primeira etapa da descoberta de padrões frequentes, a fase de Pré-processamento (foco desse trabalho). Relacionadas à fase de Mineração de Dados, as seções3.3 e 3.4 apresentam o problema de mineração de estruturas (seqüências, árvores, grafos) e a mineração de padrões com o uso de restrição. Na sequência, a fase de Análise de Padrões, outro foco do trabalho, será tratada na seção 3.5. O uso de interfaces em sistemas de mineração de dados também é relatado nesse capítulo, na seção 3.6. As última seção, seção 3.7, refere-se a aplicação da mineração de padrões nas áreas de *XML Mining* e *Web Mining*.

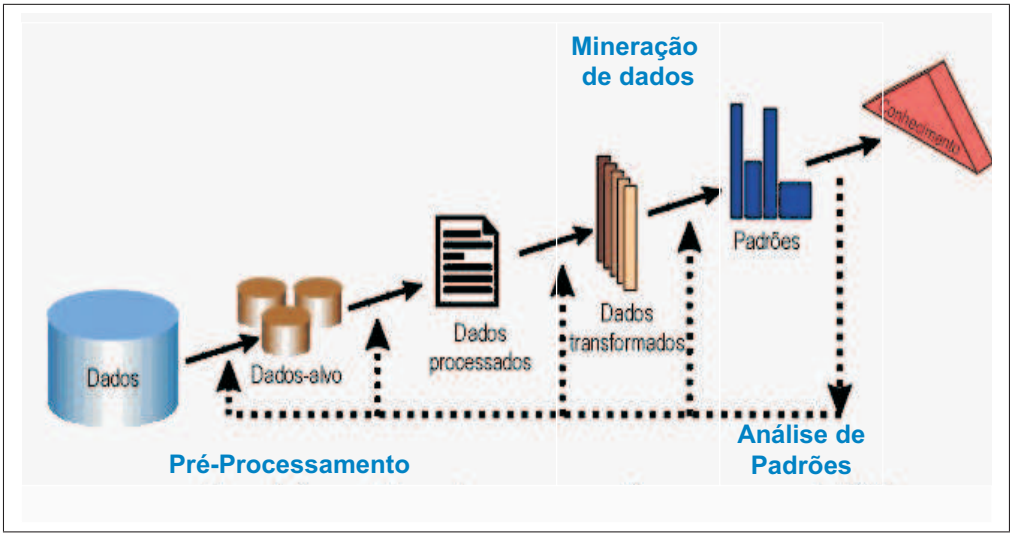

Figura 3.1: Fases da Descoberta do Conhecimento

3.2 Pré-processamento

Os dados fornecidos para o processo de mineração são dados brutos que precisam ser preparados antes de chegarem até aos algoritmos de mineração. A fase de pré-processamento é uma etapa onde serão empregadas técnicas que farão essa preparação. Ela visa não só a organização

dos dados, mas também representá-los de uma maneira que possa minimizar os custos computacionais e produzir padrões mais significativos. Segundo [Han and Kamber 2000], as técnicas de pré-processamento podem ser usadas para fazer a limpeza, integração, transformação e redução dos dados:

- Limpeza: Faz a retirada de ruídos, preenche valores que estejam faltando, identifica ou remove $outliers$ e resolve inconsistências.
- Integração: É feita quando há múltiplas fontes de dados e há a necessidade de se fazer a integração de várias bases de dados (banco de dados, datawarehouse, arquivos, etc).
- Transformação: Altera a forma dos dados colocando-os em um formato apropriado para a mineração.
- Redução: Representa os dados de uma forma reduzida, mantendo a integridade dos dados originais .

Na dissertação apresentada em [Marquardt 2006], foram desenvolvidos mecanismos para auxiliar na fase de pré-processamento da mineração de uso da Web aplicada ao ensino à distância. O estudo procurou analisar se as técnicas de pré-processamento que são utilizadas no ambiente de comércio eletrônico podem ser aplicadas de forma apropriada a outros contextos como o de ensino à distância. Um protótipo contendo as funcionalidades específicas de pré-processamento de dados para o ambiente de educação a distância foi implementado nesse trabalho. A proposta foi reduzir o tempo e os esforços gastos em atividades de pré-processamento e de estabelecer uma relação mais próxima entre os objetivos de mineração e os resultados que são obtidos.

3.3 Mineração de Estruturas: Seqüências, árvores e Grafos

3.3.1 Mineração de Seqüências

A mineração de padrões seqüenciais tem como objetivo a descoberta de informações que tem tendência de ocorrerem sequencialmente, tais como compras de um determinado cliente de uma loja, acessos de um usuário de um site, seqüências de DNA, etc.

Uma das aplicações de mineração de padrões seria descobrir atráves do histórico de itens adquiridos por clientes a tendência de compras futuras. Por exemplo, poderia ser descoberto através da mineração de dados presentes em uma loja de computadores que um número considerável de clientes que compraram computador, compraram monitores de *LCD* em um tempo posterior. Essa informação poderia ser usada em campanhas de marketing direcionado.

A mineração de seqüências foi apresentada em [Agrawal and Srikant 1995], onde padrões sequênciais são minerados a partir de uma base de dados contendo transações de compras de clientes. Nesse trabalho são propostos três algoritmos para solução do problema de mineração de seqüências, destacando o algoritmo AprioriALL.

O algoritmo AprioriALL se baseia na propriedade *Apriori* desenvolvida originalmente para algoritmos de mineração de regras de associação [Agrawal et al. 1993]. Para regras de associação a propriedade *Apriori* é usada na geração de itemsets freqüentes, já na mineração de seqüências ela é usada na geração de itemsets e seqüências freqüentes.

Como resultado do processo de mineração, o algoritmo AprioriALL retorna todas as seqüências de dados que possuem determinado padrão e que satisfaz um percentual mínimo de ocorrência na base de dados. O percentual mínimo de ocorrência (suporte) é fornecido pelo usuário. O algoritmo GSP, que apresenta uma melhor perfomance para a mineração de seqüências que o algoritmo AprioriALL, foi proposto em [Srikant and Agrawal 1996].

O desempenho melhor do algoritmo GSP é explicado pelo fato de possuir um algoritmo de poda mais eficiente que o algoritmo AprioriAll, eliminando assim mais padrões candidatos e diminuindo a carga na fase de cálculo de suporte dos padrões. Enquanto o algoritmo AprioriALL poda os candidatos testando se uma subsequência obtida através da retirada de um itemset é freqüente, o algortimo GSP testa a subsequência obtida com a retirada de apenas um item e dessa forma poda mais candidatos.

3.3.2 Mineração de Árvores

A mineração de árvores permite que sejam minerados padrões que possuem uma estrutra mais complexa do que a dos padrões seqüenciais. Na mineração de seqüências aplicada a mineração de uso da Web é possível minerar padrões que contêm uma sequência de páginas que foram acessadas por determinado número de usuários, enquanto que a mineração de árvores aplicada

a esse mesmo contexto permite estabelecer a relação hierárquica entre as páginas acessadas.

O problema da mineração de dados para dados semiestruturados modelados por árvores, tem sido amplamente estudado nos últimos 5 anos [Asai et al. 2003, Termier et al. 2002, Zaki 2002]. Dentre os trabalhos de mineração de árvores, o algoritmo FREQT [Asai et al. 2003] minera os padrões de árvores ordenados de uma coleção de dados semiestruturados onde os nós da árvore que representa os dados são *labels*. Foram feitos experimentos, aplicando esse algoritmo na extração de subestruturas freqüentes de um conjunto de páginas Web. O algoritmo é limitado apenas a extração da estrutura dos dados, outros componentes de dados semiestruturados como texto e atributos não são considerados.

O algoritmo TreeFinder [Termier et al. 2002] foi desenvolvido para encontrar padrões de árvore freqüentes em uma coleção de documentos XML. No algoritmo TreeFinder, antes de aplicar o processo de mineração, os dados são clusterizados formando clusters de árvores com dados semelhantes. A partir de cada um dos clusters gerados os padrões de árvore freqüentes são minerados.

Em [Zaki 2002] é definido o algoritmo TreeMiner, que é considerado um dos mais eficientes algoritmos de mineração de árvores. Como características principais, o TreeMiner utiliza a busca em profundidade na fase de geração de candidatos e possui uma estrutura de armazenamento de dados em memória que evita a varredura da base de dados para cálculo de suporte a partir do segundo passo. Essas duas características contribuem para sua eficiência.

O Tree Miner, assim como os outros algoritmos aqui citados, foi testado para uma aplicação em dados reais. O contexto da aplicação foi a Mineração do Uso da Web onde a base de dados corresponde aos *logs* de acesso de um site. Cada uma das árvores, neste contexto representa a navegação do usuário pelo *website*.

3.3.3 Mineração de Grafos

Certas bases de dados possuem características estruturais cujos dados são compostos por segmentos e pelo relacionamento entre eles, esses dados podem ser representados por um grafo. Como uma estrutura de dados geral, grafos cujos nós são *labels*, podem ser usados na modelagem de subestrururas de padrões mais complexas [Yan and Han 2002]. A mineração de grafos consiste basicamente em: dada uma base de dados de grafos e um suporte mínimo descobrir

todos os padrões de grafos freqüentes.

Em [Washio and Motoda 2003] foram introduzidos os fundamentos teóricos e o estado da arte de mineração de dados aplicada a grafos. Dados semiestruturados como *tags* de texto, seqüências e árvore ordenadas ou desordenadas são subclasses de grafos em geral. O algoritmo GSpan [Yan and Han 2002] foi introduzido para minerar subgrafos freqüentes de uma grande base de dados e para reduzir a complexidade do problema de mineração de grafos, somente subgrafos conectados são considerados nesse estudo.

3.4 Mineração com Restrições

A mineração de dados feita usando apenas como controle o suporte poderá retornar uma grande quantidade de padrões desinteressantes dificultando a análise dos padrões. O uso da restrição no processo de mineração visa ter como resultado padrões mais próximos do interesse do usuário. Através do uso de restrições o usuário especificará o tipo de padrão que interesse minerar. Nos algoritmos relatados na seção 3.3 o processo de mineração de padrões é efetuado sem o uso de nenhum mecanismo de restrição. A única condição imposta para que o padrão seja minerado e que ele seja freqüente, isto é, que ele satisfaça um suporte mínimo.

Restrições de padrões são condições estabelecidas pelo usuário para que sejam produzidos padrões de acordo com seu interesse. Pode-se dividi-las em duas categorias que são as restrições de geração e as restrições de validação. O primeiro tipo de restrição são aquelas usadas na Fase de Geração dos algoritmos de mineração, reduzindo assim o número de padrões gerados. As restrições de validação só são verificadas na fase de validação dos algoritmos, isto é, caso todos os padrões candidatos sejam gerados. Após a geração verifica-se os padrões satisfazem a restrição.

Nesta seção serão abordadas formas de representar restrições tanto para a mineração de seqüências, quanto para a mineração de árvores. Serão mostrados ainda dois trabalhos que fazem uso de restrições na mineração, um para a proposta de mineração de seqüências e o outro para a mineração de árvores.

3.4.1 Representação da Restrição

Para minerar padrões com restrição é necessário representar de alguma forma as características que os padrões de interesse devem apresentar. Os dois algoritmos de mineração de dados com restrição abordados na seção 3.4.2 usam dois mecanismos de representação da restrição. O primeiro faz uso de expressões regulares para representar a restrição na mineração de padrões seqüenciais e o segundo utiliza autômato de árvores para minerar padrões arborescentes.

A partir de uma expressão regular $\mathcal{R}$ sempre é possível construir um autômato finito determinístico $\mathcal{A}_{\mathcal{R}}$ tal que $\mathcal{A}_{\mathcal{R}}$ aceita exatamante as palavras geradas por $\mathcal{R}$. Informalmente, um autômato finito determinístico é uma máquina de estados finito com: um estado inicial q_0 e um ou mais estados aceitos bem definidos, e transições deterministas entre os estados dos símbolos de um dado alfabeto.

Autômatos de árvores não determinísticos podem ser usados na pesquisa envolvendo documentos XML de diferentes formas tais como base para linguagens de esquema e validação de esquemas; como mecanismo de avaliação de padrões de linguagens; como algoritmo de consulta e checagem de tipo; como o uso de linguagens regulares de string para lidar com o não determinismo.

Uma DTD, *Document Type Definition*, pode ser definida como um conjunto de regras que definem quais tipos de dados e entidades podem existir em um documento XML, ou seja, ela esquematiza a estrutura das informações dentro de um documento. Em [Murata et al. 2005] encontra-se que uma DTD pode ser representada pela gramática de árvore local, conceito este definido em [Takahashi 1975], a partir da qual naturalmente constrói-se um autômato de árvore local.

3.4.2 Algoritmos de Mineração com Restrição

Algoritmo SPIRIT

Os algoritmos da família SPIRIT fazem uso de expressões regulares como forma de restringir os padrões seqüenciais gerados [Garofalakis et al. 1999]. Esses algoritmos são baseados na técnica de mineração Apriori dos algoritmos de mineração de seqüências apresentados na seção 3.3, porém são gerados apenas padrões que satisfazem uma restrição $\mathcal{R}$ fornecida através de

uma expressão regular ou de uma autômato finito determinístico.

A família SPIRIT é formada por quatro algoritmos, assim nomeados: SPIRIT(N), SPIRIT(L), SPIRIT(V) e SPIRIT($\mathcal{R}$). A diferença entre eles está no nível de relaxamento da restrição $\mathcal{R}$. O nível de relaxamento da restrição estabelece se a geração de padrões será feita de uma maneira mais ou menos restritiva. O artigo [Garofalakis et al. 1999] detalha cada um dos níveis e verifica a performance experimentalmente, encontrando o relaxamento de $\mathcal{R}$ mais apropriado.

Algoritmo CobMiner

A introdução de um método para mineração de árvores baseada em restrições foi apresentada em [Silva 2007, de Amo et al. 2007]; tendo como propósito permitir ao usuário especificar o formato dos padrões de árvore que ele está interessado em minerar. O algoritmo proposto, CobMiner, incorpora as restrições do usuário ao processo de mineração, de forma a produzir somente padrões que as satisfaçam. O uso de restrição reduz amplamente o número de padrões gerados, diminuindo o custo computacional da fase de pós-processamento.

Em termos gerais, o algoritmo CobMiner é um algoritmo de mineração de árvores freqüentes e válidas com respeito a uma restrição imposta pelo usuário. Ele é baseado no algoritmo TreeMiner de mineração de árvores apresentado em [Zaki 2002], mas inclui no processo de mineração o contexto das restrições, conforme proposta apresentada em [Garofalakis et al. 1999] para mineração de seqüências.

O algoritmo CobMiner faz uso das restrições dentro do processo de mineração, nas fases de geração e poda de candidatos. A restrição do usuário é convertida em um autômato de árvore local, que representa naturalmente um mecanismo de especificação de um conjunto de árvores. Assim como os algoritmos da família SPIRIT, o algoritmo considera ainda um nível de *relaxamento* para a restrição do autômato de árvore local $\mathcal{A}$, tornando-a menos restritiva.

3.5 Análise de Padrões

A fase de análise dos padrões freqüentes visa dois objetivos distintos, que seria a interpretação dos padrões e a recuperação de padrões. A interpretação de padrões busca o desenvolvimento de técnicas que auxiliem no entendimento dos padrões minerados, pois estes nem sempre se

encontram em uma forma que tenha algum significado para quem irá analisá-los. Já a recuperação de padrões refere-se a métodos usados para procurar por padrões específicos dentre os padrões freqüentes. A necessidade de aplicação desse método se deve ao fato de que a quantidade de padrões minerados pode ser muito grande, o que dificulta a análise dos resultados ou a localização de padrões alvo.

A dissertação apresentada em [Vanzin 2004] propôs um mecanismo que atua na duas fases do processo de análise de padrões seqüenciais de navegação. Esse trabalho utiliza Ontologia de Domínio para efetuar a recuperação e interpretação dos padrões de uma aplicação da Mineração do Uso da Web. Para a interpretação de padrões a proposta do trabalho foi mapear padrões seqüenciais de URLs em padrões conceituais; e interpretar os padrões conceituais através da análise exploratória da semântica dos padrões. A recuperação dos padrões seqüenciais foi feita através da definição de filtros utilizando a Ontologia de Domínio definida para o projeto. O trabalho propôs ainda o desenvolvimento de um ambiente que auxilia na fase de análise de padrões, incorporando os mecanismos propostos e verificando a validade dos mesmos.

3.6 Interface

A interface pode ser entendida como sendo a parte do sistema onde o usuário poderá fazer interações, fornecer dados e visualizar resultados das operações efetuadas. A visualização de resultados é o processo de transformação da informação em uma forma visual de forma a habilitar o usuário a observá-la, entendê-la e navegar por ela [Auber 2003]. Pesquisas realizadas pela comunidade de visualização da informação(InfoViz) mostram que a representação visual dos dados possibilita uma análise mais rápida destes por parte do usuário final. As razões para isso estão em um fato científico, o cérebro humano tem maior facilidade em compreender informações visuais do que informações textuais. Isso implica que na análise de um grande conjunto de dados, a representação visual é mais eficiente do que a representação através de textos.

O Tulip [Delest et al. 2004] é um sistema de código aberto que possui uma interface amigável para a visualização e manipulação de grandes árvores e grafos. Ele fornece ao usuário a visualização dessas estruturas, permite ainda a navegação interativa e diferentes opções de layout na representação dos dados. O sistema possui duas maneiras de extrair subgrafos, de forma in-

terativa ou através do uso de algoritmos baseados na teoria de grafos e combinatória, incluindo vários tipos de clusters.

3.7 Aplicações de Mineração de Árvores

3.7.1 *XML Mining*

A linguagem XML surgiu como um novo padrão para troca de dados na Web. Recentemente XML tem adquirido grande aceitação tanto pela área comercial quanto pela área de pesquisa. A vantagem no uso dessa linguagem inclui o fato do modelo de dados semi estruturados adotado pelos documentos XML trazer mais estrutura, flexibilidade e riqueza semântica na representação dos dados [Yang et al. 2003].

Com o grande aumento de informações disponíveis em XML, surgiu a necessidade de desenvolvimento de linguagens e ferramentas para gerenciar coleções de documentos XML, bem como a necessidade de minerar informações interessantes destes documentos. Muitos problemas na descoberta de conhecimento em repositórios XML tem sido investigados, sendo alguns deles a descoberta de estrutura de documentos, descoberta de padrões freqüentes e a descoberta de padrões de consulta freqüentes.

Descoberta de Estrutura de Documentos: Fazer a descoberta da estrutura de documentos XML é a finalidade do trabalho apresentado em [Papakonstantinou and Vianu 2000]. Como problema tem-se que muitos documentos XML não têm seus dados organizados, ou seja, eles não possuem uma DTD(*Document Type Definition*). Conhecer a DTD em comum em conjuntos de documentos XML poderia otimizar o armazenamento e métodos de busca dos documentos e de seus conteúdos. O algoritmo apresentado infere DTDs de visões de dados XML, usando uma abstração que foca no conteúdo da estrutura do documento. As visões são definidas por uma linguagem de consulta que produz uma lista de documentos selecionados, e a partir de documentos semelhantes a DTD é inferida.

Descoberta de Padrões freqüentes: Por descoberta de padrões freqüentes em documento XML entende-se a busca por subárvores freqüentes que ocorrem em um ou mais documentos.

O algoritmo FREQT apresentado em [Asai et al. 2002] poderia ser usado em uma base de dados de subárvores de um ou mais documentos XML. Os objetivos em se descobrir subárvores freqüentes de um ou mais documentos são várias. Por exemplo, dada uma base de dados com informações de artigos de um repositório de artigos armazenada em um documento XML, onde cada artigo possui a classificação qualis do congresso onde foi publicado, a mineração das subárvores freqüentes desse documento poderia informar que pesquisadores em universidades do Sudeste tendem a ter um grande número de publicações em congressos Qualis *A*.

Outro algoritmo que foi aplicado na busca por padrões de árvore freqüentes em uma coleção de documentos XML foi o TreeFinder [Termier et al. 2002]. Esse algoritmo separa as árvores de entrada em clusters, agrupando-as pela semelhança entre o conjunto de *labels* de cada árvore e por sua estrutura. Os padrões de árvore freqüentes ocorrem na coleção de árvores de forma direta ou aproximada. A inclusão aproximada preserva a relação de ancestral entre os nós mas não necessariamente a relação de pai. A escolha por utilizar a inclusão aproximada se baseia em que possíveis variações da posição de um *label* aninhado de um documento XML para outro ainda mantêm a semelhança entre as estruturas das árvores.

Descoberta de Padrões de Consultas freqüentes: As expressões de caminho e padrões de árvores selecionados por predicados, que especificam o relacionamento entre as estruturas arborescentes (hierarquia estrutural) dos documentos, são características básicas de linguagens de consulta para documentos XML como XPath [Clark and Derose 1999] ou XQuery [Chamberlin 2003]. Na descoberta de padrões de consulta freqüentes tem-se como motivação o fato do processo de consulta em documentos XML poder ser computacionalmente caro. As consulta a esses documentos envolvem navegações através de árvores, que podem possuir profundidades consideráveis. Assim, a descoberta e armazenamento em *cache* de consultas frequentemente executadas podem aprimorar a performance do processo de consulta em documentos XML.

O algoritmo FastXMiner [Yang et al. 2003] foi definido para descobrir subárvores freqüentes de consultas XML. O trabalho [Yang et al. 2003] estuda o impacto do uso de cache de consultas XML na melhoria da perfomance de sistema de gerenciamento XML, fazendo o armazenamento em cache das consultas mais freqüentes. Apesar do fato de uma consulta XML ser representada

por uma árvore, elas possuem caracteres especiais como "*" e "//" o que justificou a necessidade de criação de um algoritmo específico para a mineração de consultas e não o uso dos algoritmos já existentes de mineração de árvores [Asai et al. 2002, Termier et al. 2002, Zaki 2002].

3.7.2 *Web Mining*

A *Web Mining* ou Mineração na Web visa o desenvolvimento de ferramentas e métodos para análise e descoberta de conhecimento de dados na Web. Os trabalhos sobre *Web Mining* que foram apresentados em [Kosala and Blockeel 2000, Cooley 2000] dividem a Mineração na Web em 3 categorias: Mineração do Conteúdo da Web, Mineração da Estrutura da Web e Mineração do Uso da Web. A Mineração do Conteúdo da Web consiste na extração de informações que sejam interessantes sobre o conteúdo, dados e documentos da Web. Na Mineração da Estrutura da Web a aquisição de informações é feita através da topologia, organização e estrutura do *web-site*. E na Mineração do Uso da Web a descoberta de informações interessantes é feita através dos *logs* de acesso Web, que representa o comportamento do usuário durante a navegação por um *website*.

Considerando a Mineração do Uso da Web há diversas aplicações desenvolvidas. Dentre as principais motivações para o desenvolvimento dessas aplicações estão a personalização, o melhoramento do sistema, modificações no site, business intelligence e a caracterização do uso da Web [Cooley 2000].

- Personalização: Conhecendo os hábitos de navegação de usuários na Web é possível desenvolver diversas aplicações, fazer recomendações dinâmicas baseadas em seu perfil e provêr produtos, serviços ou informações sobre produtos e serviços.

- Melhoramento do sistema: O usuário de uma aplicação Web espera que o sistema possa oferecer a ele um serviço de alta qualidade e com alta perfomance. Os padrões de uso da Web podem ser usados para compreender o comportamento de tráfego de informações de uma rede, o que pode ser usado para fazer o balanceamento de carga ou distribuição de dados. Os padrões freqüentes de navegação podem ser usados ainda para identificar fraudes, quebras no sistema e detectar intrusão.

- Modificação de site: Os padrões freqüentes de navegação do usuário podem ser usados pelo projetista do *website* para fazer alterações no site ou acrescentar novos dados.

- Business Intelligence: Informações de como os clientes estão usando um site Web é de grande importância para campanhas de marketing e e-commerce. Estas informações podem fornecer tendência de compras de produtos e podem ser usadas em propagandas.

- Caracterização do Uso da Web: A mineração do uso da Web foca no desenvolvimento de técnicas que podem predizer o comportamento do usuário enquanto ele interage com a Web.

A mineração do uso da Web é dividida em três fases que são a fase Pré-processamento, Descoberta de Padrões e Análise de Padrões [Cooley 2000]. Os padrões de navegação podem ser expressos no formato de seqüências ou de árvores, sendo que os padrões expressos em árvore possuem uma estrutura mas detalhada. Enquanto os padrões seqüenciais freqüentes representam as seqüencias de acessos freqüentes, os padrões de árvore representam as estruturas desses acessos.

A mineração de padrões seqüenciais de navegação com intuito de melhorar a estrutura de um *website* é apresentado em [Srikant and Yang 2001]. O trabalho procura tratar o problema de que muitos *website*s possuem uma organização diferente da esperada pelos usuários. Ele propõe um algoritmo que encontra de forma automática as páginas de um *website* que estão em uma localização diferente da que o usuário esperava encontrar. Para identificar essas página, o algoritmo considera que o usuário irá retornar caso não encontre a página no local esperado e o ponto de retorno é o local onde o usuário esperava encontrar a página.

O algoritmo TreeMiner de mineração de árvores foi aplicado à mineração do uso da Web [Zaki 2002, Punin and Krishnamoorthy 2002]. Nessa aplicação os relatórios de *logs* dos servidores Web foram descritos através da linguagem LOGML [Punin and Krishnamoorthy 2002, Punin and Krishnamoorthy 1998], que possui um vocabulário XML para expressar estruturalmente o conteúdo de um arquivo de log de navegação de uma maneira compacta. A definição da linguagem LOGML teve por finalidade facilitar o processo de mineração e o armazenamento adicional de informações extraídas dos *logs* de navegação Web. Um documento LOGML possui três seções, a primeira seção é um grafo das visitas dos usuários para páginas Web e hiperlinks;

a segunda possui informações adicionais de relatórios de log como páginas mais visitadas, *user agents* mais usados e palavras chaves mais usadas; e a terceira parte é o relatório das sessões de usuário que é um subgrafo do grafo de log.

Um estudo sobre a combinação entre mineração do uso da Web e Web semântica é feito em [Berendt et al. 2004]. O estudo procura fazer a análise das seguintes questões: como poderia a Mineração semântica do Uso da Web melhorar os resultados da mineração do uso da Web tradicional pela exploração da nova estrutura semântica da Web; e, como a construção da Web semântica pode fazer uso das técnicas de mineração Web. Um entendimento verdadeiro da semântica do uso da Web precisa considerar não somente a informação armazenada nos servidores de logs, mas também o significado que é constituído pelo conjuntos e seqüências de páginas acessadas.

A mineração do uso da Web para um contexto específico é tratado em [Machado 2002]. Este trabalho propõe a aplicação da mineração do uso da Web na avaliação de sites de ensino à distância. Como forma de validar essa aplicação foi feito um estudo de caso, e a partir deste foi proposto um modelo de processo para mineração do uso da Web em ambientes de ensino a distância. A mineração de dados buscou identificar padrões de uso de dados Web através das interações dos alunos com o site, produzindo padrões sequênciais e regras de associação. Os dados foram minerados usando o software comercial Intelligent Miner da IBM.

Capítulo 4

A Linguagem VisTree

4.1 Introdução

Um fator importante a considerar no desenvolvimento de técnicas de mineração de dados é a maneira como um usuário com pouco conhecimento técnico poderia usufruir desse recurso. Para tanto, além da criação de algoritmos complexos que se encarreguem de resolver problemas de mineração é necessário fornecer ao usuário uma maneira simples de acesso e utilização desses algoritmos. Os algoritmos de mineração de dados quando aplicados em dados reais, pressupõem uma fase de pós-processamento que fará a preparação dos dados de entrada para o formato esperado por eles.

A preparação dos dados para a mineração de árvores inclui a transformação da aplicação proposta em uma base de dados de árvores. Considerando mais especificamente a mineração de árvores com restrição, utilizando como mecanismo de restrição o autômato de árvore, pedir ao usuário que forneça diretamente esse dado iria restringir o uso do algoritmo de mineração a pessoas que conheçam exatamente o que é um autômato de árvore. Além disso, mesmo considerando que os usuários que farão uso do algoritmo saibam identificar o que é um autômato de árvore, por sere uma estrutura extremamente complexa a tarefa de montar a restrição diretamente no formato de um autômato de árvore tornar-se-ia limitada e muitas vezes inviável.

A melhor forma de lidar com essa questão é prover uma maneira do usuário fornecer a restrição de uma forma mais intuitiva, e assim, uma aplicação se encarregaria de pré-processar esse dado e faria a tarefa de transformá-lo em um autômato de árvore. A questão agora é de

que maneira isso poderia ser feito. Partindo do problema de mineração em questão, que é mineração de padrões de árvore, uma maneira de especificar o tipo de padrão de árvore que o usuário deseja minerar é através de um "desenho" contendo o "molde" para os padrões que deverão ser obtidos.

Outra etapa no processo de mineração de dados com ou sem restrição é a análise de padrões ou pós-processamento. A fase de análise dos padrões freqüentes visa dois objetivos distintos, que é a interpretação dos padrões e a recuperação de padrões. A interpretação de padrões busca através de mecanismos obter o entendimento dos padrões minerados, pois estes nem sempre se encontram em uma forma que tenha algum significado para quem irá analisá-los. Já a recuperação de padrões refere-se a métodos usados para procurar por padrões específicos dentre os padrões freqüentes. A necessidade de aplicação desse método se deve ao fato que a quantidade de padrões minerados pode ser muito grande o que dificulta a análise dos resultados ou a localização de padrões alvo.

Nessa etapa, considerando mais especificamente como objetivo a recuperação de padrões, objeto desse trabalho, também é importante fornecer ao usuário ferramentas que facilitem esse processo. Considerando ainda a mineração de árvores com restrição, os padrões de árvores freqüentes poderiam ser armazenados em um documento XML, pois esse tipo de documento é naturalmente representado por uma árvore; os padrões são então subárvores do documento. Essa é uma forma de estruturar as informações presente nos padrões. Dessa forma, a recuperação dos padrões poderia ser feita através de uma linguagem de consulta para XML, como a XQuery [Chamberlin 2003]. No entanto, ter-se-ia aqui também uma limitação de uso, pois somente usuários que conhecessem a linguagem de consulta XQuery estariam habilitados a efetuar essa tarefa. De forma análoga à entrada da restrição, a consulta aos padrões de árvore freqüentes poderia ser "desenhada" por uma árvore que expressasse as condições da consulta, e, que pudesse ser transformada via aplicação em uma consulta XQuery.

Para atender os dois casos foi definida uma linguagem visual de especificação de classes de padrões arborescentes, a linguagem VisTree. A proposta da linguagem é representar, através de uma árvore padrão a estrutura e conteúdo que os padrões de árvore devem possuir para serem retornados como resultado de uma consulta ou como resultado de um processo de mineração de árvores com restrição. Nas seções a seguir serão apresentadas as características da linguagem

VisTree. A seção 4.2 trata a sintaxe e semântica da linguagem, e alguns exemplos de uso da linguagem. A seção 4.3 compara a linguagem VisTree com XQuery e depois com autômatos de árvore. As duas últimas seções 4.4 e 4.5 mostram o uso da VisTree para especificação de restrições e para entrada de consultas respectivamente.

4.2 Sintaxe e Semântica da Linguagem VisTree

A linguagem VisTree é uma linguagem de especificação de classes de padrões arborescentes em geral. A linguagem é formada por expressões que apresentam um formato de árvore *especial*, chamadas *e-vtree* (*Expression Visual Tree*). A aplicação de uma *e-vtree* em uma base de dados de árvore retorna padrões de árvore que satisfazem a expressão. Nesta seção serão descritas a sintaxe e semântica da linguagem VisTree, isto é, as regras que definem as expressões da linguagem e o significado atribuído a cada expressão.

Definição 4.2.1 (Expressão *e-vtree*) Considere um alfabeto Σ composto dos seguintes subconjuntos:

- Um conjunto $\mathcal{P}$, cujos elementos são chamados de *predicados*:

  $\mathcal{P}$ = {*igual*, *contem*, *comecaPor*, *terminaCom*, *pertence*, *intervalo*, *maior*, *menor*}. Os predicados *igual*, *contem*, *comecaPor*, *terminaCom*, *maior*, *menor* e *pertence* são ditos predicados *unários de primeira ordem* e *intervalo* é dito predicado *binário*.

- Um conjunto finito $\mathcal{L}$, cujos elementos são chamados de *labels*.

- Um conjunto finito $\mathcal{N}$, cujos elementos são chamados de *nós*.

- Os símbolos lógicos $\vee$, $\wedge$ e $\neg$.

- Os símbolos $/$ e $//$ chamados de *links*.

- O símbolo $*$, chamado de *curinga*, e um símbolo $\perp$ chamado *raiz*.

Seja $N \in \mathcal{N}$. Uma *condição simples sobre o nó* N é uma expressão de um dos seguintes tipos:

- $N\ \theta\ l$ onde θ é um predicado unário e $l \in \mathcal{L}$
- $N\ \theta\ l$ onde θ é um predicado unário e $l \subseteq \mathcal{L}$
- $N\ \theta\ (l_1, l_2)$ onde θ é um predicado binário e l_1 e $l_2 \in \mathcal{L}$.

Para simplificar a notação, denotaremos uma condição simples sobre o nó N simplesmente por $\theta(l)$, ou $\theta(l_1, l_2)$ ou $\theta(l_1, l_2, ..., l_k)$ omitindo a denominação do nó N. Uma *condição sobre o nó* N é definida recursivamente por: (1) uma condição simples sobre N é uma condição sobre N; (2) Se c_1, c_2 e c são condições sobre o nó N então $c_1 \vee c_2$, $c_1 \wedge c_2$ e $\neg c$ são condições sobre o nó N.

As expressões da linguagem VisTree são chamadas de *e-vtrees* e são definidas da seguinte maneira: uma *e-vtree* é uma árvore ordenada ($\mathcal{N}'$, $\mathcal{A}$), onde $\mathcal{N}' \subseteq \mathcal{N}$, juntamente com duas funções P e L, onde:

P: $\mathcal{N}' \rightarrow\ \mathcal{P} \cup \{*\}$, tal que para todo $N \in \mathcal{N}'$, $P(N)$ é uma condição sobre o nó N ou o símbolo $*$.
L: $\mathcal{N}' \rightarrow \{/, //, \perp\}$, tal que $L(N) = \perp$ se e somente se N é o nó raiz da árvore.

Lembrando que a *profundidade* de uma *e-vtree* e, denotada por $Prof(e)$ é o número de níveis de e, de forma análoga a definição de profundidade de uma árvore(ver seção 2.1, capítulo 2).

Uma *e-vtree* e pode ser utilizada para especificar um conjunto de subárvores de uma base de dados T de árvores. Seja T um conjunto de árvores com *labels* em L. Nota-se que há diferença entre uma árvore com *labels* em L e uma *e-vtree* e. Essa diferença pode ser observada no exemplo 4.2.1.

Exemplo 4.2.1 *A Figura 4.1 ilustra uma árvore A com* labels *em L e uma* e-vtree *e. As funções P e L associadas a* e-vtree *e são:*
$P(N_0) = *$
$P(N_1) = igual(article) \vee igual(book)$
$P(N_2) = comecaPor(\text{“}\textit{ti}\text{”}) \wedge terminaCom(\text{“}\textit{tle}\text{”})$
$P(N_3) = igual(author)$

$P(N_4) = comecaPor(\text{“}A\text{”})$

$L(N_0) = \perp$
$L(N_1) = //$
$L(N_2) = /$
$L(N_3) = /$
$L(N_4) = /$

Note que na árvore A os labels *de cada nó são simbolos de L e numa e-vtree os* labels *de cada nó são condições sobre o nó. Além disto, em uma e-vtree existem dois tipos de arestas / ou // ligando um nó pai a um nó filho, enquanto que em uma árvore comum existe apenas um tipo de aresta.*

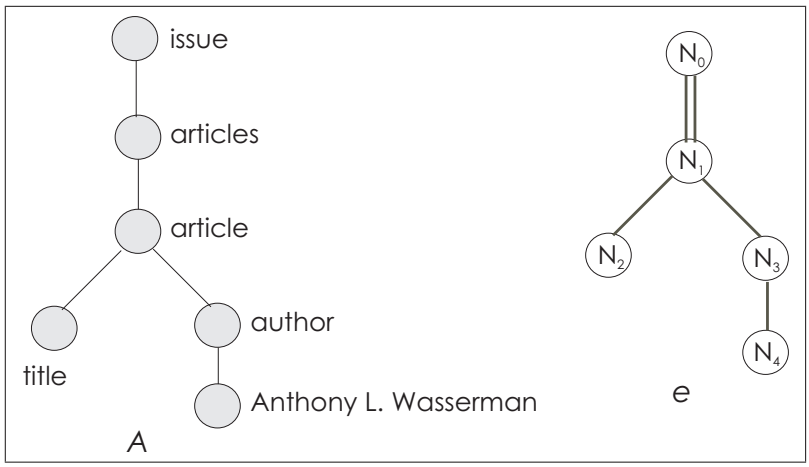

Figura 4.1: Exemplo de uma árvore com *labels* e de uma *e-vtree*.

O conjunto de subárvores associado à uma expressão textite-vtree e é determinado de acordo com a interpretação das condições especificadas sintaticamente. A definição 4.2.2 apresenta formalmente a especificação da semântica de uma expressão *e-vtree*. Antes de apresentar essa definição, pode-se partir de um exemplo da linguagem SQL para entender a semântica de uma expressão *e-vtree* da linguagem VisTree. O exemplo 4.2.2 faz uma analogia entre o que é sintaxe e semântica na linguagem SQL e na linguagem VisTree.

Exemplo 4.2.2 *Considere o arquivo books.xml apresentado no capítulo 2(figura 2.7). Se alguns itens desse arquivo fossem armazenados em um banco de dados relacional teria-se uma representação dos dados semelhante à mostrada na tabela 4.1, aqui denominada de tabela books.*

Title	Author	Year	Price
Everyday Italian	Giada de Laurentiis	2005	30.00
Harry Potter	J. K. Rowling	2005	29.99
Learning XML	Erik T. Ray	2003	39.95

Tabela 4.1: Tabela Books

Uma consulta SQL sobre esses dados poderia ser feita com o intuito de selecionar o título e autor dos livros publicados no ano de 2005. A representação sintática dessa expecificação na linguagem SQL é apresentada abaixo:

$$SELECT\ title, author$$
$$FROM\ books$$
$$WHERE\ year = 2005$$

Por outro lado, a interpretação dessa consulta (semântica), seria o resultado retornado pela consulta SQL apresentado na tabela 4.2:

Title	Author
Everyday Italian	Giada de Laurentiis
Harry Potter	J. K. Rowling

Tabela 4.2: Resultado da Consulta SQL

De forma análoga tem-se como exemplo uma base de dados de árvores apresentada na figura 4.2. Sobre essa base de dados poderia se fazer uma consulta por árvores que tivessem um nó com o label A, que por sua vez possuisse como descendente um nó com o label B e outro com o label C. A sintaxe de uma e-vtree que representa essa consulta pode ser vista na figura

4.3. A interpretação dessa consulta retornaria como resultado as árvores que satisfazem a e-vtree. O conjunto de árvores que satisfazem uma e-vtree é entendido como sendo a semântica da e-vtree.

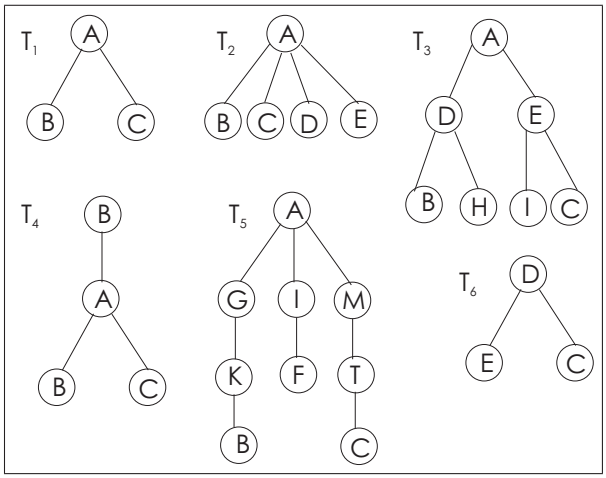

Figura 4.2: Base de Dados de Árvores

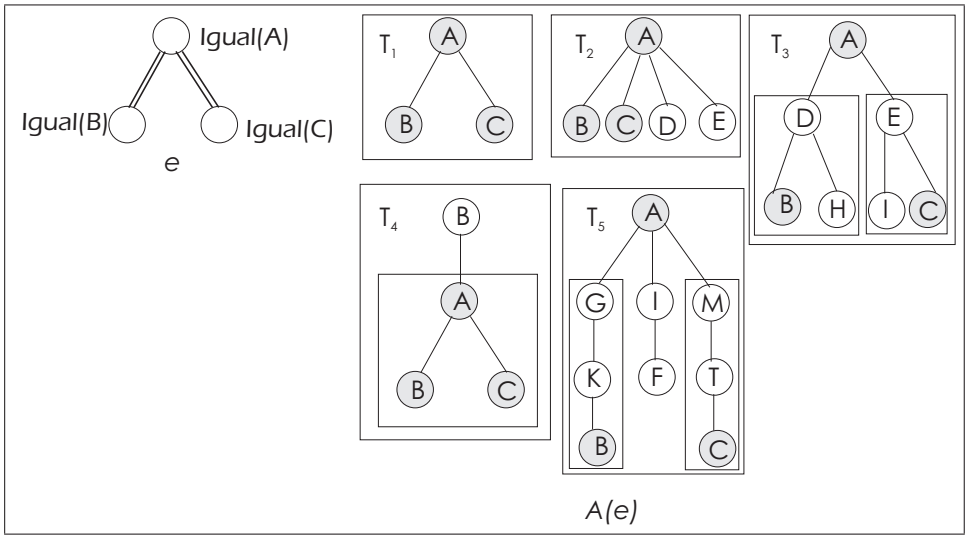

Figura 4.3: Exemplo de uma *e-vtree* e de um conjunto de árvores *A* aceitas por ela.

Definição 4.2.2 (Semântica da Linguagem VisTree) Seja T um conjunto de árvores com *labels* em L e e uma *e-vtree*. Um conjunto de árvores $A(e)$ associado a T e que satisfazem a expressão e, é definido por indução sobre a estrutura da expressão e com se segue:

- Se e só tem um único nó, o nó raiz n_0 então $P(n_0)$ é definido como sendo o conjunto

de todas as árvores com *labels* em $\mathcal{L}$, com um único nó n_0 com *label*(n_0) $\in \mathcal{L}$, tal que *label*(n_0) satisfaz a condição expressa em e e é *label* de alguma árvore em T.:

- Seja e uma *e-vtree* de profundidade $k > 1$ e suponhamos que já tenham sido definidos os conjuntos de árvores $A(e')$ com *labels* em L associados às *e-vtrees* e' de profundidade $< k$. Será definido agora o conjunto de árvores $A(e)$ associado a e e com profundidade igual a k.

  Seja n_0 o nó raiz de e e n_1, n_2, ... n_m seus nós filhos.

  Seja $A(e)$ o conjunto de todas as árvores t, tais que existe $t' \in T$ e t é subárvore de t' onde:

  1. raiz de t satisfaz o predicado $P(N_0)$
  2. a raiz de t tem n filhos $\overline{n_1}, ..., \overline{n_m}$
  3. Se $L(N_i) = /$, então deve existir uma subárvore t'' de t com raiz $\overline{n_i}$ que satisfaz $P(N_i)$

     Se $L(N_i) = //$, nesse caso a ligação entre N_i e N_0 corresponde a uma ligação ancestral-descendente em e, o que indica duas possibilidades:

     - Existe uma subárvore t'' de t com raiz $\overline{n_i}$ que satisfaz $P(N_i)$ (idem a $L(N_i) = /$)
     - Existe uma subárvore t'' de t com raiz $\overline{n_i}$ que possui um descendente que satisfaz a condição associada ao nó N_i da *e-vtree*

4.3 Comparação da Linguagem VisTree com Linguagens de Consultas XML e com Autômatos de Árvore

4.3.1 Comparação da VisTree com a XQuery

Toda consulta VisTree possui uma consulta XQuery equivalente. Uma expressão *e-vtree* da VisTree é convertida em uma consulta XQuery que é então executada e retorna os padrões que satisfazem as condições impostas na consulta. A conversão é feita considerando que cada predicado possui uma ou mais funções associadas em XQuery. A tabela 4.2 mostra a relação entre predicados e funções XQuery. As funções assim como os predicados são ligadas pelo

uso dos conectivos lógicos: And, OR ou NOT. O caracter curinga é sempre associado a função *exists*.

Predicado	Função XQuery Equivalente
Contem(l)	*contains*+*name* para *tag*, e *contains* para conteúdo
Igual(l)	*compare*+*name*
Começapor(l)	*starts-with*+*name*
Terminacom(l)	*ends-with*+*name*
Menor(l)	$No \rightarrow label < l_1$ para valores inteiros, *compare* para *string*
Maior(l)	$No \rightarrow label > l_1$ para valores inteiros, *compare* para *string*
Intervalo(l_1, l_2)	$No \rightarrow label > l_1$ e $No \rightarrow label < l_2$ para valores inteiros, *compare* para *string*
Pertence($l_1, l_2, ..., l_k$)	*Exists*(l_1) OR *Exists*(l_2) OR ... *Exists*(l_k)

Tabela 4.3: Relação entre predicados Vistree e funções XQuery.

O algoritmo *ETree2XQuery*, representado pela Figura 4.4, faz a conversão de uma *e-vtree* para uma expressão *XQuery* baseando-se no conjunto FLOWR, que representa uma ordem básica de seleção utilizada pela XQuery.

Algoritmo *ETree2XQuery*: O primeiro procedimento recebe como entrada a raiz da *e-vtree*. O conjunto de predicados da expressão é representado pelo conjunto P, que armazenará todos os predicados da *e-vtree*. A consulta XQuery é montada através da concatenação de várias informações que irão formar uma *string* correspondente a expressão FLOWR da consulta. A expressão começa com o construtor de elemento <*result*>. Em seguida, inclui-se a cláusula *for*, que especifica em qual nó da árvore do documento a consulta será aplicada. A próxima claúsula a ser inserida é a cláusula *where some*, que testa a posição hierárquica e estrutural de cada um dos *labels*, aplicando os testes nas variáveis correspondentes a eles. O mapeamento das variáveis e o preenchimento do conjunto P são feitos de forma recursiva através do procedimento *ScanETree*, que varre a árvore, em pré-ordem, coletando informações de cada nó. A lista M representa um conjunto de variáveis XQuery, que estão relacionados ao índice

de cada nó e através do mesmo é feito o mapeamento das variáveis. Após todas as variáveis estarem mapeadas e relacionadas, é concatenada à expressão a clausula satisfies, que testará as variáveis com seus respectivos predicados. A cláusula *return*, que retornará o nó que satisfaz todas as condições da *e-vtree*, é a última a ser acrescentada à expressão, juntamente com a *tag* de fechamento *</result>*, que delimita cada elemente retornado.

Um exemplo de consulta expressa em VisTree e sua expressão FLOWR equivalente é mostrado na Figura 4.5. A consulta irá retornar árvores que contenham padrões que possuam uma raiz qualquer com um descendente $article$, que por sua vez possui um filho $authors$ que deve possuir ao menos dois filhos com o *label* $author$. A expressão FLOWR é então montada da seguinte forma: na cláusula *for* define um valor para a variável $root$ que indica que a busca deverá ser feita no documento *SigmodRecord.xml* em subárvores que estão um nível abaixo da *tag* raiz do documento. Cada um dos *labels* da *e-vtree* foi mapeado em uma variável (* $\rightarrow a$, *articles* $\rightarrow b$, *authors* $\rightarrow c$, etc) como pode ser visto na cláusula *where*. As condições da textite-vtree são testadas na cláusula *satisfies*. O caracter curinga é mapeado na função *exists* e o predicado *igual* dos outros nós da expressão na função *compare*.

4.3.2 Comparação da VisTree com o Autômato de árvore

Nem toda expressão da linguagem VisTree possui um autômato de árvore local equivalente. E, somente um subconjunto da linguagem pode ser usado para representar um autômato de árvore local. Como o alfabeto do autômato deve ser formado por *labels* bem definidos, expressões que possuam predicados como $comecaPor$, $terminaCom$, $maiorQue$, $menorQue$, etc; não podêm ser transformadas em um autômato de árvore. Esse fato restringe o uso a um único predicado da linguagem, o $igual$. Para representar os autômatos locais será então utilizada uma classe de expressões VisTree, que é um subconjunto da linguagem VisTree chamada VisTreeAutomata. A classe VisTreeAutomata é formada por expressões de árvore específicas denominadas *e-vtreeA* cuja sintaxe é definida como a seguir:

Definição 4.3.1 (Expressão *e-vtreeA*) Considere um alfabeto Σ_A, tal que $\Sigma_A \subset \Sigma$, composto dos seguintes subconjuntos:

- Um conjunto finito L_A, cujos elementos são chamados de *labels*.

Procedure ETree2XQuery($\mathcal{R}$)

1. P := $\emptyset$; ($P := p_1, ..., p_n$ / $p_i \in$ Predicados da e-vtree)
2. E := "<result>{for $root in doc(" + XMLDoc + ")/*/*";
3. E += "where some ";
4. *ScanETree*($\mathcal{R}$)
5. E += " satisfies " + p_1;
6. **For each** $p_i \in P$ - p_1 **do**
7. E += " and " + p_i;
8. E += "return $root}</result>";

Procedure ScanETree($\mathcal{R}$)

1. M := {variaveis XQuery};
2. $P := P \cup R \rightarrow$Predicate;
3. $EXPT$:= False;
4. **If** $\mathcal{R}\rightarrow$Index = 0 **Then**
5. E += $M_{R \rightarrow Index}$ + " in $root//*";
6. **Else begin**
7. E += ", " + $M_{R \rightarrow Index}$ + " in " + $M_{R \rightarrow Father \rightarrow Index}$ + $\mathcal{R}\rightarrow$Link + "*";
8. $F := \mathcal{R} \rightarrow$Father;
9. $Seja\ C$:= {$F \rightarrow$Childs} - $\mathcal{R}$; (C = $c1$ ... cn);
10. **If** $C \neq \emptyset$ **Then begin**
11. $EXPT$:= True;
12. E += " except(" + $M_{c1 \rightarrow Index}$;
13. **For each** $c_i \in C$ - c_1 **do**
14. E := " union " + $M_{c_i \rightarrow Index}$;
15. **If** $EXPT$ = True **Then**
16. EXP := ")";
17. **End If;**
18. **End Else;**
19. $Seja\ C$:= {$\mathcal{R} \rightarrow$Childs}; ($C = c_1 ... c_n$)
20. **If** $C \neq \emptyset$ **Then**
21. **For each** $c_i \in C$ **do**
22. *ScanETree(c_i)*;

Figura 4.4: Algoritmo de conversão da *e-vtree* para uma expressão XQuery

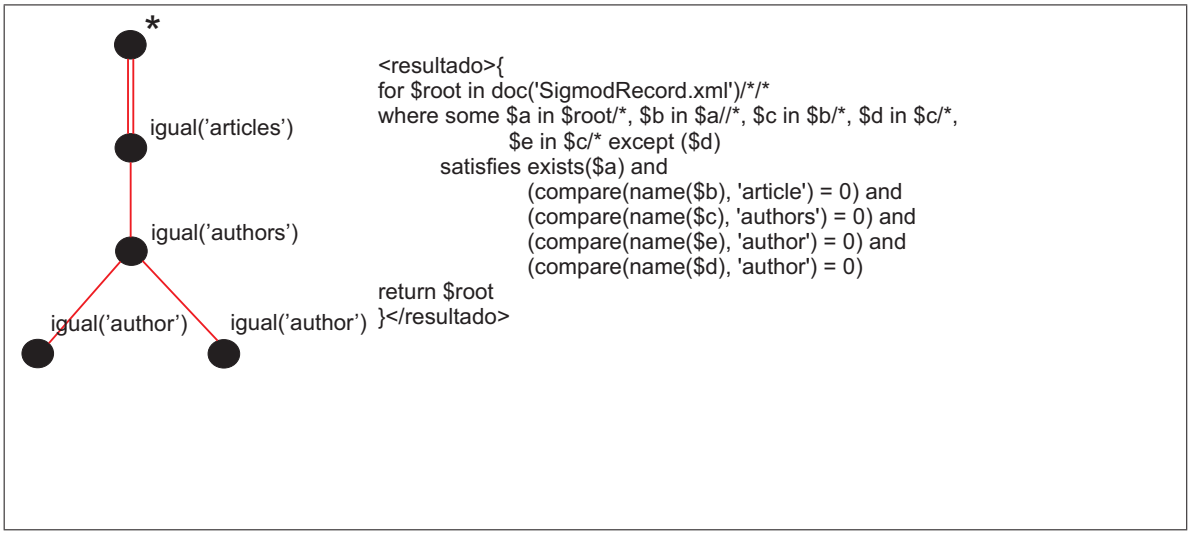

Figura 4.5: Exemplo de uma *e-vtree* e uma expressão XQuery equivalente.

- Um conjunto finito $\mathcal{N}_{\mathcal{A}}$, cujos elementos são chamados de *nós*.
- O símbolo lógico $\vee$.
- Os símbolos $/$ e $//$ chamados de *links*.
- O símbolo $*$, chamado de *curinga*, e um símbolo $\perp$ chamado *raiz*.

A conversão de uma *e-vtreeA* em um autômato de árvore é feita pelo algoritmo *ETree2Automata*, os procedimentos do algoritmo são apresentados nas figuras 4.6,4.7 e 4.8. O funcionamento geral do algoritmo é mostrado a seguir:

Algoritmo *ETree2Automata*: A conversão da *e-vtreeA* é feita em duas fases. Na primeira fase a *e-vtreeA* é convertida em um arquivo que contêm o alfabeto e as condições associadas a cada nó da expressão. Na segunda fase esse arquivo é convertido em um autômato de árvore. Os procedimentos *writeAlpha* e *writeRules* são usados na criação do arquivo de regras. O procedimento *writeAlpha* escreve no arquivo o alfabeto da *e-vtreeA* e *writeRules* escreve as regras associadas às condições de cada um dos nós da expressão. O procedimento varre de forma recursiva a expressão *e-vtreeA*, iniciando pela raiz. Para esse nó inicial é criada uma regra que leva o *label* raiz nas condições dos seus nós filhos. Para os nós filhos do nó raiz é gerada uma regra para cada um dos *labels* que fazem parte da condição do nó c_i, e cada *label*

dessa condição é associada as condições dos nós filho de c_i. Se o tipo de ligação entre o nó e seu filho for ancestral-descendente então é adicionado ainda a condição "$|x$". O procedimento *writeAutomata* monta um arquivo contendo o alfabeto, estados e funções de transição a partir do arquivo de regras.

Procedure $ETree2Automata(\mathcal{R})$

1. $\mathcal{L}$:= {labels da *e-vtreeA*}; ($\mathcal{L}$:= $l_1 ... l_n$);
2. $writeAlpha(\mathcal{L}, File_R)$;
3. $writeRules(R, File_R)$;
4. $writeAutomata(File_R, File_A)$;

Procedure writeAlpha($\mathcal{L}$,$File_R$)

1. $write($"A", $File_R)$;
2. **For each** $a_i \in \mathcal{L}$
3. $write($" "$+a_i, File_R)$;
4. $write(newline, File_R)$;

Figura 4.6: Algoritmo de conversão da *e-vtree* para um Autômato de Árvore

Exemplo 4.3.1 *Supondo uma e-vtree que possua como raiz o label* $book$*, e que esse nó tenha dois filhos. O filho da esquerda é o label* $title$ *e o da direita qualquer nó que possua um label que começa por "p". As funções de transição para um autômato equivalente a essa expressão são:*

$\delta(q_0, book) = q_1.q_2$;

$\delta(q_1, title) = \varepsilon$;

$\delta(q_2,' p...') = \varepsilon$;

Nota-se que não há um label bem definido para o estado q_2*, e que a função de transição para esse estado não é uma função de transição válida.*

A relação de descendência entre os nós pode ser representado no autômato pelo uso da recursividade. O exemplo dado a seguir mostra uma *e-vtree* que pertence a classe de expressões VisTree que podem ser usadas para definir um autômato. A expressão possui nó com relação de descendência. Note que o automâto possui regras recursivas.

Exemplo 4.3.2 *As funções de transição dadas a seguir representam as funções de transição de um autômato de árvore equivalente a e-vtree que possui uma tag $book$ como raiz, que por sua vez possui como descendente o label $author$. O label $author$ poderá ser um descendente direto ou indireto do label book; por essa razão o estado q_2 é recursivo.*

$\delta(q_0, book) = q_1 + q_2$;

$\delta(q_1, author) = \varepsilon$;

$\delta(q_2, x) = q_1 + q_2$;

4.3.3 Comparação da VisTree com outras Linguagens

Nessa sessão será feita uma comparação informal entre a linguagem VisTree e as linguagens XPath e Tree Pattern.

VisTree e XPath: Consultas expressas pela linguagem XPath podem retornar desde árvores inteiras até nós específicos; enquanto que consultas expressas na linguagem VisTree retornam sempre um padrão de árvore.

A Figura 4.9 mostra um exemplo de uma árvore de consulta. Em (a) tem-se o documento XML onde será aplicada a consulta, a árvore da consulta é mostrada em (b) e o resultado da consulta XPath para o documento pode ser visto em (c). Nota-se que para a consulta XPath foi retornado apenas os nós $title$ do documento, com seus respectivos conteúdos; uma consulta semelhante na linguagem VisTree retornaria uma árvore.

O único predicado da linguagem VisTree que possui equivalência em XPath para qualquer tipo de nó é o predicado $igual$. Outros predicados como $contem$, $maiorQue$, $menorQue$ e $intervalo$ possui equivalência na linguagem XPath apenas para conteúdos e não para *tags*. Os valores de *tags* passados nas consultas devem ser sempre valores exatos; e a linguagem não permite o uso de variáveis em suas consultas.

A principal diferença entre as linguagens está no fato de XPath retornar todos os nós a partir do último nó do caminho, como vimos no exemplo da Figura 4.9. O caminho da consulta em XPath também deve iniciar na raiz do documento; se fosse fornecido a *tag* $book$ como *tag* inicial do caminho a consulta não funcionaria. No exemplo da Figura 4.9, o último nó do caminho é $title$ e todos nós que começam por essa *tag* são retornados pela consulta XPath.

Para uma consulta expressa em VisTree toda a árvore do documento que satisfaz a consulta é retornada. O ponto a partir do qual os padrões serão retornados é informado durante a consulta. Considerando a mesma consulta feita em XPath, se a consulta em VisTree fosse feita a partir do nível 0, ou seja, a partir da *tag* $bookstore$, a consulta retornaria toda árvore que tem $bookstore$ como raiz. Caso a consulta fosse feita a partir do nível, iniciando na *tag* $book$ nada é retornado.

Vistree e TreePattern: A consulta genérica representada pelo *tree pattern* da Figura 4.10(a) irá verificar primeiro se a raiz do nó possui um *label* a, se ele não possuir um conjunto vazio é retornado; caso contrário retorna todos os descendentes que possuem um filho b com um descendente d, e um filho c.

A Figura 4.10(b) mostra um *tree pattern* que representa uma consulta onde as árvores retornadas deverão conter o nó *articles*, com um dos descedentes *article* que por sua vez terá como descendente *paragraph*, e como outro descendente também *article* que poderá aparecer várias vezes (um conjunto de nós *article*) e tem como descendente a *tag section*.

4.4 VisTree na Entrada da Restrição

Na entrada da restrição pressupõe-se que o usuário saiba representar visualmente o molde dos padrões que ele deseja minerar. O uso da restrição definirá que tipo de padrões serão produzidos durante a fase de geração. A linguagem VisTree possui os termos que poderam ser usados nessa representação, porém como visto na seção 4.3.2 nem todos os termos da linguagem poderam ser usados. A restrição deve ser expressa em termos de uma expressão árvore que possa ser convertida em um autômato de árvore local.

A expressão árvore(*e-vtree*) da restrição é criada iniciando-a com um nó raiz; para cada um dos nós a expressão do nó indica as condições que deverá satisfazer. A expressão de um nó da árvore de restrição não poderá conter outros predicados além do $Igual$, pois os outros predicados não podem ser representados por um autômato de árvore local. Uma expressão e de um nó da árvore de restrição será formada então pelas possibilidades de variação dos *labels* do nó. A ligação entre os nós, de acordo com a linguagem VisTree, poderá ser de filho ou descendente.

Na Figura 4.11 tem-se um exemplo de uma árvore de restrição e de alguns padrões que a satisfazem. A expressão referente a cada um dos nós da árvore de restrição aparece ao lado do nó; assim a expressão $e_1 = igual(A)$ esta relacionada a raiz, $e_2 = igual(*)$ ao nó filho à esquerda e $e_3 = igual(E) \vee igual(F)$ ao nó filho à direita. A expressão árvore será formada pelas expressões dos nós e pela ligação entre os nós.

4.5 VisTree na Entrada da Consulta

Enquanto a entrada da restrição será usada para produzir padrões de árvore específicos, a entrada da consulta será utilizada para a partir de padrões freqüentes resultantes de um processo de mineração filtrar padrões específicos. A entrada da consulta pode utilizar de todos os recursos da linguagem, pois toda expressão da linguagem VisTree possui uma expressão equivalente na linguagem XQuery. Dessa forma a entrada de uma consulta poderá fazer uso de todos os predicados, conectivos e demais funcionalidades da linguagem VisTree. A montagem de uma expressão árvore de consulta é semelhante a de uma expressão árvore de restrição, diferenciando apenas nas possibilidades de uso de recursos da linguagem em cada um dos casos, mas ambas produzirão árvores similares.

A *e-vtree* é definida inicialmente por um nó raiz, que representa também o nó raiz do padrão. São definidas ainda as condições que o nó deverá satisfazer, através do uso de um ou mais predicados ligados por conectivos. Quando o nó puder ser qualquer, é usado o caracter curinga $*$. O segundo passo é a inserção de links, para definir os filhos e descendentes. Para cada link inserido é obrigatório uma posterior inserção dos nós, juntamente com as condições que ele deverá satisfazer. A operação de inserção de links e nós se repete até que a consulta desejada esteja expressa na árvore de consulta.

Um exemplo de uma consulta expressa em VisTree e aplicada a um conjunto de padrões freqüentes pode ser visto em 4.12. A consulta busca por padrões que possuam qualquer *label* na raiz e que possua um descendente cujo *label* comece por “D” ou “F”. No conjunto de padrões freqüentes estão destacados os padrões que são retornados como resultado da consulta.

```
Procedure writeRules(R,File_R)
1.  If R->Index = 0 Then Begin
2.      write("R"+R→ label, File_R);
3.      If R→ child ≠ ∅ Then Begin
4.          If R→ child → index > 0 Then
5.              write(".", File_R);
6.              For each R→ child ∈ C do begin
7.                  If R→ child → link ="/" Then
8.                      write(R→ child → Condition, File_R);
9.                  Else
10.                     write(R→ child → Condition+"|x", File_R);
11.             End For
12.     End If
13.     Else
14.         write("−1", File_R);
15. End If
16. Else
17.     For each label l_i ∈ R do begin
18.         write("R"+l_i, File_R);
19.     If R→ child ≠ ∅ Then Begin
20.         If R→ child → index > 0 Then
21.             write(".", File_R);
22.             For each R→ child ∈ C do begin
23.                 If R→ child → link ="/" Then
24.                     write(R→ child → Condition, File_R);
25.                 Else
26.                     write(R→ child → Condition+"|x", File_R);
27.             End For
28.     End If
29.     Else
30.         write("−1", File_R);
31.     C = {R->Childs}; (C = c_1...c_n)
32.     If C ≠ ∅ Then
33.         For each c ∈ C do
34.             writeRules(c);
```

Figura 4.7: Procedimento que cria um arquivo de regras.

Procedure writeAutomata($File_R$,$File_A$)

1. $\mathcal{A}$:= {alfabeto do automato}; ($\mathcal{L}$:= $l_1...l_n$);
2. N := {codigo dos labels do alfabeto}; (N:= $n_1...n_k$);
3. R := {regras do automato};
4. $state := 0$; $state_l := 0$;$state_r := 1$;
5. $\mathcal{A}$:=$read(alphabet, File_R)$;
6. $write$(“A”, $File_A$);
7. **For each** $a_i \in \mathcal{A}$ **do begin**
8. mapAlfa(a_i, n_i);
9. $write$(“ ”+$n_i, File_A$);
10. **End For**
11. $write(newline, File_A)$;
12. $write$(“S”, $File_A$);
13. **For each** $Rule \in File_R$ **do begin**
14. $R+ = read(Rule, File_R)$;
15. $write(state, File_A)$; $state++$;
16. **End For**
17. $write(newline, File_A)$;
18. **For each** $rule \in R$ **do begin**
19. $splitRule(rule)$;
20. $label := rule_{left}$;
21. $write(state_l + map[label], File_A)$; $state++$;
22. **If** $rule_{right} \neq$“-1” **Then**
23. $write(mapR(rule_{right}, state_R), File_A)$;
24. **Else**
25. $write$(“-1”, $File_A$);
26. $write(newline, File_A)$;
28. **End For**

Figura 4.8: Procedimento que cria um arquivo contendo o autômato de árvore a partir de um arquivo de regras.

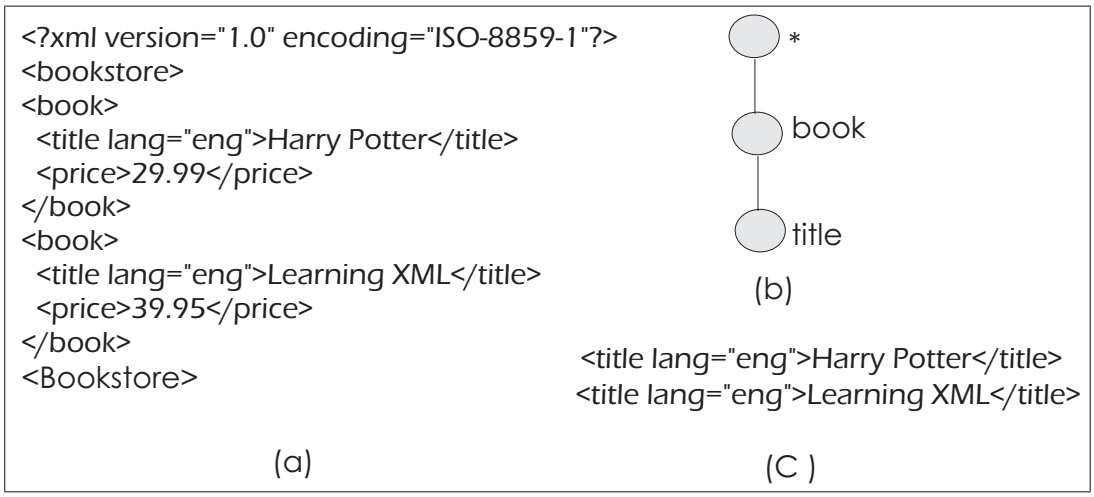

Figura 4.9: (a) Documento XML; (b) árvore de consulta; (c) Resultado da consulta XPath.

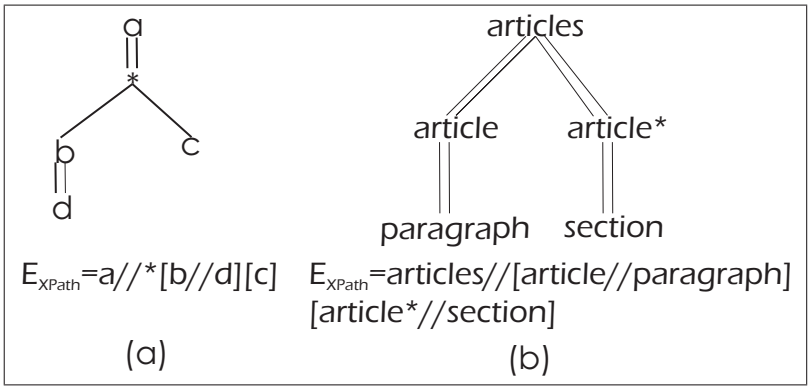

Figura 4.10: (a) Tree Pattern de uma consulta genérica e expressão XPath equivalente; (b) Tree Pattern de uma consulta real e expressão XPath equivalente.

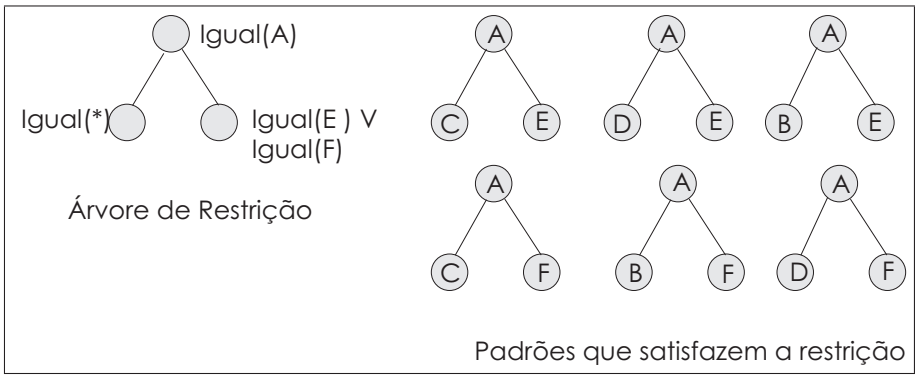

Figura 4.11: Exemplo de uma Árvore de Restrição; Padrões aceitos pela restrição.

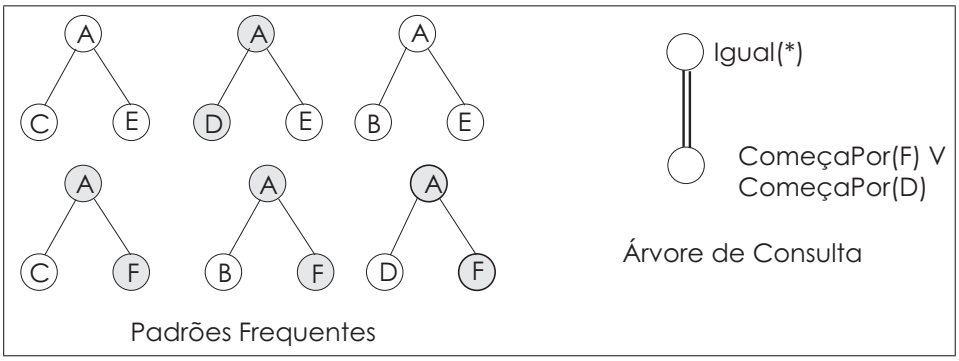

Figura 4.12: Exemplo de Conjunto de Padrões freqüentes; Exemplo de Árvore de Consulta.

Capítulo 5

O Sistema CobMiner

5.1 Introdução

O sistema CobMiner é um sistema de mineração de padrões arborescentes com restrições. O nome CobMiner(*Constraint based Miner*) vem do algoritmo de mineração com restrição utilizado pelo sistema e que motivou a criação do mesmo [de Amo et al. 2007]. Como finalidade da criação deste sistema tem-se a possibilidade de aplicação do algoritmo CobMiner em dados reais. Para isso, o sistema CobMiner é formado por módulos que atuam nas fases de pré e pós-processamento, se encarregando da preparação dos dados para os algoritmos de mineração de árvores e da recuperação de padrões. Os módulos que compõem o sistema CobMiner são: Módulo da Interface, Módulo de Entrada de Restrições, Módulo de Pré-Processamento de Dados, Módulo de Mineração e Módulo de Análise de Padrões.

Mais do que uma simples aplicação do algoritmo em dados reais o sistema oferece uma interface amigável para entrada de dados. A entrada da restrição e consulta é feita de uma forma intuitiva através do uso da linguagem VisTree, possibilitando a validação do uso dessa linguagem para esse propósito. Em cada um dos módulos foram definidos e implementados algoritmos para efetuarem as atividades propostas. A arquitetura do sistema formada por módulos permite de uma maneira simples, a integração de novas aplicações.

Nesse capítulo serão apresentadas as características do sistema CobMiner. Começando pela Arquitetura do Sistema na seção 5.2 e em seguida são detalhados cada um dos módulos. A seção 5.4 tratará do módulo de entrada da restrição, e a seção 5.5 do funcionamento do módulo de pré-processamento. O módulo de mineração de árvores, que faz uso do algoritmo de mineração CobMiner, será descrito na seção 5.6, e o módulo de análise de padrões na seção 5.7. A ultima seção, seção 5.3, mostrará o módulo de interface.

5.2 Arquitetura do Sistema

A arquitetura do sistema CobMiner é constituída pelos módulos do sistema conforme apresentado na figura 5.1 e 5.2. Note que, nessas duas imagens a estrutura de módulos é a mesma, alterando apenas o fluxo de informação entre os módulos. A ordem em que os módulos aparecem indica a interação entre eles. Segundo a funcionalidade de cada um dos módulos, eles são classificados em:

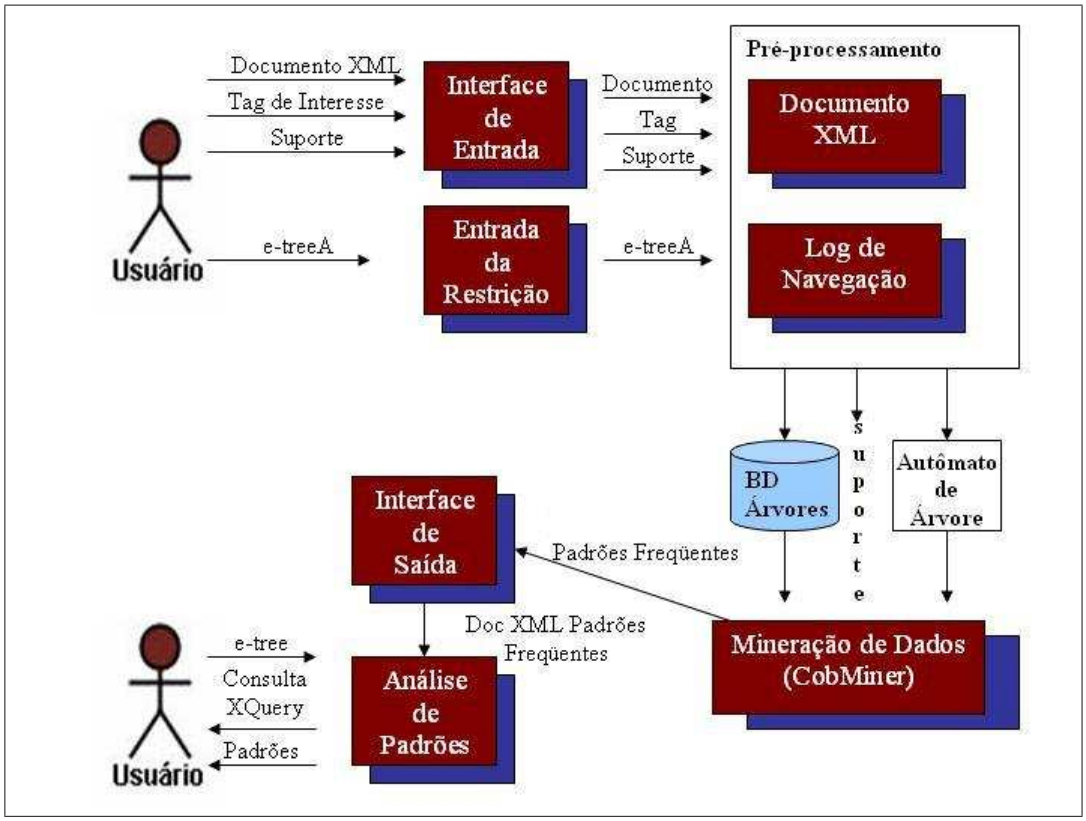

Figura 5.1: Arquitetura sistema para Documentos XML

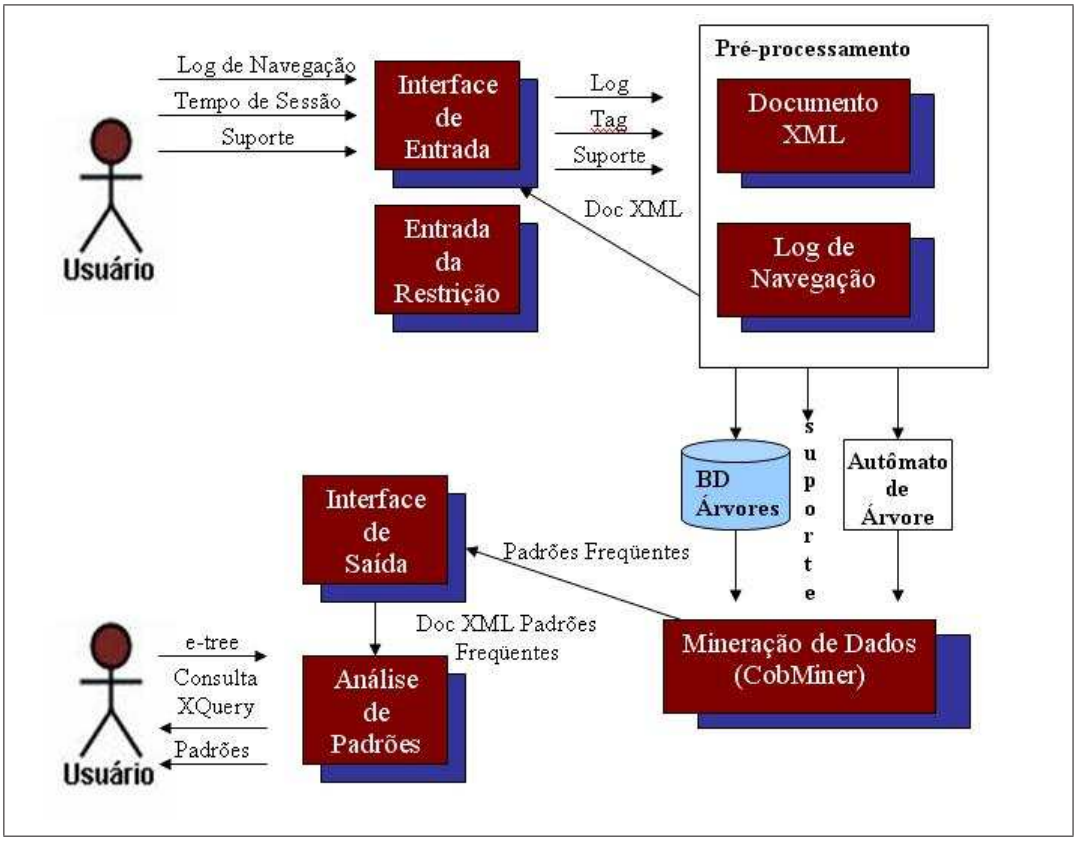

Figura 5.2: Arquitetura Sistema para *Logs* de Navegação

- Módulo de Interface de Entrada (adaptada para os dois estudos de caso propostos):

  Consiste em uma interface que permite a especificação das tarefas de mineração com ajustes das medidas de interesse (suporte), escolha dos tipos de dados a serem minerados (documentos XML, logs), carregamento dos dados a serem minerados, especificação do autômato de árvore permitindo ao usuário guiar o processo de mineração.

- Módulo de Interface de Saída: Permite a visualização e análise dos padrões minerados, além da apresentação de gráficos estatísticos diversos.

- Módulo de Entrada das Restrições: A entrada das restrições utiliza a linguagem visual VisTree e tem como objetivo permitir ao usuário a especificação de restrições dos padrões que serão minerados na fase de mineração. As restrições entradas pelo usuário através da linguagem VisTree são transformadas em um autômato de árvore que será utilizado como dado de entrada na fase de mineração.

- Módulo de Pré-Processamento dos Dados: Os dados que serão minerados são submetidos a pré-processamentos específicos para cada tipo de dado. Nesta dissertação tratamos dois estudos de caso específicos onde a mineração de padrões arborescentes com restrições pode ser utilizada. Para cada um destes dois estudos de casos foram implementados algoritmos de pré-processamento dos dados.

  Pré-processamento de documentos XML: Tem como objetivo transformar documentos XML em uma base de dados de árvores. As árvores dessa base de dados são representadas por um formato de string esperado como entrada do módulo de mineração.

  Pré-processamento de logs: Tem como objetivo selecionar informações contidas em *logs* de navegação e transformar tais informações em árvores de acesso que irão compor a base de dados. Nesse caso, as árvores também serão representadas por strings.

- Módulo de Mineração: Utiliza o Algoritmo CobMiner de mineração de padrões arborescentes com restrições semânticas nos padrões especificados por autômatos de árvore. O projeto e implementação do CobMiner é objetivo de uma outra dissertação de mestrado [Silva 2007]. O algoritmo CobMiner recebe como entrada um conjunto de árvores representadas através de strings, um autômato de árvore e limite minimo de suporte e retorna o conjunto de todos os padrões arborescentes freqüentes com relação a este limite mínimo de suporte e que satisfazem o autômato de árvore dado.

- Módulo de Análise dos Padrões: Utiliza a linguagem visual VisTree como mecanismo de consultas dos padrões minerados.

5.3 Módulo de Interface

A criação de uma interface tem como objetivo facilitar o uso do sistema por parte de usuários. A interface visa abstrair as complexidades envolvidas na execução dos algoritmos, permitindo que o usuário interaja de uma forma mais intuitiva com o sistema. Através da Interface são fornecidos os parâmetros de entrada do algoritmo de mineração de árvores para aplicação em dados reais, e são feitas também as análises sobre os dados de saída; que serão os padrões de árvore freqüentes.

O módulo de interface pode ser dividido em duas partes; uma delas é a interface de entrada dos dados e a outra a interface de saída dos dados. No módulo de entrada temos a escolha do tipo do documento que será minerado, a escolha de qual documento será minerado, especificação da restrições e do suporte. O módulo de saída tem por função a visualização e análise dos padrões minerados, sendo que a análise será feita por recuperação de padrões através de consultas.

5.3.1 Interface de Entrada de Dados

A interface está preparada para dois tipos de aplicações: *Logs* de Navegação Web e Documentos XML; mas outras aplicações podem ser inseridas com pequenos ajustes na interface. As atividades do módulo de interface de entrada foram representadas através de um diagrama de atividades UML, que pode ser visto na figura 5.3.

Os documentos XML já se encontram em um formato de árvore, então as atividades a serem efetuadas nesse caso são: fornecer o documento XML, a *tag* de interesse, restrição, algoritmo de mineração e o suporte. O documento XML é representado por uma árvore e será dividido de acordo com uma *tag* de interesse em subárvores. Essas subárvores possuem como raiz a *tag* de interesse e irão formar a base de dados usada no processo de mineração. Antes de fazer a escolha da *tag* de interesse, deverá ser informado o nível hierárquico em que a *tag* aparece na árvore do documento XML. Por convenção a *tag* raiz do documento é considerada como sendo o nível zero, as *tags* que se encontram um nível abaixo a ela nível um, e assim por diante.

Conforme o nível escolhido é apresentanda uma listagem com as *tags*, para a escolha da *tag* de interesse. A *tag* de interesse é passada para o módulo de pré-processamento de Documentos XML, que fará a representação das subárvores no formato esperado pelos algoritmos de mineração. Na figura 5.4 é mostrado um exemplo de um arquivo XML. Supondo que o nível escolhido seja dois, e a *tag article* seja a de interesse. Nesse documento há quatro subárvores que possuem *article* como *tag* raiz, a base de dados será composta por essas subárvores. A figura 5.5 mostra separadamente as quatro subárvores que formarão a base de dados.

Os *logs* de navegação por sua vez têm que ser transformados para se obter as árvores de navegação de usuários. O resultado dessa transformação será um documento XML contendo as árvores de acesso dos usuários, que comporão o documento de entrada. Os outros dados de entrada serão a restrição, o algoritmo de mineração e o suporte. A interface de entrada para

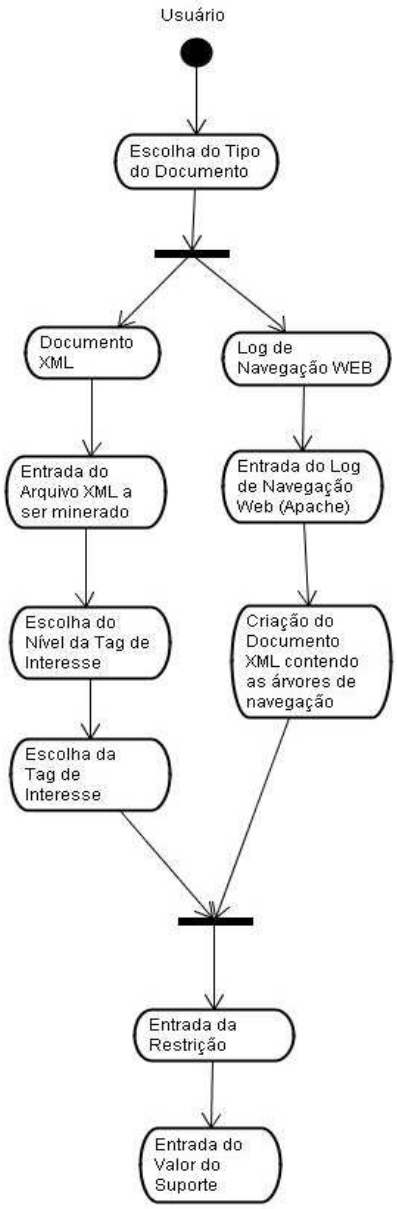

Figura 5.3: Diagrama de atividades da Interface de Entrada

os *logs* de navegação Web difere na aparência apenas por não haver nesse caso, a necessidade de fornecer uma *tag* de interesse. As árvores de navegação são delimitadas pela mesma *tag* <*user_session*>.

A entrada da restrição, que será melhor detalhada na seção 3.4, é feita de acordo com a linguagem VisTree e será representada visualmente na interface de acordo com as opções fornecidas pelo usuário. Não há distinção na maneira como é feita a entrada da restrição para os dois

```
<?xml version="1.0" encoding="UTF-8"?>
<SigmodRecord>
<issue>
<volume>11</volume>
<number>1</number>
<articles>
<article>
  <title>Annotated Bibliography on Data Design.</title>
  <keyword>data</keyword>
  <initPage>45</initPage>
  <endPage>77</endPage>
  <authors>
    <author position="00">Anthony I. Wasserman</author>
    <author position="01">Karen Botnich</author>
  </authors>
</article>
<article>
  <title>Architecture of Future Data Base Systems.</title>
  <keyword>data</keyword>
  <initPage>30</initPage>
  <endPage>44</endPage>
  <authors>
    <author position="00">Lawrence A. Rowe</author>
    <author position="01">Michael Stonebraker</author>
  </authors>
</article>
<article>
  <title>Database Directions III Workshop Review.</title>
  <keyword>database</keyword>
  <initPage>8</initPage>
  <endPage>8</endPage>
  <authors>
    <author position="00">Tom Cook</author>
  </authors>
</article>
<article>
  <title>Errors in 'Process Synchronization in Database Systems'.</title>
  <keyword>database</keyword>
  <initPage>9</initPage>
  <endPage>29</endPage>
  <authors>
    <author position="00">Philip A. Bernstein</author>
    <author position="01">Marco A. Casanova</author>
    <author position="02">Nathan Goodman</author>
  </authors>
</article>
</articles>
</issue>
</SigmodRecord>
```

Figura 5.4: Exemplo de um Documento XML

estudos de caso propostos. A figura 5.6 mostra a interface de entrada do sistema CobMiner, à esquerda da tela está o documento XML, no centro a árvore de restrição expressa na linguagem VisTree, e à direita as opções de inserção da árvore de restrição.

5.3.2 Interface de Saída de Dados

A interface de saída de dados recebe como entrada o arquivo de resultado, contendo os padrões freqüentes. Esse arquivo será um documento XML, o que possibilita a visualização dos padrões freqüentes pelos usuários de uma forma mais estruturada e o que permitirá a recuperação de padrões usando consultas XML. A interface mostrará o arquivo XML e fornecerá os recursos

Árvore 1

```
<article>
 <title>Annotated Bibliography on Data Design.</title>
 <keyword>data</keyword>
 <initPage>45</initPage>
 <endPage>77</endPage>
 <authors>
  <author position="00">Anthony I. Wasserman</author>
  <author position="01">Karen Botnich</author>
 </authors>
</article>
```

Árvore 2

```
<article>
 <title>Architecture of Future Data Base Systems.</title>
 <keyword>data</keyword>
 <initPage>30</initPage>
 <endPage>44</endPage>
 <authors>
  <author position="00">Lawrence A. Rowe</author>
  <author position="01">Michael Stonebraker</author>
 </authors>
</article>
```

Árvore 3

```
<article>
 <title>Database Directions III Workshop Review.</title>
 <keyword>database</keyword>
 <initPage>8</initPage>
 <endPage>8</endPage>
 <authors>
  <author position="00">Tom Cook</author>
 </authors>
</article>
```

Árvore 4

```
<article>
 <title>Errors in 'Process Synchronization in Database Systems'.</title>
 <keyword>database</keyword>
 <initPage>9</initPage>
 <endPage>29</endPage>
 <authors>
  <author position="00">Philip A. Bernstein</author>
  <author position="01">Marco A. Casanova</author>
  <author position="02">Nathan Goodman</author>
 </authors>
</article>
```

Figura 5.5: Base de Dados

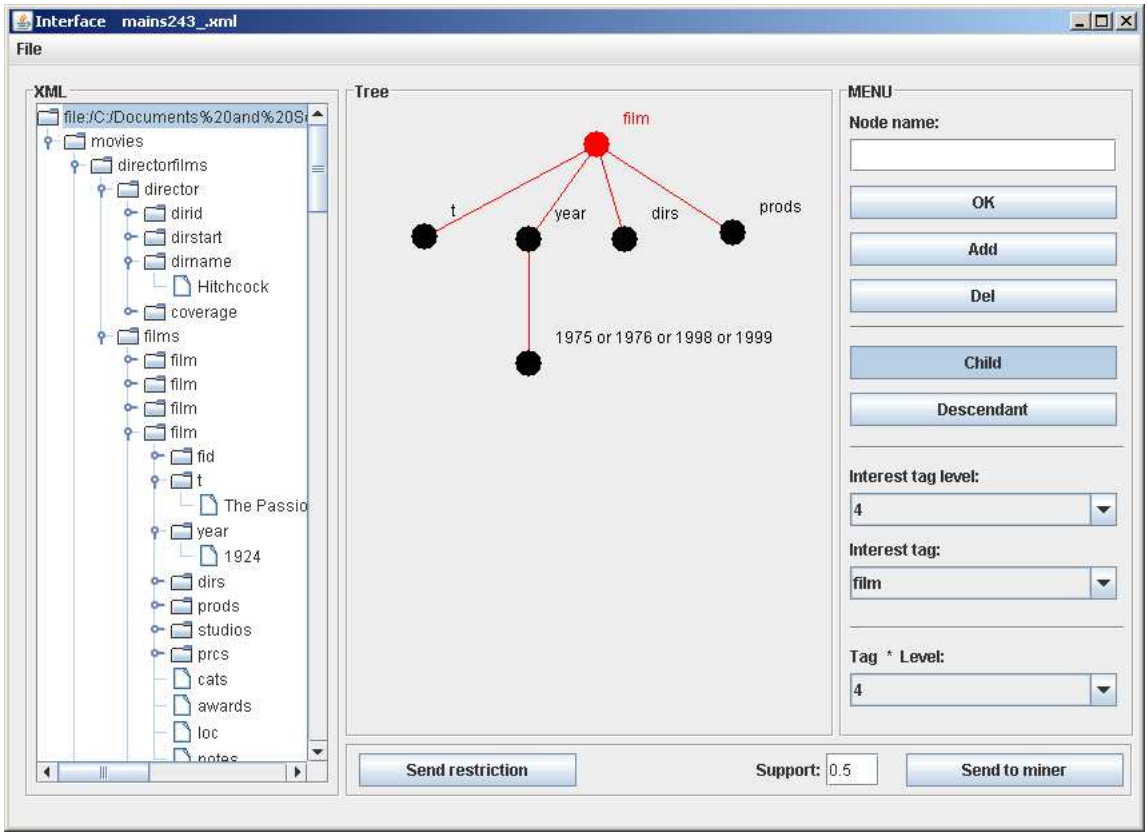

Figura 5.6: Interface de Entrada dos dados

necessários para a entrada da consulta (módulo de análise de padrões).

Para a entrada da árvore de consulta expressa na linguagem VisTree, a interface possui

uma área onde a árvore de consulta é desenhada. Nessa interface é possível especificar todas as características da *e-tree* que representa a consulta. A interface possui um painel com os predicados da linguagem, a posição do nó e o tipo de relacionamento entre os nós. O usuário monta a consulta e a envia para o módulo de análise de padrões.

A interface de saída de dados pode ser vista na figura 5.7. No centro da tela tem-se a árvore de consulta do usuário; montada segundo as opções do painel à direita. Nesse exemplo, o usuário está interessado em minerar padrões freqüentes que possuam uma raiz qualquer, tenha um descendente $article$ que por sua vez possui como filhos um nó que comece com ti e termine com tle e o outro com o nome $author$. A árvore de consulta é passada para o módulo de análise de padrões, onde será convertida em uma consulta XQuery.

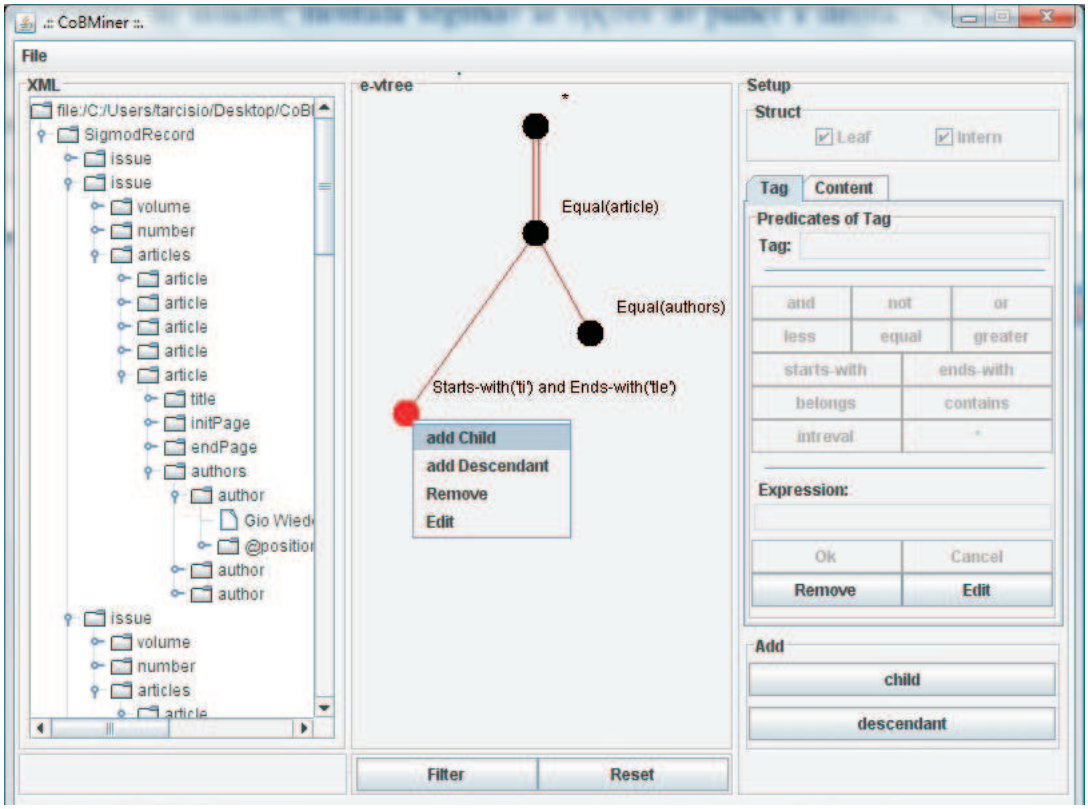

Figura 5.7: Interface de Saída dos dados

5.4 Módulo de Entrada de Restrições

O módulo de entrada de restrições fornecerá ao usuário, pelo uso da linguagem VisTree, a possibilidade de expressar visualmente uma restrição. Esse módulo se utiliza do módulo de interface para a interação com o usuário. A forma de entrada da restrição será através da *expressão árvore e-tree* da linguagem. O usuário fornecerá a estrutura e conteúdo dos nós da árvore de restrição. O módulo de pré-processamento receberá essa restrição em um arquivo montado a partir da *e-tree* correspondente à restrição, a partir desse arquivo o automâto de árvore local correspondente é criado.

O primeiro nó da *e-tree* será a raiz que deverá ser única. Após a inserção da raiz há a possibilidade de inserção de nós filhos e para cada um desses nós haverá uma expressão associada contendo as condições que os nós deverão satisfazer.

O arquivo criado pelo módulo de entrada de restrição conterá os *labels* e a relação entre eles na mesma forma em que aparecem na *e-tree*. A relação entre os *labels* é expressa de uma maneira semelhante ao conceito de DTD (*Document Type Definition*), que é usada para representar a estrutura de documentos XML. Uma DTD especifica, para cada tipo de objeto, a sequência permitida de tipos desse objeto. As DTDs podem especificar também informações tais como atributos ou conteúdo especial para cada tipo [Papakonstantinou and Vianu 2000]. Elas correspondem a gramáticas de árvore local [Murata et al. 2005], onde declarações de tipos elementos correspondem a regras de produção. Os autômatos de árvore local, usados pelo algoritmo CobMiner também correspondem a gramáticas de árvore local.

A proposta inicial era representar a restrição como uma DTD e depois converter a DTD em um autômato de árvore local. A conversão de DTD para autômato é feita considerando que cada declaração de tipo corresponde a uma regra de produção, o que torna essa conversão simples. Como não é possível descrever conteúdo através de uma DTD, essa idéia foi abandonada. Optou-se por fazer a conversão da restrição para um arquivo que representasse as características da *e-tree*, e desse arquivo para o automâto de árvore local. Entretanto, o conceito de que cada *tag* corresponde a uma regra de produção que foi usada.

5.5 Módulo de Pré-processamento

O módulo de pré-processamento tem por finalidade transformar os dados reais no formato esperado pelo algoritmo CobMiner. Como em outros algoritmos de mineração, no CobMiner os dados são representados de uma forma simplificada. A base de dados de árvore é um arquivo onde cada linha contêm o identificador da árvore, seu tamanho, e a string que representa a árvore. O autômato de árvore é esperado também em um arquivo onde a primeira linha contêm os *labels* do alfabeto, a segunda os estados e a partir da terceira linha são representadas as funções de transição.

O propósito geral do módulo de processamento seria então, a partir dos dados reais, que incluem o documento que será minerado e a restrição fornecida pelo usuário transformá-los e criar os arquivos de entrada do algoritmo CobMiner. O pré-processamento para as duas aplicações nesse ponto funcionam praticamente da mesma forma, a diferença está no fato de que em Documentos XML a base de dados é quebrada pela *tag* de interesse e nos *Logs* de Navegação as árvores começam sempre abaixo da *tag* <*user_session*>. A aplicação em *Logs* de Navegação Web possui ainda um pré-processamento adicional que é a obtenção das árvores de navegação dos usuários tendo como entrada um arquivo de *Log* do Servidor Web Apache; esse procedimento é efetuado durante a entrada de dados e retorna para a interface um arquivo XML com as árvores de navegação. É através deste arquivo que os dados de entrada são fornecidos.

O funcionamento do módulo de pré-processamento geral poderia ser representado segundo a ordem das tarefas da seguinte forma:

1. Cria a árvore DOM(*Document Object Model*) do documento XML.

2. Faz a codificação de *labels* mapeando os em códigos númericos.

3. Monta a base de dados no formato esperado pelo algoritmo CobMiner e armazena-a em um arquivo.

4. Monta o autômato de árvore no formato esperado pelo algoritmo CobMiner e armazena-o em um arquivo.

5.5.1 Pré-processamento de Documentos XML

O pré-processamento de um Documento XML irá extrair do arquivo as subárvores que formaram a base de dados a ser minerada e converterá a restrição de entrada em um autômato de árvore. Para formar a base de dados, o documento XML é representado como sendo uma única árvore, e a partir de uma *tag* de interesse t, são extraídas as subárvores que possuem como raiz t. Os *labels* dos nós são codificados em números naturais e as subárvores são transformadas em *strings*.

Um exemplo contendo parte de um documento XML, duas subárvores do documento e a base de dados no formato esperado pelo algoritmo CobMiner pode ser visto na figura 5.8. A *tag* de interesse nesse caso é *article*, o documento possui duas subárvores que tem como raiz essa *tag*, t_1 e t_2. O mapeamento dos *labels* em nós é feito da seguinte forma: *article* $\rightarrow$ 7, *title* $\rightarrow$ 8, *Architectural ...* $\rightarrow$ 18, *InitPage* $\rightarrow$ 10, etc.

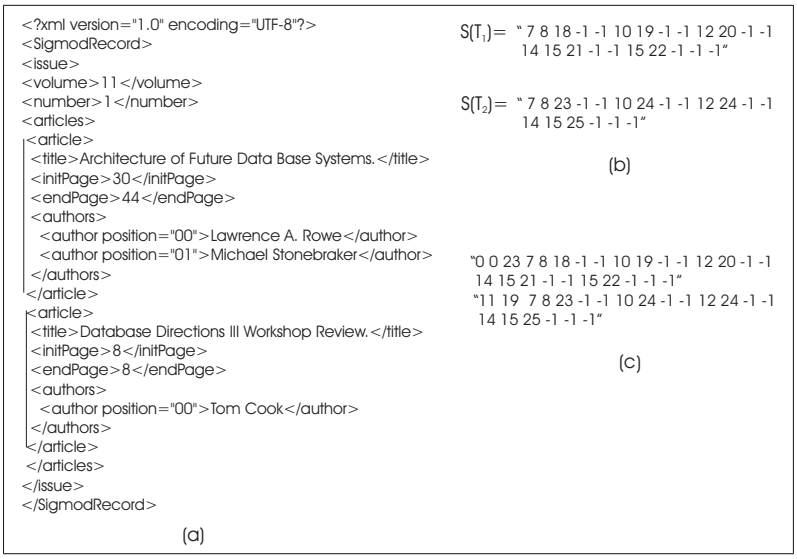

Figura 5.8: (a)Documento XML;(b)Subárvores do documento;(c)Base de dados;

O autômato é obtido a partir da entrada da restrição feita através da interface. O módulo de pré-processamento recebe um arquivo com as informações da restrição, onde consta o alfabeto e regras de produção, com os *labels* ainda não codificados. O módulo de pré-processamento faz então a codificação dos *labels* (obedecendo a mesma codificação das subárvores), e a criação de um arquivo contendo o alfabeto, estados e regras de produção, no formato esperado pelos algoritmos de mineração.

O autômato de árvore local dado a seguir é um exemplo de restrição para o documento XML da figura 5.8, que indica que o usuário está interessado apenas em minerar padrões onde apareçam o autor “Jefrey D.Ullman” ou “Michael Stonebreaker”. O arquivo gerado pelo módulo de interface, e o arquivo do automâto correspondente são mostrados na figura 5.9.

$\delta(q_0, article) = q_1q_2q_3q_4$; $\delta(q_1, title) = \varepsilon$;

$\delta(q_2, initPage) = \varepsilon$; $\delta(q_3, endPage) = \varepsilon$;

$\delta(q_4, authors) = q_5$; $\delta(q_5, author) = (q_6 + q_7)$;

$\delta(q_6, Jeffrey\ D.\ Ullman) = \varepsilon$; $\delta(q_7, Michael\ Stonebraker) = \varepsilon$;.

A1 article|title|initPage	A 7 8 10 12 14 15 102 176
A2 endPage|authors|author	S 0 1 2 3 4 5
A3 Jefrey D. Ullman|Michael Stonebraker	R 0 7 (1.2.3.4)
R article(title,initPage,endPage,authors)	R 18 -1
R title [-1]	R 2 10 -1
R initPage [-1]	R 12 -1
R endPage [-1]	R 14 15
R authors [author]	R 15 (102.176)
R author [Jeffrey D. Ullman|Michael Stonebraker]	R 102 -1
	R 176 -1
(a)	(b)

Figura 5.9: (a)Arquivo da restrição;(b)Arquivo do Autômato;

5.5.2 Pré-processamento de *Logs* de Navegação Web

No contexto da mineração do uso da Web, a fase de pré-processamento dos dados é um pouco mais complexa. Podemos considerar que os *logs* de navegação Web passam por dois tipos de pré-processamento; o primeiro parte de um arquivo de *log* do servidor Web Apache e monta um arquivo XML contendo as árvores de navegação do usuário.; o outro cria a base de dados e o autômato de árvore para o algoritmo CobMiner, de uma maneira semelhante ao módulo de pré-processamento de documentos XML. A primeira parte do pré-processamento de *logs* consiste nas seguintes tarefas:

1. Limpeza do dados
2. Identificação das Sessões de Usuário
3. Categorização
4. Transformação

A limpeza de dados faz a retirada de entradas irrelevantes do *log* de navegação, como entrada para imagens (.gif,.jpg,etc), folhas de estilo (.css). São consideradas relevantes as entradas que representem um acesso a uma página do website. A segunda tarefa é a identificação de uma sessão de usuário; considera-se que um usuário é identificado pelo IP e que os usuários do *website* não são validados por nenhum mecanismo de login. As entradas do *log* de navegação são ordenadas pelo número de IP e pela data de acesso. Cada sessão é identificada pelo tempo de navegação do usuário, o tempo máximo estabelecido para ser considerado uma mesma sessão foi 30 minutos.

A terceira tarefa é a categorização, onde cada página acessada é mapeada em uma categoria. O objetivo dessa tarefa seria simplificar a estrutura das árvores e gerar padrões mais fáceis de serem interpretados. As informações utilizadas na categorização deverão estar armazenadas em uma tabela, na forma de um documento XML fornecido como dado de entrada. A tabela contêm todos os endereços do *website* na forma em que poderão aparecer no arquivo de *log* de acesso; para cada página existe uma categoria relacionada.

O último passo consiste em transformar os acessos dos usuário em árvores. O algoritmo utilizado para efetuar essa tarefa foi baseado no algoritmo apresentado em [Ivancsy and Vajk 2006]. A árvore de acesso correspondente a sessão do usuário é construída da seguinte maneira: a primeira entrada do *log* corresponde à raiz da árvore. As próximas entradas são inseridas seguindo o percurso em profundidade da árvore até que uma entrada se repita ou termine a sessão do usuário. Caso ocorra a repetição, a próxima entrada do *log* é inserida como filho da primeira ocorrência dessa entrada na árvore.

A figura 5.10 ilustra as 4 fases de pré-processamento dos *logs* de acesso; na primeira parte da figura(a) é mostrada parte de um log de acesso Web com 5 entradas, que são de uma mesma sessão de usuário, pois o tempo de navegação não ultrapassa os 30 minutos. Em (b) uma tabela ilustra a relação entre endereço e categoria, exemplificando o mapeamento que seria feito para as páginas acessadas que aparecem no arquivo de *log* (a). Na figura 5.10(c) é possível ver a árvore de acesso para a sessão de usuário em questão .

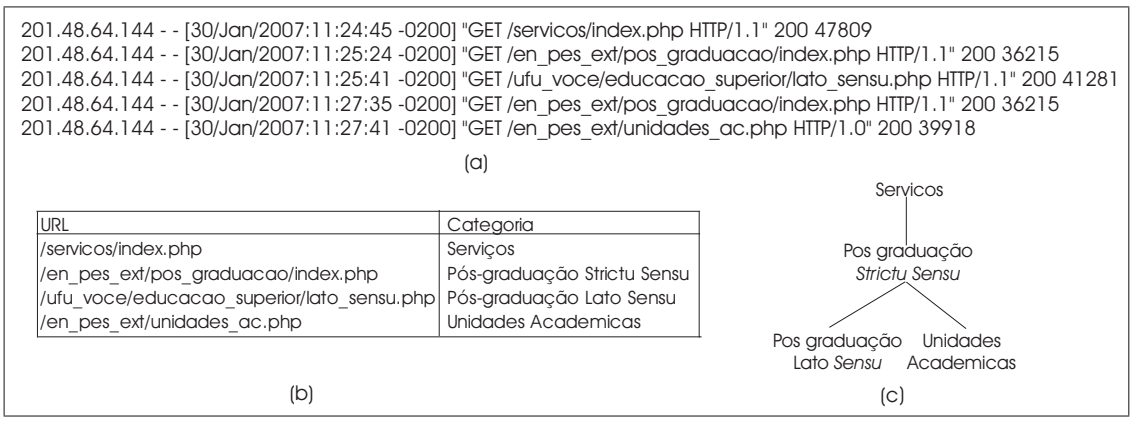

Figura 5.10: (a) Parte de um arquivo de log de acesso; (b)Uma tabela de categorias; (c) Uma arvore de acesso

5.6 Módulo de Mineração

O módulo de mineração recebe os dados de entrada e faz a chamada de um do algoritmo de mineração de árvores CobMiner. Os dados de entrada dos algoritmos serão a base de dados de árvore, onde cada árvore é representada por uma string; o autômato de árvore que especifica a restrição e o valor do suporte. A saída do algoritmo é armazenada em um arquivo que conterá os padrões freqüentes, representados pelos *labels* codificados e no formato *strings*. Esse módulo também tem por função receber o resultado da execução do algoritmo e armazenar os padrões de árvore freqüentes em um documento XML.

Para a criação do documento XML de resultado, é necessário fazer a decodificação dos *labels* e a transformação dos padrões em um formato XML. A decodificação dos *labels* é feita através de uma estrutura criada previamente pelo módulo de pré-processamento, onde cada código é mapeado em um *label*. A criação do documento XML do resultado é feita utilizando essa estrutura. É definido primeiro um nó para o documento, e para cada linha do arquivo de resultado, que representa cada um dos padrões freqüentes, é construída uma *tag* $< padrao >$. Essa *tag* delimitará a árvore do padrão, que será construída associando uma *tag* para cada *label* e respeitando a hierarquia dos *labels* no padrão.

A figura 5.11 exemplifica uma transformação de um resultado do algoritmo de mineração em um documento XML. Na figura 5.11(a) pode-se ver os padrões freqüentes resultantes. Os padrões são armazenados na ordem em que são calculados e retornados pelo algoritmo. Cada

linha do arquivo representa um padrão que se encontra no formato de uma *string*. Os mesmos padrões estão representados no formato XML na figura 5.11(b), onde o documento começa com uma *tag* raiz, denominada $< root >$ e coloca cada um dos padrões, já com os *labels* decodificados, entre a *tag* $< padrao >$.

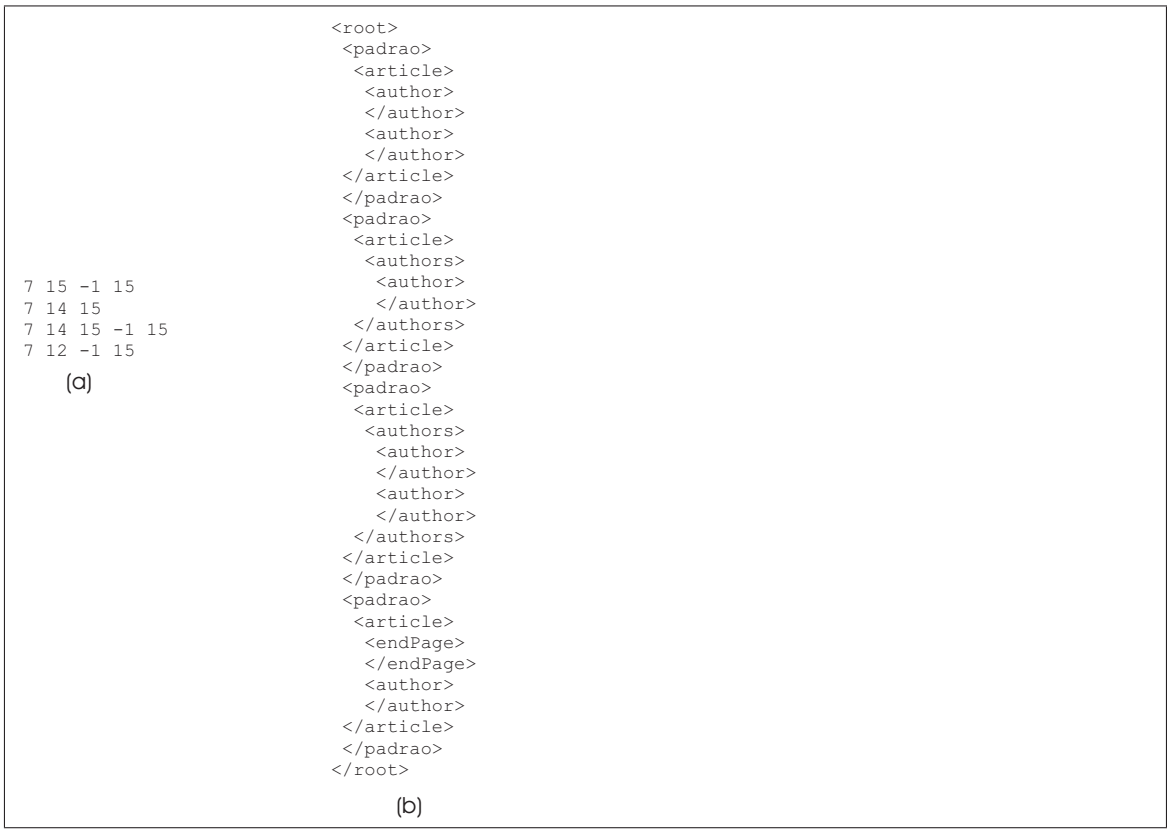

Figura 5.11: Base de Dados

5.7 Módulo de Análise dos Padrões

No capítulo 2 foi descrita a fase de análise de padrões em um processo de mineração. Essa fase é usada para fazer a interpretação e a recuperação de padrões. Para as duas aplicações propostas (Documentos XML e *logs* de Navegação Web), considera-se que a interpretação de padrões não seria necessária, pois na maioria de Documentos XML os dados possuem *tags* com nomes auto-descritivos e para os *logs* de Navegação a categorização das páginas contribui para que sejam produzidos padrões mais fáceis de serem interpretados. O foco desse trabalho para a

fase de análise de padrões é a recuperação de padrões, que visa permitir ao usuário a seleção de padrões específicos entre os padrões freqüentes retornados pelo processo de mineração.

As consultas aos padrões freqüentes devem ser expressas visualmente pela linguagem VisTree. A interface de saída de dados disponibiliza os recursos da linguagem (predicados, conectivos, links, etc) para que o usuário desenhe a árvore de consulta. A árvore de consulta será uma *e-tree* (*expressão árvore*) da linguagem VisTree e será o molde que os padrões deverão obedecer para serem retornados como resultado da consulta. A definição da árvore de consulta é semelhante a de árvore de restrição apresentada na seção 3.4. A diferença está no fato de que a árvore de restrição será transformada em um autômato de árvore local e deve obedecer às regras de definição dessa estrutura. Já a árvore de consulta será convertida em uma consulta XQuery, e o que se tem como regra nesse caso é que toda árvore de consulta tem que possuir uma consulta XQuery equivalente.

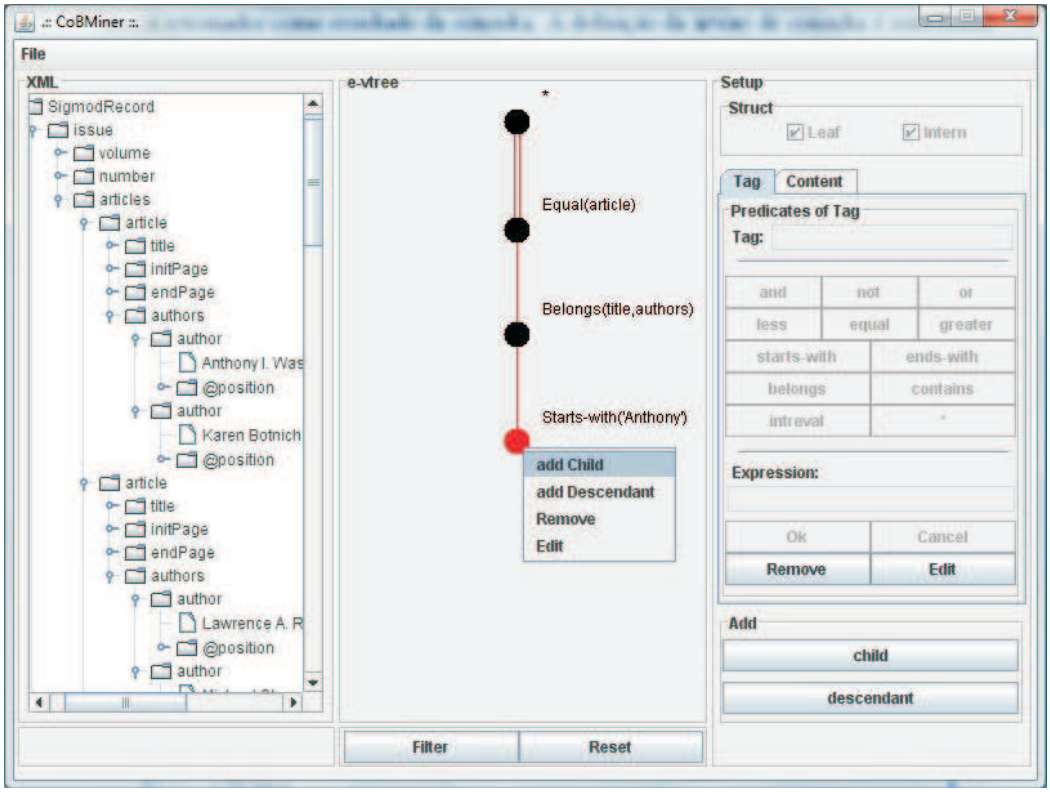

Figura 5.12: Exemplo de uma consulta que será processada pelo Módulo de Análise de Padrões

A criação da árvore de consulta parte da raiz, para a qual são definidas as condições sobre o nó. As condições para um nó são representadas por uma expressão da linguagem. Após definir a

raiz, outros nós poderão ser adicionados à árvore de consulta. O próximo passo após a inserção da raiz é então a inserção do tipo de ligação entre a raiz e o novo nó (filho ou descendente). O novo nó é acrescentado à árvore depois que uma expressão para o nó é fornecida. O processo se repete para cada um dos nós até que a árvore de consulta esteja pronta. Para que uma consulta possa ser enviada, a *e-tree* deve estar completa, ou seja, não podem haver *links* não conectados a um nó, ou nós sem uma condição definida. A árvore de consulta é então convertida em uma expressão FLOWR da linguagem XQuery.

A conversão em uma expressão FLOWR é feita segundo um algoritmo. Pela interface de saída é possível visualizar a expressão FLOWR equivalente a árvore de consulta, e os padrões do resultado. A figura 5.12 mostra a interface de saída do sistema com uma consulta ao arquivo XML *SigmodRecord.xml*. A consulta é montada com as opções do painel à direita da interface. O módulo de análise de padrões monta um arquivo XML com os padrões resultantes da consulta. Na construção do arquivo ele constrói a *tag* $< resultado >$ que irá delimitar os padrões resultado. Na figura 5.13 é possível ver parte do documento XML com o resultado da consulta da figura 5.12.

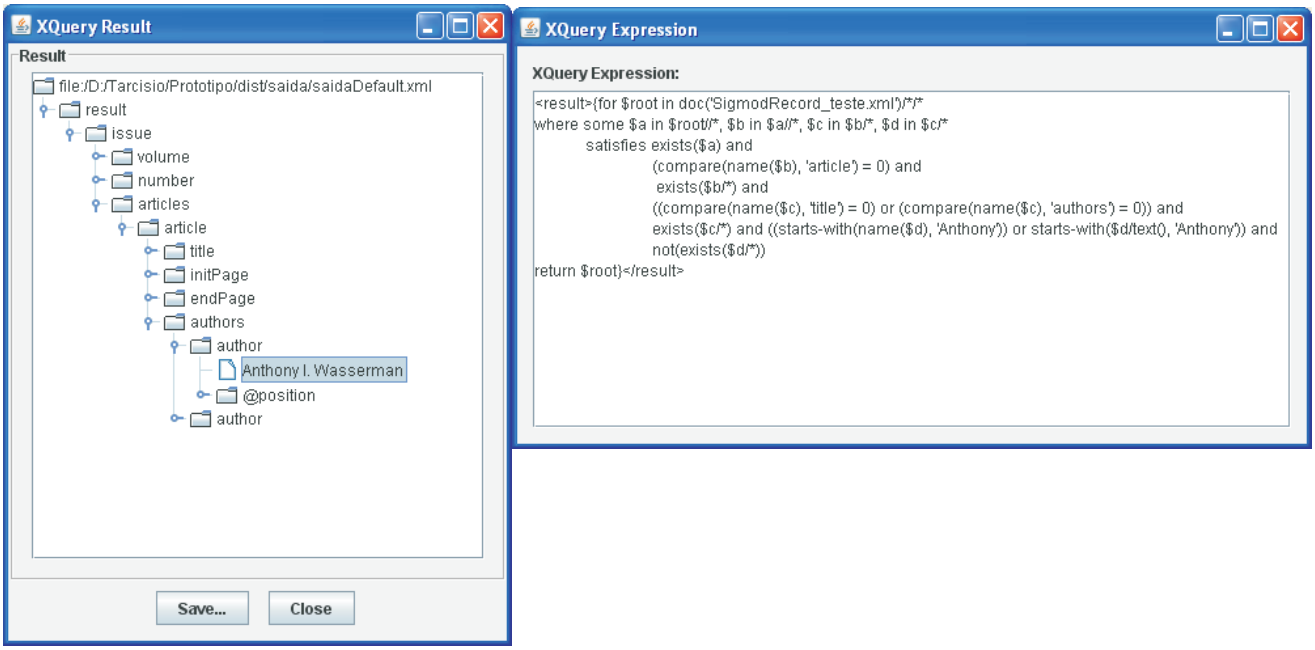

Figura 5.13: Resultado da Consulta; Expressão XQuery da Consulta expressa na figura 5.12

Capítulo 6

Estudos de Caso

6.1 Introdução

A forma de validar e registrar alguns resultados obtidos com o trabalho desenvolvido foi através de dois estudos de caso aplicados à área de *XML Mining* e *Web Mining*. Neste capítulo serão apresentados testes que foram realizados no sistema proposto, utilizando documentos XML da base de dados de filmes IMDB e *logs* de acesso do site da Universidade Federal de Uberlândia. O sistema minerou padrões de árvore freqüentes dos documentos XML contendo estruturas e conteúdos que satisfizessem as especificações do usuário, e as árvores de acesso freqüentes no website.

6.2 Aplicação em *XML Mining*

Uma das aplicações propostas foi a mineração de subárvores freqüentes de um documento XML visando descobrir estruturas e conteúdos que ocorrem com uma determinada freqüência e obdecem à restrição fornecida pelo usuário. Os documentos XML usados no estudo de caso foram obtidos da base de dados *IMDB*[1], que é considerada a maior base de dados de filmes da Internet. Ela contêm informações sobre filmes, atores, diretores, roteristas, estúdios, entre outras. Foram escolhidos 3 documentos XML dessa base de dados, com o tamanho variando de 3479 a 48937 árvores. Os arquivos escolhidos foram:

[1] http://infolab.stanford.edu/pub/movies/dtd.html.

- people55.xml: Lista de 3479 pessoas famosas em filmes;
- mains243.xml: Lista principal de 12114 filmes;
- casts124.xml: Lista de artistas que atuaram em determinado filme, com 48937 entradas; mostrando os atores e seus papéis em 9000 filmes e 2700 diretores;

Para cada um dos 3 arquivos foram definidas as *tags* de interesse e a árvore de restrição. O arquivo *mains243.xml* foi dividido em 12114 subárvores, que possuem como raiz a tag <film>, formando a base de dados DB-Mains. A base de dados DB-People será formada pelas 3479 subárvores do documento *people55.xml* que possuem como raiz a tag <person>. O arquivo *casts124.xml* por sua vez irá formar a base de dados DB-Casts com 48937 subárvores do documento, e com tag raiz igual a <m>.

Na base de dados DB-People, a mineração deverá retornar apenas padrões que contenham como informação de uma pessoa o nome, código da função, nome de família, apelido, data de nascimento, país de origem e colegas (pessoas que trabalharam com eles no mesmo filme). A restrição sobre o conteúdo é feita quanto ao nome do colega que deverá ser um dentre os quatro nomes: Hitchcock, Mirta Ibarra, Elizabeth Montegomery ou Bertolucci. O autômato A-People validará as árvores que satisfazem essa restrição:

Autômato de Árvore A-People $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}\}$

$\Sigma =$ {person, pname, pcode,familynm, givennm, dob, background, rels, workedwith, colleague, name, Hitchcock, Mirta Ibarra, Elizabeth Montegomery, Bertolucci}

$\delta(q_0, person) = q_1.q_2.q_3.q_4.q_5.q_6.q_7$;

$\delta(q_1, pname) = \varepsilon$;

$\delta(q_2, pcode) = \varepsilon$;

$\delta(q_3, familynm) = \varepsilon$;

$\delta(q_4, givennm) = \varepsilon$;

$\delta(q_5, dob) = \varepsilon$;

$\delta(q_6, background) = \varepsilon$;

$\delta(q_7, rels) = q_8$

$\delta(q_8, workedwith) = q_9$;

$\delta(q_9, colleague) = q_{10}$;

$\delta(q_{10}, name) = q_{11}$;

$\delta(q_{11}, Hitchcock) = \varepsilon$;

$\delta(q_{11}, MirtaIbarra) = \varepsilon$;

$\delta(q_{11}, ElizabethMontegomery) = \varepsilon$;

$\delta(q_{11}, Bertolucci) = \varepsilon$;

A restrição sobre a base de dados DB-Mains, restringe a mineração a padrões que possuam em sua estrutura as seguintes informações: titulo, diretor, produtor, processo usado para fazer o filme (facultativo) e ano. O conteúdo da *tag* "ano"também é restringido pelo ano de produção dos filmes. O interesse é em filmes dos anos de 1975, 1976, 1998 ou 1999. A presença do conteúdo no entanto é facultativa. O autômato que representa essa restrição é o automâto A-Mains dado a seguir:

Autômato de Árvore A-Mains $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

$\Sigma =$\{film, t, year, dirs, prods, prcs, 1975, 1976, 1998, 1999\}

$\delta(q_0, film) = q_1.q_2.q_3.q_4.q_5*$;

$\delta(q_1, t) = \varepsilon$;

$\delta(q_2, year) = q_6*$;

$\delta(q_3, dirs) = \varepsilon$;

$\delta(q_4, prods) = \varepsilon$;

$\delta(q_5, prcs) = \varepsilon$;

$\delta(q_6, 1975) = \varepsilon$;

$\delta(q_6, 1976) = \varepsilon$;

$\delta(q_6, 1998) = \varepsilon$;

$\delta(q_6, 1999) = \varepsilon$;

Para a base de dados DB-Casts, a restrição produzirá padrões onde aparecem as seguintes informações: artista membro, identificador do filme, título do filme, nome artístico, tipo do papel, descrição curta do papel do artista no filme, nome do ator. Restrições sobre o conteúdo são feitas considerando apenas os artistas cujo tipo de papel é indefinido (*und*). O autômato que molda os padrões para essa restrição é o A-Casts:

Autômato de Árvore A-Casts $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

$\Sigma =$ {m, f, t,a, p, r, n,und}

$\delta(q_0, m) = (q_1.q_2.q_3.q_4.q_5.q_6)$;

$\delta(q_1, f) = \varepsilon$;

$\delta(q_2, t) = \varepsilon$

$\delta(q_3, a) = \varepsilon$;

$\delta(q_4, p) = q_7$;

$\delta(q_5, r) = \varepsilon$;

$\delta(q_6, n) = \varepsilon$;

$\delta(q_7, und) = \varepsilon$;

6.2.1 Análise dos Resultados

Nos documentos XML, como os dados são semiestruturados, a base de dados contendo suas subárvores apresentam estruturas semelhantes. Como há um alto grau de semelhança entre as estruturas das árvores, se a mineração for feita considerando apenas a estrutura dos padrões, padrões freqüentes são retornados mesmo se utilizado um valor maior de suporte. Já a repetição de conteúdos de uma subárvore para outra irá depender da informação armazenada no documento. Os conteúdos das *tags* são valores que repetem com uma freqüência menor, e dependendo da informação que ele está representando um conteúdo poderá ocorrer uma única vez em toda base de dados.

Nesse trabalho foi proposta a mineração de documentos XML levando em conta a estrutura dos padrões e os conteúdos das *tags*. Dessa forma, o usuário pode especificar não só as *tags* de seu interesse e a relação hierárquica entre elas, mas também fazer restrições quanto ao conteúdo que elas devem apresentar. O estudo de caso levou em consideração o fato de que um conteúdo que representa uma informação que ocorre pouco na base de dados não tem chances de ser freqüente. As restrições para cada uma das bases de dados foram especificadas restringindo somente conteúdos de *tags* que repetem em uma freqüência maior.

Alguns padrões freqüentes encontrados podem ser vistos na figura 6.1. O primeiro padrão,

figura 6.1(a), é um padrão freqüente e válido, segundo à restrição, obtido da mineração da base de dados *BD-People* com o uso do automato *A-People*. Nessa base de dados, $1,5\%$ das pessoas possuem como informações o nome, código da função, nome de família, apelido, data de nascimento, país de origem e tiveram como colega de trabalho Hitchcock.

O segundo padrão, figura 6.1(b), foi obtido da mineração da base de dados *BD-Mains* com o uso do autômato *A-Mains*. O padrão apresentado aparece em $2,2\%$ das árvores da base de dados. O padrão indica que das informações dos filmes armazenadas no arquivo, os filmes que possuem as *tags* título, diretor, produtor e ano, e que foram produzidos no ano de 1998, possuem uma freqüência de $2,2\%$.

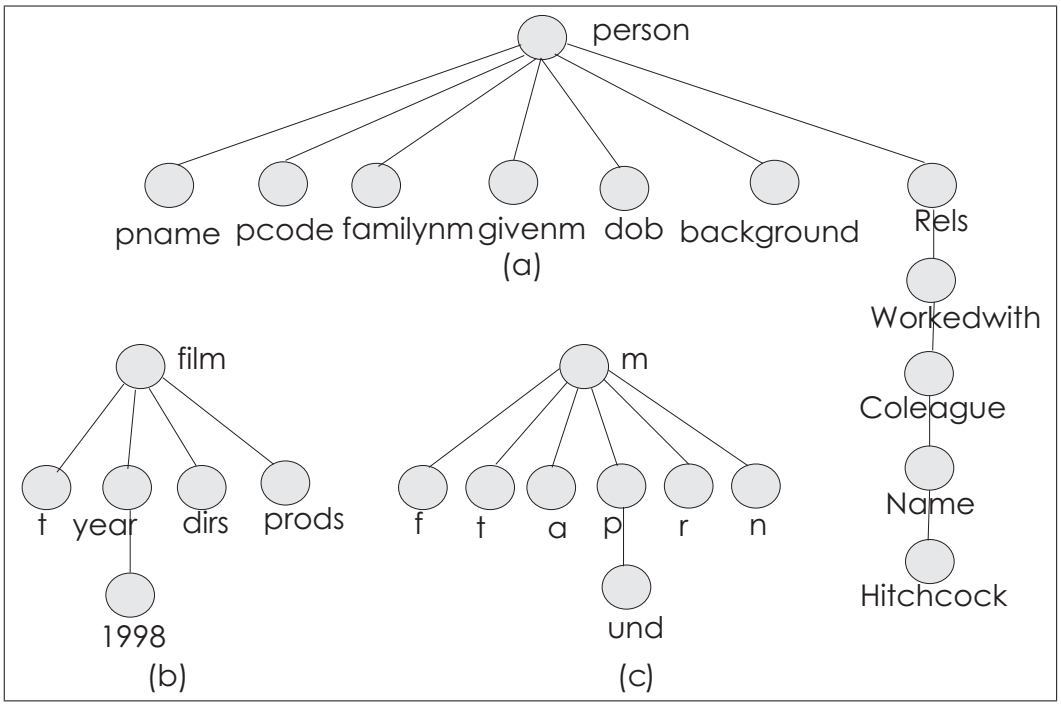

Figura 6.1: (a)Padrão freqüente da base de dados *BD-People*;(b)Padrão freqüente da base de dados *BD-Mains*;(c)Padrão freqüente da base de dados *BD-Casts*

O último padrão, figura 6.1(c), é um padrão freqüente da base de dados *BD-Casts*, minerado segundo o autômato *A-Casts*. As informações contidas no padrão são: artista membro, identificador do filme, título do filme, nome artístico, tipo do papel (com conteúdo igual a indefinido), descrição curta do papel do artista no filme, nome do ator. Os resultados demonstraram que artistas que possuem as informações listadas anteriormente e um papel indefinido, ocorrem com uma freqüência de $16,6\%$ nas árvores da base de dados.

6.3 Aplicação em *Web Mining*

A aplicação desenvolvida para *Web Mining*, teve como objetivo testar o sistema proposto na Mineração do Uso da Web. Nesse contexto, foram mineradas as árvores de acesso dos usuários de um *website*, utilizando restrições como forma de direcionar o tipo de padrão de árvore retornado pelo processo de mineração. Os *logs* de navegação foram fornecidos pela Universidade Federal de Uberlândia[2], e correspondem aos acessos desse *website* no mês de julho de 2007, no período entre 3/07/2007 à 22/07/2007.

Os arquivos de *logs* de acesso, usados no estudo de caso, são arquivos no formato do servidor Web Apache. Os *logs* fornecidos para esse estudo encontravam-se agrupados por dia, ou seja, um arquivo para cada dia de acesso. A primeira tarefa para a preparação dos dados foi a concatenação dos *logs* em arquivos que representassem os acessos ao *website* da UFU por um período de uma semana. Posteriormente, foram feitas as tarefas de pré-processamento para a transformação dos *logs* em árvores de acessos dos usuários, armazenando-as em um documento XML. Desse arquivo XML foram obtidas as bases de dados no formato esperado pelos algoritmos de mineração. As bases de dados que formaram o estudo de caso foram:

- DB-UFULogA: Base de dados com 4582 árvores, formada pelos acesso dos usuários feitos no período entre 3/07/2007 à 9/07/2007;
- DB-UFULogB: Base de dados com 5765 árvores,formada pelos acesso dos usuários feitos no período entre 10/07/2007 à 16/07/2007;
- DB-UFULogC: Base de dados com 4028 árvores, formada pelos acesso dos usuários feitos no período entre 17/07/2007 à 22/07/2007;

Para a mineração com restrição dessas bases de dados foram definidos 4 autômatos, um para cada base de dados e um autômato geral que foi aplicado em todas as bases de dados. O primeiro autômato corresponde ao autômato aplicado especificamente na base de dados *DB-UFULogA*. De acordo com esse autômato, são considerados interessantes no processo de mineração somente os padrões que contêm as sessões de usuário começando pela página “Pesquisa e Pos-graduação” , que seguem para página “Pos-graduação Strictu Sensu”, dessa para a página

[2]http://www.ufu.br.

"Apresentação Mestrado em Educação" e acessa qualquer outra página na sequência, depois retorna 3 níveis até a página "Pesquisa e Pos-graduação" e acessa qualquer outra página; ou vice-versa. O autômato *A-UFULogA*, corresponde à representação dessa restrição:

Autômato de Árvore A-UFULogA $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma =$ {Pesquisa e Pos-Graduacao, Pos-Graduacao Strictu Sensu, Apresentacao Mestrado em Educacao, {P}}

onde:$\{P\} = Conjunto\ de\ todas\ as\ paginas\ do\ Website$

$\delta(q_0, Pesquisa\ e\ Pos - Graduacao) = (q_1.q_2)|(q_2.q_1);$

$\delta(q_1, Pos - Graduacao\ Strictu\ Sensu) = q_3;$

$\delta(q_2, x) = \varepsilon$

$\delta(q_3, Apresentacao\ Mestrado\ em\ Educacao) = q_4;$

$\delta(q_4, x) = \varepsilon;$

O autômato da base de dados *DB-UFULogB*, restringe os padrões à acessos feitos à paginas do portal de notícias da UFU. O acesso do usuário deverá iniciar pela página "Noticias Gerais". O próximo acesso poderá ser feito tanto a página de notícias sobre "Concursos" quanto de "Cursos", retornando à página "Noticias Gerais" e seguindo ou para a página de notícias da "Pos-graduação" ou para a de"Educação Básica". No autômato *A-UFULogB* dado a seguir, tem-se a restrição descrita anteriormente:

Autômato de Árvore A-UFULogB $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma =$ {Noticias Gerais, Concursos, Cursos, Pos-Graduacao, Educacao Basica}

$\delta(q_0, Noticias\ Gerais) = (q_1|q_2).(q_3|q_4);$

$\delta(q_1, Concursos) = \varepsilon;$

$\delta(q_2, Cursos) = \varepsilon$

$\delta(q_3, Pos - Graduacao) = \varepsilon;$

$\delta(q_4, Educacao\ Basica) = \varepsilon;$

Na base de dados *DB-UFULogC* a mineração dos acessos foi feita restringindo-os a padrões

que começam pela página “UFU e Você”, seguindo para a página “Vida Acadêmica” e depois para “Calendário Acadêmico”. Da página “Calendário Acadêmico”, o usuário poderá retornar dois níveis até a página “UFU e Você” e depois acessar uma das páginas: “Serviços UFU”, “Mapa Portal” ou “Busca Avançada”, ou encerrar a navegação. O autômato *A-UFULogC* representa essa restrição:

Autômato de Árvore A-UFULogC $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$\Sigma =$ {UFU e Voce,Vida Academica, Busca Avancada, Calendario Academico, Servicos UFU, Mapa Portal}

$\delta(q_0, UFU\ e\ Voce) = q_1.(q_2|q_3|q_4)*$;

$\delta(q_1, Vida\ Academica) = q_5$;

$\delta(q_2, Servicos\ UFU) = \varepsilon$

$\delta(q_3, Mapa\ Portal) = \varepsilon$;

$\delta(q_4, Busca\ Avancada) = \varepsilon$;

$\delta(q_5, Calendario\ Academico) = \varepsilon$;

Além dos autômatos mostrados anteriormente, foi definido ainda um autômato geral que foi testado em todas as três bases de dados. O autômato *A-UFUGeral*, corresponde à restrição que se caracteriza por minerar somente padrões de acessos que se iniciam na página “Pesquisa e Pos-graduação”, segue para a página “Pos-graduação Lato Sensu” e depois segue ou não (facultativo, poderá não ocorrer ou ocorrer uma ou várias vezes) para uma outra página qualquer do *website*; depois de chegar nesse ponto, retorna até a página “Pesquisa e Pos-graduação” e acessa qualquer outra página do *website*. A representação formal do autômato *A-UFUGeral* é:

Autômato de Árvore A-UFUGeral $= (Q, q_0, \Sigma, \delta)$

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma =$ {Pesquisa e Pos-Graduacao, Pos-Graduacao Lato Sensu, {P}}

onde:$\{P\} = Conjunto\ de\ todas\ as\ paginas\ do\ Website$

$\delta(q_0, Pesquisa\ e\ Pos - Graduacao) = q_1.q_2$;

$\delta(q_1, Pos - Graduacao\ Lato\ Sensu) = q_3*$;

$\delta(q_2, x) = \varepsilon$

$\delta(q_3, x) = \varepsilon$

6.3.1 Análise dos Resultados

As bases de dados com as árvores de acesso caracterizam-se por apresentar pouca semelhança estrutural entre si, ao contrário das bases de dados dos documentos XML. Essa diferença se deve ao fato de que a forma de acesso de cada usuário é variável, podendo ocorrer situações onde dois usuários acessaram o mesmo conjunto de páginas porém em ordem diferente, o que produzirá árvores de acesso diferentes. Na árvore de acesso não há portanto uma posição hieráquica fixa de um *label* (página acessada). Ele poderá ocorrer em diferentes níveis em árvores de acesso diferentes. Por essa razão, a estrutura dos padrões de árvore de acesso se repetem com uma freqüência bem menor que a estrutura dos padrões de Documentos XML, e foram minerados com valores de suporte menores do que os aplicados no estudo de caso em *XML Mining*.

Uma outra diferença entre as características dos dados nos dois estudos de caso está no conjunto de *labels* das bases de dados. No estudo de caso aplicado em *XML Mining*, cada base de dados possui um conjunto de *labels* diferentes. Em *Web Mining* todas as bases de dados são formadas por acessos feitos a um mesmo *website*, havendo grande semelhança entre o conjunto de *labels* de uma base para outra. Essa semelhança possibilitou o uso de uma restrição geral, que pode ser aplicada nas três bases de dados analisadas no estudo.

Na figura 6.2 são mostrados quatro exemplos de padrões freqüentes encontrados nas bases de dados analisadas no estudo de caso. O padrão da figura 6.2(a), é um padrão freqüente e válido da base de dados *BD-UFULogA*, segundo o autômato *A-UFULogA*. A árvore de acesso indica que nesse período foram freqüentes acessos que iniciaram na página "Pesquisa e Pos-graduação", seguiram para a página de informações sobre cursos de mestrado ("Pos-graduação Strictu Sensu") e depois para a página de informações do Mestrado em Educação ("Apresentação Mestrado em Educação") acessando em seguida a página de disciplinas desse mesmo mestrado ("Disciplinas Mestrado em Educação") e retornando até a página inicial para acessar então a página de informações sobre os cursos de especialização ("Pos-graduação Lato Sensu").

O segundo padrão, figura 6.2(b), foi minerado na base de dados *BD-UFULogB* com a aplicação da restrição representada pelo autômato *A-UFULogB*. A restrição limitou a mineração

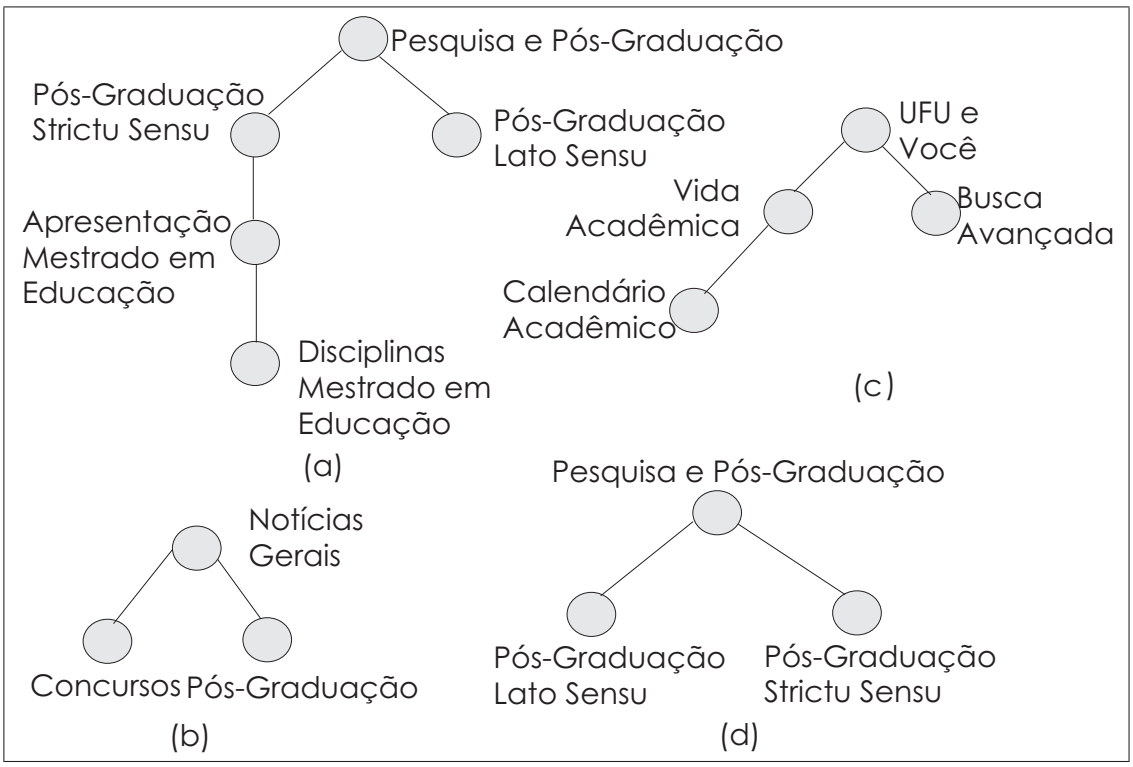

Figura 6.2: (a)Padrão freqüente da base de dados *BD-UFULogA*;(b)Padrão freqüente da base de dados *BD-UFULogB*;(c)Padrão freqüente da base de dados *BD-UFULogC*; (d)Padrão freqüente nas 3 bases de dados

a padrões de acesso do portal de notícias do *website*. O padrão minerado demonstra que no período de tempo analisado, um dos acessos freqüentes nesse contexto foi o que começava pela página de notícias gerais ("Noticias Gerais"), seguia para a página de notícias sobre concursos ("Concursos"), retornava a página de notícias gerais e finalizava o acesso com a página de notícias sobre os cursos de pós-graduação da instituição ("Pos-graduação").

A figura 6.2(c) mostra um padrão freqüente da terceira base de dados *BD-UFULogB*. Esse padrão foi gerado de acordo com à restrição fornecida pelo autômato *A-UFULogC*. A árvore de acesso representada pelo padrão mostra ser freqüentes acessos que partem da página "UFU e Você", seguem para a página de informações e serviços para os alunos da instituição ("Vida Acadêmica") e acessam a página que contêm os calendários da universidade ("Calendário Acadêmico"); retornam até a página "UFU e Você" e acessam o serviço de busca do portal ("Busca Avançada").

O último padrão, figura 6.2(d), foi minerado com à restrição geral do autômato *A-UFUGeral*

em todas as três bases de dados, o que indica que ele foi freqüente nas três semanas de acesso consideradas no estudo de caso. O padrão de acesso se inicia na página de "Pesquisa e Pos-graduação" seguindo para a página dos cursos de especialização da universidade ("Pos-graduação Lato Sensu"), retornando à página inicial e acessando em seguida a página dos curso de mestrado "Pos-graduação Strictu Sensu").

Capítulo 7

Conclusões e Trabalhos Futuros

Um dos grandes desafios da Mineração de Dados é fornecer ao usuário formas de realizar esse processo sem exigir conhecimentos específicos. Para isso, faz-se necessária a realização de algumas abstrações por parte de interfaces e aplicações facilitando o uso dos algoritmos de mineração e possibilitando a utilização destes por um grupo maior de usuários. A proposta principal deste trabalho foi a definição de uma linguagem visual para especificar classes de padrões arborescentes que pudesse ser usada tanto para especificar restrições na fase de pré-processamento de mineração de padrões de árvore freqüentes com restrição, quanto para efetuar consultas em uma fase de pós-processamento.

A criação da linguagem VisTree teve como objetivo a representação visual do mecanismo de restrição usado pelo algoritmo CobMiner, e de consultas usadas na recuperação de padrões. Em uma fase de pré-processamento através do uso da linguagem, a árvore com o molde dos padrões que deverão ser minerados é montada, e posteriormente é representado por um autômato de árvore. Dessa forma o usuário não terá que ter conhecimento do que é um autômato de árvore para efetuar a mineração dos dados. Ele poderá fazer uso da linguagem visual para representar à restrição de uma maneira intuitiva. Um processo semelhante também ocorre na representação da consulta, na fase de recuperação de padrões, onde uma árvore de consulta é usada para definir a consulta sobre os padrões freqüentes e é posteriormente convertida em uma consulta XQuery.

O sistema CobMiner foi desenvolvido para se ter um ambiente onde o uso da linguagem pudesse ser testado. Nesse sistema foram escolhidas duas aplicações em dados reais para fazer a mineração de padrões de árvore com restrição, sendo elas a mineração de Documen-

tos XML e de *logs* de Navegação Web. No desenvolvimento do sistema foram criados os algoritmos da fase de pré-processamento que fazem as atividades comuns a qualquer atividade de pré-processamento e que adicionalmente faz a conversão da restrição em um autômato de árvore. A fase de mineração é feita por um algoritmo de mineração de padrões de árvore com restrição previamente definido em um outro trabalho de dissertação de mestrado [Silva 2007, de Amo et al. 2007], o algoritmo CobMiner. Na fase de pós-processamento do sistema, os algoritmos para a análise de padrões possibilitam as consultas aos padrões freqüentes em VisTree e faz a tradução da árvore de consulta em uma expressão XQuery.

Dois estudos de caso foram feitos com a aplicação da linguagem VisTree na especificação de classes de padrões arborescentes em *XML Mining* e *Web Mining*. As bases de dados escolhidas foram documentos XML da IMDB(*Internet Movie Database*) e *logs* de navegação do site da Universidade Federal de Uberlândia e cada um dos estudos foi constituído por 3 bases de dados. Os documentos XML minerados contêm informações relacionadas a filmes e os *logs* de acesso a navegação dos usuários do *website* por um período de uma semana cada. As base de dados obtidas dos documentos XML possuiam tamanho variando entre 3479 a 48937 árvores e os *logs* de navegação de 4028 a 5765.

Os trabalhos futuros que foram identificados relacionam-se tanto às aplicações descritas nesta dissertação como à criação de novas aplicações. Dentre as possibilidades de trabalhos futuros tem-se:

- Aplicação da mineração em documentos XML, tendo como entrada vários documentos XML semelhantes. No estágio atual o sistema aceita como entrada apenas um documento XML;

- Mineração de árvores de acesso com restrição em um sistema Web onde há validação do usuário, como sistemas de comércio eletrônico, educação a distância, etc.

- Criação de outro módulo de aplicação para fazer a mineração de padrões arborescentes com restrição em um ambiente de Bioinformática, utilizando-a na mineração estruturas de RNA.

Referências Bibliográficas

[Agrawal et al. 1993] Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C.

[Agrawal and Srikant 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann.

[Agrawal and Srikant 1995] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan. IEEE Computer Society Press.

[Amer-Yahia et al. 2002] Amer-Yahia, S., Cho, S., Lakshmanan, L. V. S., and Srivastava, D. (2002). Tree pattern query minimization. *The VLDB Journal*, pages 315–331.

[Asai et al. 2002] Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., and Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. In *SDM '02: Proceedings of the Second SIAM International Conference on Data Mining 2002*, pages 158–174.

[Asai et al. 2003] Asai, T., Arimura, H., Uno, T., Nakano, S.-i., and Satoh, K. (2003). Efficient tree mining using reverse search.

[Auber 2003] Auber, D. (2003). Tulip:A huge graph visualisation framework. In Mutzel, P. and Jünger, M., editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag.

[Berendt et al. 2004] Berendt, B., Hotho, A., and Stumme, G. (2004). Usage mining for and on the semantic web. In *Data Mining Next Generation Challenges and Future Directions*, pages 461–481. AAAI Press, Boston.

[Chamberlin 2003] Chamberlin, D. (2003). Xquery: a query language for xml. In *SIGMOD '03: Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 682–682, New York, NY, USA. ACM Press.

[Che and Liu 2005] Che, D. and Liu, Y. (2005). Efficient minimization of xml tree pattern queries. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, page 447, Washington, DC, USA. IEEE Computer Society.

[Chen et al. 2003] Chen, Z., Jagadish, H. V., Lakshmanan, L. V. S., and Paparizos, S. (2003). From tree patterns to generalized tree patterns: On efficient evaluation of xquery. In *VLDB*, pages 237–248.

[Chevalet and Michot 1992] Chevalet, C. and Michot, B. (1992). An algorithm for comparing rna secondary structures and searching for similar substructures. *Computer Applications in the Biosciences*, pages 215–225.

[Clark and Derose 1999] Clark, J. and Derose, S. J. (1999). Xml path language XPath version 1.0. *W3C Recomendation*.

[Cooley 2000] Cooley, R. (2000). *Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data*. PhD thesis, Department of Computer Science, University of Minnesota.

[Cooley et al. 1999] Cooley, R., Mobasher, B., and Srivastava, J. (1999). Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, pages 5–32.

[de Amo and Felício 2007] de Amo, S. and Felício, C. Z. (2007). Using tree automata for xml mining and web mining with constraints. In *3° Workshop de Algoritmos e Aplicações de Mineração de Dados*.

[de Amo et al. 2007] de Amo, S., Silva, N. A., Silva, R. P., and Fernandes, F. (2007). Contraint-based tree pattern mining. In *22th Brazilian Symposium on Databases*.

[Delest et al. 2004] Delest, M., Munzner, T., Auber, D., and Domenger, J.-P. (2004). Exploring infovis publication history with tulip. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'04)*, Washington, DC, USA. IEEE Computer Society.

[Garofalakis et al. 1999] Garofalakis, M. N., Rastogi, R., and Shim, K. (1999). SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234.

[Han and Kamber 2000] Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann.

[Ivancsy and Vajk 2006] Ivancsy, R. and Vajk, I. (2006). Frequent pattern mining in web log data. *Acta Polytechnica Hungarica, Journal of Applied Science at Budapest Tech Hungary, Special Issue on Computational Intelligence*, pages 77–90.

[Katz et al. 2003] Katz, H., Chamberlin, D., Kay, M., Wadler, P., and Draper, D. (2003). *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kosala and Blockeel 2000] Kosala, R. and Blockeel, H. (2000). Web mining research: a survey. *SIGKDD Explor. Newsl.*, pages 1–15.

[Machado 2002] Machado, L. d. S. (2002). Mineração do uso da web na educação a distância: Propostas para a condução de um processo a partir de um estudo de caso. Master's thesis, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.

[Marquardt 2006] Marquardt, C. G. (2006). Apoio ao pré-processamento de dados da mineração do uso em ambientes de ensino na web. Master's thesis, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.

[Miklau and Suciu 2004] Miklau, G. and Suciu, D. (2004). Containment and equivalence for a fragment of xpath. *J. ACM*, 51:2–45.

[Murata et al. 2005] Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of xml schema languages using formal language theory. *ACM Trans. Inter. Tech.*, pages 660–704.

[Neven 2002] Neven, F. (2002). Automata theory for xml researchers. *SIGMOD Record*, 31(3):39–46.

[Olteanu et al. 2002] Olteanu, D., Meuss, H., Furche, T., and Bry, F. (2002). Xpath: Looking forward. In *EDBT '02: Proceedings of the Worshops XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers*, pages 109–127, London, UK. Springer-Verlag.

[Papakonstantinou and Vianu 2000] Papakonstantinou, Y. and Vianu, V. (2000). DTD inference for views of XML data. In *PODS 2000*, pages 35–46, New York, NY, USA. ACM Press.

[Punin and Krishnamoorthy 2002] Punin, J. R. and Krishnamoorthy, Mukkai S.and Zaki, M. J. (2002). Logml: Log markup language for web usage mining. In *WEBKDD '01: Revised Papers from the Third International Workshop on Mining Web Log Data Across All Customers Touch Points*, pages 88–112, London, UK. Springer-Verlag.

[Punin and Krishnamoorthy 1998] Punin, J. R. and Krishnamoorthy, M. S. (1998). Wwwpal - a system for analysis and synthesis of web pages. In *Proceedings of the WebNet 98 Conference*.

[Punin et al. 2001] Punin, J. R., Krishnamoorthy, M. S., and Zaki, M. J. (2001). Web usage mining: Languages and algorithms. In *Studies in Classification, Data Analysis, and Knowledge Organization*. Springer-Verlag.

[Shapiro and Zhang 1990] Shapiro, B. A. and Zhang, K. Z. (1990). Comparing multiple rna secondary structures using tree comparisons. *Computer Application in Bioscience*, pages 309–318.

[Silva 2007] Silva, N. d. A. (2007). Cobminer - mineração de padrões arborescentes com restrição. Master's thesis, Faculdade de Computação, Universidade Federal de Uberlândia.

[Srikant and Agrawal 1996] Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag.

[Srikant and Yang 2001] Srikant, R. and Yang, Y. (2001). Mining web logs to improve website organization. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 430–437, New York, NY, USA. ACM Press.

[Takahashi 1975] Takahashi, M. (1975). Generalizations of regular sets and their application to a study of context-free languages. *Information Control*, pages 1–36.

[Termier et al. 2002] Termier, A., Rousset, M.-C., and Sebag, M. (2002). Treefinder: a first step towards xml data mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining 2002*, pages 450–457, Washington, DC, USA. IEEE Computer Society.

[Vanzin 2004] Vanzin, M. (2004). Mecanismo de apoio a interpretação e recuperação de padrões do uso da web baseados em ontologia de domínio. Master's thesis, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.

[Washio and Motoda 2003] Washio, T. and Motoda, H. (2003). State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, pages 59–68.

[Yan and Han 2002] Yan, X. and Han, J. (2002). gSpan: Graph-based substructure pattern mining. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724, Washington, DC, USA. IEEE Computer Society.

[Yang et al. 2003] Yang, L. H., Lee, M.-L., and Hsu, W. (2003). Efficient mining of xml query patterns for caching. In *VLDB*, pages 69–80.

[Zaki 2002] Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *KDD 2002*, pages 71–80, New York, NY, USA. ACM Press.