UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Lucas Antonio Toledo Godoy

Extração de Dados de Produtos em Páginas de Comércio Eletrônico

São José do Rio Preto
2015

Lucas Antonio Toledo Godoy

Extração de Dados de Produtos em Páginas de Comércio Eletrônico

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador: Prof. Dr. Ivan Rizzo Guilherme
Coorientador: Prof. Dr. Daniel C. G. Pedronette

São José do Rio Preto
2015

Godoy, Lucas Antonio Toledo.
Extração de dados de produtos em páginas de comércio eletrônico / Lucas Antonio Toledo Godoy. -- São José do Rio Preto, 2015
103 f. : il., tabs.

Orientador: Ivan Rizzo Guilherme
Coorientador: Daniel C. G. Pedronette
Dissertação (mestrado) – Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação - Matemática. 2. World Wide Web (Sistema de recuperação da informação) 3. Comércio eletrônico. 4. Algoritmos de computador. I. Guilherme, Ivan Rizzo. II. Pedronette, Daniel Carlos Guimarães. III. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. IV. Título.
CDU – 518.72

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Lucas Antonio Toledo Godoy

Extração de Dados de Produtos em Páginas de Comércio Eletrônico

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Ivan Rizzo Guilherme
IGCE - DEMAC - UNESP - Rio Claro
Orientador

Prof. Dr. Fabrício Aparecido Breve
IGCE - DEMAC - UNESP - Rio Claro

Prof. Dr. Jurandy Gomes de Almeida Junior
ICT - UNIFESP - São José dos Campos

São José do Rio Preto
26 de fevereiro de 2015

Ao meu pai, que sempre acreditou que o estudo é a chave para o sucesso e que viu de perto todo o meu esforço durante o desenvolvimento deste trabalho. Apesar de não estar mais conosco, sempre estará em nossas lembranças como o maior exemplo de honestidade e caráter que tivemos.

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida e por me conceder saúde e serenidade para lidar com os desafios tanto da vida acadêmica quanto da profissional.

Aos meus pais e familiares por todo o apoio e incentivo e pela presença nos momentos de alegria e, sobretudo, nos momentos difíceis que encaramos durante o tempo em que este trabalho foi desenvolvido.

Ao meu orientador, Prof. Dr. Ivan Rizzo Guilherme, por ter me ajudado a tomar as decisões mais acertadas e pela paciência que demonstrou durante os obstáculos e contratempos que surgiram em minha vida pessoal durante o desenvolvimento deste trabalho.

Aos amigos da MStech, empresa em que trabalhei no início do mestrado, que sempre incentivou a capacitação dos seus colaboradores e me permitiu conciliar as atividades profissionais e acadêmicas.

Por fim, agradeço ao IBOPE, empresa na qual trabalhei durante grande parte do tempo em que este trabalho foi desenvolvido e que devido às atividades realizadas no dia-a-dia, me permitiu a escolha do tema desta dissertação, além de fornecer subsídios para a montagem da base de páginas de comércio eletrônico empregada nos experimentos deste trabalho.

As alturas alcançadas e mantidas pelos grandes homens não foram encontradas de maneira súbita. Enquanto seus companheiros dormiam, os grandes homens estavam labutando no meio da noite.
(Henry Wadsworth Longfellow)

Resumo

A extração de dados em páginas Web é um importante problema que começou a ganhar força a partir da metade da década de 90. Um dos subdomínios dessa categoria de extração de dados possui como foco os produtos em páginas de vendas *online*, dada a riqueza de informações disponibilizadas pelas lojas de *e-commerce*, ou comércio eletrônico, em seus portais de vendas. A extração dos dados dos produtos contidos nessas páginas, como nomes e preços, permite a criação de uma grande variedade de outras ferramentas que façam uso de tais dados com o objetivo de fornecer uma interpretação semântica a eles, como comparações entre preços praticados por diferentes lojas ou análises de hábitos de consumo. Diversas abordagens têm sido empregadas para se chegar à correta extração dos dados de interesse das páginas, fazendo uso de uma gama variada de técnicas para alcançarem seus objetivos, sendo que a técnica de *Tree Matching* apresenta grande destaque devido aos bons resultados. Este trabalho teve como objetivo implementar e avaliar o uso da técnica de *Tree Matching* para a extração de dados de produtos, especificamente o nome do produto, seu preço e, porventura, o preço promocional, em páginas de comércio eletrônico, a fim de determinar sua aplicabilidade a um sistema comercial. Foram propostas melhorias ao processo de extração com a finalidade de reduzir o tempo de resposta e aumentar a acurácia do algoritmo *Generalized Simple Tree Matching*. Resultados experimentais demonstraram uma precisão na extração dos dados de produtos na ordem de 93.6% sobre as páginas contidas na base *Ecommerce DB* e um ganho médio no tempo de resposta na ordem de 36% quando as páginas são reduzidas pelos métodos propostos neste trabalho.

Palavras-chave: Extração de Informações, Extração de Dados Estruturados, Páginas de Comércio Eletrônico, Tree Matching.

Abstract

Web data extraction is an important issue which started becoming a strong line of study in the mid 90s. A subdomain of that category of study is the product data extraction from online sales pages, given the wealth of information provided by stores through their websites. Data extraction of products contained in these kind of pages, like product name and prices, enables the creation of a wide variety of other tools that are able to use such data in order to provide a semantic interpretation to them, such as prices comparison among different stores and consumption habits analysis. Several approaches have been applied to reach the target data extraction from Web pages. These approaches, in turn, use a wide range of techniques to reach their goals, and Tree Matching technique has great prominence due to its good results. This dissertation aimed to implement and evaluate the Tree Matching technique for the extraction of product data, specifically the product name, its price and, perhaps, the promotional price, on e-commerce pages, in order to determine its applicability to a commercial system. Improvements have been proposed to the extraction process in order to reduce the response time and increase the accuracy of the *Generalized Simple Tree Matching* algorithm. Experimental results demonstrated that the extraction process got an accuracy of about 93.6% on pages contained in *Ecommerce Database* and an average gain in response time of about 36% when the pages were reduced by the methods proposed in this study.

Keywords: Information Extraction, Structured Data Extraction, E-commerce Web Pages, Tree Matching.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de algoritmos

Lista de abreviaturas e siglas

AFN Autômato Finito Não Determinístico

API Application Programming Interface

ARPANet Advanced Research Projects Agency Network

DOM Document Object Model

G-STM Generalized Simple Tree Matching

HTML HyperText Markup Language

IDE Integrated Development Environment

JSON-LD JavaScript Object Notation for Linked Data

RDF Resource Description Framework

STM Simple Tree Matching

URL Uniform Resource Locator

WWW World Wide Web

XML Extensible Markup Language

Sumário

1 Introdução

O rápido crescimento da Web nas últimas duas décadas fez com que ela se tornasse a maior fonte pública de dados em todo o mundo. A diversidade de dados e, consequentemente, as informações que esses dados quando organizados podem trazer é exorbitante, o que faz da Web, dentre outras características, uma excelente fonte de informações sobre os mais diversos tipos de conteúdo existentes.

Uma importante utilização da Web reside na criação de portais de comércio eletrônico, os chamados portais de *e-commerce*. Grandes volumes de informações referentes a produtos têm sido disponibilizados nos mais diversos formatos nos portais mantidos pelas lojas especializadas no assunto. A qualidade das informações publicamente fornecidas por essas lojas é bastante significativa, já que devem ser oferecidos aos usuários todos os detalhes sobre os produtos vendidos, que vão desde a descrição até o preço do produto, caso contrário os consumidores *online* não poderiam efetuar suas compras com a confiabilidade e segurança que desejam.

Em decorrência da grande competição entre as empresas de comércio eletrônico e da necessidade de medições quanto à qualidade e integridade dos serviços oferecidos pelas lojas *online*, tem sido necessário o desenvolvimento de programas que visam facilitar a busca de produtos por consumidores, a avaliação dos hábitos dos compradores ou para a definição de estratégias de preços frente aos concorrentes. Todos esses programas requerem a extração dos dados das páginas dos portais de comércio eletrônico. O processo de extração dos dados é uma tarefa bastante complexa devido à diversidade de estruturas e conteúdos das páginas, já que cada portal emprega um estilo próprio para a construção de seu *website*. Há também a dificuldade decorrente do grande volume de páginas a serem avaliadas.

A necessidade do desenvolvimento de sistemas para a extração de dados em páginas na Web surgiu na metade da década de 90, a partir da demanda das empresas e usuários pela busca de produtos contidos nessas páginas. Como consequência dessa demanda, dentre as principais linhas de pesquisa seguidas para tal finalidade, três delas ganharam força (LIU, 2011):

1. **Abordagem manual**: observando a própria página e seu respectivo código-fonte, o programador (o ser humano) identifica os padrões da página e escreve um programa que extrai os dados que são o alvo da análise (ZHAI; LIU, 2005; LIU, 2011). Obviamente essa abordagem não é escalável para um grande número de *sites*, dada a dificuldade na análise de tamanha quantidade de páginas por um ser humano. Além disso, sempre há a possibilidade de que o *layout* das páginas seja alterado periodicamente, o que acarretaria uma tarefa de reescrita do código sempre que isso

acontecesse;

2. **Construção de um wrapper**: no contexto da extração de informações de páginas Web, um *wrapper* é um programa que tem a capacidade de extrair dados de conteúdos estruturados (LIU, 2011), no caso, páginas HTML (*HyperText Markup Language*). Nesse tipo de abordagem um conjunto de regras é estabelecido a partir de uma coleção de páginas previamente rotuladas por um processo manual, denominado *indução* e, posteriormente, o conjunto de regras é aplicado às demais páginas (ZHAI; LIU, 2006; LIU, 2011). Diferentemente da abordagem manual, nesta abordagem o operador humano analisa um grupo reduzido de páginas e é o *wrapper*, a partir das regras inferidas desse grupo, quem vai identificar os padrões nas páginas;

3. **Extração automática**: trata-se de uma abordagem na qual, dada uma única página ou um conjunto de páginas como entrada, são aplicadas técnicas que automaticamente encontram padrões e gramáticas provenientes da estruturação natural dos dados nas páginas Web (ZHAI; LIU, 2005; LIU, 2011). Como esta abordagem elimina a necessidade de interação manual e rotulação, ela é altamente escalável para a extração de dados e também auto-adaptável às mudanças que podem ocorrer na estrutura das páginas com o decorrer do tempo.

O processo da separação, análise e da extração de dados em páginas Web é uma subárea de estudo de *Data Mining* (mineração de dados), denominada *Web Data Mining* (LAENDER et al., 2002a). A mineração de dados é comumente definida como sendo o processo da descoberta de padrões ou extração de conhecimentos provenientes da análise de alguma fonte de dados (LIU, 2011), que pode vir a ser bancos de dados, textos, imagens e, claro, páginas Web.

Diversas técnicas vêm sendo utilizadas para a mineração de dados em páginas Web. As técnicas mais comuns, segundo Liu (2011), são: o **aprendizado supervisionado** (*supervised learning*), também conhecido como classificação; o **aprendizado não supervisionado** (*unsupervised learning*), também conhecido como agrupamento (*clustering*); o **aprendizado semissupervisionado** (*semi-supervised learning*); a mineração por meio de **regras de associação** (*association rule mining*); a mineração de **padrões sequenciais** (*sequential pattern mining*). Todas essas técnicas se encaixam nas linhas de pesquisa de construção de *wrappers* ou extração automática, podendo, inclusive, serem mesclas de ambas.

O processo de mineração de dados em páginas Web é bastante similar ao processo de *data mining* tradicional, mas possui a peculiaridade de que, ao invés da análise dos dados ser feita sobre bases já coletadas e armazenadas, as bases de dados agora são as próprias páginas Web, que podem sofrer constantes mudanças. A mineração de dados em páginas Web envolve também a localização e a extração do que é dado útil em meio ao

conteúdo das páginas, tarefa que geralmente envolve a análise de um grande volume de dados.

As informações disponibilizadas *online* sobre os produtos vendidos, quando extraídas e organizadas por um serviço autônomo de captura de produtos entre lojas, que faz uso das abordagens discutidas acima, são muito úteis, pois funcionam como porta de entrada para o desenvolvimento de ferramentas especializadas na análise sobre indicadores, como os preços praticados entre lojas, medições de audiência entre páginas, estimativas de lucros, categorização de produtos por quantidade de vendas, dentre outras inúmeras possibilidades de organização semântica desses dados. Isso tudo somente é possível se houver a correta extração das informações pertinentes aos produtos das páginas públicas de vendas das lojas de *e-commerce*.

1.1 Objetivos

Este trabalho tem como objetivo o estudo, a implementação e a melhoria das técnicas para a extração de dados de produtos contidos em páginas de vendas *online* das mais diversas lojas especializadas no assunto. Utilizando o resultado do estudo foram identificadas as estratégias e os algoritmos mais adequados e eficazes para a identificação e a extração dos nomes dos produtos bem como seus preços e preços promocionais. Posteriormente, os algoritmos identificados foram implementados. A avaliação dos algoritmos foi realizada por meio do emprego de bases de dados de páginas Web de vendas *online*.

1.2 Motivação

As maiores lojas de varejo tanto no Brasil quanto no exterior vêm apostando cada vez mais em vendas realizadas pela Web, dada a comodidade e a crescente confiança dos consumidores nessa categoria de comércio. Dessa forma, tais lojas investem em sites de *e-commerce* nos quais disponibilizam seus produtos ao público.

Toma-se como premissa que seja preciso analisar quais produtos estão disponíveis em páginas de comércio eletrônico das mais diversas lojas baseando-se na visualização dessas páginas por um grande grupo de usuários. Os usuários desse grupo possuem ferramentas instaladas em seus computadores que capturam as páginas de comércio eletrônico visualizadas, utilizando um navegador de Internet, e as enviam para um servidor central, a fim de que os produtos contidos nessas páginas sejam extraídos. Empiricamente foi determinado que o volume de páginas visualizadas por esse grupo e que chegam para terem seus produtos extraídos seja na ordem de 1000 páginas por minuto. Diante de tal situação fica muito difícil a contratação de pessoas que deem conta de tamanha vazão de páginas sem o auxílio de uma ferramenta computacional que empregue, por exemplo,

técnicas de Inteligência Artificial capazes de automatizar o processo ou pelo menos parte dele.

Ciente dessa situação e analisando as diversas páginas de *e-commerce* das mais variadas lojas, percebe-se que, apesar do estilo próprio de cada loja em construir sua página, há claramente um padrão no que diz respeito à estruturação dessas páginas, tanto no que se refere à linguagem HTML, que é estruturada por natureza, quanto nas seções apresentadas dentro das páginas de *e-commerce*, sendo que a totalidade delas apresenta uma ou mais das seguintes seções, também chamadas de indicadores: produtos em detalhe, produtos em vitrine, produtos sugeridos, produtos em carrinho e produtos comprados.

Diante de tais circunstâncias é necessário a elaboração de métodos e técnicas capazes de determinar áreas em uma página de vendas na Web nas quais se concentram os dados de interesse e, posteriormente, efetuar a extração desses dados.

1.3 Estrutura da Dissertação

Além deste capítulo introdutório, que apresenta os objetivos e motivações para o desenvolvimento do trabalho, esta dissertação se divide também em outros 7 capítulos.

O Capítulo 2 apresenta a evolução histórica do tema proposto nesta dissertação, além de destacar os principais trabalhos em cada linha de pesquisa encontrada na literatura especializada no assunto.

No Capítulo 3 é apresentada de forma conceitual a estrutura de uma página Web, bem como as formas de disposição de conteúdos em páginas, além da formalização do modelo utilizado em sua construção. Por fim, o capítulo detalha o modelo DOM, que é uma das peças mais importantes para a viabilização das ideias propostas neste trabalho.

O Capítulo 4 possui foco na elucidação das técnicas de *Tree Matching* para a extração de padrões em páginas Web, utilizando como estrutura para essa busca de padrões as árvores construídas a partir do código HTML das páginas. São apresentados os algoritmos *Simple Tree Matching*, ou simplesmente STM, e *Generalized Simple Tree Matching*, chamado G-STM, que são dois algoritmos especializados na busca de padrões em árvores, expondo-se suas principais características e limitações. O capítulo termina com a discussão de uma melhoria incorporada ao algoritmo G-STM com o intuito de incrementar sua acurácia, adequando-o ao ambiente de atuação proposto neste trabalho, que são as páginas de produtos dos portais de comércio eletrônico.

Por sua vez, o Capítulo 5 apresenta as etapas do processo de extração de dados em páginas Web e propõe estratégias para reduzir seu tempo de resposta.

O Capítulo 6 descreve a metodologia seguida neste trabalho, os métodos utilizados, com detalhes acerca dos parâmetros adotados e o material empregado para avaliação dos

métodos.

No Capítulo 7 são apresentados e discutidos os resultados dos experimentos que avaliam as etapas do processo de extração proposto neste trabalho. Os experimentos englobam todas as etapas, desde o *pré-processamento*, cujo foco é a construção do modelo DOM e a redução do tamanho das páginas, o *processamento*, para extração das regiões de dados, e o *pós-processamento*, cujo objetivo é a extração dos dados dos produtos.

Por fim, no Capítulo 8 são expostas as conclusões do autor embasadas pelos resultados obtidos, enumeradas as contribuições deste trabalho e apontados os trabalhos futuros que podem ser desenvolvidos a partir dos resultados dos estudos apresentados nesta dissertação.

2 Evolução Histórica e Principais Trabalhos

Este capítulo tem o objetivo de apresentar as fases históricas da extração de informações em páginas Web, mostrando as evoluções ocorridas acerca do estudo do tema durante os anos e contextualizando o presente trabalho quanto ao estado da arte.

2.1 Evolução Histórica

O interesse por extrair objetos bem como seus atributos de páginas da Web é tão antigo quanto a própria Web.

Desde as primeiras conexões da ARPANet (*Advanced Research Projects Agency Network*), em 1969 (LIU, 2011), até o advento da Internet e, posteriormente, sua popularização com o surgimento da Web em 1989 (LIU, 2011), também conhecida pelo acrônimo WWW (*World Wide Web*), o objetivo de interligar computadores formando uma grande rede sempre foi o de disseminar informações, inicialmente sigilosas quando da sua criação mas que, com o passar do tempo, foi provando-se uma forma bastante eficaz de compartilhamento de informações ao público em geral, haja vista a diversidade quanto ao uso com que a Internet foi sendo tratada até chegarmos aos dias de hoje.

A partir dos anos 90, a consolidação da Web não só dentro do ambiente acadêmico mas também em grandes corporações fez com que o crescente número de documentos (artigos, pesquisas, documentos confidenciais) circulando dentro dos sistemas de informação dessas instituições despertassem significativo interesse público (LABSKY, 2008) e, devido ao próprio processo de popularização da Web ocorrido com o passar dos anos, muitos desses documentos tornaram-se públicos de fato, disseminando, dessa forma, os conhecimentos adquiridos e desenvolvidos por essas instituições.

Com o gradual aumento da quantidade de informações disponibilizadas na Web fez-se necessário o desenvolvimento de ferramentas capazes de gerenciar, organizar e distribuir tais informações ao público, de forma que pudessem ser absorvidas sem grande esforço. Assim, o primeiro documento público descrevendo a linguagem HTML, chamado *"HTML Tags"*, foi disponibilizado em 1991 na própria Web pelo físico e também responsável pela criação da *World Wide Web*, Tim Berners-Lee (RAGGETT et al., 1998). A linguagem HTML forneceu uma forma simples e estruturada de construção de páginas Web, possibilitando que, a partir de então, a Web começasse a tomar a forma e a aparência a qual conhecemos hoje, ou seja, disponibilizando informações na forma de páginas (*Web Pages*).

A natureza estruturada das páginas Web permite que duas observações sejam feitas a respeito dos objetos nelas contidos, observações estas também compartilhadas

por Furche et al. (2012). Primeiramente, os conteúdos (ou objetos) contidos em páginas são tipicamente apresentados como listas, tabelas, *grids*, ou então outras estruturas de repetição que fazem uso de um *template*, ou seja, um modelo comum para a organização do conteúdo. Em segundo lugar, as páginas Web são planejadas e construídas com o objetivo de que os usuários consigam identificar rapidamente os objetos e seus respectivos atributos contidos na página. Por isso, as páginas procuram empregar elementos visuais e vocabulários simplificados ao apresentarem objetos de um mesmo domínio como, por exemplo, ofertas de produtos, nas quais são mostradas somente a imagem e o preço de cada item. Essas observações são de grande valia para o desenvolvimento de sistemas capazes de extrair informações de páginas Web. Assim, se nos primórdios da Web a linguagem HTML foi concebida para sanar a preocupação sobre a forma como as informações seriam disponibilizadas ao público, a construção de extratores de informações, por sua vez, tem como preocupação o oposto, ou seja, a partir de páginas publicamente disponibilizadas na Web, extrair dados de interesse nelas contidos, indexando informações que fazem parte das bases de dados anteriormente acessíveis somente ao ambiente interno de instituições.

Os primeiros sistemas de extração de dados em páginas Web eram baseados em regras de extração integralmente construídas por processos manuais (FURCHE et al., 2012), em outras palavras, pr meio de programas escritos por um ser humano após este analisar a estrutura das páginas, bem como seu código-fonte, buscando informações visuais que remetessem aos locais onde os dados estavam concentrados.

Durante a segunda metade da década de 90, técnicas de aprendizado de máquina proliferaram na área da extração de dados (FERRARA et al., 2014), diminuindo significativamente a necessidade da utilização de processos manuais na extração, mas não chegando a eliminá-los em sua totalidade. Tal mudança, apesar de ser considerada um grande avanço (e de fato foi), trouxe consigo um empecilho: a necessidade da utilização de grandes volumes de dados para o treinamento dos métodos de aprendizado de máquina.

Logo após o ano de 2001, época na qual, segundo Labsky (2008), os conceitos de Web Semântica foram introduzidos, parte das pesquisas com extração de dados em páginas direcionaram o foco para o desenvolvimento de processos capazes de automaticamente inserir anotações semânticas nas páginas, além de explorar o uso de ontologias nos processos de extração.

Também durante os anos 2000, mais especificamente na segunda metade da década, surgiram técnicas que visavam a construção de *wrappers* para a extração de dados, técnicas essas que diferenciavam entre si pela robustez e tentativa de simplificação do processo de inferência de regras, as quais tinham por objetivo a sintetização do que era útil à extração, dentre todo o conteúdo da página. Ainda assim, tais técnicas exigiam a intervenção humana em uma etapa inicial de rotulação dos dados e treinamento chamada **indução**, na qual o ser humano mostrava ao *wrapper* onde estavam os dados e assim, a partir daí,

as regras podiam ser inferidas. Devido a essa etapa inicial de rotulação e treinamento para a construção de regras, essas abordagens de extração são consideradas abordagens de aprendizado supervisionado (LIU, 2011).

Paralelamente aos modelos de *wrappers* que possuíam intervenção humana, surgiram tentativas de automatização completa do processo de extração, seja pela confecção de algoritmos de construção de *wrappers* de forma autônoma (ARLOTTA et al., 2003), quanto pelo emprego de outros tipos de técnicas, como a utilização de métodos de reconhecimento de padrões visando a determinação de regiões de interesse na página, para então, dentro de tais regiões, extrair os dados de interesse (JINDAL; LIU, 2010). As técnicas de extração automatizada de dados têm como grande objetivo a eliminação completa da interação humana junto ao processo de extração, mas também focam na construção de algoritmos autoadaptáveis às mudanças que podem ocorrer rotineiramente na estrutura da página com o passar do tempo. Abordagens dessa categoria não trabalham com dados previamente rotulados e precisam descobrir sozinhas quais dados devem extrair. Por esses motivos são consideradas abordagens de extração não supervisionadas ou automáticas (LIU, 2011).

Diversos outros modelos de extração de dados foram concebidos com o intuito de mesclar as melhores características das abordagens de aprendizado supervisionado com a abordagem automática, maximizando os pontos fortes e suprimindo as falhas de cada uma. Tais modelos ficaram conhecidos como abordagens semi-automáticas (ASHISH; KNOBLOCK, 1997; ESTIÉVENART et al., 2006). As abordagens semi-automáticas para a extração de dados permitem que algumas de suas etapas sejam totalmente automatizadas – como a localização de regiões de interesse dentro das páginas por meio de técnicas de reconhecimento de padrões – mas permitem também que outras etapas possuam interação humana, como a consulta a um operador caso o algoritmo de extração encontre um dado na página mas fique em dúvida se o mesmo é ou não um dado de interesse ao domínio do problema.

É importante ressaltar que, apesar das abordagens supervisionadas e semi-automáticas preverem interação com um ser humano, esta se dá em escalas distintas e se apresenta de forma muito mais contundente na abordagem que emprega o aprendizado supervisionado, pois tal abordagem exige o treinamento a partir de um número significativo de exemplos previamente rotulados. A abordagem semi-automática, por sua vez, não exige muitos exemplos rotulados (LIU, 2011), tampouco interação humana em boa parte de suas etapas, a não ser no que diz respeito à determinação da semântica dos dados extraídos (ESTIÉVENART et al., 2006) pois, mesmo que uma abordagem semi-automática se esforce ao máximo para realizar sua tarefa de forma autônoma, ainda assim é bastante difícil que os dados de interesse sejam extraídos corretamente sem que haja o conhecimento prévio do domínio do problema. Por exemplo, no domínio da extração de dados de produtos em páginas de comércio eletrônico é bastante comum que estruturas que contêm repetições

de padrões de construção, tais como menus e tabelas, sejam confundidas com vitrines de produtos por abordagens que empregam a técnica de busca por padrões sequenciais, pois os algoritmos, caso não possuam conhecimento prévio que os permitam diferenciar tais estruturas, acabam por retornar todas elas como resultados válidos. Assim, faz-se necessária a introdução de formas de inserção de conhecimento às abordagens durante a realização das sus tarefas, seja pela introdução de heurísticas ao processo de extração seja pela interação humana nos momentos em que o algoritmo gerou uma dúvida. É exatamente esse tipo de interação que é prevista nas abordagens semi-automáticas mas não nas abordagens supervisionadas.

A Tabela 1 resume as principais características das abordagens supervisionada, semi-automática e automática, realçando a aplicabilidade das características sobre cada abordagem.

Tabela 1 – Aplicabilidade de cada característica nas abordagens supervisionada, semi-automática e automática. A quantidade de *check marks* indica a importância da característica à abordagem, sendo que três *check marks* indicam máxima importância. Itens marcados com "-" indicam inexistência.

Abordagem	Fase de treinamento	Grande volume de páginas	Eficiência	Eficácia
Supervisionada	√√√	√	√	√√√
Semi-automática	√	√√	√√	√√
Automática	-	√√√	√√√	√

As próximas seções visam destacar os principais trabalhos desenvolvidos em cada fase histórica da construção do conhecimento acerca da extração de dados em páginas Web e contextualizar o presente trabalho quanto ao estado da arte nessa linha de conhecimento.

2.2 Principais Trabalhos

Uma forma bastante empregada na literatura para a classificação dos trabalhos na área da extração de dados em páginas Web reside no grau de interação humana que tais trabalhos propõem e observa-se que esse grau de interação é diretamente proporcional à eficácia de cada método desenvolvido. Abordagens com grande acurácia em seus resultados normalmente exigem elevado nível de interação humana (abordagens de aprendizado supervisionado), enquanto abordagens com mínima intervenção ou ausência de intervenção humana (abordagens não supervisionadas ou automáticas) geralmente possuem menor eficácia. Após a proliferação de técnicas utilizando a abordagem supervisionada e também a abordagem não supervisionada ou automática, estudos foram realizados com o intuito de juntar as duas técnicas, dando origem à abordagem semi-automática, objetivando a mínima interação com um ser humano porém com a máxima eficácia. Apesar da diversidade de abordagens, todas elas possuem uma premissa em comum: assumem que os dados

disponíveis na Web estão dispostos de forma estruturada nas páginas, já que tais páginas frequentemente são populadas por meio de *templates* que, por sua vez, são populados por dados provenientes de alguma base interna (FURCHE et al., 2012).

Em relação à utilização das abordagens de aprendizado supervisionado, diversos autores direcionaram o foco para a geração de *wrappers*, como os desenvolvidos nos trabalhos de Hsu e Dung (1998), Muslea et al. (1999) e Kushmerick (2000) e nos sistemas WIEN (KUSHMERICK et al., 1997) e STALKER (MUSLEA et al., 1998). As desvantagens desta abordagem na geração de *wrappers* reside no elevado tempo que a rotulação de páginas exige, rotulação essa que consiste em indicar ao algoritmo de geração do *wrapper* aonde estão os dados de interesse em um conjunto específico de páginas. Além disso, a manutenção dos *wrappers* é custosa, já que as páginas podem mudar de estrutura e *layout* a qualquer momento. Mesmo trabalhos que tentaram resolver esse problema, como o de Gulhane et al. (2011), exigem cerca de 20 páginas por *site* para a geração de regras, considerando que a maioria dos *sites* possuam uma gama de cerca de 10 modelos diferentes quanto à estrutura visual. Uma tentativa de adaptação automática de *wrappers* a mudanças realizadas em páginas foi feita por Ferrara e Baumgartner (2011) e obteve relativo sucesso.

Quanto às abordagens não supervisionadas ou automáticas, destacam-se as desenvolvidas por Crescenzi et al. (2002), Simon e Lausen (2005), Liu e Zhai (2005), Zhai e Liu (2006) e Jindal e Liu (2010). Os três últimos trabalhos utilizam uma técnica chamada *Tree Matching*, que consiste em montar uma árvore em que cada nó representa uma *tag* do código HTML da página e então comparar os nós nos diferentes níveis para que seja determinada a similaridade entre duas subárvores distintas. Tal técnica é utilizada na extração de padrões dentro das páginas a fim de localizar quais regiões possuem dados de interesse. Quanto aos sistemas que utilizam essa abordagem, podem ser citados o RoadRunner (CRESCENZI et al., 2001) e o EXALG (ARASU; GARCIA-MOLINA, 2003). As abordagens não supervisionadas geralmente sofrem uma carência de orientação sobre quais partes da página contêm informações relevantes, pois as páginas Web muitas vezes possuem partes irrelevantes ao domínio da aplicação mas que constituem estruturas regulares, como menus e rodapés e, devido a isso, acabam levando os algoritmos de extração a cometerem erros.

No campo da abordagem semi-automática podem ser destacados os sistemas *Retrozilla* (ESTIÉVENART et al., 2006) e os trabalhos de Ashish e Knoblock (1997) e Laender et al. (2002b). Todos eles implementam interfaces gráficas para a comunicação com um usuário em alguma etapa do processo de extração de dados que necessite de uma mínima interferência humana.

Métodos que fazem uso de anotações semânticas bem como ontologias para a extração de dados podem ser encontrados nos trabalhos de Furche et al. (2012), Furche et al. (2014), Akmal et al. (2014) e Vandic et al. (2014).

Outros trabalhos que merecem destaque não quanto à aplicabilidade das técnicas mas devido à importância histórica são os trabalhos de Garcia-Molina et al. (1997), como exemplo de processo de extração manual, e Freitag (2000), como exemplo de trabalho que utiliza aprendizado de máquina para a extração.

2.3 Considerações Finais

Neste capítulo foram apresentadas a evolução histórica e as principais linhas de pesquisa desenvolvidas na área da extração de dados em páginas na Web até o momento, sendo destacadas as características mais relevantes de cada abordagem. Acerca da técnica utilizada, este trabalho baseia-se na desenvolvida por Jindal e Liu (2010) que, por sua vez, emprega a técnica *Tree Matching* para a localização de regiões de dados dentro de páginas Web. Nesta dissertação são introduzidas melhorias para a adaptação da técnica ao modelo de negócios ora proposto, que são os portais de comércio eletrônico, bem como otimizações que visam minimizar o tempo de resposta do algoritmo desenvolvido por Jindal e Liu (2010).

O próximo capítulo foca-se em aspectos um pouco mais formais quanto à estrutura de páginas Web, em outras palavras, às características de construção de páginas Web, as quais permitem que técnicas como a do *Tree Matching* possam realizar seu trabalho com exatidão. O capítulo seguinte apresenta também o modelo DOM, que é a primeira grande peça do fluxo desenvolvido neste trabalho e que culminará na extração de dados em páginas na Web.

3 Aspectos Conceituais de Páginas na Web

Dados estruturados em páginas na Web frequentemente indicam que certas regiões das páginas contêm informações importantes. Esses dados geralmente são retirados de grandes bases e são dispostos nas páginas por meio do uso de *templates* e, a partir de então, passam a ser chamados de registros de dados ou *data records* (ZHAI; LIU, 2005). Neste capítulo são apresentados os aspectos conceituais relacionados à estrutura das páginas Web bem como o modelo de dados empregado em sua construção. Este capítulo apresenta também o modelo DOM, que é peça-chave para o desenvolvimento de técnicas de extração de padrões em páginas Web estruturadas na forma de árvores.

3.1 A Estrutura das Páginas na Web

As páginas disponibilizadas na Web são construídas utilizando-se a linguagem HTML (*HyperText Markup Language*). A linguagem HTML é composta por uma série de símbolos de marcação que dizem ao navegador Web como ele deve interpretar os textos englobados por esses símbolos (SHANNON, 2007). A linguagem HTML é dita *HyperText* devido ao modo como ela permite que seja feita a navegação entre diferentes conteúdos, que se dá a partir do clique em áreas especiais de texto denominadas *hyperlinks*. O motivo de ser chamada *hyper* deve-se ao fato de a linguagem permitir que os *links* redirecionem o usuário para qualquer outro conteúdo na Web de maneira não linear, ou seja, não há uma ordem específica para que os *links* sejam clicados (SHANNON, 2007). A linguagem HTML também é tida como uma linguagem de *markup* pois as *tags* permitidas (estruturas que compõem a linguagem) marcam o conteúdo ao qual estão associadas, indicando que o referido conteúdo compõe um determinado tipo de texto (SHANNON, 2007), tal como negrito ou itálico, por exemplo. Finalmente, HTML é tida como uma linguagem de fato pois, assim como as demais linguagens de programação conhecidas, é formada por um conjunto bem definido de palavras que, juntas, formam uma sintaxe (SHANNON, 2007).

As páginas Web escritas em HTML consistem de textos, *tags* e *links* para outras páginas, imagens ou vídeos. A maioria das *tags* HTML trabalham em pares, ou seja, uma *tag* de abertura (<>) e uma *tag* de fechamento (</>), respectivamente (LIU, 2011). Dentro de cada par de *tags* pode haver outros pares de *tags*, construindo-se dessa forma estruturas aninhadas. Na linguagem HTML não há *tags* designadas para cada tipo de dado (cadeias de caracteres, números inteiros, valores binários e assim por diante) pois a linguagem não foi concebida com o intuito de codificação ou armazenamento de dados. Sendo assim, qualquer *tag* pode ser usada para qualquer tipo de dado. As *tags* de abertura podem também conter atributos dentro delas, os quais aplicam certos comportamentos ao

texto que englobam, como estilos e padrões de formatação, por exemplo.

3.2 Disposição de Conteúdos em Páginas na Web

As páginas Web que exibem dados recuperados de bases por meio da utilização de *templates* para a disposição e formatação desses dados são comumente chamadas de **páginas ricas em dados** (LIU, 2011). Tais tipos de páginas são o alvo deste trabalho, pois as páginas das lojas de comércio eletrônico, das quais os dados de produtos são capturados, se encaixam nessa categoria. Sendo assim, existem dois tipos principais de páginas Web ricas em dados a saber (LIU, 2011):

1. **Listas**: páginas com listas de dados contêm diversas listas de objetos, ou seja, repetições de um modelo preenchido com dados diferentes, porém de uma mesma categoria. A Figura 1 ilustra a página de uma loja de vendas *online* contendo duas categorias de listas de produtos, uma na horizontal (parte central da página) e outra na vertical (lado direito da página). Dessa forma, a página utilizou dois *templates* diferentes para dispor os dados dos produtos, constituindo, assim, duas regiões de dados distintas.

2. **Detalhe**: páginas de detalhe focam somente um único item em especial, a fim de apresentarem mais informações sobre determinado objeto. A Figura 2 exemplifica uma página de detalhe de uma loja de vendas *online*. Tais páginas contêm todas as informações de um produto, como a imagem, a descrição, o preço, formas de pagamento, etc.

Frequentemente há casos em que uma página Web pode apresentar tanto uma seção de detalhe quanto uma lista, ou seja, promover uma mistura de regiões de dados, como mostra a Figura 3. Páginas com essa disposição dos dados exigem maior inteligência por parte das técnicas de localização dos dados-alvo a fim de não confundirem os padrões utilizados na disposição dos produtos e, consequentemente, misturarem dados de produtos não adjacentes na página.

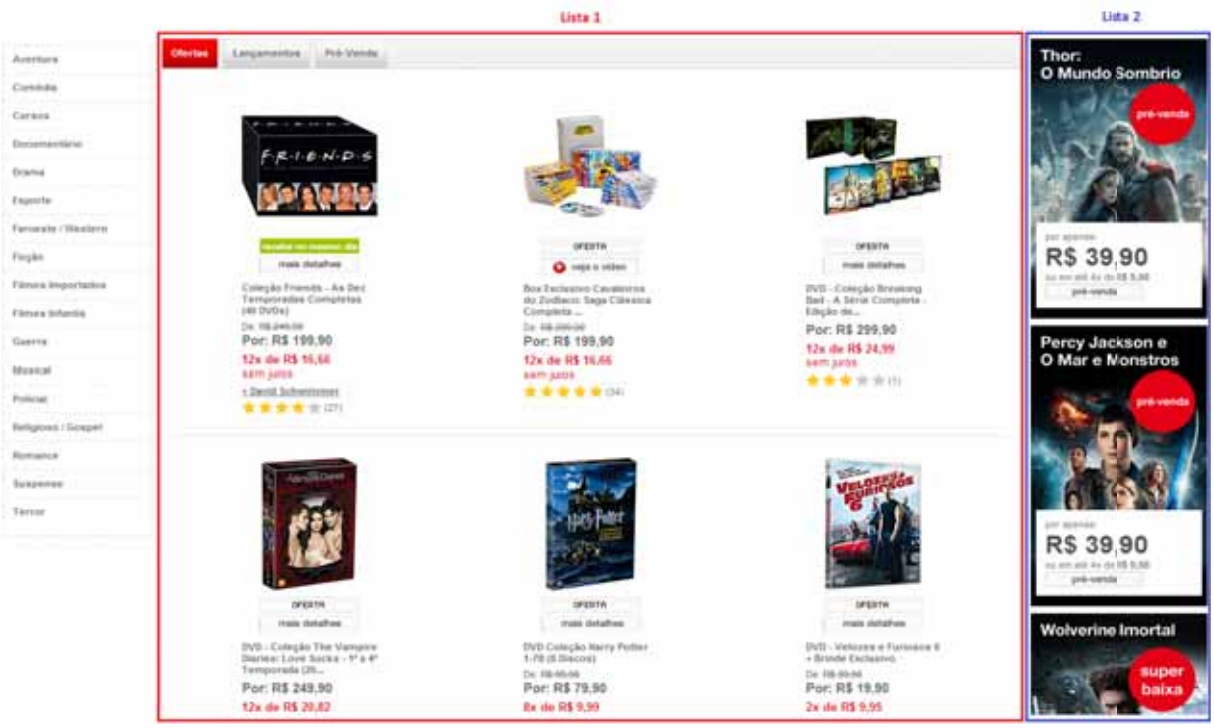


Figura 1 – Segmento de uma página de vendas na Web com dois tipos de listas: produtos dispostos horizontalmente na região central da página e produtos dispostos verticalmente no lado direito da página.

Figura 2 – Segmento de uma página de vendas na Web mostrando um produto em detalhe.

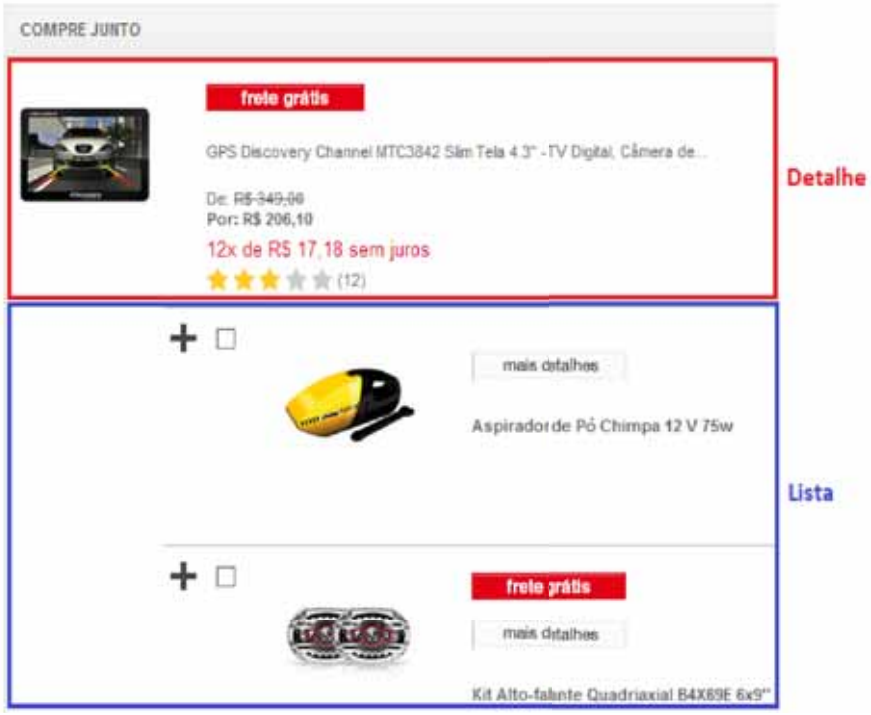


Figura 3 – Segmento de uma página de vendas na Web mostrando um produto em detalhe e demais produtos em listas.

3.3 O modelo de dados

Analisando-se as páginas Web, percebe-se que a maioria delas pode ser modelada como um conjunto de relações aninhadas, que são objetos tipados os quais permitem desdobramentos na forma de conjuntos e tuplas (LIU, 2011). Tais objetos podem ser definidos da seguinte maneira (JINDAL; LIU, 2010):

- Há um conjunto de **tipos básicos**, $B = \{B_1, B_2, \ldots, B_k\}$. Cada elemento B_i é um tipo atômico, ou seja, indivisível, e seu domínio, denotado por $dom\,(B_i)$ é um conjunto de constantes;
- Se $T_1, T_2, ..., T_n$ são tipos básicos ou conjuntos de tipos básicos, então $[T_1, T_2, \ldots, T_n]$ é um tipo **tupla** cujo domínio $dom\,([T_1, T_2, \ldots, T_n]) = \{[v_1, v_2, \ldots, v_n]\,|v_i \in dom\,(T_i)\}$;
- Se T é um tipo tupla, então $\{T\}$ é um tipo **conjunto** cujo domínio $dom\,(\{T\})$ é o conjunto de potência de $dom\,(T)$.

No contexto das páginas Web, que não especificam os tipos de seus dados, B_i é usualmente uma porção de texto, uma imagem, um vídeo e assim por diante. Logo, de acordo com as definições acima, a porção de código HTML contida no Código 1 poderia ser representada pelo tipo tupla contendo dados aninhados, apresentado no Código 2 e chamado de *produto*. No Código 2, *nome* e *imagemProduto*, que integram a tupla *produto*, são

tipos básicos; *precos* é uma tupla formada pelo conjunto de tipos básicos *preco* e *precoPromo*.

```
<div class="produto">
  <div class="imagemProduto"><img src="imagens/forthoseabouttorock.png" />
      Imagem do produto
  <p>
    <div class="nome"><b><h4>AC/DC ao vivo - For Those About To Rock</h4></
       b></div>
  </p>
  <div class="preco">R$199,90</div>
  <div class="precoPromo">R$109,90</div>
</div>
```

Código 1 – Porção de código HTML especificando um produto.

```
produto [
    nome: texto;
    imagemProduto: imagem;
    precos: {[ preco: ponto flutuante;
         precoPromo: ponto flutuante;]}
]
```

Código 2 – Exemplo de tupla com dados aninhados.

Sabendo-se da natureza estruturada das páginas Web e tendo em mente os tipos básicos bem como os conjuntos e tuplas definidos acima, uma página pode ser naturalmente representada como uma árvore contendo as seguintes especificações (LIU, 2011):

- Um tipo básico B_i é uma árvore formada somente por uma folha, ou então um nó;
- Um tipo tupla $[T_1, T_2, \ldots, T_n]$ é uma árvore enraizada em um nó do tipo tupla com n subárvores ou árvores filhas, uma para cada elemento T_i;
- Um tipo conjunto $\{T\}$ é uma árvore enraizada em um nó do tipo conjunto contendo uma subárvore.

Uma **instância** de um tipo T é simplesmente um elemento de $dom\,(T)$. Dessa forma, instâncias podem ser representadas como árvores de forma que (LIU, 2011):

- Uma instância (constante) de um tipo básico é uma árvore formada somente por uma folha;
- Uma instância de uma tupla $[v_1, v_2, \ldots, v_n]$ é uma árvore enraizada em um nó do tipo tupla com n nós filhos ou subárvores representando os valores dos atributos $v_1, v_2, \ldots, v_n$;

- Uma instância do tipo conjunto $\{e_1, e_2, \ldots, e_n\}$ forma um conjunto de nós com n filhos ou subárvores representando os elementos $e_1, e_2, \ldots, e_n$ do conjunto.

Uma instância de um tipo tupla é comumente chamada de **registro** (de dados) no universo da pesquisa de extração de dados. Já uma instância de um tipo conjunto é chamada de **lista**, pois em uma página Web os registros dentro de um conjunto são dispostos em uma ordem específica. O modelo de dados descrito nesta seção permite a tradução de uma página Web em uma estrutura hierarquizada na forma de uma árvore, o que torna muito mais fácil a manipulação dos componentes da página, como evidencia o modelo DOM, elucidado na próxima seção.

3.4 Modelo DOM

Quando uma página HTML é carregada pelos navegadores mais modernos é realizado um mapeamento da estrutura da página denominado **documento-objeto**. Neste mapeamento, o código HTML da página é traduzido para um objeto na forma de uma árvore, como a descrita na seção anterior, onde há um nó raiz o qual é o proprietário de todos os demais nós: elementos, textos, atributos, estilos de formatação e, inclusive, comentários (W3SCHOOLS, 2014). Dessa forma, trabalhando-se com objetos, torna-se muito mais fácil o acesso aos elementos da página, a partir da chamada de propriedades e métodos em qualquer linguagem, já que tal modelo é independente de plataforma, permitindo que programas e *scripts* dinamicamente acessem e atualizem o conteúdo, a estrutura e o estilo de documentos.

O DOM (*Document Object Model*) é uma API (*Application Programming Interface*) que implementa o mapeamento de documentos HTML e XML (*Extensible Markup Language*). Tal API foi desenvolvida com o objetivo de facilitar o acesso e a manipulação de elementos contidos nesses tipos de documentos (W3, 2014), definindo sua estrutura lógica e a forma como são acessados. Fazendo-se uso do DOM, um desenvolvedor torna-se capaz de criar documentos e navegar pela sua estrutura, além de adicionar, modificar ou excluir elementos e conteúdos. Todos os elementos presentes em uma página podem ser acessados, com pequenas restrições a casos bastante específicos, o que faz do DOM um modelo bastante popular no emprego de técnicas que precisam percorrer a estrutura de páginas HTML, como é o caso da extração de dados em páginas Web.

Para ilustrar como o DOM é construído a partir de uma página HTML, consideremos a porção de código representada pelo Código 3. Cada *tag* (<TABLE>, <TBODY>, <TR> e <TD>) é convertida em um objeto (ou nó) na árvore. A árvore que representa o DOM da página é ilustrada pela Figura 4.

```
<TABLE>
  <TBODY>
    <TR><TD>Shady Grove</TD><TD>Aeolian</TD></TR>
    <TR><TD>Over the River, Charlie</TD><TD>Dorian</TD></TR>
  </TBODY>
 </TABLE>
```

Código 3 – Porção de código HTML para a geração do DOM (NICOL et al., 2001).

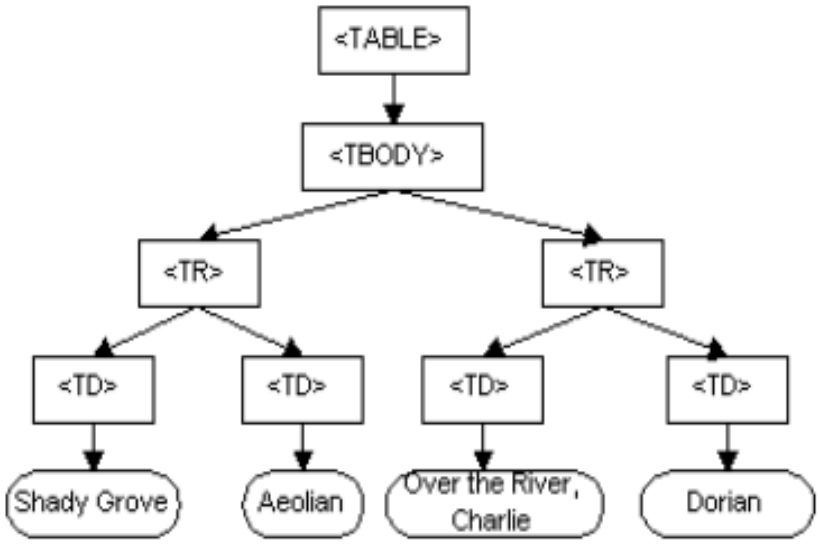


Figura 4 – DOM estruturado na forma de uma árvore (NICOL et al., 2001).

Vale ressaltar que os nós criados no exemplo da Figura 4 não representam tipos de dados, mas sim objetos os quais possuem funções e uma identidade própria, como elementos formadores de tabelas e elementos textuais. De acordo com Nicol et al. (2001), o DOM, sendo um modelo de objeto, define os seguintes elementos:

- As interfaces e objetos utilizados para representar e manipular um documento;
- A semântica de cada interface e objeto, incluindo comportamentos e atributos. É importante salientar que a referida semântica não diz respeito ao significado dos dados contidos no modelo, mas sim no papel que cada elemento desempenha dentro da estrutura, como o tipo do nó, por exemplo, que abrange nós de comentários, nós de texto, elementos de uma tabela e assim por diante;
- As relações e colaborações entre as interfaces e objetos.

Todo documento que segue o modelo DOM contém zero ou 1 nó do tipo *doctype* e zero ou mais instruções de processamento ou comentários. O nó do tipo documento atua como a raiz do documento como um todo (NICOL et al., 2001). Embora o DOM não

especifique que o modelo deva ser implementado na forma de uma árvore, haja vista que é um modelo conceitual que deve ser implementado da maneira mais conveniente, essa foi a estrutura que melhor se adaptou às proposições do modelo e é utilizada amplamente em sua construção. Uma das mais importantes propriedades do DOM é o seu **isomorfismo estrutural**. Assim, se duas implementações distintas do modelo são usadas para se criar a representação de um documento, ambas **devem** resultar no mesmo modelo estrutural.

3.5 Considerações Finais

Este capítulo teve o objetivo de sumarizar e formalizar as observações feitas a respeito da construção de páginas Web que as consolidam como formas estruturadas para a exibição de dados e que, exatamente por mostrarem-se estruturadas, permitem que técnicas de reconhecimento de padrões e mineração de dados possam ser empregadas em seus conteúdos.

O próximo capítulo visa justamente elucidar as técnicas baseadas na busca de padrões contidos nas estruturas de uma página Web para que, a partir daí, os dados de interesse possam ser extraídos dessas estruturas.

4 Extração de Padrões em Páginas Web

Este capítulo apresenta inicialmente os motivos que inspiraram o uso de técnicas de *Tree Matching* para a extração de padrões, padrões estes que remetem às regiões as quais contêm registros de dados (*Data Records*) dentro de páginas Web, representadas na forma de árvores como foi visto no Capítulo 3. Em seguida são apresentadas as técnicas de *Tree Matching*, começando-se da utilizada no trabalho de Yang (1991), chamada STM (*Simple Tree Matching*), apontando-se suas limitações, as quais contribuíram para o desenvolvimento da técnica apresentada por Jindal e Liu (2010), chamada G-STM (*Generalized Simple Tree Matching*), que é uma das bases deste trabalho e que, diferentemente de outras abordagens, consegue lidar com listas de dados. Por fim, é proposta uma melhoria ao algoritmo *Generalized Simple Tree Matching* com o intuito de aumentar sua acurácia.

4.1 Introdução ao Tree Matching

O Capítulo 3 apresentou o modelo DOM, modelo no qual as estruturas de uma página Web podem ser descritas na forma de uma árvore, também conhecida como grafo *acíclico* (pois não possui ciclos), *não orientado* (não há uma ordem específica para percorrer os vértices) e *conexo* (existe caminho entre quaisquer dois de seus vértices) (REIS et al., 2004). A técnica *Tree Matching* atua sobre um tipo particular de árvore, denominada *árvore enraizada ordenada e rotulada* (REIS et al., 2004).

Uma árvore é dita *enraizada* quando possui um único nó origem (raiz) que é fixo, ou seja, tal nó sempre será a raiz da árvore. Por sua vez, uma árvore é dita *enraizada* e *ordenada* quando é uma árvore enraizada cuja ordem relativa dos nós filhos de um nó pai é fixa, ou seja, a ordem na qual os nós filhos de um determinado nó aparecem na árvore é sempre a mesma. Já uma árvore *enraizada ordenada* e *rotulada* é aquela árvore enraizada e ordenada a qual possui um rótulo, ou um nome, associado a cada um de seus nós. Tendo em vista essas três premissas, as árvores provenientes do modelo DOM, apresentado no Capítulo 3, se encaixam perfeitamente nessa categoria de árvores e podem então ser analisadas por técnicas de *Tree Matching*. Tais árvores possuem sempre um único nó raiz (nó cuja *tag*, ou pela definição, rótulo, é *<html>*), a hierarquia entre as *tags* é sempre a mesma e, por fim, todos os nós são rotulados visto que possuem uma *tag* HTML associada a cada um deles como, por exemplo, *<body>*, *<header>*, e assim por diante.

O conceito embutido nas técnicas de *Tree Matching* é o de buscar avaliar similaridades estruturais entre regiões de uma mesma árvore enraizada ordenada e rotulada, regiões essas tomadas duas a duas, determinado-se então uma pontuação para cada par de nós similares entre si, sendo que cada nó do par pertence a uma região. A similaridade entre

esses nós é determinada principalmente pela comparação da *tag* HTML que representa cada nó, como uma * * ou um *<li>*, por exemplo. Outros elementos ou atributos, como identificadores e classes, também podem ser levados em consideração ao se determinar o quão similar um nó é em relação a outro, dependendo da implementação de *Tree Matching* utilizada. Assim, quanto mais próxima for a pontuação obtida em relação ao número total de nós de cada região avaliada, mais similar tais regiões são entre si. Tal conceito de similaridade remete-nos ao reconhecimento de padrões dentro da estrutura dessas árvores, visto que padrões são estruturas que se repetem dentro de uma estrutura maior que as engloba.

Assim, sabendo-se que páginas de comércio eletrônico utilizam *templates* (modelos) para a disposição dos produtos e que tais *templates* se repetem formando listas de dados (ou regiões de dados), o motivo da utilização do *Tree Matching* neste trabalho reside unicamente na busca por tais padrões de dados dentro da estrutura da árvore e, consequentemente, dentro da página.

A Figura 5 ilustra o objetivo do *Tree Matching* no que diz respeito à extração de padrões. A figura apresenta uma vitrine de uma página de comércio eletrônico contendo três produtos. A vitrine como um todo (representada na figura pelo contorno contínuo) corresponde a uma região de dados (*Data Region*) dentro da página que, por sua vez, poderá conter diversas outras regiões de dados como essa. Cada conjunto de dados, contendo a imagem do produto, sua descrição e os respectivos preços, constitui um registro de dado (*Data Record*), representado pelo contorno pontilhado na figura. A porção de código HTML que corresponde a essa vitrine está contida no Código 4, no qual foram suprimidos itens de estilo e formatação meramente por questões de espaço na página.

Figura 5 – Vitrine contendo três produtos. A região de dados é representada pelo contorno contínuo enquanto os registros de dados são representados pelos contornos pontilhados.

O código HTML contido no Código 4 apresenta padrões de repetição e ordenação quanto às *tags* filhas de cada *tag* *<li>*, de forma que cada produto disposto nessa vitrine na página Web é constituído pelo mesmo conjunto de *tags* que, por sua vez, estão dispostas

na mesma ordem, como pode ser observado pelos fragmentos destacados pelos contornos pontilhados no código. Cada conjunto de *tags* forma um *template* e são esses *templates* que a técnica do *Tree Matching* visa encontrar dentro da página, como será visto a partir da próxima seção.

```
1  <div id="geral_divitems">
2    <ul id="geral_ul">
3      <li crmwa\_mdc="Home|Vitrine 1|1|">\break<div class="neemurc-carousel-
         item">
4      <div class="neemurc-carousel-item-product"><div><a href="..."><img
           src="..."></a></div>
5      <div class="neemurc-carousel-item-divcontent">\break
6        <p class="title-carousel" title="Livro\ -\ Doctor\ Who:\ The\ Vault
           :\ Treasures\ From\ The\ First\ 50\ Years">
7          <a href="...">Livro-Doctor\ Who:The\ Vault:Treasures\ From\ The
             ...</a></p>
8        <div class="neemurc-carousel-item-rating"></div>
9        <div class="the-price">
10         <span class="neemurc-carousel-item-old-price">
11           <span class="neemurc-carousel-item-old-price-label">De:\ </span
               ><span class="neemurc-carousel-item-old-price-value">R$\
               179,00</span>
12         </span><br><span class="price-carousel">R$\ 102,03<span>\ no\
             boleto</span></span>
13       </div>
14       <div class="price-carousel"><span>ou\ 5x\ de\ R\$\ 21,48\ sem\
           juros</span></div>
15     </div></div></div>
16   </li>
17
18   <li crmwa\_mdc="Home|Vitrine 1|2|"><div class="neemurc-carousel-item">
19     <div class="neemurc-carousel-item-product"><div><a href="..."><img
           src="..."></a></div>
20     <div class="neemurc-carousel-item-divcontent">
21       <p class="title-carousel" title="Livro\ -\ Doctor\ Who:\ Eleventh\
           Doctor">
22         <a href="...">Livro-Doctor\ Who:Eleventh\ Doctor's\ Sonic...</a><
             /p>
23       <div class="neemurc-carousel-item-rating"><img src="..."> <span>(3)
           </span></div>
24       <div class="the-price">
25         <span class="neemurc-carousel-item-old-price">
26           <span class="neemurc-carousel-item-old-price-label">De:\ </span
               ><span class="neemurc-carousel-item-old-price-value">R\$\
               32,90</span>
27
```

```
28          </span><br><span class="price-carousel">R\$\ 27,90</span>
29        </div>
30        <div class="price-carousel"><span>ou\ 1x\ de\ R\$\ 27,90\ sem\
              juros</span></div>
31      </div></div></div>
32    </li>
33    ...
34    <li crmwa\_mdc="Home|Vitrine 1|3|"><div class="neemurc-carousel-item">
35      <div class="neemurc-carousel-item-product"><div><a href="...."><img
            src="...."></a></div>
36      <div class="neemurc-carousel-item-divcontent">
37        <p class="title-carousel" title="Livro\ -\ Doctor\ Who:\ Shada\ -\
              A\ Aventura\ Perdida\ de\ Douglas\ Adams">
38          <a href="....">Livro-Doctor\ Who:Shada-A\ Aventura\ Perdida\ de...
                </a></p>
39        <div class="...."><img src="...."> <span>(8)</span></div>
40        <div class="the-price">
41          <span class="neemurc-carousel-item-old-price">
42            <span class="neemurc-carousel-item-old-price-label">De:\ </span
                  ><span class="neemurc-carousel-item-old-price-value">R\$\
                   39,90</span>
43          </span><br><span class="price-carousel">R\$\ 19,90</span>
44        </div>
45        <div class="price-carousel"><span>ou\ 1x\ de\ R\$\ 19,90\ sem\
              juros</span></div>
46        <span>Veja\ em\ </span> <span>outras\ lojas</span></a>
47      </div></div></div>
48    </li>
49    ...
50  </ul>
51 </div>
```

Código 4 – Porção de código HTML constituindo uma vitrine que contém três produtos.

4.2 Simple Tree Matching (STM)

O algoritmo *Simple Tree Matching* (STM) foi inicialmente proposto para a comparação entre dois programas de computador no campo da engenharia de software (ZHAI; LIU, 2005). O STM avalia a similaridade entre duas árvores A e B a partir da determinação da correspondência máxima (*matching*) entre elas, fazendo uso de técnicas de **Programação Dinâmica** com complexidade $O(n_1 n_2)$, onde n_1 e n_2 são os tamanhos das árvores A e B respectivamente.

A Programação Dinâmica pode ser definida, simplificadamente, como *recursão com apoio de uma tabela* (PARBERRY, 1995). Ao invés de todos os subproblemas de um

problema maior serem resolvidos de forma recursiva, tais subproblemas são resolvidos sequencialmente e suas soluções são armazenadas em uma tabela que auxiliará na composição do resultado final (PARBERRY, 1995). A técnica da Programação Dinâmica é particularmente útil em problemas nos quais se faz necessário o conceito de "dividir para conquistar". Assim como em um algoritmo recursivo, também na Programação Dinâmica cada instância do problema é resolvida a partir da solução de instâncias menores, ou seja, de subinstâncias da instância original, mas a característica distintiva da Programação Dinâmica em relação à recursão simples é a tabela que armazena as soluções das várias subinstâncias (FEOFILOFF, 2014). O segredo, portanto, é resolver os subproblemas na ordem correta para que, sempre que a solução de um subproblema se fizer necessária, ela já esteja disponível na tabela de resultados. Um algoritmo puramente recursivo, em outras palavras, aquele que não é estruturado de forma a usar Programação Dinâmica, é tipicamente ineficiente para problemas como o apresentado neste trabalho, isso porque refaz, muitas vezes, a solução de cada subproblema, dado que cada nível da árvore necessita de comparações entre todos os nós filhos dos níveis inferiores e o resultado das comparações deve ser elevado ao nível imediatamente superior.

Três definições são importantes para o entendimento dos conceitos contidos no algoritmo *Simple Tree Matching*. São elas:

1. Definição de mapeamento entre duas árvores: sejam A e B duas árvores e i e j o *i-ésimo* e o *j-ésimo* nó de A e B, respectivamente, quando a árvore é percorrida em pré-ordem. Um mapeamento M entre a árvore A, de tamanho m, e a árvore B, de tamanho n, é um conjunto de pares ordenados (i, j), sendo um elemento de cada árvore, que satisfazem as seguintes condições, para todo $(i_1, j_1), (i_2, j_2) \in M$:

   - $i_1 = i_2$ se, e somente se, $j_1 = j_2$;
   - i_1 está à esquerda de i_2 se, e somente se, j_1 estiver à esquerda de j_2;
   - i_1 é um ancestral de i_2 se, e somente se, j_1 for um ancestral de j_2.

2. Definição de *matching* máximo: sejam A e B duas árvores e $i \in A$, $j \in B$ dois nós contidos nas árvores A e B respectivamente. O *matching* entre duas árvores é definido como o mapeamento M, como definido acima, tal que, para todo o par $(i, j) \in M$, em que i e j não são nós raízes, o par $(pai(i), pai(j)) \in M$. O *matching* máximo é o mapeamento que contém o número máximo de pares;

3. Definição do número de pares no *matching* máximo: também chamado, neste trabalho, de pontuação de *matching* ou valor de *matching*, trata-se de um refinamento da definição anterior. Assim, sejam $A = R_A : \langle A_1, A_2, ..., A_m \rangle$ e $B = R_B : \langle B_1, B_2, ..., B_n \rangle$ duas árvores enraizadas ordenadas e rotuladas, onde R_A e R_B são as raízes de A e B, e A_i, B_j são a *i-ésima* e a *j-ésima* subárvore percorridas em pré-ordem no

primeiro nível (nível logo abaixo da raiz) de A e B respectivamente. Seja $W_{A,B}$ o número de pares no *matching* máximo entre as árvores A e B. Se R_A e R_B contiverem símbolos idênticos, $W_{A,B}$ é dado por $m(\langle A_1, A_2, ..., A_m\rangle, \langle B_1, B_2, ..., B_n\rangle) + 1$, em que $m(\langle A_1, A_2, ..., A_m\rangle, \langle B_1, B_2, ..., B_n\rangle)$ é o número de pares no *matching* máximo entre $\langle A_1, A_2, ..., A_m\rangle$ e $\langle B_1, B_2, ..., B_n\rangle$. Se R_A e R_B contiverem símbolos distintos, $W_{A,B} = 0$.

Assim, a determinação do valor de $W_{A,B}$ pode ser resumida da seguinte maneira, com uso da Programação Dinâmica:

$$W_{A,B} = \begin{cases} 0 & \text{se } R_A \neq R_B \\ m(\langle A_1, A_2, ..., A_m\rangle, \langle B_1, B_2, ..., B_n\rangle) + 1 & \text{caso contrário;} \end{cases}$$

Já m, por sua vez, é obtido da seguinte maneira:

$$m(\langle\rangle, \langle\rangle) = 0;$$
$$m(s, \langle\rangle) = m(\langle\rangle, s) = 0;$$

$$\begin{aligned} &m(\langle A_1, A_2, ..., A_m\rangle, \langle B_1, B_2, ..., B_n\rangle) = \\ &\quad max(m(\langle A_1, A_2, ..., A_{m-1}\rangle, \langle B_1, B_2, ..., B_{n-1}\rangle) + W_{A_m,B_n}, \\ &\qquad m(\langle A_1, A_2, ..., A_m\rangle, \langle B_1, B_2, ..., B_{n-1}\rangle), \\ &\qquad m(\langle A_1, A_2, ..., A_{m-1}\rangle, \langle B_1, B_2, ..., B_n\rangle)). \end{aligned}$$

O Algoritmo 1 foi desenvolvido tomando-se como base as definições acima, nas quais os referidos símbolos são as *tags* HTML que rotulam cada nó. As raízes das árvores A e B são comparadas primeiramente (linha 2). Se as raízes contiverem símbolos distintos ($R_A \neq R_B$), então as árvores não são correspondentes (não há *matching*) e $W_{A,B} = 0$. Por outro lado, se as raízes contiverem símbolos idênticos, então o algoritmo encontra, recursivamente, o número de pares no *matching* máximo entre as subárvores filhas no nível logo abaixo das raízes de A e B e o salva na matriz W (linha 16), que faz o papel da tabela na Programação Dinâmica.

A matriz auxiliar M é criada no algoritmo com uma linha e uma coluna extra, que são preenchidas com valores zero (linhas 8 a 13). Isso se dá meramente por uma questão de simplificação na implementação do algoritmo, a fim de que fosse evitado o uso de estruturas condicionais extras na função *max* para a obtenção do *matching* máximo (linha 16). A matriz M funciona como um artifício para que o número de pares que formam o *matching* máximo entre os nós comparados vá sendo acumulado da esquerda para a direita, de cima para baixo, conforme for sendo calculado (linha 16) em cada interação, até que

chegue ao último elemento de M ($M[m, n]$), o qual conterá, consequentemente, o valor do *matching* máximo dentre todas as subárvores filhas de A e B, considerando que A e B possuam m e n subárvores filhas, respectivamente.

Algoritmo 1: SimpleTreeMatching(A, B)

```
Entrada: árvores A e B
Saída: pontuação de similaridade entre as árvores A e B
1  início
2      se raízes de A e B contêm símbolos distintos então
3          retorna 0;
4      fim
5      senão
6          m ← número de subárvores filhas no primeiro nível de A;
7          n ← número de subárvores filhas no primeiro nível de B;
8          para i=0 até m faça
9              M[i, 0] ← 0;
10         fim
11         para j=0 até n faça
12             M[0, j] ← 0;
13         fim
14         para i=1 até m faça
15             para j=1 até n faça
16                 M[i, j] ← max(M[i, j − 1], M[i − 1, j], M[i − 1, j − 1] + W[i, j]);
17                 onde W[i, j] ← SimpleTreeMatching(A_i, B_j)
18             fim
19         fim
20         retorna M[m, n] + 1;
21     fim
22 fim
```

Logo, dados dois nós contidos em níveis equivalentes nas árvores A e B, a matriz W armazenará, para cada nível da recursão (o que equivale ao nível da árvore em que a análise está sendo feita), o valor do *matching* máximo calculado para cada um dos pares de subárvores $(\langle A_i \rangle, \langle B_j \rangle)$ filhas de A e B, onde $0 \leq i < n$ e $0 \leq j < m$, para que, então, o algoritmo selecione aqueles mais similares a fim de que componham o *matching* máximo de A e B, o que equivale ao cálculo de m na terceira definição. O Algoritmo 1 é chamado para cada nível das árvores de entrada, até que o nível com os nós folha seja atingido, haja vista que para se obter o *matching* máximo de uma subárvore é necessário primeiramente obter o *matching* máximo das suas filhas, das filhas dessas filhas e assim por diante até o nível das folhas. Quando tal fato ocorre, os resultados começam a ser retirados da pilha de recursão e são devolvidos para o nível imediatamente acima, a fim de serem armazenados na matriz W do referido nível.

Nota-se que o algoritmo não prevê comparações entre nós de níveis diferentes, tampouco permutações de nós nos mesmos níveis, ou seja, o algoritmo foca-se exclusivamente

em computar a similaridade entre duas árvores e não a quantidade de modificações que uma árvore deve sofrer a fim se que se transforme em outra por meio de operações de inserção, permutação ou exclusão de nós, o que elevaria a complexidade de tempo do algoritmo para $O(n_1 n_2 h_1 h_2)$ (ZHAI; LIU, 2006) contra uma complexidade de tempo na ordem de $O(n_1 n_2)$ obtida pelo STM, onde n_1 e n_2 são os números de nós das árvores A e B e h_1 e h_2 são as profundidades máximas das árvores A e B.

4.2.1 Exemplo de execução do Simple Tree Matching

A fim de simular a execução do *Simple Tree Matching*, são utilizadas como exemplos as árvores A e B representadas pela Figura 6, que possuem como suas raízes os nós N1 e N15, respectivamente.

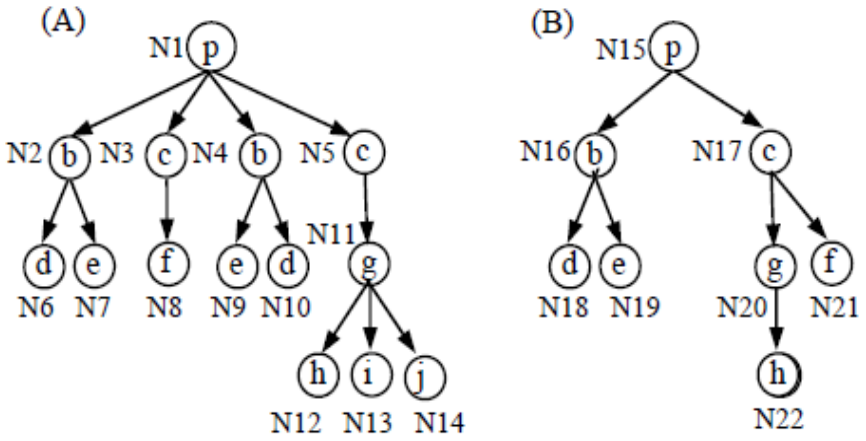


Figura 6 – Árvores A e B como exemplo de execução do algoritmo STM (ZHAI; LIU, 2005).

Para determinar o *matching* máximo entre as duas árvores (A e B), suas raízes N1 e N15 são as primeiras a serem comparadas e, como possuem símbolos idênticos, $M_{1,15}[4,2]+1$ é retornado como o valor do *matching* máximo entre as árvores A e B (linha 20 do Algritmo 1), onde 4 e 2 são as quantidades de subárvores filhas das raízes N1 e N15. No Algoritmo 1, linha 16, a matriz $M_{1,15}$ (representada pela Tabela 2) é computada com base na matriz $W_{1,15}$ (representada pela Tabela 3), em que cada entrada $W_{1,15}[i,j]$, por sua vez, é o *matching* máximo entre a *i-ésima* e a *j-ésima* subárvore filha de N1 e N15, respectivamente, no nível logo abaixo das raízes. As matrizes representadas pelas Tabelas 4 a 7 mostram o processo recursivo de determinação do *matching* entre as subárvores filhas de N1 e N15 até o nível dos nós folha, nos quais as células mais relevantes estão sombreadas. Nota-se que as linhas e colunas com valores iguais a zero nas tabelas que representam as matrizes M são provenientes das inicializações efetuadas pelas linhas 9 e 12 no Algoritmo 1.

Tabela 2 – Matriz M para cálculo do *matching* entre os nós 1 e 15.

	0	N16	N16 a N17
0	0	0	0
N2	0	3	3
N2 a N3	0	3	5
N2 a N4	0	3	5
N2 a N5	0	3	6

Tabela 3 – Matriz W com as comparações entre os nós N2-N16, N2-N17, N3-N16, N3-N17, N4-N16, N4-N17, N5-N16 e N5-N17, para o cálculo dos valores da Tabela 2.

	N16	N17
N2	3	0
N3	0	2
N4	2	0
N5	0	3

Tabela 4 – Matriz M para cálculo do *matching* entre os nós 5 e 17.

	0	N20	N20 a N21
0	0	0	0
N11	0	2	2

Tabela 5 – Matriz W com as comparações entre os nós N11-N20 e N11-N21, para o cálculo dos valores da Tabela 4.

	N20	N21
N11	2	0

Tabela 6 – Matriz M para cálculo do *matching* entre os nós 11 e 20.

	0	N22
0	0	0
N12	0	1
N12 a N13	0	1
N12 a N14	0	1

Tabela 7 – Matriz W com as comparações entre os nós N12-N22, N13-N22 e N14-N22, para o cálculo dos valores da Tabela 6.

	N22
N12	1
N13	0
N14	0

Sabendo-se que a pilha de recursão do Algortimo 1 começa a ser esvaziada sempre que um nível de nós folha é atingido, são tomadas como ponto de partida, a fim de exemplificação, as subárvores N12, N13 e N14, filhas da subárvore cuja raiz é o nó N11 e a subárvore N22, filha da subárvore cuja raiz é N20. O algoritmo realiza as seguintes comparações entre os nós, sempre formando pares entre nós de árvores distintas, ou seja, um nó filho da subárvore N11 e um nó filho da subárvore N20: N12-N22, N13-N22 e N14-N22. O resultado dessas comparações está disposto na Tabela 7. Como somente os nós N12 e N22 possuem *tags* idênticas, a célula que representa essa comparação recebe valor 1, sendo que as demais recebem valor zero. Os valores dispostos na Tabela 7 são então utilizados para a composição da matriz M, representada pela Tabela 6, por meio da formulação apresentada na linha 16 do Algoritmo 1. Como houve similaridade somente entre os nós N12 e N22, o valor do *matching* máximo entre as subárvores filhas de N11 e N20 é igual a 1, como explicita a célula sombreada pertencente à Tabela 6.

Com o término da execução do algoritmo nesse nível da recursão, o valor do *matching* máximo recém calculado é então transportado para o nível imediatamente acima, sendo acrescido de 1 unidade, já que além da similaridade calculada entre os filhos de N11 e N20, os próprios nós N11 e N20 são similares (linha 20 do algortimo), como evidencia a célula sombreada na matriz representada pela Tabela 5, matriz essa que mostra as comparações entre os nós filhos de N5 e N17. Tal esquema de programação deixa claro que as comparações entre os nós N5 e N17 devem ser efetuadas somente após serem feitas as comparações entre os filhos desses nós pois, caso contrário, não haveria uma forma de se saber quais os valores do *matching* desses filhos, fato esse que justifica o emprego da recursão no algoritmo.

Com a Tabela 5 devidamente preenchida, calcula-se então o *matching* máximo entre os nós N5 e N17. Primeiramente, verifica-se o *matching* entre os nós N11 e N20, que já foi calculado e, de acordo com a Tabela 5, possui valor 2. Tal valor é então carregado para a céula imediatamente à direita e, como o *matching* entre os nós N11 e N21 possui valor zero (os nós não são similares entre si), o valor do *matching* máximo entre os filhos de N5 e N17 é 2, o que é verdade, já que somente os pares de nós (N12, N22) e (N11, N20) são similares.

Da mesma forma como foi feito para os nós N11 e N20, o valor do *matching* máximo entre N5 e N17 é trasportado para a matriz W do nível de recursão imediatamente acima, acrescido de 1 unidade (célula sombreada N5-N17 na Tabela 3). As demais células apresentadas na Tabela 3 que possuem valor diferente de zero são provenientes da comparação entre os nós N2 e N16 (valor 3), N3 e N17 (valor 2) e N4 e N16 (valor 2), calculados da mesma forma que os demais valores apresentados até aqui.

O valor final do *matching* máximo, ou seja, o número de pares no *matching* máximo entre as árvore A e B, como evidencia a célula sombreada na matriz representada pela Tabela 2, é igual a 6. Vale ressaltar que, apesar das subárvores N3 e N17 possuírem certa similaridade entre seus nós, o objetivo do algoritmo é calcular o *matching* **máximo** entre subárvores, de forma que, como o *matching* máximo da subárvore cuja raiz é N5 já foi obtido na comparação com a subárvore cuja raiz é N17, o *matching* entre N3 e N17 não faz parte do *matching* final, caso contrário estaria sendo computado em duplicidade. Outro ponto a se considerar é que o *matching* deve ser computado apenas uma vez para cada subárvore em relação àquela que for mais similar a ela. Por isso, o *matching* entre as subárvores N4 e N16 não é levado em consideração no cálculo do *matching* máximo, pois ele já foi considerado durante a comparação entre N2 e N16 (valor 3 é maior que valor 2).

4.2.2 Limitações do Simple Tree Matching

Antes das limitações do algoritmo *Simple Tree Matching* serem discutidas, o conceito de **listas** no contexto das páginas Web será definido. Listas são denominadas

como repetições sucessivas e contíguas de um mesmo padrão, ou seja, de um mesmo *template*. Basicamente, as listas seguem o mesmo conceito de regiões de dados, que já foram definidas anteriormente. Esta seção emprega tal terminologia devido à literatura especializada se referir desta maneira no que diz respeito às limitações que são discutidas aqui.

Apesar do algoritmo *Simple Tree Matching* ter sido desenvolvido com a finalidade de encontrar padrões em árvores e listas não serem nada além do que repetições sucessivas de padrões, o tamanho variável de listas dentro de uma página prejudica o resultado desse algoritmo caso algumas considerações extras não sejam adicionadas. As árvores A e B da Figura 7 são tomadas como exemplos para elucidar o problema. Se o algoritmo *Simple Tree Matching* fosse aplicado somente da maneira como foi descrito na Seção 4.2, a pontuação de *matching* obtida a partir da comparação entre as duas subárvores ilustradas na figura não refletiria a realidade.

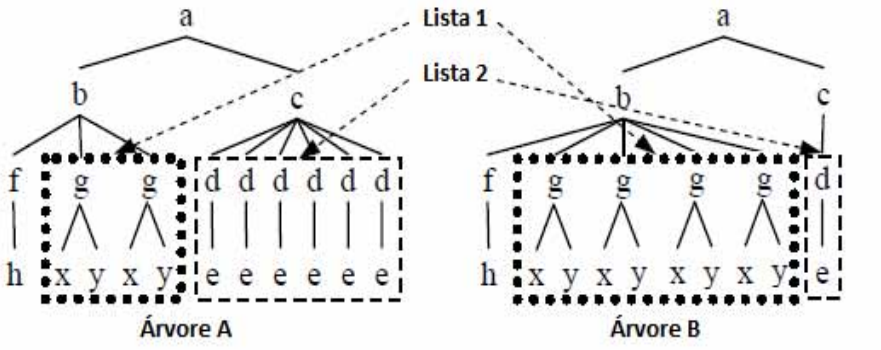


Figura 7 – Árvores A e B exemplificando as limitações do algoritmo STM na detecção de listas. Adaptado de Jindal e Liu (2010).

Facilmente é possível observar que os nós filhos de b tanto da árvore A quanto da árvore B constituem padrões. O mesmo ocorre com os nós filhos de c. As árvores A e B possuem os mesmos padrões em suas listas, contudo a quantidade de itens pertencentes a essas listas não é a mesma. O algoritmo apresentado na Seção 4.2 conseguiria correlacionar, por exemplo, somente os dois primeiros conjuntos de nós filhos de b das duas subárvores, mas não os dois últimos filhos de b pertencentes à árvore B, já que estes últimos não possuem itens relacionados na mesma posição na árvore A. Tal fato, influenciaria negativamente na pontuação obtida já que, se expandirmos tal conclusão também aos filhos de c, de um total de 23 nós na árvore A e 19 nós na árvore B, obteríamos uma pontuação igual a 13, o que corresponde a aproximadamente 57% de *matching* em relação à árvore com maior número de nós, indicando baixa similaridade, já que mais de 40% dos nós ficaram de fora do *matching*.

A próxima seção apresenta o algoritmo *Generalized Simple Tree Matching*, desen-

volvido com o objetivo de sanar as deficiências do algoritmo *Simple Tree Matching* quanto à detecção de listas.

4.3 Generalized Simple Tree Matching (G-STM)

O algoritmo *Generalized Simple Tree Matching*, proposto por Jindal e Liu (2010), tem o objetivo de adaptar o algoritmo *Simple Tree Matching* para que o tamanho variável de listas em páginas Web não influencie negativamente na pontuação obtida durante a comparação entre duas árvores construídas a partir dessas páginas.

Sendo assim, é necessário incluir no algoritmo *Simple Tree Matching*, apresentado na Seção 4.2, uma função responsável pelo tratamento de listas com tamanhos variáveis, logo após o término do cálculo das pontuações de cada conjunto de subárvores filhas de duas árvores A e B dadas como entrada. São essas pontuações preliminares que fornecem os subsídios necessários à função de detecção de listas a fim de que ela faça as devidas adaptações às pontuações finais. A estratégia para a detecção de listas emprega uma técnica de construção de expressões regulares que generalizam os nós pertencentes às listas, incluindo símbolos opcionais e representações de repetições, mantendo somente uma instância (elemento) da lista que fará o papel de representante das demais, para que seus tamanhos variáveis não sejam mais um empecilho à detecção de padrões.

O Algoritmo 2 mostra o pseudo-código do novo algoritmo, que é chamado de "*Generalized*" *Simple Tree Matching* porque é capaz de lidar com quaisquer tamanhos de regiões de dados, **generalizando** as listas contidas nas páginas Web. No mais, o algoritmo *Generalized Simple Tree Matching* funciona da mesma forma que o *Simple Tree Matching*, comparando pares de subárvores e determinando suas pontuações de similaridade por meio do emprego da Programação Dinâmica. Percebe-se que a linha 19 do Algoritmo 2 faz chamada a uma função especializada na detecção de listas, a qual faz uso da matriz W como forma da função detectar onde se encontram as listas dentro da estrutura das árvores que estão sendo comparadas. A matriz auxiliar M, por sua vez, utiliza agora a matriz W após os ajustes das pontuações, a fim de que o cálculo do número de pares no *matching* máximo já seja feito com as pontuações corretas. Observa-se também que o algoritmo retorna agora, além da pontuação do *matching* entre duas subárvores, dois outros valores: $nodes_A$ e $nodes_B$. Os dois novos valores, que também são armazenados na matriz W, refletem a quantidade normalizada de nós, que é utilizada na detecção das listas, como será visto adiante.

4.3.1 A Função de Detecção de Listas

A função de detecção de listas foi a principal inovação inserida no algoritmo STM para que ele se tornasse o G-STM. A ideia básica dessa função, apresentada no Algoritmo

Algoritmo 2: GeneralizedSimpleTreeMatching(A, B)

Entrada: árvores A e B
Saída: pontuação de similaridade entre as árvores A e B, quantidade de nós na árvore A e quantidade de nós na árvore B

```
1  início
2      se raízes de A e B contêm símbolos distintos então
3          retorna (0, A.nodes, B.nodes);
4      fim
5      senão
6          m ← número de subárvores filhas no primeiro nível de A;
7          n ← número de subárvores filhas no primeiro nível de B;
8          para i=0 até m faça
9              M[i, 0] ← 0;
10         fim
11         para j=0 até n faça
12             M[0, j] ← 0;
13         fim
14         para i=1 até m faça
15             para j=1 até n faça
16                 W[i, j] ← GeneralizedSimpleTreeMatching(W, A_i, B_j);
17             fim
18         fim
19         (W, nodes_A, nodes_B) ← DetectaListas(W, A, B);
20         para i=1 até m faça
21             para j=1 até n faça
22                 M[i, j] ← max(M[i, j − 1], M[i − 1, j], M[i − 1, j − 1] + W[i, j].score);
23             fim
24         fim
25         retorna (M[m, n] + 1, nodes_A, nodes_B);
26     fim
27 fim
```

3, é utilizar a matriz W, que contém o valor do *matching* proveniente das comparações entre todas as subárvores filhas de uma árvore A com as filhas de uma árvore B, para determinar quais subárvores de A são similares às de B. A cada subárvore filha de A similar a uma filha de B é atribuído um símbolo único e a cada subárvore distinta atribui-se um símbolo diferente. O resultado dessa atribuição são duas cadeias de símbolos, uma para as subárvores filhas de A e outra para as subárvores filhas de B. Ambas as cadeias são utilizadas na geração de gramáticas, que auxiliarão na detecção de listas.

Um ponto a ser ressaltado no algoritmo é o conceito de similaridade entre subárvores aplicado na linha 16, o qual determina se um par de subárvores receberá ou não um mesmo símbolo. Tal conceito difere do conceito de similaridade aplicado aos Algoritmos 1 e 2, que consideram somente as *tags* HTML contidas nos nós analisados e determinam as pontuações que serão armazenas em W. Na atribuição de símbolos, além das *tags* serem

Algoritmo 3: DetectaListas(W, A, B)

Entrada: matriz W, árvores A e B

Saída: matriz W ajustada, quantidade ajustada de nós da árvore A e quantidade ajustada de nós da árvore B

```
1  início
2      k ← número de linhas da matriz W;
3      n ← número de colunas da matriz W;
4      Define-se uma matriz temporária W_norm, contendo k linhas e n colunas em que
       W_norm[i, j] contém a pontuação normalizada da i-ésima subárvore filha de A
       (S_Ai) e da j-ésima subárvore filha de B (S_Bj).
5      para i=1 até k faça
6          para j=1 até n faça
7              W_norm[i, j] ← W[i, j].score/max(W[i, j].nodes_A, W[i, j].nodes_B);
8          fim
9      fim
10     para i=1 até k faça
11         para j=1 até n faça
12             score_norm ← W_norm[i, j];
13             max_Ai ← max(W_norm[i, 1..n]);
14             max_Bj ← max(W_norm[1..k, j]);
15             // max_Ai(max_Bj) é a pontuação normalizada máxima que a subárvore
               S_Ai (S_Bj) possui com qualquer subárvore filha da árvore B (A)
16             se score_norm > τ1 e score_norm/max_Ai > τ2 e score_norm/max_Bj > τ2
               então
17                 S_Ai e S_Bj são consideradas similares, então recebem um mesmo
                   símbolo
18             fim
19         fim
20     fim
21     para todo S_Ai e S_Bj que não receberam símbolos faça
22         S_Ai e S_Bj recebem símbolos distintos
23     fim
24     string1 ← cadeia formada pelos símbolos dados às subárvores filhas de A;
25     string2 ← cadeia formada pelos símbolos dados às subárvores filhas de B;
26     g_A ← GeraGramatica(string1);
27     g_B ← GeraGramatica(string2);
28     se g_A possui os mesmos padrões "+" que g_B então
29         retorna (W, nodes_A, nodes_B) ← AtualizaW(W, g_A, g_B);
30     fim
31     senão
32         retorna (W, A.nodes, B.nodes);
33     fim
34 fim
```

idênticas, são utilizadas também pontuações normalizadas, calculadas e armazenadas na matriz W_{norm} (linha 7), para que discrepâncias resultantes de diferentes tamanhos de subárvores sejam atenuadas. Entende-se por pontuações normalizadas a razão entre a pontuação (*matching*) armazenada em W na posição referente ao par comparado e o número máximo de nós dentre as subárvores que formam o par. Cada posição da matriz W terá uma posição correspondente contendo a pontuação normalizada na matriz W_{norm}.

O Algoritmo 3 pode ser dividido em duas partes. A primeira, que engloba as linhas 2 a 25, é especializada em gerar as cadeias de caracteres a partir das subárvores filhas de cada árvore fornecida como entrada, dispondo os símbolos atribuídos a cada subárvore lado a lado na ordem em que aparecem. A segunda, que se estende da linha 26 até a linha 33, tem o objetivo de construir as gramáticas que descrevem as cadeias de caracteres geradas pela primeira parte e então analisar as expressões regulares que representam as gramáticas a fim de saber se há equivalências entre elas. A geração das expressões regulares será abordada na próxima seção.

Assim, para que duas subárvores recebam o mesmo símbolo, declarando-as similares, o algoritmo de detecção de listas de Jindal e Liu (2010) especifica dois limiares (*thresholds*), τ_1 e τ_2, ambos utilizados pela linha 15.

O limiar τ_1, fixado pelos autores do algoritmo em 50%, define o *score* ou a pontuação normalizada mínima que duas subárvores devem possuir em relação àquela que detém o maior número de nós do par. Por exemplo, se a matriz W indica que duas subárvores, uma contendo três nós e outra contendo quatro nós, foram comparadas e receberam pontuação de valor três, indicando que três pares de nós dessas árvores são idênticos, a pontuação normalizada calculada e armazenada na matriz W_{norm} para essas subárvores será de 75% ($W[i,j].score/max(W[i,j].nodes_A, W[i,j].nodes_B) = 3/4 = 0.75$). Isso se faz necessário para que se garanta um nível de qualidade mínimo, indicando que pelo menos 50% dos nós presentes em uma lista formada pelas subárvores filhas de uma árvore A estejam também presentes na outra lista formada pelas subárvores filhas de uma árvore B.

O limiar τ_1 possui um valor considerado baixo (50%), mas que deve ser mantido dessa forma devido às características dos nós que constituem as listas em páginas Web. Tais nós contendo *tags* como, por exemplo, *<li>, <tr>, <td>*, entre outras, constituem árvores com profundidades pequenas (empiricamente percebeu-se que não costumam ultrapassar três nós) e a inserção de um único nó opcional abaixo dessas *tags*, por exemplo, pode fazer com que o valor da pontuação normalizada armazenada caia consideravelmente. Na prática, o valor de 50% dado ao limiar supre de forma consistente o objetivo proposto a ele desde que um segundo limiar (τ_2) também seja adotado a fim de ser usado em conjunto com esse primeiro.

Ao se estipular um limiar de 50% entre a relação do *matching* de duas subárvores com seus números de nós, como foi definido acima, abre-se a possibilidade de que subárvores que

não formam listas sejam consideradas listas, recebendo um mesmo símbolo na construção da cadeia de caracteres que será usada na geração da gramática. Para atenuar esse problema, um segundo limiar (τ_2) foi estabelecido por Jindal e Liu (2010) e teve seu valor determinado empiricamente e fixado no algoritmo em 70%. Esse segundo limiar garante que somente as subárvores filhas de A e B que possuírem o mínimo de 70% de nós idênticos quando comparados às subárvores de maior *matching* dentre todos os filhos de A e B estejam habilitadas a receberem o mesmo símbolo que elas. Este critério é estabelecido baseado na observação de que, se uma subárvore forma uma lista com outras subárvores, então sua pontuação com a subárvore cuja comparação rendeu a pontuação máxima deve ser muito próxima das pontuações obtidas na comparação com todas as demais subárvores que também formam essa mesma lista, pois se tais subárvores não fossem bastante similares entre si, não formariam a lista.

Por exemplo, uma subárvore A_i filha de uma árvore A é comparada a uma subárvore B_j filha de uma árvore B a fim de se determinar se receberão ou não um mesmo símbolo. A pontuação relativa ao número de nós armazenada na matriz W_{norm} é maior que 0.5 o que garante certa similaridade mas, como já foi visto, não é suficiente. Faz-se necessário, então, encontrar uma forma de "unir" essas subárvores e todas as demais subárvores que porventura formarem listas sob o mesmo símbolo tanto para as subárvores filhas de A quanto para as filhas de B. A maneira encontrada foi a eleição de subárvores que representem um ponto comum entre as subárvores candidatas a receberem um mesmo símbolo. São então eleitas duas subárvores, uma dentre as filhas de A, aquela que possui a maior pontuação normalizada em relação à A_i (max_{A_i} no Algoritmo 3) e outra, dentre as filhas de B, aquela que possui a maior pontuação normalizada em relação à B_j (max_{B_j} no Algoritmo 3). Dessa forma, se o quociente formado pela pontuação normalizadas de A_i e max_{A_i} e o quociente formado pelas pontuações normalizadas de B_j e max_{B_j} for maior que o limiar τ_2, ambas as subárvores recebem o mesmo símbolo. O algoritmo garante que, no decorrer de suas iterações, as demais subárvores que forem similares à A_i e B_j e, consequentemente, similares à max_{A_i} e max_{B_j}, também recebam esse mesmo símbolo.

Ambos os limiares são necessários porque, embora os elementos contidos nas listas compartilhem o mesmo *template*, pode haver itens opcionais dentro dos *templates*, aparecendo em certos elementos da lista mas em outros não. O limiar τ_2, por exemplo, permite que até 30% de itens não contidos no *template*, que possui grande chance de ser a subárvore com maior *matching*, seja tolerado.

4.3.2 Construção das Gramáticas para a Detecção de Listas

O Algoritmo 3 utiliza duas cadeias de caracteres construídas a partir da apuração das similaridades entre subárvores para gerar gramáticas regulares que descrevem a linguagem dessas cadeias e, posteriormente, utiliza as expressões regulares deduzidas a

partir dessas gramáticas para a detecção de listas.

A fim de que os termos *linguagem*, *gramática* e *expressões regulares* não sejam simplesmente citados sem critério algum, esta seção aborda brevemente o significado de cada um deles. Segundo Partee et al. (1990 apud REAL; ANDRADE, 2011) o conceito de *linguagem* é matematicamente definido como sendo um conjunto finito de palavras, morfemas ou alguma sequência finita de caracteres, ou seja, é um conjunto finito de cadeias de caracteres. Uma *gramática*, por sua vez, de acordo com Real e Andrade (2011) é definida como um modelo ou técnica utilizada para gerar uma linguagem, em outras palavras, é a gramática quem vai ditar as regras que determinarão quais são as cadeias de caracteres que fazem e aquelas que não parte da linguagem. Já as *expressões regulares*, de acordo com Ramos (2008), são notações alternativas utilizadas para representar a classe das linguagens regulares simbolizando, de forma mais concisa, as regras que compõem a gramática. O termo "regulares" que pode estar associado a linguagens, gramáticas e expressões refere-se a uma classe de linguagens mais simples (RAMOS, 2008), que são reconhecidas ou geradas a partir de algoritmos com baixa complexidade, sem uso de estruturas auxiliares de memória.

As expressões regulares fazem uso de operadores aplicados às palavras, como os quantificadores "?", "*" e "+", que indicam a repetição da palavra zero ou uma vez, zero ou mais vezes e uma ou mais vezes, respectivamente; o operador "." indica concatenação, mas pode ser omitido; o operador "|" indica alternativas para a escolha da palavra que o precede. Além disso, usa-se parênteses para a indicação de agrupamentos ou precedência de operações.

Agora que os principais conceitos teóricos os quais serão abordados nesta seção já foram apresentados, parte-se para a explicação de como eles ajudarão na detecção de listas, sendo que culminarão na geração de expressões regulares. Sabendo-se que uma lista é uma sequência de registros de dados que seguem um mesmo *template* D, pode-se então dizer que a expressão regular que a representa será $(D)^+$ (JINDAL; LIU, 2010). A expressão regular que descreve uma lista, de acordo com Jindal e Liu (2010), pode ser definida da seguinte maneira:

- Existe um conjunto de símbolos atômicos, $S = S_1, S_2, ..., S_n$;
- Se $D_i \in D_1, D_2, ..., D_k$ é um símbolo atômico ou uma expressão regular de uma lista, então $D_1 D_2 ... D_k$ é uma expressão regular que define um registro,
- Se D é uma expressão regular de um registro, então $(D)^+$ é uma expressão regular de uma lista.

Cada símbolo presente na expressão regular representa uma subárvore e não um item de dados a ser extraído, que sempre estará em um nó folha. O primeiro passo para

a construção da gramática e, consequentemente, da expressão regular que representa as subárvores, é a construção de um *Autômato Finito Não Determinístico* (AFN) que reconheça a linguagem. Por exemplo, supõe-se que o algoritmo de detecção de listas tenha gerado a seguinte cadeia de símbolos para as subárvores filhas de uma árvore qualquer: *abaab*. Os passos para a construção do Autômato Finito Não Determinístico que reconhece essa cadeia estão representados na Figura 8.

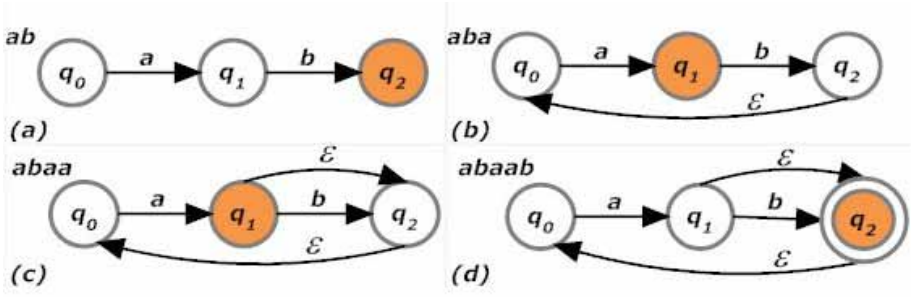


Figura 8 – Passos para a geração do AFN a partir da cadeia de símbolos *abaab*. Os símbolos à esquerda representam o que já foi processado e os estados sombreados representam o estado corrente (JINDAL; LIU, 2010).

Um autômato finito M, como o da Figura 8, é determinado por uma quíntupla $M = (Q, \Sigma, \delta, q0, F)$. O autômato da figura possui três estados, sendo que o conjunto de estados, representado na quíntupla por Q, é $Q = \{q_0, q_1, q_2\}$. O estado inicial é q_0 e o conjunto F de estados aceitos (estados finais) é $F = \{q_2\}$. O conjunto de símbolos reconhecidos pelo autômato é formado por todos os símbolos distintos presentes na cadeia de símbolos de entrada e, no exemplo da Figura 8, será $\Sigma = \{a, b, \epsilon\}$. As setas direcionadas que interligam os estados na figura representam as transições aceitas (símbolo δ na quíntupla) formando, portanto, as regras da gramática. Vale ressaltar que o símbolo ϵ representa uma transição vazia, ou seja, uma transição utilizada somente para forçar uma mudança de estado, sem o emprego de um dos símbolos presentes na cadeia de caracteres de entrada.

Nota-se que na forma como os autômatos finitos reconhecedores das linguagens produzidas pelas subárvores filhas de uma árvore qualquer são construídos, todas as disjunções (transições com símbolos diferentes que saem de um mesmo estado e que levam a um mesmo estado) são do tipo $(b|\epsilon)$ e nunca do tipo $(a|b)$, com dois símbolos diferentes de ϵ. Esse tipo de transição, quando traduzidas para expressões regulares, se tornarão expressões do tipo $(b^?)$ as quais denotam que o símbolo b do exemplo é opcional. Expressões regulares desse tipo são chamadas na literatura especializada de *union-free regular expressions* (JINDAL; LIU, 2010) ou expressões regulares livres de uniões.

Notam-se também outras características interessantes quando os autômatos construídos para o ambiente de atuação dos algoritmos deste trabalho, que são as páginas Web, são analisados mais a fundo. Tais autômatos possuem sempre um único estado inicial e um

único estado final. Além disso, os autômatos são gerados sempre a partir de uma linguagem fixa, já que para cada conjunto de subárvores filhas de uma subárvore qualquer, uma nova cadeia de caracteres é gerada e essa cadeia é única. Essas características, somadas aos tipos de disjunções que tais autômatos apresentam, nos permitem concluir que, apesar de o algoritmo de detecção de listas construir um Autômato Finito Não Determinístico, o resultado final possui traços de determinismo, que tornam o autômato menos complexo quanto à sua tradução para expressões regulares (RAMOS, 2008). A expressão regular inferida a partir do exemplo da Figura 8 será: $(ab^?)^+$.

A geração das expressões regulares tem o único objetivo de auxiliar na determinação de equivalências entre cadeias de caracteres geradas a partir da atribuição de símbolos às subárvores filhas de duas árvores dadas como entrada. A equivalência entre as cadeias de caracteres é determinada baseando-se nos padrões de repetições contidos nas expressões regulares geradas a partir delas, que serão sempre assinalados por grupos marcados com o sinal "+", indicando repetições de uma ou mais vezes de um padrão nessas cadeias de caracteres. Por exemplo, se duas árvores geraram para suas subárvores filhas as cadeias de caracteres *eaaabcccd* e *aaaaabccd*, as expressões regulares geradas para tais cadeias serão $e^?a^+bc^+d$ e a^+bc^+d, respectivamente. Isolando-se as partes que representam as listas nas expressões regulares (repetições de padrões), temos a^+ e c^+ para ambas, indicando que regiões contendo *templates* e, portanto, registros de dados, foram encontradas.

4.3.3 Ajustes nas Pontuações dos Itens Pertencentes às Listas

A fim de que listas com quantidades diferentes de itens não influenciem negativamente na determinação do *matching* entre duas subárvores, como foi visto na Seção 4.2.2, as pontuações e os números de nós contidos na matriz W devem ser ajustados.

O modo escolhido para tratar as listas, tanto neste trabalho quanto no trabalho de Jindal e Liu (2010), é manter somente uma instância do *template* que se repete, revisando as linhas e colunas correspondentes a todas as subárvores que integram a lista na matriz W. Isso é feito calculando-se, para cada lista, a média da pontuação obtida com a comparação de uma instância na primeira árvore com todas as instâncias na segunda árvore. As entradas na matriz W correspondentes à primeira instância da lista em ambas as árvores são atualizadas com a pontuação recalculada, enquanto as demais linhas e colunas correspondentes às outras instâncias da lista recebem valor zero.

O Algoritmo 4 demonstra como as novas pontuações são calculadas e como as entradas são atualizadas na matriz W.

Algoritmo 4: AtualizaW(W, g_A, g_B)

Entrada: matriz W, expressões regulares provenientes das gramáticas geradas para as subárvores filhas das árvores A e B

Saída: matriz W ajustada nas posições em que existem listas, quantidade ajustada de nós da árvore A e quantidade ajustada de nós da árvore B

```
1  início
2      nodes_A = 1; // recebe 1 por causa dos nós raízes
3      nodes_B = 1;
4      para cada grupo marcado com o operador "+" em g_A ou g_B faça
5          //S_A e S_B são os conjuntos de subárvores filhas de A e B, respectivamente,
           os quais foram associados os mesmos símbolos nas instâncias do grupo.
           S_A = {S_Ai}, i = 1...x;
6          S_B = {S_Bj}, j = 1...y;
7          para cada par de conjuntos (S_A e S_B) faça
8              avgScore = 0;
9              avgNodes = 0;
10             count = 0;
11             para cada subárvore S_Ai em S_A e S_Bj em S_B faça
12                 avgScore+ = W[i, j].score;
13                 avgNodes+ = W[i, j].nodes_1 + W[i, j].nodes_2;
14                 count + +;
15             fim
16             avgScore = avgScore/count;
17             avgNodes = avgNodes/(2*count);
18             W[i,j].score = avgScore; para i=j=1
19             W[i,*].score = 0; para i>1
20             W[*,j].score = 0; para j>1
21             nodes_A+ = avgNodes;
22             nodes_B+ = avgNodes;
23         fim
24     fim
25     para cada subárvore filha de A que não está dentro de um grupo faça
26         nodes_A+ = número de nós na subárvore;
27     fim
28     para cada subárvore filha de B que não está dentro de um grupo faça
29         nodes_B+ = número de nós na subárvore;
30     fim
31     retorna (W, nodes_A, nodes_B);
32 fim
```

4.3.4 Exemplo de execução do Generalized Simple Tree Matching

As árvores A e B utilizadas como forma de ilustrar as limitações do algoritmo *Simple Tree Matching* na Seção 4.2.2, e reproduzidas novamente na Figura 9 por questões de conveniência, são utilizadas como exemplo, agora para a execução do G-STM. O mesmo cenário que mostrou as deficiências do STM é empregado para averiguar o comportamento

do G-STM frente aos mesmos desafios.

Para que o exemplo não fique muito extenso, são descritos os passos para a obtenção do *matching* somente entre as subárvores filhas de b na árvore A e das subárvores filhas de b na árvore B. O *matching* para as demais subárvores é obtido da mesma maneira. Como o cálculo do *matching* no algoritmo *Generalized Simple Tree Matching* é feito da mesma forma que no algoritmo *Simple Tree Matching*, até que chegue ao ponto no qual são tratadas as listas, não há necessidade de que todo o processo seja abordado novamente. O exemplo é então iniciado justamente a partir da linha 19 do algoritmo *Generalized Simple Tree Matching*, que é o ponto de entrada para o tratamento de listas.

A matriz W contendo as pontuações de *matching* que chega à linha 19 do Algoritmo 2 está representada na Tabela 8, na qual os valores subscritos associados a b representam a ordem em que as subárvores filhas da árvore b aparecem na Figura 9.

A matriz W é então enviada ao Algoritmo 3, juntamente com as árvore A e B que estão sendo comparadas no nível de recursão atual. O algoritmo calcula primeiramente a matriz W_{norm} contendo as pontuações normalizadas a partir da matriz W (linhas 5 a 9). Os elementos da matriz W_{norm} estão representados pela Tabela 9.

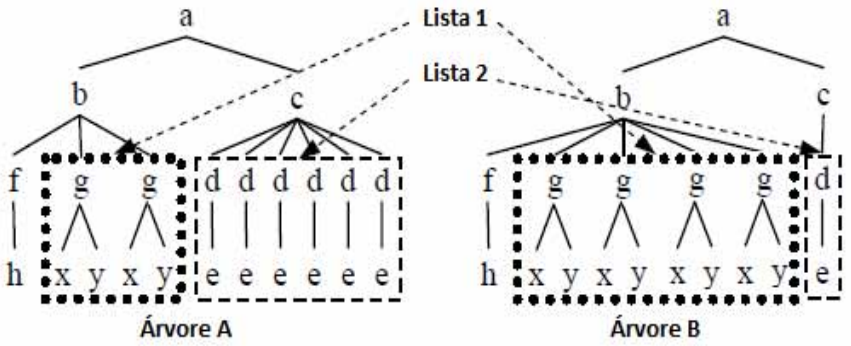


Figura 9 – Árvores A e B dadas como entradas ao algoritmo G-STM. Adaptado de Jindal e Liu (2010).

Tabela 8 – Matriz W contendo o *matching* entre as subárvores filhas de b na árvore A e as subárvores filhas de b na árvore B. Cada elemento da matriz é uma tupla na qual o primeiro valor é o *matching*, o segundo é o número de nós de b em A e o terceiro é o número de nós de b em B.

	b_1	b_2	b_3	b_4	b_5
b_1	(2, 2, 2)	(0, 2, 3)	(0, 2, 3)	(0, 2, 3)	(0, 2, 3)
b_2	(0, 3, 2)	(3, 3, 3)	(3, 3, 3)	(3, 3, 3)	(3, 3, 3)
b_3	(0, 3, 2)	(3, 3, 3)	(3, 3, 3)	(3, 3, 3)	(3, 3, 3)

As linhas 10 a 23 do Algoritmo 3 utilizam as pontuações normalizadas da matriz W_{norm} para a construção das cadeias de caracteres que simbolizam as subárvores filhas de

Tabela 9 – Matriz W_{norm} contendo o *matching* normalizado entre as subárvores filhas de b na árvore A e as subárvores filhas de b na árvore B.

	b_1	b_2	b_3	b_4	b_5
b_1	1	0	0	0	0
b_2	0	1	1	1	1
b_3	0	1	1	1	1

A e B. Com os limiares τ_1 e τ_2 fixados em 0.5 e 0.7, respectivamente, esse bloco de código produzirá as seguintes cadeias, considerando que os símbolos são atribuídos de acordo com a ordem do alfabeto: *abb* e *abbbb*. Como a Seção 4.3.1 abordou, subárvores similares devem receber um mesmo símbolo e subárvores não similares devem receber símbolos distintos. Visivelmente, as subárvores enraizadas em f tanto da árvore A quanto de B são similares. O mesmo pode-se dizer das subárvores enraizadas em g. Pode-se afirmar também que as subárvores enraizadas em f são claramente diferentes daquelas enraizadas em g. Por essas razões as subárvores enraizadas em f receberam o símbolo a e as enraizadas em g receberam o símbolo b.

Com as cadeias de caracteres que constituem a linguagem produzida pelas subárvores devidamente determinadas, o próximo passo é a geração das expressões regulares que representam essa linguagem. No Algoritmo 3, tanto g_A (linha 26) quanto g_B (linha 27) conterão a expressão ab^+. O padrão de repetição é b^+ para ambas, o que indica que há presença de uma lista e a matriz W precisará ser ajustada pelo Algoritmo 4.

A matriz W terá todas as suas entradas referentes às subárvores enraizadas em g atualizadas pelo Algoritmo 4. Após a passagem pela linha 13 do algoritmo, *avgScore* terá valor 3 (24/8) e avgNodes terá valor 3 (48/16). A entrada b_2xb_2 na Tabela 8 terá então seu valor referente ao *matching* atualizado para 3 e as demais entradas pertencentes aos itens da lista (b_2xb_3, b_2xb_4, b_2xb_5, b_3xb_2, b_3xb_3, b_3xb_4 e b_3xb_5) terão suas pontuações zeradas. A matriz W, após as atualizações terem sido realizadas, está representada na Tabela 10.

Tabela 10 – Matriz W após as entradas referentes aos itens da lista terem sido atualizadas.

	b_1	b_2	b_3	b_4	b_5
b_1	(2, 2, 2)	(0, 2, 3)	(0, 2, 3)	(0, 2, 3)	(0, 2, 3)
b_2	(0, 3, 2)	(3, 3, 3)	(0, 3, 3)	(0, 3, 3)	(0, 3, 3)
b_3	(0, 3, 2)	(0, 3, 3)	(0, 3, 3)	(0, 3, 3)	(0, 3, 3)

Percebe-se que a lista fica representada em W somente por um de seus elementos (b_2xb_2), elemento esse que carregará os valores médios do *matching* e do número de nós da lista toda. É como se a lista possuísse agora um único nó, eliminado as discrepâncias causadas pelas diferenças quanto ao número de elementos que listas formadas por um mesmo *template* podem causar, como a Seção 4.2.2 exemplificou.

O Algoritmo 4 retornará então uma tupla formada pela matriz W acima e pela

quantidade ajustada de nós para as árvores A e B, que será igual a 6 para ambas (3 nós da subárvore recém ajustada enraizada em g, 2 nós da subárvore enraizada em f, que não foi alterada, e um nó referente à raiz b). O algoritmo *Generalized Simple Tree Matching* receberá W e o utilizará para o cálculo do *matching* máximo entre o primeiro filho de a na árvore A e o primeiro filho de a na árvore B, que será igual a 6. Percebe-se que, diferentemente do que ocorreu durante a execução do *Simple Tree Matching*, aqui já conseguiu-se um *matching* de 100%, visto que o *matching* obteve um valor igual a 6 e o número de nós ajustados também é igual a 6.

O valor final do *matching* máximo entre as árvores A e B do exemplo, após a execução do *Generalized Simple Tree Matching*, é igual a 10. A quantidade de nós para cada árvore enraizada em a tanto na árvore A quanto na B também é igual a 10, o que garante 100% de *matching* pelo algoritmo eliminado-se, portanto, as discrepâncias de resultados causadas por listas não tratadas.

4.3.5 Melhoria Proposta ao G-STM

Analisando-se mais profundamente o algoritmo de detecção de listas apresentado na Seção 4.3.1, pode-se perceber uma pequena diferença em relação ao modo como os números de nós das árvores A e B são retornados quando há atualizações nas pontuações e nos números de nós (linha 29) e quando não há (linha 32). Talvez pela forma da representação do valor do número de nós, não está claro como este valor é calculado para o caso em que não há atualizações.

O fato pertinente é que o cálculo incorreto desses valores pode influenciar o algoritmo como um todo. Por exemplo, se em alguns níveis abaixo dos nós que estão sendo comparados foram encontradas listas, então houve atualizações dos valores de pontuação de *matching* e, consequentemente, os números de nós filhos também foram atualizados. Essa é a função do método de atualização das pontuações explicado na Seção 4.3.3. Tais valores devem ser elevados aos níveis superiores da recursão, porém isto não está claro pois o uso da notação $A.nodes$ e $B.nodes$ dá a entender que estão sendo retornados sempre os números totais de nós em cada nível em detrimento do número ajustado de nós naqueles níveis nos quais foram detectadas listas. Logo, conforme os níveis da árvore forem sendo percorridos, o cálculo dos valores das pontuações normalizadas contidos na matriz W_{norm} (linha 7), ficará prejudicado, já que os valores $W[i,j].nodes_A$ e $W[i,j].nodes_B$ poderão ser bem maiores do que deveriam, o que influencia para o aumento da quantidade de itens Falsos Negativos nos resultados, ou seja, itens que deveriam ser extraídos mas não foram identificados pelo algoritmo.

Assim, este trabalho integra a função *NormalizaNumeroNos* (Algoritmo 5) ao algoritmo *GeneralizedSimpleTreeMatching* (Algoritmo 2, Seção 4.3). A nova função é inserida logo após o cálculo de cada elemento da matriz W (linha 16), para que seja obtido

o número de nós, agora chamado de *número de nós normalizados*, imediatamente após o cálculo de cada valor de pontuação para as subárvores comparadas em cada nível. Tal número de nós foi chamado de "normalizado" porque não reflete a quantidade real de nós contidos em uma subárvore e sim a quantidade recalculada para os casos nos quais há listas em qualquer nível da subárvore. Se não houver listas entre os nós pertencentes a uma subárvore, a quantidade normalizada de nós será igual à quantidade real.

Algoritmo 5: NormalizaNumeroNos(W, A, B, i, j)

Entrada: matriz W, subárvores A e B, índices i e j cujas pontuações foram recém calculadas.

Saída: subárvores A e B com suas propriedades *normalizedNodes* atualizadas.

```
1  início
2      similarity ← W[i,j].score/max(W[i,j].nodes_A, W[i,j].nodes_B);
3      se similarity > A.similarity então
4          A.similarity ← similarity;
5          A.normalizedNodes ← W[i,j].nodes_A;
6      fim
7      se similarity > B.similarity então
8          B.similarity ← similarity;
9          B.normalizedNodes ← W[i,j].nodes_B;
10     fim
11     retorna (A, B);
12 fim
```

Para que a implementação sugerida funcione, as árvores A e B devem possuir duas novas propriedades, chamadas *normalizedNodes* e *similarity*, além da propriedade *nodes* já prevista no algoritmo original, amplamente utilizada nos algoritmos do Capítulo 4. A propriedade *normalizedNodes*, preenchida pelas linha 5 e 9 do Algoritmo 5, armazenará o valor normalizado de nós calculados para cada subárvore analisada e é este valor que será repassado ao nível superior pela linha 25 do algoritmo *GeneralizedSimpleTreeMatching* a partir de então. Já a propriedade *similarity*, calculada pela linha 2 do algoritmo, armazenará o valor de similaridade obtido a partir da comparação de um nó filho da subárvore A com um nó filho da subárvore B para a propriedade associada à arvore A e o oposto para a propriedade associada à arvore B, configurando uma busca pelos nós com maiores similaridades, haja vista que o número normalizado de nós deve ser calculado em relação à subárvore mais similar àquela que está sendo comparada. A variável *similarity* é calculada pela razão entre a pontuação recém obtida armazenada em *W[i,j]* e o número máximo de nós entre essas subárvores, lembrando que *W[i,j]* contém o valor do *matching* entre a *i-ésima* subárvore filha de A e a *j-ésima* subárvore filha de B calculada na linha 16 do *Generalized Simple Tree Matching*.

A função *NormalizaNumeroNos* é chamada para cada comparação feita para as subárvores filhas de A e de B, já que é chamada dentro dos laços definidos pelas linhas 14

e 15 do algoritmo *Generalized Simple Tree Matching*. O valor da similaridade calculado sempre ficará compreendido no intervalo entre 0 e 1 e, quanto mais próximo de 1, mais similar dois nós são entre si. A comparação entre as subárvores filhas de A com as subárvores filhas de B que render o maior valor de similaridade é aquela a qual emprestará seus valores de quantidades de nós normalizados à propriedade *normalizedNodes*, melhorando assim a precisão na extração de padrões.

4.4 Considerações Finais

Este capítulo apresentou inicialmente um panorama acerca da busca por padrões dentro de páginas Web por meio do emprego de técnicas de *Tree Matching*. Foram discutidas, primeiramente, as características do algoritmo *Simple Tree Matching* bem como suas limitações a respeito do tamanho variável de listas dentro da estrutura das páginas, limitações essas as quais incentivaram o desenvolvimento do algoritmo *Generalized Simple Tree Matching* por Jindal e Liu (2010). O *Generalized Simple Tree Matching*, por sua vez, reduz (ou resume), as instâncias que se repetem nas listas para apenas um representante, o qual será o responsável por preservar as características da lista por meio do emprego da média entre os valores do *matching* e do número de nós entre todos os elementos da lista. O capítulo é concluído com a sugestão de uma melhoria a ser incorporada ao algoritmo *Generalized Simple Tree Matching* visando um ganho no que diz respeito à acurácia do *matching* entre duas subárvores.

O próximo capítulo apresenta a metodologia empregada no desenvolvimento do trabalho, destacando as etapas que, juntas, formam o processo de extração de dados de produtos em páginas de comércio eletrônico. São apontadas algumas observações e melhorias anexadas ao processo como um todo, a fim de adaptar a técnica de *Tree Matching* ao ambiente de atuação proposto, otimizando-a quanto ao tempo de resposta.

5 Extração de Dados de Produtos em Páginas de Comércio Eletrônico

Neste capítulo é apresentado todo o fluxo do processo de extração de dados de produtos em página Web de comércio eletrônico adotado neste trabalho. Cada uma das etapas que compõem o fluxo do processo de extração são formadas por tarefas internas, as quais são abordadas e discutidas em detalhes.

5.1 Fluxo de Extração de Dados em Páginas de Comércio Eletrônico

Com o objetivo de aprimorar o processo de extração de padrões para atuar especificamente em páginas de vendas em portais de comércio eletrônico e então extrair os produtos dessas páginas, o fluxo de extração foi definido em três etapas: etapa de pré-processamento (preparação da página para a extração de padrões), etapa de processamento (execução do G-STM para a extração de padrões) e uma etapa de pós-processamento (filtragem dos resultados e extração dos produtos). A etapa de pré-processamento possui o intuito de reduzir o tamanho da página e, consequentemente, a quantidade de estruturas a serem comparadas pela etapa de processamento que, por sua vez, utiliza o algoritmo G-STM para a extração dos padrões contidos nessas páginas. Essa redução no tamanho das páginas objetiva diminuir significativamente o tempo de resposta do algoritmo. A etapa de pós-processamento propõe meios para que o algoritmo consiga lidar com ruídos que porventura possam ocorrer nos resultados, evitando que menus, ou outras estruturas de repetição encontradas no código HTML da página, venham a ser interpretados como produtos. A Figura 10 mostra o diagrama do fluxo de extração de dados de produtos proposto neste trabalho, destacando cada etapa e suas tarefas internas.

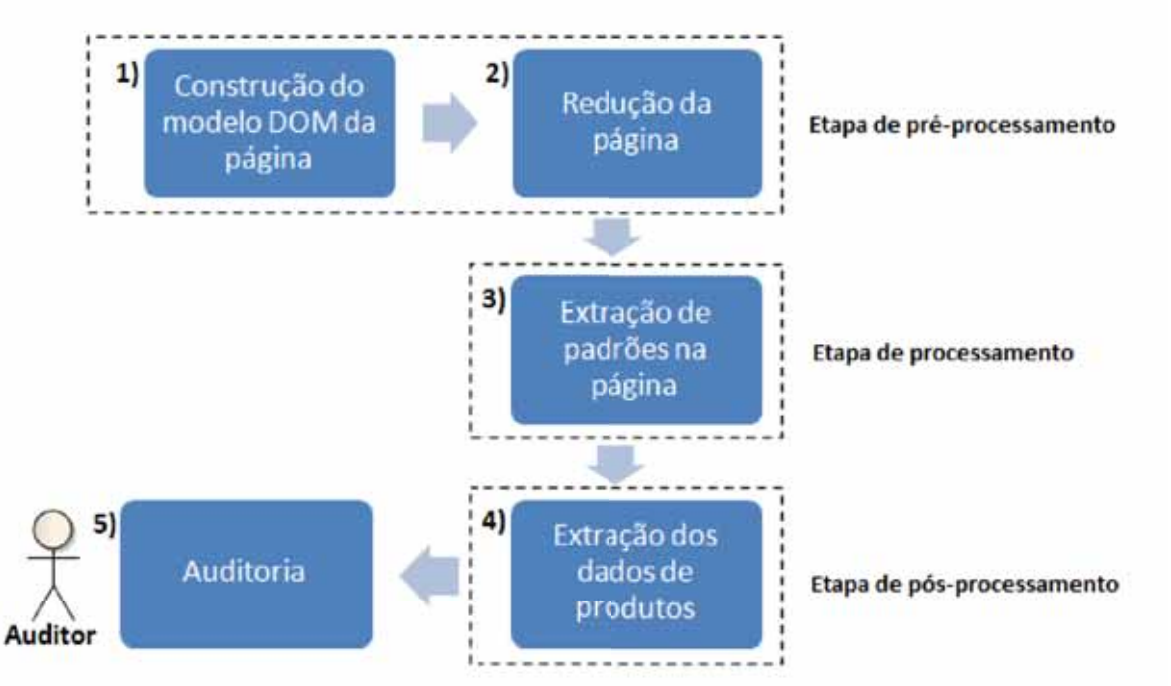


Figura 10 – Diagrama do fluxo de extração de dados de produtos.

As próximas seções discutem cada etapa apresentada no diagrama acima.

5.2 Etapa de Pré-Processamento

A etapa de Pré-Processamento desempenha duas tarefas: Construção do modelo DOM da página (item 1 da Figura 10) e Redução da página (item 2 da Figura 10). O conteúdo técnico empregado na primeira tarefa desta etapa foi discutido na seção 3.4, quando foram abordados os aspectos conceituais referentes às páginas Web. Já a segunda tarefa da etapa de pré-processamento, que ocorre após o modelo DOM da página ser construído devido às facilidades que o modelo proporciona no que diz respeito à manipulação de elementos que compõem a estrutura da página Web, é discutida a seguir.

5.2.1 Limpeza de Características Visuais no Código HTML

A extração de regiões de interesse dentro de páginas HTML realizadas tanto pelo algoritmo STM quanto pelo G-STM são feitas a partir de comparações das estruturas que formam essas páginas, estruturas essas que se tornam nós de uma árvore após serem convertidas pelo modelo DOM. Como tais estruturas são analisadas uma a uma para toda página, torna-se importante a retirada de qualquer estrutura que se saiba, *a priori*, não conter os dados alvo da extração, que são os dados de produtos. A retirada de estruturas que não contêm dados de interesse reduz significativamente a quantidade de comparações efetuadas pelos algoritmos de *Tree Matching* já que, atuando sobre páginas cada vez menores, serão criados, proporcionalmente ao tamanho da página, menos níveis

de recursão, menos itens nas tabelas de pontuações e, sobretudo, os resultados serão disponibilizados em tempos menores.

Assim, este trabalho utiliza uma etapa de pré-processamento para a eliminação de estruturas logo após a página ter sido convertida para o modelo DOM. Isso se dá porque a manipulação da página no formato DOM é muito mais prática e menos complexa do que se fosse tratada como HTML puro ou texto. Desse modo, depois de serem retiradas todas as estruturas que não possuem importância semântica à extração de dados, a página resultante será entregue ao algoritmo de *Tree Matching*. As estruturas eliminadas pela etapa de pré-processamento compreendem, em sua maioria, estruturas de estilo e formatação de texto, que agregam somente valor visual à página e não contribuem, neste trabalho, para a extração de dados. As estruturas eliminadas são as seguintes:

- Nós da árvore que possuem a *tag <script>*, bem como todos os nós filhos desses nós;
- Nós da árvore que possuem a *tag <style>*, bem como todos os nós filhos desses nós;
- Nós da árvore que possuem a *tag <meta>*, bem como todos os nós filhos desses nós.

São eliminadas também estruturas que representam cabeçalhos, como o exemplificado pela Figura 11, e rodapés das páginas, como o exemplo da Figura 12, que apresentam apenas a identidade visual e informações de contato das lojas e também não são de interesse deste trabalho. Cabeçalhos e rodapés são identificados no código HTML pelas seguintes estruturas:

- Nós da árvore que possuem a *tag <header>*, bem como todos os nós filhos desses nós;
- Nós da árvore que possuem a *tag <footer>*, bem como todos os nós filhos desses nós.

Figura 11 – Exemplo de cabeçalho de uma página de comércio eletrônico.

Além dos nós da árvore citados acima, são excluídos também todos aqueles que possuem o atributo *.hidden*, como o nó representado abaixo. Nós com esse atributo não ficam visíveis ao usuário e, por isso, mesmo que representem produtos, não são do interesse deste trabalho, já que o foco são as informações que um ser humano que navega pela página de vendas *online* possa observar.

* *

Figura 12 – Exemplo de rodapé de uma página de comércio eletrônico.

5.2.2 Utilizando o Vocabulário de Produtos do Schema.org

O Schema.org[1] é uma associação formada por grandes empresas da Web com a finalidade de definir padrões de metadados, os quais podem ser usados para a anotação de páginas Web. Os metadados definidos no *Schema.org* contêm *tags* que formam vocabulários que, por sua vez, foram concebidos com o intuito de auxiliar as principais ferramentas de busca existentes, como o Google, Bing e Yahoo!, a aprimorarem seus resultados, refinando-os e dispondo as informações buscadas de uma forma mais amigável e precisa aos usuários.

Tags HTML são apenas interpretadas pelos navegadores, que renderizam as informações contidas e as apresentam ao usuário dispondo-as nas páginas, não fornecendo meios de inferir o significado dessas informações. Por exemplo, o fragmento HTML *<h2>Dogville</h2>* é interpretado pelo navegador de forma a apresentar o texto "Dogville" em um formato de cabeçalho do tipo 2. No entanto, fica impossível saber do que se trata o termo "Dogville", podendo ser um filme, o nome de um bairro ou de uma cidade. Os vocabulários propostos pelo *Schema.org* foram construídos com o objetivo de sanar as dúvidas quanto aos significados dos termos encontrados em páginas Web e, para adicionarem informações às páginas, tais vocabulários fazem uso de formatos padronizados como o *Microdata*[2], o RDF[3] ou o JSON-LD[4]. Dos padrões citados, o *Microdata* é o mais comum, pois é constituído de uma série de atributos adicionados às *tags*, atributos esses introduzidos junto ao HTML5, que tornam mais simples e intuitivas as anotações, haja vista que as páginas Web já estão no formato HTML.

O padrão *Microdata* é composto basicamente por três elementos, adicionados ao código HTML como atributos de *tags*: *itemscope*, *itemtype* e *itemprop*. O atributo *itemscope* tem o objetivo de dizer aos navegadores que o bloco de código englobado por determinada *tag* é um bloco de interesse. O atributo *itemtype* tem o objetivo de dizer qual o tipo do item que o bloco se refere e alguns desses tipos já são pré-determinados pelo *Schema.org*, como o tipo *Movie*, o tipo *Book*, o tipo *Place* e o tipo que é de interesse

[1] http://www.schema.org. Último acesso em 23 de fevereiro de 2015.
[2] http://www.w3.org/TR/microdata. Último acesso em 23 de fevereiro de 2015.
[3] http://www.w3.org/RDF. Último acesso em 24 de fevereiro de 2015.
[4] http://en.wikipedia.org/wiki/JSON-LD. Último acesso em 23 de fevereiro de 2015.

deste trabalho, *Product*. O *itemtype* é definido na forma de uma URL como, por exemplo, *itemtype="http://schema.org/Product"*. Já o atributo *itemprop* tem o objetivo de rotular propriedades de um determinado item, como nome de um produto e o seu preço.

O seguinte fragmento de código HTML exemplifica um produto anotado com o vocabulário *Product* em uma página de comércio eletrônico:

```
<div itemscope itemtype="http://schema.org/Product">
  <img itemprop="image" src="dell-30in-lcd.jpg" alt="A Dell UltraSharp
      monitor"/>
  <span itemprop="name">Dell UltraSharp 30 LCD Monitor</span>
  <div itemprop="aggregateRating"
    itemscope itemtype="http://schema.org/AggregateRating">
    <span itemprop="ratingValue">87</span>
    out of <span itemprop="bestRating">100</span>
    based on <span itemprop="ratingCount">24</span> user ratings
  </div>
  <div itemprop="offers" itemscope itemtype="http://schema.org/
      AggregateOffer">
    <span itemprop="lowPrice">$1250</span>
    to <span itemprop="highPrice">$1495</span>
    from <span itemprop="offerCount">8</span> sellers
  </div>
  Sellers:
  <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    <a itemprop="url" href="save-a-lot-monitors.com/dell-30.html">
      Save A Lot Monitors - $1250</a>
  </div>
  <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    <a itemprop="url" href="jondoe-gadgets.com/dell-30.html">
      Jon Doe's Gadgets - $1350</a>
  </div>
</div>
```

Código 5 – Fragmento de código HTML com um produto anotado de acordo com o *Schema.org*.

As anotações que ajudam os motores de buscas podem também ajudar o processo de extração de dados de páginas Web proposto neste trabalho, desde que as páginas estejam anotadas com o padrão proposto no *Schema.org* e que esses padrões estejam aplicados corretamente, já que se a página foi anotada incorretamente ou as anotações não foram usadas da forma como foram especificadas, as próprias lojas serão prejudicadas diante das ferramentas de buscas. Sabendo disso, produtos anotados são poderosos subsídios para a diminuição do tempo de resposta da extração de padrões já que somente as porções da página anotadas com *itemscope* precisarão ser analisadas. Assim, a abordagem desenvolvida

neste trabalho faz uso, quando disponíveis, dos atributos propostos pelo *Schema.org* com a finalidade de reduzir o número de comparações feitas pelo algoritmo de *Tree Matching*, fornecendo ao algoritmo somente árvores montadas a partir de blocos de código HTML anotados, ao invés da página toda.

5.3 Etapa de Processamento

A etapa de processamento (item 3 da Figura 10) possui como figura central o algoritmo *Generalized Simple Tree Matching* proposto por Jindal e Liu (2010), descrito na Seção 4.3. Esta é a etapa que consome mais tempo quando comparada às etapas de pré e pós-processamento, já que nela a árvore que representa a página Web analisada é percorrida e comparações são feitas entre seus nós à procura de padrões que se traduzem em registros de dados. Todo o conteúdo teórico desta etapa foi abordado no Capítulo 4. A versão do algoritmo empregada nesta etapa é a que foi proposta neste trabalho, que engloba a alteração feita com o objetivo de melhorar sua acurácia.

5.4 Etapa de Pós-Processamento

A etapa de pós-processamento ocorre imediatamente após os registros de dados, que são as estruturas que se repetem na página, terem sido extraídas pelo algoritmo de *Tree Matching* na etapa de processamento. A etapa de pós-processamento recebe toda a variedade de registros de dados encontrados na etapa anterior e tem o intuito de refinar os resultados, selecionando as regiões de dados que contém produtos e então, a partir delas, extrair de fato os produtos.

5.4.1 Extração dos Dados dos Produtos

Esta etapa tem o objetivo de eliminar do conjunto de resultados aqueles itens que não contêm dados de produtos, ou seja, Falsos Positivos. Na maioria dos casos os Falsos Positivos são estruturas provenientes de menus (Figura 13), *banners* (Figura 14), ou outros itens de repetição que, exatamente por compartilharem estruturas similares entre seus itens, são reconhecidos como padrões de dados pelos algoritmos de *Tree Matching*. Para que essas estruturas não componham o resultado final do processo de extração de padrões em páginas Web são definidas algumas heurísticas a partir da observação das diferenças encontradas em resultados corretos e em Falsos Positivos.

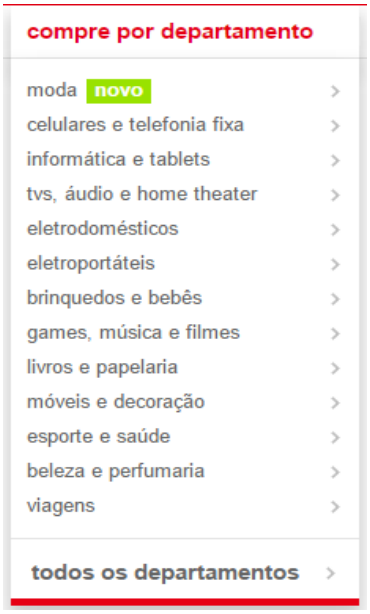


Figura 13 – Exemplo de menu em uma página de comércio eletrônico.

Figura 14 – Exemplo de *banners* em uma página de comércio eletrônico.

Este trabalho apresenta quatro heurísticas definidas empiricamente a partir da observação sucessiva das estruturas que, de fato, contêm produtos em páginas Web (Verdadeiros Positivos) das mais diferentes lojas de comércio eletrônico. São elas:

1. Produtos nunca são encontrados em subárvores com altura menor do que três níveis;
2. Nomes de produtos são compostos por, no mínimo, três palavras;
3. Os produtos de interesse deste trabalho são aqueles que possuem um preço associado, ou seja, a estrutura que contém um produto sempre deve possuir outra estrutura cujo texto será um valor numérico em suas adjacências, podendo incluir ou não o símbolo *"$"* antes do valor numérico;
4. Se a página implementar os metadados do *Schema.org*, as propriedades dos produtos serão marcadas com o elemento *itemprop*, como visto na Seção 5.2.2, e seus valores

serão *"name"* (referente ao nome do produto) e *"price"* (referente ao preço do produto).

Como pode haver casos em que listas são encontradas dentro de outras listas, ou seja, regiões de dados dentro de outras regiões de dados, de acordo com a forma com que cada página foi implementada, os dados referentes aos produtos são extraídos, neste trabalho, do menor número de nós possível, todos filhos de um mesmo nó pai. Em outras palavras, os dados serão capturados da menor subárvore possível que respeite as heurísticas definidas acima. Assim, se os dados dos produtos que são de interesse (nome e preços) contiverem outras informações associadas, como especificações técnicas ou formas de pagamento, estas não serão consideradas. Isso ocorre porque o algoritmo, após já ter encontrado os dados que procura, descarta as demais estruturas (nós da árvore) que estão dentro da mesma região de dados extraída pelo *Tree Matching*, na qual já foram encontrados os dados do produto.

O trabalho de Zhai e Liu (2006) que é uma das referências do trabalho desenvolvido por Jindal e Liu (2010), quando encontra uma região de dados em um nível superior que abrange outras regiões de dados em níveis inferiores, escolhe sempre a região de nível superior. Tal critério pode provocar a inclusão de muitos dados espúrios nos resultados. O algoritmo proposto neste trabalho procede a escolha baseado nas próprias informações contidas nas regiões de dados, fornecendo resultados de maior qualidade considerando o ambiente de atuação ora proposto.

Outro ponto a ser considerado é a escolha do preço e do preço promocional, caso exista. O preço promocional obviamente sempre terá seu valor inferior ao do preço normal, logo dentre os dois valores encontrados pela etapa de pós-processamento, o preço promocional será o de menor valor. Caso somente um valor seja encontrado nas proximidades do nome do produto, este será o preço normal e o preço promocional será nulo.

Apesar de simples, as regras definidas acima garantem que boa parte dos resultados Falsos Positivos retornados pelos algoritmos de *Tree Matching* sejam descartados, ajudando a melhorar o índice de acerto do processo de extração de dados de produtos em páginas Web.

5.5 Auditoria nos dados coletados

Todas as técnicas apresentadas neste trabalho não necessitam da interação humana para desemprenharem seu papel, constituindo-se técnicas totalmente autônomas. As heurísticas definidas na Seção 5.4.1 foram introduzidas com o objetivo de reduzir a carência da abordagem automática para a extração de padrões no que diz respeito ao conhecimento

do domínio do problema e diminuir, dessa forma, a quantidade de itens Falsos Positivos nos resultados.

O processo de auditoria proposto no fluxo do trabalho para a extração de dados de produtos em páginas de comércio eletrônico, apresentado na Figura 10, consiste de uma validação por parte de um operador humano, o auditor, acerca do grau de acerto dos dados extraídos, podendo este reprovar o resultado caso não esteja de acordo com as premissas do nome do produto, preço do produto e preço promocional caso exista. O processo de auditoria, portanto, é o único ponto do processo proposto neste trabalho que prevê interação com um ser humano.

O papel da figura do auditor, no fluxo de extração, é o de analisar os dados dos produtos provenientes da etapa de pós-processamento à procura a Falsos Positivos. O presente trabalho não considera a auditoria como uma tarefa da etapa de pós-processamento porque ela não consiste de um método de extração em si, funcionando mais como uma etapa complementar ao processo de extração como um todo, com o intuito de minimizar a ocorrência de Falsos Positivos para que o processo apresentado possa ser introduzido a um sistema de escala comercial, por exemplo, que exige o máximo grau de acerto com o mínimo de intervenção externa. Portanto, ao analisar-se a etapa de auditoria, deixa-se a esfera que engloba somente o processo de extração e adentra-se ao conceito de um sistema de extração, já pensando em um produto empregado em escala comercial. O processo de extração em si não necessita da auditoria para desempenhar suas tarefas; contudo, a auditoria pode ajudar o processo de extração a incrementar a eficácia de suas tarefas.

Para que o auditor possa avaliar a qualidade dos dados dos produtos extraídos, o sistema que emprega o processo de extração proposto deve possuir uma interface gráfica de comunicação com o auditor. Os resultados do processo de auditoria podem então ser tratados de duas maneiras, excludentes entre si:

1. Falsos Positivos são somente descartados, não sendo apresentados como parte do resultado final da extração e não contribuindo para a melhora da acurácia do sistema;
2. Falsos Positivos são utilizados na inferência de regras que auxiliarão o processo na extração dos produtos de novas páginas, melhorando a acurácia do sistema.

Obviamente a segunda maneira é muito mais útil ao processo de extração do que a primeira. A segunda maneira faz com que o sistema aprenda com os erros cometidos, pois o auditor, ao mostrar ao processo de extração seus erros, permite que este não mais os cometa.

Toda essa discussão remete aos tipos de abordagens apresentados no Capítulo 2, que basicamente são a abordagem de aprendizado supervisionado, a abordagem automática e a abordagem semi-automática. Pois bem, se somente o processo de extração proposto neste

trabalho for analisado, a abordagem se encaixa no conceito de uma abordagem automática. Todavia, se a etapa de auditoria for considerada e o processo de extração for inserido no contexto de um sistema computacional, este se comporta como um sistema semi-automático, já que permite interação humana, mesmo que mínima, e prevê, somente quando necessária, uma espécie de treinamento acerca do que são Falsos Positivos, que visa auxiliar as etapas de pré-processamento, processamento e pós-processamento, a desempenharem suas tarefas.

5.6 Considerações finais

Este capítulo apresentou as etapas propostas neste trabalho para a formalização do fluxo do processo para a extração de dados de produtos em páginas de comércio eletrônico. Tais etapas são interdependentes e devem ser executadas em ordem, já que etapas posteriores necessitam dos resultados de etapas anteriores a elas. Foram propostas tentativas de melhorias no processo de extração como um todo, com o intuito de reduzir o tempo de resposta e aumentar a taxa de acerto dos métodos pertencentes a cada etapa. O próximo capítulo apresenta o material utilizado para a avaliação das abordagens contidas neste trabalho, além da metodologia empregada para o desenvolvimento de cada etapa que compõe o processo de extração de dados de produtos ora proposto.

6 Material e Métodos

Neste capítulo são apresentados o material utilizado nos experimentos, composto pelas bases de dados de páginas Web e ferramentas computacionais, além da metodologia proposta para realizar a extração de dados de produtos em páginas de comércio eletrônico.

6.1 Material

Para determinar a efetividade do fluxo de extração e também comparar o desempenho das técnicas de extração de padrões de páginas Web utilizando o algoritmo *Generalized Simple Tree Matching* otimizado para o tempo de resposta e acurácia, proposto neste trabalho, com o algoritmo *Generalized Simple Tree Matching* original, apresentado por Jindal e Liu (2010), foram utilizadas duas bases de dados: *Ecommerce DB Versão 1.0*, especialmente desenvolvida para este trabalho, e a base *TBDW (*Testbed for Information Extraction from Deep Web*) versão 1.02*[1], que é uma base pública de páginas Web. A base *Ecommerce DB Versão 1.0* foi criada para avaliar a abordagem proposta em relação ao cenário do mercado de comércio eletrônico no Brasil. A base *TBDW versão 1.02* foi utilizada para comparar a abordagem proposta com outras abordagens disponíveis na literatura.

6.1.1 Base de páginas TBDW versão 1.02

A base de páginas *Testbed for Information Extraction from Deep Web* foi construída com o objetivo de permitir a avaliação de programas capazes de extrair informações estruturadas de páginas Web, em outras palavras, programas chamados *wrappers*. Tal base foi desenvolvida por pesquisadores da Kyushu University, do Japão, e do CSIRO Mathematical and Information Sciences, da Austrália (YAMADA et al., 2004). Utilizou-se para a realização deste trabalho a versão 1.02 da base *TBDW*, já que essa mesma versão foi empregada para a avaliação dos resultados do algoritmo *Generalized Simple Tree Matching* no trabalho de Jindal e Liu (2010).

A base *TBDW* é formada por conjuntos de páginas contendo resultados de buscas coletados junto a 51 bases de dados dos mais variados assuntos selecionadas de forma aleatória na Web. Essas 51 bases compreendem informações armazenadas em bancos de dados que não estão acessíveis às ferramentas de buscas convencionais. Tais bases foram selecionadas por meio de duas ferramentas especializadas (*Turbo10 Search Engine*[2] e

[1] http://daisen.cc.kyushu-u.ac.jp/TBDW. Último acesso em 19 de fevereiro de 2015.
[2] http://turbo10.com. Último acesso em 19 de fevereiro de 2015.

CompletePlanet[3]) que atuam na chamada *Deep Web*[4]. Foram utilizadas também páginas contidas em uma lista montada a partir do trabalho de Cope et al. (2003), desenvolvido junto à Universidade Nacional da Austrália.

Quanto às bases encontradas na *Deep Web*, foram escolhidas somente aquelas que possuíam uma interface para a busca de resultados, ou seja, um campo no qual pudesse ser inserida uma *query* de busca, assim como ocorre nos sites de busca mais populares atualmente, como o *Google* ou o *Bing*. As bases também deveriam disponibilizar páginas de resultados para essas buscas, como as das Figuras 15 e 16. Os resultados dessas buscas foram então armazenados na forma de páginas HTML e estão organizados na base *TBDW* em subdiretórios numerados de 1 a 51 possuindo, cada um, um conjunto de 5 páginas. Além das páginas armazenadas no formato HTML, cada diretório possui arquivos descrevendo as páginas de resultados encontrados. Tais arquivos foram criados a partir da identificação manual dos resultados, no que diz respeito à quantidade de regiões de dados contidas na página, quantidade de registro de dados, quantidade de itens em cada registro, além da *query* utilizada na busca e da URL de busca obtida. Tais parâmetros são de grande valia durante os experimentos realizados a partir da base *TBDW*, pois facilitam a avaliação dos testes realizados, já que não é necessário que o pesquisador que utiliza a base para seus testes precise se preocupar em abrir página a página da base a fim de verificar os itens nela contidos, bastando que o sistema de avaliação leia os parâmetros contidos em tais arquivos.

Em resumo, as páginas que compõem a base *TBDW* foram selecionadas em dois passos: primeiramente foram escolhidas as bases contidas na *Deep Web* a serem utilizadas, por meio das ferramentas citadas e então, após essa escolha, foram coletadas as páginas de resultados obtidas a partir da busca por termos escolhidos pelos autores e submetidos à interface de busca dessas bases.

Do total de 51 bases utilizadas na geração das páginas que compõem a base *TBDW*, quatro delas não são bases escritas em língua inglesa. As páginas das 51 bases possuem tamanho médio de 29 kilobytes contendo, em média, 18 registros de dados por página.

[3] http://completeplanet.com. Último acesso em 19 de fevereiro de 2015.

[4] Deep Web é o termo dado ao conteúdo que raramente é mostrado em resultados de ferramentas de busca convencionais, pois os motores desses sites de busca não conseguem atingir o nível da Web no qual estão as bases de dados que armazenam tais informações, visto que não existem páginas que possuem *links* que direcionam a elas (IFFAT; SAMI, 2010).

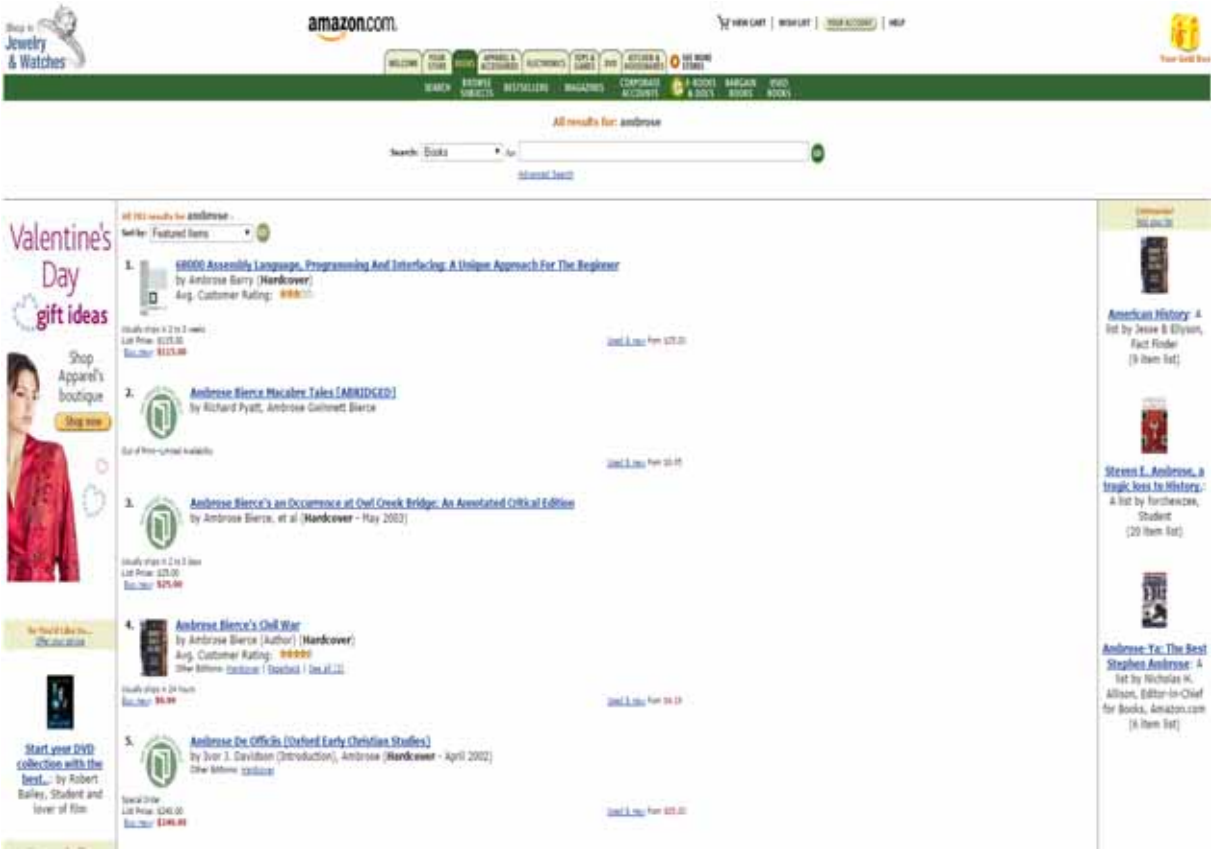


Figura 15 – Exemplo de página na base *TBDW* contendo resultados de uma busca pelo termo "*ambrose*".

Figura 16 – Exemplo de página na base *TBDW* contendo resultados de uma busca pelo termo "*indonesia*".

A Tabela 11 resume as características da base *TDBW* em sua versão 1.02.

Tabela 11 – Características da base de páginas *TBDW versão 1.02*.

Característica	**Quantidade**
Domínios distintos	51
Páginas por domínio	5
Média de registros de dados por página	18

6.1.2 Base de páginas Ecommerce DB versão 1.0

A base de páginas *Ecommerce DB*, em sua primeira versão, é uma base de dados de páginas de comércio eletrônico criada especialmente para os experimentos deste trabalho. Esta base visa simular o ambiente mais próximo do real possível para os experimentos, já que a base *TBDW versão 1.02* não é uma base especializada em comércio eletrônico, sendo útil apenas aos experimentos com o algoritmo de *Tree Matching*, mas não aos experimentos de extração de dados de produtos. A base *Ecommerce DB*, por sua vez, pode ser utilizada em ambas as modalidades de experimentos.

A base é dividida em 10 diretórios separados por loja, cada um contendo o código HTML de 20 páginas de vitrine das oito maiores lojas de comércio eletrônico do Brasil, segundo números da publicação *online* E-commercebrasil (2011). As lojas incluídas na base são as seguintes: *Lojas Americanas*, *Submarino*, *Netshoes*, *Ponto Frio*, *Magazine Luiza*, *Casas Bahia*, *Walmart* e *Livraria Saraiva*. Além dessas oito lojas, foram adicionadas à base também as lojas *Decathlon* e *Fast Shop*, devido à forma como os dados dos produtos são dispostos em suas estruturas, que diferem um pouco das demais e são uma boa oportunidade para por à prova o processo de extração de produtos.

Optou-se por incluir somente páginas de vitrine das lojas devido ao elevado número de produtos e, consequentemente, registros de dados, que esse tipo de página possui, em contrapartida às páginas de detalhe, carrinho e produtos comprados, que possuem apenas um produto ou então um número reduzido de produtos. Vale ressaltar que a lista de lojas utilizada foi publicada no ano de 2011 e utilizou-se a ferramenta *Google Adplanner* para a seleção das maiores lojas na época. Já os códigos HTML das páginas dessas lojas foram coletados entre os meses de agosto e outubro de 2014. A coleta das páginas foi feita por meio do aplicativo *Ibope Emeter* em sua versão 0.14.43[5]. A escolha desse aplicativo, que é de propriedade do grupo IBOPE[6] (IBOPE Pesquisa de Mídia Ltda.), foi motivada pela capacidade que ele possui quanto à captura do código HTML renderizado da página somente após o carregamento completo da mesma, a partir da monitoração de um painel

[5] http://painelinternet.ibope.com.br/AcordoParticipacao.aspx. Último acesso em 20 de fevereiro de 2015.

[6] www.ibope.com.br. Último acesso em 20 de fevereiro de 2015.

de cerca de 5000 usuários que navegaram regularmente em sites de comércio eletrônico e possuem o aplicativo instalado em suas máquinas.

A Tabela 12 resume as características da base *Ecommerce DB* em sua versão 1.0.

Tabela 12 – Características da base de páginas *Ecommerce DB versão 1.0*.

Característica	**Quantidade**
Domínios distintos	10
Páginas por domínio	20
Média de registros de dados por página	29

As Figuras 17 e 18 são exemplos de duas das 200 páginas subdivididas em 10 diretórios que compõem a base *Ecommerce DB*.

Figura 17 – Exemplo de página de uma loja de departamentos contida na base *Ecommerce DB*.

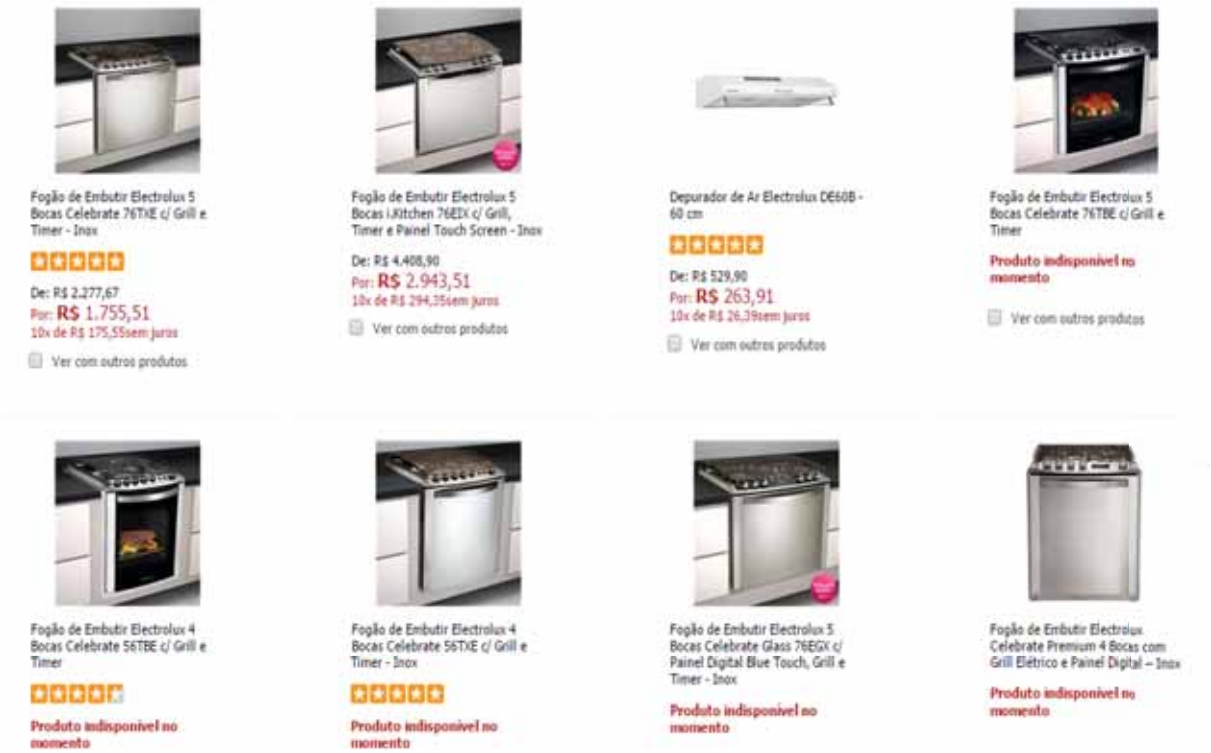


Figura 18 – Exemplo de página contendo uma vitrine pertencente à base *Ecommerce DB*.

6.1.3 Hardware e Software

Todos os experimentos foram realizados em um computador pessoal com a seguinte configuração: sistema operacional Microsoft Windows 8.1, processador Intel i7-3537U com clock de 2.0 GHz, memória RAM de 8 GB e HD de 500 GB.

Foram utilizados os seguintes softwares para implementação e execução dos experimentos:

- Java SE versão 7: linguagem de programação utilizada;
- *Spring Framework* versão 3.5: framework baseado na linguagem Java que possibilita a aplicação dos conceitos de inversão de contexto e injeção de dependências;
- IDE para programação *Spring Tool Suite*, versão 3.6: *Integrated Development Environment* com suporte ao *Spring Framework* baseada na plataforma Eclipse Kepler SR2 versão 4.3.2;
- *Jsoup* 1.8.1[7]: biblioteca escrita em linguagem Java para a construção do modelo DOM e manipulação de árvores.

[7] http://jsoup.org/

6.2 Métodos

O presente trabalho propõe um passo além em relação à aplicação de um algoritmo de *Tree Matching* para a captura de padrões dentro de uma página Web. O algoritmo foi empregado dentro de um ambiente específico de atuação – as páginas de vitrines de comércio eletrônico – e, a partir de seus resultados, foram extraídos os dados de interesse. Assim, somente a aplicação e a validação de técnicas de *matching* entre estruturas de páginas apresentadas em Jindal e Liu (2010) que, por si só, é uma tarefa bastante complexa, não foi suficiente para que os objetivos propostos por esse trabalho fossem atingidos, sendo que foram necessárias também a aplicação das tarefas propostas no capítulo anterior. O algoritmo *Generalized Simple Tree Matching* descrito no Capítulo 4 foi avaliado experimentalmente pelos autores utilizando-se algumas bases de páginas, dentre as quais a base *TBDW versão 1.02*, em Jindal e Liu (2010). Embora os resultados obtidos tenham sido bastante promissores, o foco dos experimentos foi somente a etapa da extração de registros de dados de páginas Web, não importando qual a natureza desses dados, podendo ser menus, banners, dentre outros padrões encontrados nas páginas avaliadas, o que é típico em abordagens não supervisionadas ou automáticas.

Este trabalho incorporou, após um profundo estudo de técnicas de *Tree Matching*, uma etapa de limpeza e redução de páginas de vendas *online* antes de fornecê-las a um algoritmo de *Tree Matching* e também uma etapa de filtragem de dados de produtos posterior à extração de padrões em páginas por um algoritmo de *Tree Matching*.

A Figura 19 apresenta o diagrama com as etapas da metodologia utilizada para a realização deste trabalho, todas com o intuito de tornar possível a execução das etapas de pré-processamento, processamento e pós-processamento apresentadas no Capítulo 5.

Figura 19 – Diagrama da metodologia proposta.

As particularidades e os parâmetros utilizados em cada etapa apresentada na Figura 19 são apresentados a seguir.

6.2.1 Preparação das Páginas para Extração de Dados

O primeiro passo para extração de dados em páginas Web baseado em técnicas de *Tree Matching* é a construção do modelo DOM dessa página (item 1 na Figura 19). Para isso foi utilizada uma biblioteca de código aberto (*Open Source*) desenvolvida na linguagem Java chamada *JSoup*. A fim de atribuir novas propriedades e estender funcionalidades a essa biblioteca e, dessa forma, possibilitar a realização das propostas contidas neste trabalho, utilizou-se o padrão de projeto *Decorator* (GAMMA et al., 2008) sobre a classe principal dessa biblioteca.

Os métodos de retirada de itens visuais e demais estruturas inúteis à extração de dados, bem como os reconhecedores de vocabulários do *Schema.org* (item 2 na Figura 19), também foram implementados em linguagem Java, seguindo as proposições discutidas nas Seções 5.2.1 e 5.2.2. O algoritmo é composto por três passos principais: i) Retirada de cabeçalhos e rodapés; ii) Retirada de *scripts*, *links*, itens de estilo, metadados e estruturas ocultas; iii) Procura por elementos do vocabulário do *Schema.org*.

A Figura 20 mostra um fragmento de uma página antes e depois da redução.

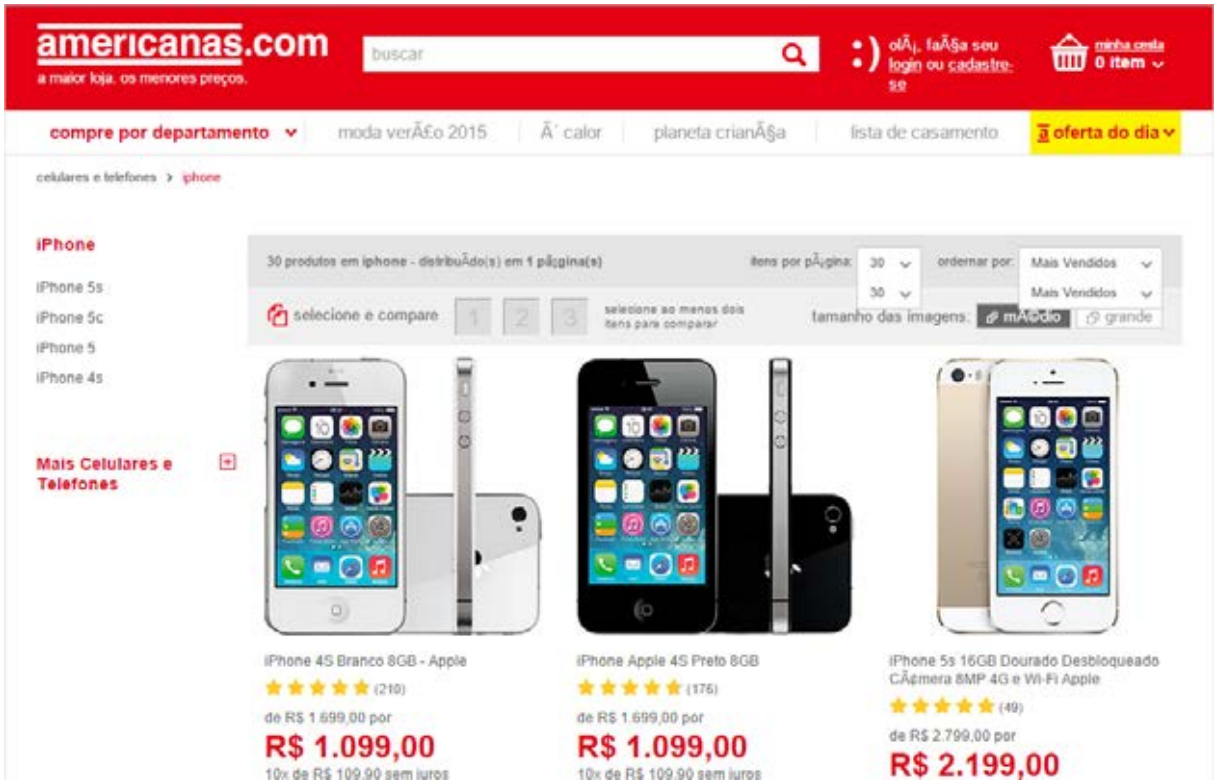
6.2.2 Extração de Padrões em Páginas Web

O algoritmo escolhido para desempenhar a tarefa de extração de padrões dentro de páginas de comércio eletrônico (item 3 na Figura 19) foi o algoritmo *Generalized Simple Tree Matching* apresentado na Seção 4.3. Toda a implementação do algoritmo foi realizada utilizando-se a linguagem Java e foi baseada nos pseudoalgoritmos dos métodos *GSTM*, *Detect-Lists* e *UpdateW* apresentados no trabalho de Jindal e Liu (2010) e adaptados para este trabalho de acordo com o conteúdo das Seções 4.3 (Algoritmo 2), 4.3.1 (Algoritmo 3) e 4.3.3 (Algoritmo 4), já que não há implementação do G-STM disponível publicamente.

Este trabalho, assim como o de Jindal e Liu (2010), adotou a seguinte configuração de parâmetros para uso no método de detecção de listas (Algoritmo 4.3.1):

- $\tau_1 = 0.5$: limiar (ou *threshold*) referente à pontuação normalizada mínima que duas subárvores em um par, para serem consideradas similares, devem possuir em relação àquela que, desse par, possui o maior número de nós.
- $\tau_2 = 0.7$: limiar referente à proporção mínima de nós idênticos que duas subárvores em um par, para serem consideradas similares, devem possuir em relação à árvore irmã de maior valor de *matching*.

Ambos os parâmetros foram determinados empiricamente, testando-se variações nos tamanhos de páginas e, consequentemente, variações nas alturas das árvores e quantidades de nós contidos nessas árvores. Conforme explicado no Capítulo 4, o algoritmo *Generalized*



(a)

(b)

Figura 20 – Páginas de vitrine de uma loja de comércio eletrônico. (a) Página sem a redução. (b) Página após a execução dos métodos de redução.

Simple Tree Matching encontra padrões dentro das páginas e é capaz de lidar com listas (registros de dados) de tamanhos variáveis. A Figura 21 mostra os registros de dados encontrados em uma página de vitrine em uma loja de comércio eletrônico após a execução do algoritmo, sendo tais registros destacados pelas linhas tracejadas. Na figura, dois registros foram encontrados.

Figura 21 – Exemplo de uma página de vitrine de comércio eletrônico com os registros de dados encontrados destacados pelas linhas pontilhadas.

6.2.3 Extração dos Dados de Interesse

Os métodos de extração dos dados de interesse que, neste trabalho, são os dados dos produtos contidos em páginas de comércio eletrônico, assim como os demais métodos, foram implementados em linguagem Java. Tais métodos implementam as heurísticas (item 4 na Figura 19) definidas na Seção 5.4.1 por meio da aplicação de um motor de regras sobre os resultados encontrados pelo algoritmo de extração de padrões (item 3 na Figura 19). As regras possuem prioridades entre si, sendo aplicadas na seguinte ordem:

1. Procura de propriedades do *Schema.org*;
2. Altura mínima da árvore para que contenha dados;
3. Procura pelo nome do produto;
4. Procura pelos preços do produto.

A primeira regra impede a execução das demais, ou seja, se forem encontradas as propriedades de nome e preço do produto definidas pelo *Schema.org*, não há motivos para que as três demais regras sejam executadas. Caso não sejam encontradas as propriedades do *Schema.org* que são de interesse, as demais regras serão executadas e somente se as três obtiverem sucesso é que o produto será extraído, a fim de que sejam evitados dados parciais no conjunto de resultados.

6.2.4 Protocolo de testes

Os experimentos realizados para a validação das estratégias de pré-processamento e preparação das páginas para extração de dados procuraram determinar a sensibilidade quanto aos tempos de resposta da técnica de *Tree Matching* empregada neste trabalho em relação ao tamanho da página que recebe como entrada (item 5 na Figura 19). Primeiramente, o algoritmo foi executado sobre páginas completas, ou seja, sem qualquer tratamento no que diz respeito à eliminação de estruturas contidas nessas páginas (item 6 na Figura 19). Posteriormente, o mesmo algoritmo foi executado, porém dessa vez sobre páginas que passaram pela etapa de pré-processamento para a redução do seu tamanho (item 6 na Figura 19). Em ambos os casos foram anotados os tempos de execução do algoritmo até o final do processo de *matching* das estruturas de cada página para que, a partir desses resultados, fosse determinado o ganho obtido com as reduções (item 7 na Figura 19).

Já os experimentos com intuito de avaliar os métodos de extração de regiões de interesse e extração dos dados (item 6 na Figura 19) foram realizados de forma independente do experimento anterior e, para cada resultado, gerou-se uma pontuação a qual foi utilizada para a avaliação da efetividade das técnicas.

Para avaliar o desempenho dos métodos de separação das regiões de interesse foi atribuído o valor de 1 ponto para cada região corretamente extraída. O total de regiões de interesse disponível em cada página foi determinado manualmente.

Da mesma forma, foi atribuído o valor de 1 ponto para cada produto integralmente extraído da página durante a avaliação de desempenho da etapa de extração dos dados. Vale ressaltar que o produto ora referido é composto pelas propriedades *nome*, *preço* e *preço promocional*, sendo que as duas primeiras são obrigatórias e a última é opcional. Sendo assim, foram considerados como produtos corretamente extraídos somente aqueles cujas propriedades foram corretamente preenchidas, respeitando-se o critério de opcionalidade. A quantidade total de produtos contidos em cada página para a comparação com os resultados obtido pelos métodos foi determinada manualmente.

6.2.5 Indicadores de Desempenho

Para avaliar o ganho de tempo obtido no processo por meio da redução do tamanho das páginas analisadas, foram correlacionados os tempos de execução do método proposto na forma de uma razão, a qual foi traduzida em termos percentuais, como definido abaixo, em que T_a é o tempo gasto pelo método em uma situação a e T_b é o tempo gasto pelo algoritmo em uma segunda situação chamada b:

$$\left(1 - \frac{T_b}{T_a}\right) * 100 \tag{6.1}$$

Quanto mais próximo de 100 for o valor do ganho de tempo, mais eficiente a execução do algoritmo foi na situação *b* em relação à situação *a*. Por outro lado, valores negativos indicam que a situação *b* foi menos eficiente, em termos temporais, quando comparada à situação *a*.

Já para a determinação do desempenho tanto para a etapa de extração das regiões de interesse quanto para a etapa da extração dos dados de produtos das páginas, foram utilizados dois indicadores. Tais indicadores são amplamente adotados para a avaliação dos resultados obtidos por algoritmos de extração de dados, seja qual for o domínio dessa extração (resultados de sites de buscas, classificados de empregos, notícias e dados de produtos, etc). Tais indicadores foram empregados, por exemplo, nos trabalhos de Zhai e Liu (2006), Jindal e Liu (2010) e Ferrara e Baumgartner (2011).

- *Precisão* (do inglês, *Precision*): definida como a razão entre o número de itens extraídos corretamente e o total de itens extraídos. A *precisão* é dada pela equação 6.2, em que E_c é o total de itens corretamente extraídos e E_t é o número total de itens extraídos;

$$Precisão = \frac{E_c}{E_t} \tag{6.2}$$

  Os valores da *Precisao* são multiplicados por 100 nos experimentos para representação em termos percentuais. Logo, quanto mais próximo de 100% for o valor da razão que determina a *Precisão*, maior será a qualidade dos resultados;

- *Revocação* (do inglês, *Recall*): definido como a razão entre o número de itens extraídos corretamente e o total de itens que deveriam ser retornados. O indicador de *Revocação* é dado pela equação 6.3, em que E_c é o total de itens corretamente extraídos e N_t é o número total de itens disponíveis para a extração.

$$Revocação = \frac{E_c}{N_t} \tag{6.3}$$

  Da mesma forma que o indicador de *Precisão*, os valores do indicador de *Revocação* são multiplicados por 100 nos experimentos para representação em termos percentuais. Quanto mais próximo de 100% for o valor da razão que determina a *Revocação*, maior será a qualidade dos resultados.

A fim de relacionar os dois indicadores anteriores, foi utilizado um indicador auxiliar chamado *F-score*, também conhecido como $F_1 score$ ou *F-measure*, que é a média harmônica entre a *Precisão* e a *Revocação* e é definida pela igualdade 6.4. Quanto mais próximo de 1 for o valor do *F-score*, melhores serão os resultados.

$$F - score = 2 \cdot \frac{Precisão \cdot Revocação}{Precisão + Revocação} \tag{6.4}$$

6.3 Considerações Finais

Este capítulo apresentou as duas bases utilizadas para a validação dos métodos propostos neste trabalho. A base *TBDW versão 1.02* foi escolhida por já ter sido empregada em experimentos de outros autores, enquanto a base *Ecommerce DB versão 1.0* foi construída a partir de páginas de vitrine de lojas *online* brasileiras exclusivamente para este trabalho. Foram apresentados também os itens de *hardware* e *software* adotados para o desenvolvimento e execução dos experimentos, seguidos da seção da metodologia, na qual foram apresentadas as técnicas escolhidas e os parâmetros empegados nos algoritmos. Por fim, foi apresentado o protocolo de testes bem como os indicadores de avaliação adotados para a validação dos resultados obtidos nos experimentos, resultados esses que são apresentados no próximo capítulo.

7 Resultados

Neste capítulo são apresentados os experimentos realizados bem como a análise dos resultados obtidos pela aplicação das técnicas descritas neste trabalho no que diz respeito à extração de dados de produtos em páginas de comércio eletrônico.

Os experimentos foram divididos em dois grupos. O primeiro grupo refere-se à validação da extração de padrões em páginas Web e atua sobre as etapas de pré-processamento e processamento, descritas na Seções 5.2 e 5.3. Já o segundo grupo foca-se na avaliação dos resultados obtidos quanto à extração de dados de produtos das páginas de comércio eletrônico, atuando sobre a etapa de pós-processamento, descrita na Seção 5.4.

7.1 Primeiro Grupo de Experimentos

O primeiro grupo de experimentos foi criado com o objetivo de avaliar o algoritmo *Generalized Simple Tree Matching* apresentado no Capítulo 4, tanto na sua versão original quanto melhorada. O primeiro experimento desse grupo busca analisar os impactos que a limpeza de itens de estilo e formatação além da exclusão de cabeçalhos e rodapés e o uso dos conceitos do *Schema.org* imprimem ao algoritmo no que diz respeito ao seu tempo de resposta, já que é ele quem realiza as comparações entre os nós da árvore montada pelo modelo DOM da página. Já o segundo experimento desse grupo visa avaliar o impacto da mudança proposta na Seção 4.3.5 na melhoria da acurácia do algoritmo G-STM, especificamente na redução da quantidade de Falsos Negativos, ou seja, resultados que deveriam ser retornados mas não foram.

Para o primeiro experimento foi empregada a base de páginas Web *Ecommerce DB versão 1.0*, descrita no Capítulo 6. O segundo experimento emprega a base de páginas *TBDW versão 1.02*, visto que ela foi utilizada nos experimentos realizados por Jindal e Liu (2010) durante a validação do algoritmo G-STM original e apresentou bons resultados.

7.1.1 Avaliação do tempo de resposta

Este experimento utiliza o algoritmo G-STM em sua versão original. Foram definidas quatro métricas para avaliar o tempo de resposta do algoritmo G-STM, **com** e **sem** a retirada de itens de estilo e formatação além da exclusão de cabeçalhos e rodapés sobre as páginas da base *Ecommerce DB versão 1.02*. As métricas definidas são as seguintes:

- *TMM* (Tempo Médio de *Matching*): métrica definida como a média de tempo, em

segundos, do começo do *matching* até o seu término para as páginas de uma loja. Auxilia no cálculo do tempo ganho na execução do *matching* entre uma árvore completa e uma árvore reduzida;

- *TMP* (Tempo Médio do Processamento): métrica definida como a média de tempo, em segundos, desde começo da construção do modelo DOM para um página, passando pela eliminação das estruturas, até o término do *matching* para as páginas de uma loja. Esta métrica auxiliará no cálculo de quanto tempo a retirada de estruturas da árvore construída para a página toma, a fim de que seja determinado se o ganho no tempo de resposta não foi anulado pelo tempo gasto com as eliminações de nós não úteis à extração na árvore.

Tais métricas foram relacionadas ao indicador de desempenho de ganho de tempo definido na Seção 6.2.5 da seguinte maneira:

- *GM* (Ganho no *Matching*): métrica definida como o ganho de tempo no *matching*, em termos percentuais, obtido com a retirada de estruturas da página. Esta métrica é calculada a partir do quociente entre os valores obtidos para a métrica *TMM* depois e antes da retirada de estruturas, como definido abaixo, em que TMM_a é o tempo médio do processo de *matching* da página sem que as estruturas sejam retiradas e TMM_b é o tempo médio do processo de *matching* da página reduzida;

$$\left(1 - \frac{TMM_b}{TMM_a}\right) * 100 \tag{7.1}$$

- *GR* (Ganho Real): métrica definida como o ganho de tempo real, em termos percentuais, obtido com a retirada de estruturas da página. Esta métrica é calculada a partir do quociente entre os valores obtidos para a métrica *TMP* depois e antes da retirada de estruturas, como definido abaixo, em que TMP_a é o tempo médio do processamento completo da página sem que as estruturas sejam retiradas e TMP_b é o tempo médio do processamento completo da página reduzida, ou seja, sem as estruturas definidas pelas heurísticas apresentadas na Seção 5.2.1.

$$\left(1 - \frac{TMP_b}{TMP_a}\right) * 100 \tag{7.2}$$

A Tabela 13 apresenta os resultados obtidos após a execução das etapas de pré-processamento e processamento sem a eliminação e com a eliminação de estruturas das páginas Web pertencentes à base *Ecommerce DB versão 1.0*. As medições de tempo foram realizadas sempre utilizando-se o mesmo ambiente no que diz respeito aos processos e serviços executados em paralelo aos testes, sendo que o único processo do nível de usuário executado foi o processo Java referente aos algoritmos analisados, a fim de que todos os

testes fossem realizados em um ambiente controlado. Tal controle foi estabelecido para que fossem minimizadas ao máximo as interferências externas quanto à alocação de recursos da máquina, que poderiam interferir na medição dos tempos nos testes.

Tabela 13 – Tempos médios de execução obtidos nas páginas Web da base *Ecommerce DB versão 1.02*. TMM_a e TMM_b referem-se ao tempo médio somente do *matching* antes e depois da redução da página, respectivamente. Já TMP_a e TMP_b referem-se ao tempo médio do processamento total antes e depois da redução da página, respectivamente. *GM* e *GR* referem-se aos ganhos de tempo somente durante o *matching* e ganho de tempo real, respectivamente.

Página	$TMM_a(s)$	$TMP_a(s)$	$TMM_b(s)$	$TMP_b(s)$	$GM(\%)$	$GR(\%)$
Americanas	50,04	50,13	9,39	9,51	81,24	81,03
Submarino	32,68	32,87	11,71	11,89	64,17	63,83
Netshoes	39,08	39,18	17,41	17,56	55,45	55,18
Ponto Frio	21,37	21,44	18,27	18,36	14,51	14,37
Magazine Luiza	18,72	18,83	15,54	15,59	17,21	16,99
Casas Bahia	5,72	5,77	5,55	5,63	2,97	2,43
Walmart	30,52	30,57	14,58	14,64	52,23	52,11
Livraria Saraiva	10,44	10,51	9,80	9,82	6,57	6,13
Fast Shop	43,83	43,91	5,67	5,76	87,06	86,88
Decathlon	26,57	26,61	31,29	31,43	-17,76	-18,11

Os valores da coluna que indicam o Ganho Real do processo (GR) deixam claro o ganho obtido com a retirada de estruturas visuais, cabeçalhos, rodapés e itens ocultos das páginas. Trabalhando-se com páginas reduzidas, o algoritmo *Generalized Simple Tree Matching* obteve, no melhor dos casos, um ganho médio de tempo na ordem de 87% e um ganho real médio, considerando todas as lojas, na ordem de 36% quando comparado à execução do algoritmo para as mesmas páginas, porém completas. Das 10 lojas avaliadas, seis delas implementam o vocabulário de produtos do *Schema.org*: *Americanas*, *Fast Shop*, *Netshoes*, *Submarino*, *Walmart* e *Decathlon*. Não por acaso, com exceção da loja *Decathlon*, tais lojas foram as que apresentaram os maiores ganhos de tempo real, pois permitiram que o algoritmo de *Tree Matching* direcionasse seus esforços para as partes corretas do código HTML.

A diferença existente entre os valores do ganho real e do ganho no *matching* evidenciam que o tempo gasto com a redução da página, antes que ela começasse a ser percorrida pelo algoritmo de *matching*, não anula o ganho de tempo conseguido durante o *matching*.

Somente uma das 10 lojas analisadas apresentou piora em relação aos resultados obtidos com páginas reduzidas. A loja Decathlon foi adicionada à base *Ecommerce DB versão 1.0* justamente por suas características peculiares quanto à estrutura interna. No caso dessa loja em particular, parte das estruturas visuais fazem com que o algoritmo satisfaça as situações de não similaridade determinadas pela linha 1 do *Generalized Simple*

Tree Matching mais rapidamente, poupando comparações subsequentes. Isso faz com que o algoritmo termine sua execução para as páginas dessa loja mais rapidamente quando a página está completa. O acréscimo de tempo ao processo para a retirada de estruturas fez com que, nesse caso, a utilização de páginas reduzidas tomasse mais tempo do que as completas. No entanto, este não é o padrão observado nos experimentos realizados, sendo que a grande maioria apresentou um ganho real bastante satisfatório no que diz respeito ao tempo de resposta.

O gráfico da Figura 22 ilustra o comportamento constatado no experimento. A distância observada entre as barras que descrevem o comportamento do algoritmo para as páginas completas em relação às páginas reduzidas, ao longo do eixo das abcissas, deixa claro que o ganho de tempo obtido pelas propostas apresentadas neste trabalho foi bastante efetivo. O intervalo de confiança para a amostra de cada loja também é ilustrado no topo de cada uma das barras contidas no gráfico.

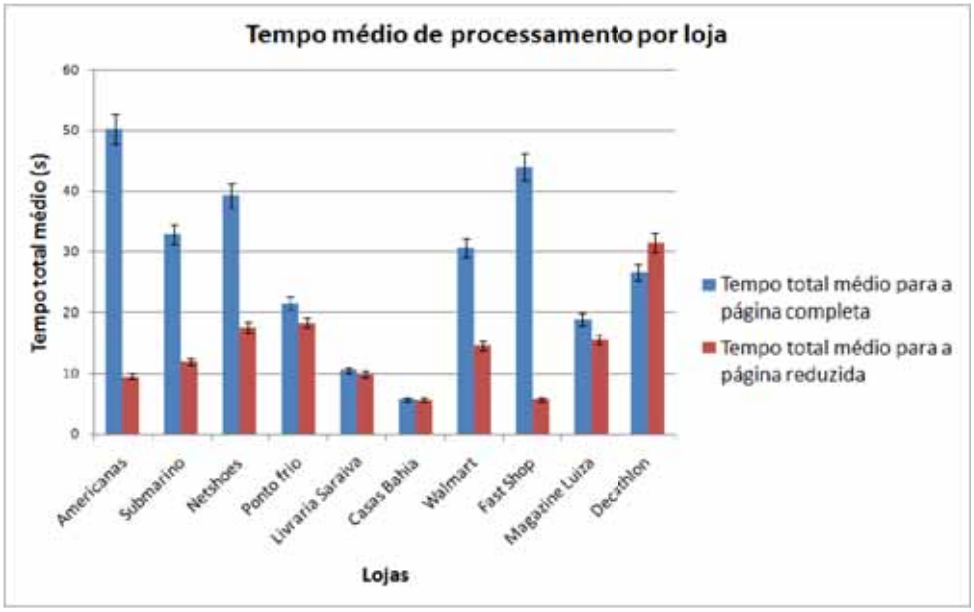


Figura 22 – Gráfico comparativo entre o tempo médio do processo de extração de dados de uma página completa e uma página reduzida.

7.1.2 Avaliação da precisão

Este experimento utiliza o algoritmo G-STM em sua versão original e em sua versão melhorada apresentada na Seção 4.3.5. O protocolo empregado no experimento que avalia a precisão do algoritmo com a melhoria proposta foi o mesmo utilizado no trabalho de Jindal e Liu (2010). A base utilizada para este experimento, inclusive, é a mesma utilizada nos testes do algoritmo original. Assim, foram escolhidas 51 páginas de um total de 255 que constituem a base. Cada uma das páginas pertence a um dos 51 domínios que compõem a base, submetendo-se então ao algoritmo uma página de cada tipo. O trabalho o qual este experimento se baseia não especifica qual página de cada um dos conjuntos de domínios

foi utilizada nos testes. Este experimento utilizou sempre a amostra de página número 1 pertencente a cada domínio.

Foram definidas três métricas com o objetivo de avaliar a acurácia do algoritmo antes e depois da melhoria proposta na Seção 4.3.5, métricas essas que foram então submetidos aos indicadores de desempenho apresentados na Seção 6.2.5. São elas:

- VP (Verdadeiro Positivo): métrica definida como a quantidade de resultados corretos encontrados;
- FP (Falso Positivo): métrica definida como a quantidade de resultados dados como corretos mas que na verdade não são;
- FN (Falso Negativo): métrica definida como a quantidade de resultados que deveriam ser dados como corretos mas que não foram extraídos pelo algoritmo.

Tais métricas foram relacionadas aos indicadores de *Precisão* e *Revocação* definidos na Seção 6.2.5 da seguinte maneira:

- *Precisão*: $E_c = VP$ e $E_t = VP + FP$. Logo:

$$Precisão = \frac{VP}{VP + FP} \tag{7.3}$$

- *Revocação*: $E_c = VP$ e $N_t = VP + FN$. Logo:

$$Revocação = \frac{VP}{VP + FN} \tag{7.4}$$

A Tabela 14 apresenta os resultados obtidos durante a execução do algoritmo *Generalized Simple Tree Matching* sobre a base de páginas *TBDW versão 1.02* sem a melhoria proposta.

Tabela 14 – Quantidade de Verdadeiros Positivos (VP), Falsos Positivos (FP) e Falsos Negativos (FN) encontrados durante o experimento com o algoritmo G-STM sem a melhoria proposta. Utilizou-se uma página de cada um dos 51 domínios contidos na base *TBDW versão 1.02*.

Domínio	**VP**	**FP**	**FN**	**Domínio**	**VP**	**FP**	**FN**
1	22	2	3	27	23	0	1
2	15	0	0	28	11	0	1
3	24	0	1	29	96	0	0
4	27	0	6	30	0	1	10
5	0	0	25	31	36	1	1
6	4	0	1	32	4	0	1
7	7	0	0	33	19	0	10
8	17	0	11	34	15	0	1
9	12	0	0	35	18	0	0
10	77	0	0	36	13	0	18
11	16	0	1	37	56	0	0
12	15	0	7	38	11	0	0
13	16	0	1	39	8	0	2
14	14	0	0	40	60	0	0
15	32	0	2	41	33	0	1
16	6	0	15	42	26	0	0
17	167	0	6	43	16	0	25
18	6	0	8	44	46	0	0
19	6	0	51	45	14	0	1
20	16	0	0	46	5	0	15
21	28	0	104	47	6	0	0
22	36	0	10	48	56	0	0
23	20	0	4	49	27	0	2
24	16	0	31	50	73	0	1
25	38	0	0	51	19	0	0
26	28	0	1				
TOTAL	665	2	288	**TOTAL**	691	2	90

Os valores da *Precisão*, *Revocação* e *F-score* obtidos para o experimento empregando o algoritmo sem a melhoria estão expostos na Tabela 15.

Tabela 15 – *Precisão*, *Revocação* e *F-score* obtidos pelo experimento na base *TBDW versão 1.02*, utilizando o algoritmo G-STM sem a melhoria.

Indicador de desempenho	**Valor**
Precisão	99,71%
Revocação	78,20%
F-score	0,88

A Tabela 16 apresenta os resultados obtidos durante a execução do algoritmo

Generalized Simple Tree Matching sobre a base de páginas *TBDW versão 1.02* com a melhoria proposta.

Tabela 16 – Quantidade de Verdadeiros Positivos (VP), Falsos Positivos (FP) e Falsos Negativos (FN) encontrados durante o experimento com o algoritmo G-STM implementando a melhoria proposta. Utilizou-se uma página de cada um dos 51 domínios contidos na base *TBDW versão 1.02*.

Domínio	**VP**	**FP**	**FN**	**Domínio**	**VP**	**FP**	**FN**
1	24	2	1	27	23	0	1
2	15	0	0	28	11	0	1
3	24	0	1	29	96	0	0
4	31	0	2	30	0	1	10
5	0	0	25	31	36	1	1
6	4	0	1	32	5	0	0
7	7	0	0	33	19	0	10
8	18	0	10	34	15	0	1
9	12	0	0	35	18	0	0
10	77	0	0	36	31	0	0
11	16	0	1	37	56	0	0
12	22	0	0	38	11	0	0
13	16	0	1	39	10	0	0
14	14	0	0	40	60	0	0
15	32	0	2	41	33	0	1
16	6	0	15	42	26	0	0
17	172	0	1	43	16	0	25
18	14	0	0	44	46	0	0
19	57	0	0	45	14	0	1
20	16	0	0	46	5	0	15
21	32	0	100	47	6	0	0
22	46	0	0	48	56	0	0
23	23	0	1	49	27	0	2
24	46	0	1	50	74	0	0
25	38	0	0	51	19	0	0
26	28	0	1				
TOTAL	790	2	163	**TOTAL**	713	2	68

Os valores da *Precisão*, *Revocação* e *F-score* obtidos para o experimento empregando o algoritmo melhorado estão expostos na Tabela 17.

Tabela 17 – *Precisão*, *Revocação* e *F-score* obtidos pelo experimento na base *TBDW versão 1.02* utilizando a versão melhorada do algoritmo G-STM.

Indicador de desempenho	**Valor**
Precisão	99,73%
Revocação	86,68%
F-score	0,93

A melhoria inserida no algoritmo *Generalized Simple Tree Matching* proposta na Seção 4.3.5 possui como objetivo a redução da quantidade de Falsos Negativos nos resultados das extrações de padrões em páginas Web. O indicador que relaciona o número de Falsos Negativos é a *Revocação*, logo a melhoria proposta ao algoritmo neste trabalho deve ser sentida com maior impacto nesse indicador. Analisando-se os valores da *Revocação* nas Tabelas 15 e 17, percebe-se um aumento de aproximadamente oito pontos percentuais no valor refente ao indicador para o algoritmo que implementou a mudança, evidenciando uma significativa melhora no que se refere à eficácia do método proposto por esse trabalho. Assim, subárvores da página que possuem listas em seus níveis mais baixos e não eram consideradas padrões de dados devido à incoerência nas pontuações de *matching* obtidas pelo algoritmo, a partir de então passaram a ser consideradas regiões de dados, diminuindo-se, dessa forma, a quantidade de itens Falsos Negativos retornados pelo processo de extração de padrões. A melhoria proposta ao G-STM cumpriu, portanto, as expectativas nela depositadas.

7.2 Segundo Grupo de Experimentos

O segundo grupo de experimentos utiliza a base *Ecommerce DB versão 1.0*, montada especialmente para este trabalho, a qual é constituída exclusivamente de páginas de vitrine das maiores lojas de comércio eletrônico brasileiras segundo a publicação E-commercebrasil (2011), conforme o Capítulo 6 apresentou. Essa base constitui assim o ambiente mais próximo do real possível que um processo de extração de dados de produtos em páginas de comércio eletrônico pode atuar.

O segundo grupo de experimentos foca a acurácia dos dados dos produtos encontrados e extraídos de páginas de comércio eletrônico, ou seja, o resultado final da última etapa do processo de extração (*Pós-processamento*), empregando a versão melhorada do algoritmo *Generalized Simple Tree Matching*, apresentada na Seção 4.3.5, e páginas reduzidas. Os indicadores adotados para a avaliação dos resultados da extração dos dados de produtos são os mesmos definidos na Seção 7.1.2: *Precisão*, *Revocação* e *F-score*. Para que um produto extraído seja considerado um Verdadeiro Positivo não basta que apenas seu nome ou preço sejam reconhecidos. Ambas as propriedades do produto devem ser corretamente extraídas. O preço promocional não é obrigatório. Caso apenas uma das propriedades seja extraída, o produto será considerado um Falso Positivo. Falsos Positivos compreendem os itens extraídos que são reconhecidos como produtos porém são, na verdade, outras estruturas de repetição, como menus e *banners*, por exemplo.

7.2.1 Avaliação da precisão

O algoritmo G-STM melhorado foi executado sobre a base *Ecommerce DB versão 1.0* para cada conjunto de 20 páginas pertencente a cada loja separadamente e os resultados obtidos foram submetidos às heurísticas de extração de dados ora propostas. A Tabela 18 mostra os valores obtidos para as métricas em cada uma das lojas.

Tabela 18 – Quantidade por loja de itens Verdadeiros Positivos (VP), Falsos Positivos (FP) e Falsos Negativos (FN) obtidos para o experimento na base *Ecommerce DB versão 1.0*.

Domínio	**VP**	**FP**	**FN**
Americanas	626	11	0
Submarino	690	23	39
Netshoes	796	136	2
Ponto Frio	467	33	0
Magazine Luiza	393	44	46
Casas Bahia	361	14	16
Walmart	388	19	275
Livraria Saraiva	663	26	0
Fast Shop	89	6	22
Decathlon	146	2	124

Os valores de *Precisão*, *Revocação* e *F-score* obtidos para cada loja são mostrados pela Tabela 19.

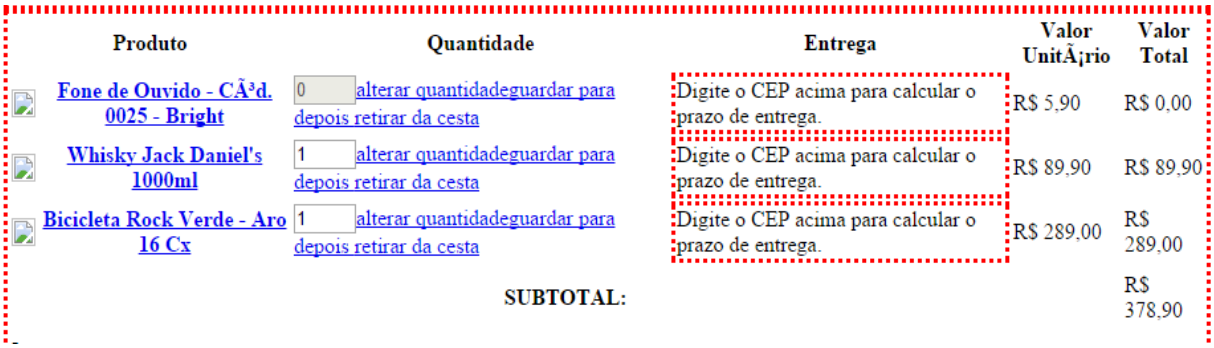
Tabela 19 – Valores de *Precisão*, *Revocação* e *F-score* obtidos pelo experimento na base *Ecommerce DB versão 1.0* acerca dos produtos extraídos pelo processo para cada loja da base.

Domínio	**Precisão** (%)	**Revocação** (%)	**F-score**
Americanas	98,27	100	0,99
Submarino	96,77	94,65	0,96
Netshoes	85,41	99,75	0,92
Ponto Frio	93,40	100	0,97
Magazine Luiza	89,93	89,52	0,90
Casas Bahia	96,27	95,76	0,96
Walmart	95,33	58,52	0,73
Livraria Saraiva	96,23	100	0,98
Fast Shop	93,68	80,18	0,86
Decathlon	98,65	54,07	0,70

Analisando-se a tabela acima, chamam a atenção os baixos valores obtidos nas medidas relacionadas ao indicador de *Revocação* para as lojas *Walmart* (58,52%) e *Decathlon* (54,07%). Tais valores ficaram consideravelmente menores em relação aos demais devido ao elevado número de resultados Falsos Negativos, ou seja, produtos que deixaram de ser extraídos, mesmo ambas as lojas fazendo parte do grupo de lojas que implementam o

Schema.org na base de páginas. Verificou-se que as regiões de dados que contém os produtos não está sendo reconhecida pelo algoritmo de *Tree Matching* e, por isso, não chega à etapa de pós-processamento a fim de terem os dados dos produtos extraídos. Tal algoritmo não reconhece as estruturas de repetição como deveria por peculiaridades na forma como os sites foram estruturados, especificamente no que diz respeito às páginas com resultados de buscas, que também configura-se uma vitrine de produtos. Os setores contendo vitrines tanto nas páginas iniciais quanto em páginas híbridas dessas lojas (páginas de carrinho com vitrine de produtos sugeridos, por exemplo), tiveram seus produtos corretamente extraídos.

A Figura 23 ilustra um exemplo de estrutura em uma página de comércio eletrônico que possui padrões de repetição de *tags* e tal estrutura é extraída da página pelo processo como se fosse um produto. Isso se dá porque as heurísticas de altura da árvore, quantidade de palavras no nome do produto e valor numérico, definidas na Seção 5.4.1, são respeitadas, mas o conteúdo não se trata de um produto, configurando um resultado Falso Positivo.



Produto	Quantidade	Entrega	Valor UnitÃ¡rio	Valor Total
Fone de Ouvido - CÃ³d. 0025 - Bright	0 alterar quantidadeguardar para depois retirar da cesta	Digite o CEP acima para calcular o prazo de entrega.	R$ 5,90	R$ 0,00
Whisky Jack Daniel's 1000ml	1 alterar quantidadeguardar para depois retirar da cesta	Digite o CEP acima para calcular o prazo de entrega.	R$ 89,90	R$ 89,90
Bicicleta Rock Verde - Aro 16 Cx	1 alterar quantidadeguardar para depois retirar da cesta	Digite o CEP acima para calcular o prazo de entrega.	R$ 289,00	R$ 289,00
	SUBTOTAL:			R$ 378,90

Figura 23 – Exemplo de Falso Positivo extraído de uma página de comércio eletrônico.

A Tabela 20 mostra os valores obtidos para a base como um todo, considerando-se todas as lojas juntas.

Tabela 20 – *Precisão*, *Revocação* e *F-score* obtido para o experimento acerca da acurácia na extração de dados de produtos na base *Ecommerce DB versão 1.0*.

Indicador de desempenho	**Valor**
Precisão	93,63%
Revocação	89,81%
F-score	0,92

Percebe-se que a medida relacionada ao indicador de *Revocação* possui o valor mais baixo dentre as três devido às duas lojas com alto número de itens Falsos Negativos nos experimentos. Tal quantidade de Falsos Negativos também reduz o valor do indicador *F-score*. No entanto, o valor da *Precisão*, considerado alto (mais de 93% dos dados extraídos foram corretos), acaba por atenuar essa redução na medida do indicador *F-score*.

8 Conclusões

Com a popularização e a crescente confiança por parte dos consumidores no comércio realizado na Web, o chamado *e-commerce* ou comércio eletrônico, essa categoria de comércio ganhou considerável força entre as grandes lojas do varejo nos últimos anos, o que levou a uma corrida para o desenvolvimento de portais de vendas *online* nos quais as lojas exibem seus estoques a fim de atraírem compradores de forma massiva.

Ciente da grande variedade de portais de vendas na Web atualmente, o processo de extração dos dados de produtos contidos nessas páginas mostra-se uma tarefa bastante complexa haja vista a diversidade de estruturas e conteúdos presentes nas páginas, já que cada portal de vendas *online* emprega um estilo próprio para a construção de seu *website*. Soma-se a tudo isso também a dificuldade decorrente do grande volume de páginas avaliadas a fim de terem seus produtos extraídos, o que torna a tarefa de extração extremamente massante quando executada por um ser humano sem o auxílio de ferramentas computacionais.

Diversas abordagens surgiram com o objetivo de se obter o melhor resultado na extração de dados de páginas Web, diferindo-se entre elas principalmente o grau de interação com o ser humano requerido para o desenvolvimento da tarefa. A abordagem não supervisionada ou automática não exige qualquer intervenção durante sua execução, o que a torna viável para uma grande quantidade de páginas de entrada; no entanto tal abordagem não é capaz de interpretar semanticamente os dados extraídos, exigindo uma etapa de pós processamento que geralmente emprega certas heurísticas para a captura dos dados de interesse. Por sua vez, a abordagem de aprendizado supervisionado apresenta grande eficácia na extração mas, diferentemente da abordagem não supervisionada, exige um elevado nível de interação com um ser humano, o que dificulta sua utilização em larga escala para elevadas quantidades de páginas. A abordagem semi-automática surgiu com o intuito de unir as melhores características das duas abordagens anteriores, exigindo a mínima intervenção possível com o ser humano, para que sejam determinados os dados de interesse dentre todo o conteúdo da página.

Este trabalho propôs contribuições para a área de estudo da abordagem automática no que se refere às estratégias de extração de padrões e dados de produtos em páginas de vitrines de lojas *online*, utilizando para isso algoritmos que determinam regiões de interesse dentro das páginas, regiões essas em que há grande possibilidade de que dados de produtos sejam encontrados. Após a determinação das regiões de interesse, parte-se então para a etapa de extração dos dados dos produtos, etapa cujo foco são o nome e seus preços. Já no que se refere a um sistema que possa ser empregado em escala comercial, este trabalho

o classifica como um sistema semi-automático, pois a interação com o ser humano prevista reside em uma etapa de auditoria que complementa o processo automático de extração, na qual os dados extraídos são aprovados ou reprovados pela figura do auditor no sistema.

Para realizar a tarefa da extração dos padrões e, consequentemente, das regiões de interesse e posteriormente dos dados dos produtos em páginas Web, foram estudadas técnicas de localização de padrões entre nós de árvores, as quais são chamadas de técnicas de *Tree Matching*. O modelo proposto neste trabalho divide o processo de extração em três etapas: a primeira, chamada *pré-processamento*, é especializada na construção de uma árvore (modelo DOM) a partir das *tags* que formam a página HTML de um site de comércio eletrônico, além de reduzir a página por meio de retirada de estruturas que não possuem regiões de dados; a segunda etapa, chamada *processamento*, é especializada na localização e extração das regiões de dados; a terceira, chamada, *pós-processamento*, é especializada na extração dos dados dos produtos das estruturas selecionadas durante a etapa de processamento.

Foram estudados nesta dissertação dois algoritmos para serem aplicados na etapa de processamento, ambos baseados na técnica de *Tree Matching*: *Simple Tree Matching* e *Generalized Simple Tree Matching*. O segundo algoritmo foi escolhido para compor o processo de extração devido a sua maior robustez e capacidade de lidar com listas de dados. Para a etapa de pós-processamento foram desenvolvidas heurísticas baseadas nas peculiaridades dos dados de interesse, heurísticas essas que atuam sobre as regiões de dados extraídas na etapa anterior.

Os resultados obtidos nos experimentos mostram que o ganho de tempo apresentado no processo de extração de padrões realizado na etapa de processamento a partir das técnicas implementadas na etapa de pré-processamento foi bastante efetivo. A base *Ecommerce DB versão 1.0* foi submetida ao processo de extração de regiões de interesse o obteve, no melhor caso, um ganho real na ordem de 87% quando comparado ao processo o qual não utilizou a etapa de pré-processamento. A média do ganho real, em termos de tempo de resposta, ficou na ordem de 36%.

A melhoria na acurácia proposta ao algoritmo *Generalized Simple Tree Matching* também apresentou ganhos significativos. O indicador de *Revocação*, que é influenciado pelas quantidades de itens Falsos Negativos nos resultados, apresentou uma melhora de aproximadamente oito pontos percentuais, evidenciando que a alteração proposta ao algoritmo G-STM causou reflexos bastante positivos no que diz respeito à localização de regiões de dados que antes não eram identificadas.

Os experimentos que visaram medir a eficácia da extração dos dados dos produtos apresentaram elevadas taxas de acerto. A *Precisão* do acerto obtida para as 10 lojas foi, em média, de 93,63%, enquanto a *Revocação* apresentou um valor de 89,81%. A medida relacionada ao indicador *F-score* obteve o valor de 0,92. Os valores das medidas referentes

aos três indicadores de desempenho apontam uma boa taxa de acerto, dado o valor da *Precisão* figurar acima da casa de 93%. No entanto, há ainda uma significativa quantidade de Falsos Negativos observada no processo, dada a diferença entre as medidas referentes aos indicadores de *Precisão* e *Revocação*. Mesmo assim, a elevada taxa de acerto permite que um sistema comercial empregue o processo de extração desenvolvido neste trabalho.

8.1 Contribuições

Ao final deste trabalho, as seguintes contribuições podem ser enumeradas:

1. Proposição de estratégias para a redução de páginas Web buscando a otimização no desempenho quanto ao tempo de resposta em algoritmos de *Tree Matching*;
2. Aperfeiçoamento do algoritmo *Generalized Simple Tree Matching* porposto por Jindal e Liu (2010) buscado melhor precisão no cálculo das pontuações entre subárvores;
3. Proposição de heurísticas para a extração de dados de produtos contidos em padrões extraídos de páginas Web pelo uso de técnicas de *Tree Matching*;
4. Aplicação pioneira dos vocabulários de produtos propostos pelo *Schema.org* na extração de dados de produtos de páginas Web empregando técnicas de *Tree Matching*;
5. Criação de uma base de dados de páginas de comércio eletrônico, *Ecommerce DB*, contendo páginas de vitrines de lojas brasileiras.

8.2 Trabalhos Futuros

Dentre as sugestões de trabalhos futuros, pode-se destacar:

- Avaliação do processo de extração de produtos para outros tipos de páginas de comércio eletrônico e não só o de vitrine, como produto em detalhe, produtos em carrinho e produtos comprados, utilizando, para isso, diversas páginas distintas para a extração de padrões, ao invés de uma única página como acontece para vitrines.
- Analisar as peculiaridades das páginas de busca das lojas *Walmart* e *Decathlon*, a fim adaptar o algoritmo de *Tree Matching* para que apresente os altos níveis de *Precisão* e *Revocação* apresentados pelas demais lojas;
- Incrementar a robustez da heurística que trata do reconhecimento do nome dos produtos, para que demais seções contento texto, como uma possível descrição do produto que esteja muito próxima ao nome não confundam o algoritmo de modo que o nome seja extraído incorretamente. Para o caso dos preços é possível melhorar

a heurística que trata da sua extração no sentido de que o preço normal e o preço promocional não sejam confundidos com outros valores, como valores de parcelas de pagamento, por exemplo;

- Produção de *templates* por loja para os padrões encontrados pelo algoritmo *Generalized Simple Tree Matching*, por meio do emprego da linguagem *XPath*. Dessa forma, páginas de uma loja que já passaram pelo algoritmo de *Tree Matching* não necessitariam passar por todo o processo novamente, a não ser que haja mudanças significativas nas estruturas da página. Essa melhoria reduziria substancialmente o tempo de resposta para a extração de dados, pois a página não precisaria ser analisada novamente, já que a expressão *XPath* tem a capacidade de localizar elementos dentro de um documento estruturado;

- Criação de um processo de reprocessamento de páginas reprovadas durante a etapa de auditoria.

Referências

AKMAL, S.; SHIH, L.-H.; BATRES, R. Ontology-based similarity for product information retrieval. *Computers in Industry*, Elsevier, v. 65, n. 1, p. 91–107, 2014. Citado na página 27.

ARASU, A.; GARCIA-MOLINA, H. Extracting structured data from web pages. In: HALEVY, A. Y.; IVES, Z. G.; DOAN, A. (Ed.). *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*. [S.l.]: ACM, 2003. p. 337–348. ISBN 1-58113-634-X. Citado na página 27.

ARLOTTA, L.; CRESCENZI, V.; MECCA, G.; MERIALDO, P.; TRE, U. R. Automatic annotation of data extracted from large web sites. In: *Proc. Sixth International Workshop on the Web and Databases (WebDB 2003)*. [S.l.: s.n.], 2003. p. 7–12. Citado na página 25.

ASHISH, N.; KNOBLOCK, C. A. Semi-automatic wrapper generation for internet information sources. In: IEEE. *Cooperative Information Systems, 1997. COOPIS'97., Proceedings of the Second IFCIS International Conference on*. [S.l.], 1997. p. 160–169. Citado 2 vezes nas páginas 25 e 27.

COPE, J.; CRASWELL, N.; HAWKING, D. Automated discovery of search interfaces on the web. In: AUSTRALIAN COMPUTER SOCIETY, INC. *Proceedings of the 14th Australasian database conference-Volume 17*. [S.l.], 2003. p. 181–189. Citado na página 73.

CRESCENZI, V.; MECCA, G.; MERIALDO, P. RoadRunner: Towards automatic data extraction from large web sites. In: APERS, P. M. G.; ATZENI, P.; CERI, S.; PARABOSCHI, S.; RAMAMOHANARAO, K.; SNODGRASS, R. T. (Ed.). *VLDB*. Morgan Kaufmann, 2001. p. 109–118. ISBN 1-55860-804-4. Disponível em: <http://www.vldb.org/conf/2001/P109.pdf>. Citado na página 27.

CRESCENZI, V.; MECCA, G.; MERIALDO, P. Roadrunner: automatic data extraction from data-intensive web sites. In: ACM. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. [S.l.], 2002. p. 624–624. Citado na página 27.

E-COMMERCEBRASIL. *TOP 10 Lojas Online Brasileiras 2010/2011*. 2011. Http://www.e-commercebrasil.org/numeros/top10-maiores-lojas-online/. 15 de Fevereiro de 2014. Disponível em: <http://www.e-commercebrasil.org/numeros/top10-maiores-lojas-online/>. Citado 2 vezes nas páginas 75 e 92.

ESTIÉVENART, F.; MEURISSE, J.-R.; HAINAUT, J.-L.; THIRAN, P. Semi-automated extraction of targeted data from web pages. In: IEEE. *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*. [S.l.], 2006. p. 48–48. Citado 2 vezes nas páginas 25 e 27.

FEOFILOFF, P. *Programação Dinâmica*. 2014. Http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html. 14 de Setembro de 2014. Disponível em: <http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html>. Citado na página 41.

FERRARA, E.; BAUMGARTNER, R. Automatic wrapper adaptation by tree edit distance matching. In: *Combinations of Intelligent Methods and Applications*. [S.l.]: Springer, 2011. p. 41–54. Citado 2 vezes nas páginas 27 e 83.

FERRARA, E.; MEO, P. D.; FIUMARA, G.; BAUMGARTNER, R. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, Elsevier, v. 70, p. 301–323, 2014. Citado na página 24.

FREITAG, D. Machine learning for information extraction in informal domains. *Machine learning*, Springer, v. 39, n. 2-3, p. 169–202, 2000. Citado na página 28.

FURCHE, T.; GOTTLOB, G.; GRASSO, G.; ORSI, G.; SCHALLHART, C.; WANG, C. Amber: Automatic supervision for multi-attribute extraction. *arXiv preprint arXiv:1210.5984*, p. 1–22, 2012. Citado 2 vezes nas páginas 24 e 27.

FURCHE, T.; GOTTLOB, G.; GRASSO, G.; GUO, X.; ORSI, G.; SCHALLHART, C.; WANG, C. Diadem: Thousands of websites to a single database. *Proceedings of the VLDB Endowment*, v. 7, n. 14, p. 1845–1856, 2014. Citado na página 27.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. [S.l.]: Bookman, 2008. (Design Patterns). ISBN 9788573076103. Citado na página 79.

GARCIA-MOLINA, H.; HAMMER, J.; MCHUGH, J. Semistructured data: The tsimmis experience. In: *Proceedings of First East-European Workshop on Advances in Database and Information Systems (ADBIS). St. Petersburg Russia*. [S.l.: s.n.], 1997. p. 22–22. Citado na página 28.

GULHANE, P.; MADAAN, A.; MEHTA, R.; RAMAMIRTHAM, J.; RASTOGI, R.; SATPAL, S.; SENGAMEDU, S. H.; TENGLI, A.; TIWARI, C. Web-scale information extraction with vertex. In: IEEE. *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. [S.l.], 2011. p. 1209–1220. Citado na página 27.

HSU, C.-N.; DUNG, M.-T. Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, Elsevier, v. 23, n. 8, p. 521–538, 1998. Citado na página 27.

IFFAT, R.; SAMI, L. K. Understanding the deep web. *Library Philosophy and Practice (e-journal)*, Libraries at University of Nebraska-Lincoln, p. 364, 2010. Citado na página 73.

JINDAL, N.; LIU, B. A generalized tree matching algorithm considering nested lists for web data extraction. In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. *SIAM Data Mining*. [S.l.], 2010. p. 930–941. Citado 24 vezes nas páginas 9, 25, 27, 28, 32, 37, 47, 48, 51, 52, 53, 54, 55, 57, 61, 67, 69, 72, 78, 79, 83, 85, 88 e 97.

KUSHMERICK, N. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, Elsevier, v. 118, n. 1, p. 15–68, 2000. Citado na página 27.

KUSHMERICK, N.; WELD, D. S.; DOORENBOS, R. Wrapper induction for information extraction. In: *Proc. IJCAI-97*. [s.n.], 1997. p. 729–737. Disponível em: <http://citeseer.nj.nec.com/kushmerick97wrapper.html>. Citado na página 27.

LABSKY, M. *Information Extraction from Websites using Extraction Ontologies*. Tese (Doutorado) — University of Economics, 2008. Citado 2 vezes nas páginas 23 e 24.

LAENDER, A.; RIBEIRO-NETO, B.; SILVA, A. da; TEIXEIRA, J. A brief survey of web data extraction tools. *SIGMOD Record*, v. 31, n. 2, p. 84–93, 2002. Citado na página 19.

LAENDER, A. H.; RIBEIRO-NETO, B.; SILVA, A. S. da. Debye–data extraction by example. *Data & Knowledge Engineering*, Elsevier, v. 40, n. 2, p. 121–154, 2002. Citado na página 27.

LIU, B. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. [S.l.]: Springer, 2011. (Data-Centric Systems and Applications). ISBN 9783642194603. Citado 8 vezes nas páginas 18, 19, 23, 25, 29, 30, 32 e 33.

LIU, B.; ZHAI, Y. Net–a system for extracting web data from flat and nested data records. In: *Web Information Systems Engineering–WISE 2005*. [S.l.]: Springer, 2005. p. 487–495. Citado na página 27.

MUSLEA, I.; MINTON, S.; KNOBLOCK, C. Stalker: Learning extraction rules for semistructured, web-based information sources. In: AAAI PRESS MENLO PARK, CA. *Proceedings of AAAI-98 Workshop on AI and Information Integration*. [S.l.], 1998. p. 74–81. Citado na página 27.

MUSLEA, I.; MINTON, S.; KNOBLOCK, C. A hierarchical approach to wrapper induction. In: ACM. *Proceedings of the third annual conference on Autonomous Agents*. [S.l.], 1999. p. 190–197. Citado na página 27.

NICOL, G.; WOOD, L.; CHAMPION, M.; BYRNE, S. *Document Object Model (DOM) Level 3 Core Specification*. World Wide Web Consortium, 2001. 08 de Fevereiro de 2014. Disponível em: <http://www.w3.org/TR/DOM-Level-3-Core/>. Citado 3 vezes nas páginas 9, 13 e 35.

PARBERRY, I. *Problems on Algorithms*. [S.l.]: Prentice Hall, 1995. ISBN 9780134335582. Citado 2 vezes nas páginas 40 e 41.

PARTEE, B.; MEULEN, A. T.; WALL, R. *Mathematical methods in linguistics*. [S.l.]: Springer, 1990. Citado na página 53.

RAGGETT, D.; LAM, J.; ALEXANDER, I.; KMIEC, M. A history of html. In: _____. *Raggett on HTML 4*. England: Addison Wesley Longman, 1998. 16 de Novembro de 2014. Disponível em: <http://www.w3.org/People/Raggett/book4/ch02.html>. Citado na página 23.

RAMOS, M. V. M. *Linguagens Formais e Autômatos*. 383 p. Monografia (Especialização) — Universidade Federal do Vale do São Francisco, Petrolina, 2008. Citado 2 vezes nas páginas 53 e 55.

REAL, L.; ANDRADE, R. *Linguagem: teoria, análise e aplicações*. 125 p. Monografia (Pós-graduação) — Universidade Federal do Paraná e Universidade de São Paulo, 2011. Citado na página 53.

REIS, D. d. C.; GOLGHER, P. B.; SILVA, A.; LAENDER, A. Automatic web news extraction using tree edit distance. In: ACM. *Proceedings of the 13th international conference on World Wide Web*. [S.l.], 2004. p. 502–511. Citado na página 37.

SHANNON, R. *What is HTML*. 2007. Http://www.yourhtmlsource.com/starthere/whatishtml.html. 09 de Fevereiro de 2014. Disponível em: <http://www.yourhtmlsource.com/starthere/whatishtml.html>. Citado na página 29.

SIMON, K.; LAUSEN, G. Viper: augmenting automatic information extraction with visual perceptions. In: ACM. *Proceedings of the 14th ACM international conference on Information and knowledge management*. [S.l.], 2005. p. 381–388. Citado na página 27.

VANDIC, D.; NEDERSTIGT, L. J.; AANEN, S. S.; FRASINCAR, F.; HOGENBOOM, F. Ontology population from web product information. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. *Proceedings of the companion publication of the 23rd international conference on World wide web companion*. [S.l.], 2014. p. 391–392. Citado na página 27.

W3. *What is the Document Object Model?* 2014. Http://www.w3.org/TR/WD-DOM/introduction.html. 09 de Fevereiro de 2014. Disponível em: <http://www.w3.org/TR/WD-DOM/introduction.html>. Citado na página 34.

W3SCHOOLS. *The HTML DOM Document Object*. 2014. Http://www.w3schools.com/jsref/dom_obj_document.asp. 09 de Fevereiro de 2014. Disponível em: <http://www.w3schools.com/jsref/dom_obj_document.asp>. Citado na página 34.

YAMADA, Y.; CRASWELL, N.; NAKATOH, T.; HIROKAWA, S. Testbed for information extraction from deep web. In: ACM. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. [S.l.], 2004. p. 346–347. Citado na página 72.

YANG, W. Identifying syntactic differences between two programs. *Software: Practice and Experience*, Wiley Online Library, v. 21, n. 7, p. 739–755, 1991. Citado na página 37.

ZHAI, Y.; LIU, B. Web data extraction based on partial tree alignment. In: ACM. *Proceedings of the 14th international conference on World Wide Web*. [S.l.], 2005. p. 76–85. Citado 6 vezes nas páginas 9, 18, 19, 29, 40 e 44.

ZHAI, Y.; LIU, B. Structured data extraction from the web based on partial tree alignment. *Knowledge and Data Engineering, IEEE Transactions on*, IEEE, v. 18, n. 12, p. 1614–1628, 2006. Citado 5 vezes nas páginas 19, 27, 44, 69 e 83.