LEANDRO SILVA ALONSO

AGENTE INTELIGENTE DE SHOPPING COMPARISON ATRAVÉS DE DETECÇÃO DE PADRÕES

LAVRAS - MG

2012

LEANDRO SILVA ALONSO

AGENTE INTELIGENTE DE SHOPPING COMPARISON ATRAVÉS DE DETECÇÃO DE PADRÕES

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Orientador

Prof. Dr. Tales Heimfarth

Co-Orientador

Prof. MSc. Tiago Amador Coelho

LAVRAS - MG

2012

LEANDRO SILVA ALONSO

AGENTE INTELIGENTE DE SHOPPING COMPARISON ATRAVÉS DE DETECÇÃO DE PADRÕES

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *24 de Outubro de 2012*

Prof. MSc. André Grützmann — Universidade Federal de Lavras

Profa. Dra. Marluce Rodrigues Pereira — Universidade Federal de Lavras

Prof. Dr. Tales Heimfarth

Orientador

Prof. MSc. Tiago Amador Coelho

Co-Orientador

LAVRAS - MG

2012

Dedicado a todos que fizeram parte dessa caminhada e, especialmente, a minha família e amigos.

AGRADECIMENTOS

Agradeço inicialmente aos meus pais, Faustino Bisso Alonso Neto e Maria Perpétua da Silva Alonso, por todo o apoio, força, incentivo e carinho. E à minha irmã, Letícia Silva Alonso, que mesmo distante sempre se fez presente. Sem vocês eu não teria chegado até aqui.

Um imenso obrigado ao professor Ahmed, com quem trabalhei todos esses anos e aprendi muito. Não poderia deixar de mencionar meu orientador Tiago, que apoiou e contribuiu muito para este trabalho. Vocês foram os melhores que eu poderia encontrar.

À minha turma de Computação (2008/1) fica meu muito obrigado e um até logo por todos os momentos vividos.

Aos meus companheiros de trabalho do NTE e do LICESA minha imensa gratidão por todo aprendizado e grandes momentos vividos. Jamais serão esquecidos.

Por fim, agradeço aos meus amigos que, de uma forma ou de outra, fizeram parte dessa caminhada. Muito obrigado!

RESUMO

Os serviços de comparação de preços, chamados de *Shopping Comparison*, têm impactado o setor de comércio *online* e oferecido informações de valor para que o consumidor efetue decisões de compra.

No entanto, extrair os dados de múltiplas lojas online mostra-se uma tarefa não trivial, tendo em vista que documentos HTML são semi-estruturados e possuem um baixo valor semântico.

Neste trabalho propõe-se um agente de *Shopping Comparison* ou *Shopbot*, que através da detecção de padrões em documentos HTML consegue determinar aonde estão os produtos e seus atributos, gerando um modelo reaproveitável capaz de traduzir os dados semi-estruturados em informações.

Os resultados indicam que o protótipo desenvolvido é completamente viável e mostrou-se funcional na maior parte das lojas analisadas.

Palavras-chave: Shopping Comparison, Shopbot, Web Content Mining.

ABSTRACT

The Shopping Comparison services has impacted the online commerce and offered valuable information for the consumer to make purchasing decisions.

However, extract the data from multiple vendors is a nontrivial task, considering that HTML documents are semi-structured and have a low semantic value.

In this work is proposed a Shopping Comparison agent, or Shopbot, that by detecting patterns in HTML documents can determine where are the products and his attributes, generating a reusable model able to translate the semi-structured data into valuable information.

The results show that the prototype developed is entirely viable and proved to be functional in most vendors analyzed.

Keywords: Shopping Comparison, Shopbot, Web Content Mining.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 Introdução

1.1 Contextualização e Motivação

A Era Digital tem inserido novas e importantes regras que regem a sociedade moderna e o atual panorama socioeconômico. Globalização, competição, informação em grandes quantidades, mudanças constantes em curto espaço de tempo, variedades de opções e vários outros termos e efeitos dessa Era têm surgido ao longo do tempo.

Com a evolução das tecnologias e o advento da Internet, o comércio se modificou através do surgimento do comércio online, também chamado de *e-commerce* ou *comércio virtual*. O consumidor, antes limitado por barreiras geográficas, passou a ter acesso a um maior conjunto de lojas, produtos e serviços através da web.

Segundo dados da empresa e-bit, o comércio eletrônico brasileiro alcançou um faturamento de R$ 18,7 bilhões em 2011. Um aumento de 26% em relação ao ano anterior, quando o faturamento foi de R$ 14,8 bilhões. O número de consumidores que compraram pela *web* ao menos uma vez é de 32 milhões e apenas no ano de 2011 foram efetuados 53,7 milhões de pedidos através de 5000 lojas online (EBIT, 2011).

Através das milhares de lojas o consumidor passou a ter mais opções, informações e ofertas do que nunca. Ele pode optar por fazer suas compras em grandes redes, lojas virtuais especializadas, catálogos virtuais, dentre outros meios. E é bombardeado com mensagens por diversos canais: televisão, Internet, rádio, redes sociais, mídia impressa, etc. (MCKENNA, 1999)

No entanto, devido à imensurável quantidade de anúncios disponíveis, a tarefa de escolher o produto desejado pelo melhor preço torna-se demasiadamente

custosa. Como um exemplo simples, suponha-se um consumidor que deseja adquirir uma *TV LCD*, ele deve selecionar uma quantidade finita de lojas que conhece; buscar o produto desejado em cada loja e, por fim, cruzar os dados obtidos a fim de encontrar o melhor custo-benefício. Embora tal tarefa possa parecer simplória, ela se torna cada vez mais complexa à medida que mais lojas são incluídas na pesquisa.

Serviços de *Shopping Comparison* se propõem a resolver esse problema: através de um serviço online baseado na web os consumidores podem encontrar produtos, preços e outras informações de múltiplas lojas online, ao invés de visitar cada uma separadamente. Os preços e produtos são apresentados na mesma interface e o consumidor pode tomar a melhor decisão, sendo então redirecionado para o fornecedor do produto para completar a transação (WAN, 2009).

Para que tais dados sejam apresentados é necessário que um agente de pesquisa de preços, chamado de *Shopbot*, automaticamente pesquise uma coleção de lojas online para coletar e comparar dados de um produto especificado (ZHANG; JING, 2011). Entretanto, não existe uma padronização para a exibição dos dados dos produtos: cada loja possui seu próprio layout, organização estrutural, tecnologias, etc. Portanto, desenvolver um bom agente de comparação (*shopbot*), capaz de coletar dados de múltiplas fontes se torna uma tarefa não trivial.

A tarefa de descobrimento e análise de informações úteis da *World Wide Web*, através da busca automática de informação de recursos disponíveis online é a definição de *Web Mining* (COOLEY; MOBASHER; SRIVASTAVA, 1997). Sendo este trabalho, portanto, pertencente a essa linha de pesquisa.

1.2 Objetivos

Este trabalho elabora um conjunto de regras e procedimentos para a detecção de padrões; identificação de atributos; extração de dados em lojas de *e-commerce* e geração de um modelo genérico e reaproveitável para cada fonte de dados.

Objetiva-se mensurar a eficiência do modelo proposto em extrair dados de produtos através da implementação de um protótipo de *shopbot* que aplica tais regras e procedimentos em uma aplicação de *Shopping Comparison*.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte maneira:

- Os capítulos 2, 3 e 4 constituem o Referencial Teórico deste trabalho. No capítulo 2 tem-se a apresentação de conceitos de *HTML*. No capítulo 3 é realizada uma apresentação sobre conceitos de *Web Mining*, focando-se em *Web Content Mining*. Por fim, no capítulo 4 é dada uma definição de *Shopping Comparison* e *Shopbot*, além dos seus impactos no comércio online;

- O capítulo 5 descreve a Metodologia utilizada no trabalho, definindo os procedimentos metodológicos e as tecnologias utilizadas;

- No capítulo 6 são apresentados com detalhes as regras e procedimentos utilizados na construção do protótipo de *shopbot* proposto neste trabalho, desde a elaboração do algoritmo até a interface do serviço de *Shopping Comparison*;

- O capítulo 7 contém os resultados obtidos e algumas discussões a respeito do que foi atingido.

2 Linguagem HTML

A *World Wide Web* é uma imensa rede de informações. Segundo o *World Wide Web Consortium*, a principal organização de padrões para a Internet, a *Web* se baseia em três mecanismos que tornam esses recursos disponíveis para o público mais amplo possível (CONSORTIUM, 1999):

1. Um esquema de nomenclatura uniforme para a localização de recursos presentes na *Web*. Ex.: *Universal Resource Identifier*, ou *URI*;

2. Protocolos, para acesso à recursos nomeados na *Web*. Ex.: *Hypertext Transfer Protocol*, ou *HTTP*;

3. *Hypertext*, para a fácil navegação entre esses recursos. Ex.: *HTML*.

HTML, que é usado para navegar entre os recursos da *Web*, é tratado na seção seguinte.

2.1 HTML

HyperText Markup Language (*HTML*) é a *língua franca* para publicação de conteúdo na *World Wide Web*. É um formato não-proprietário que pode ser criado e processado por uma ampla variedade de ferramentas, desde simples editores de texto à ferramentas mais complexas de autoração. *HTML* usa *tags* como * * e * * para estruturar o texto em títulos, parágrafos, listas, blocos, imagens, *links* para outros documentos, etc. (CONSORTIUM, 2010)

Um breve código, ou documento, *HTML* é exibido abaixo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

```
<head>
   <title>Titulo da Pagina HTML</title>
</head>
<body>
<div id="conteudo" class="bloco">
   <h1>Titulo do Texto</h1>
   <p>Paragrafo contendo texto.</p>
   <p>Segundo paragrafo com mais texto.</p>
</div>
<div class="bloco">
   <p><strong>Texto em negrito.</strong></p>
   <p><em>Texto em italico.</em></p>
</div>
</body>
</html>
```

A interpretação desse código pelo browser *Google Chrome* é exibido na FIGURA 1.

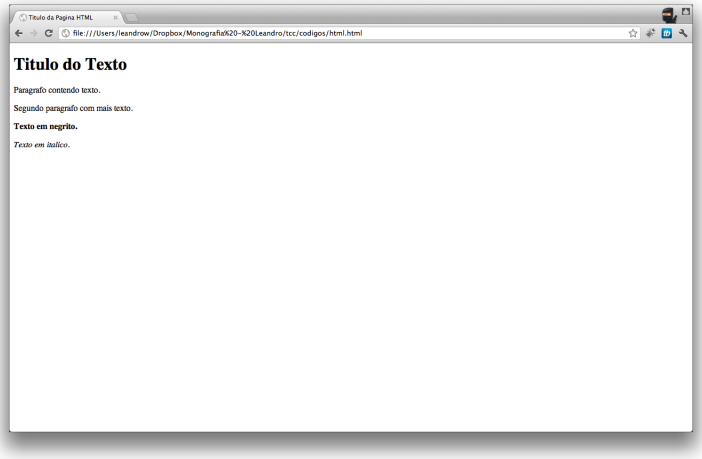

Figura 1: Interpretação de código HTML pelo browser Google Chrome.

Apesar do *HTML* possuir uma grande quantidade de tags para formatação de documentos, existe uma limitação histórica em seu projeto: as tags usadas são semi-estruturadas, elas apenas descrevem como a informação deve ser exibida mas não do que se trata os dados (WAN, 2009). Dessa forma, não é uma linguagem semântica que pode ter sua informação trivialmente extraída e interpretada por agentes computacionais.

2.2 Document Object Model

Document Object Model, abreviadamente chamado de *DOM*, é uma interface de programação de aplicações, ou *API*, para documentos *HTML* válidos e *XML* corretamente formatados. Ela define a estrutura lógica dos documentos e a maneira como serão acessados e manipulados. Através da *DOM*, programadores podem construir documentos, navegar pela sua estrutura, adicionar, modificar ou remover elementos e conteúdo (ROBIE *et al.*, 2000).

Como é uma especificação do *World Wide Web Consortium*, ou *W3C*, o objetivo da *Document Object Model* é fornecer uma *API* padrão que possa ser usada por uma ampla variedade de linguagens de programação.

No *DOM*, os documentos tem uma estrutura lógica que é muito parecida com uma árvore; mais precisamente uma "floresta", que é um conjunto de árvores. Cada documento contém zero ou um nó de declaração de tipo de documento (*DOCTYPE*), um nó raiz e zero ou mais instruções de processamento e comentários. O elemento raiz funciona como o nó raiz da árvore do documento (ROBIE *et al.*, 2000).

Como exemplo, considere o código *HTML* apresentado na seção anterior. A árvore *DOM* deste documento é apresentada na FIGURA 2. Através da manipulação desta árvore é possível adicionar novos elementos através da simples adição

de novos nós à estrutura, remover elementos através da remoção de nós e editar conteúdo textuais ou estruturais, seja alterando textos ou modificando nós.

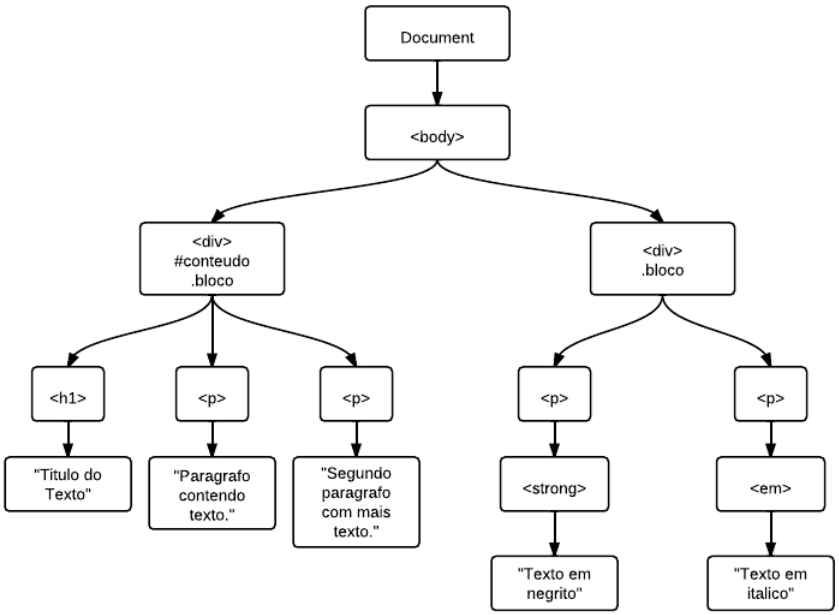

Figura 2: Reprodução de uma árvore DOM para um documento HTML.

É facilmente perceptível que a *API DOM* é de suma importância na análise de conteúdos provenientes da *World Wide Web*.

2.3 Seletores CSS

Cascading Style Sheets (*CSS*) é uma linguagem de estilo utilizada para descrever a apresentação de páginas *Web*, incluindo cores, layouts, fonte, etc (CONSORTIUM, 2011). Para que os elementos possam ser estilizados existem regras de correspondência para determinar quais regras se aplicam a elementos da árvore *DOM* (BOS *et al.*, 2011).

Um simples exemplo de um documento *CSS* é apresentado a seguir, contendo alguns seletores:

```
* {
   font-size: 12px;
}
h1 {
   color: red;
}
p {
   color: black;
}
```

O *CSS* acima especifica, através do uso do seletor "*", que qualquer elemento do documento no qual é aplicado deve ter o texto no tamanho de *12 pixels*; todos os elementos do tipo "h1" terão o texto da cor vermelha e os elementos do tipo "p" terão a cor preta, especificados através do uso de seletores homônimos a esses elementos. Na Quadro 1 é apresentada a sintaxe de alguns seletores *CSS* da especificação 2.1.

É importante ressaltar que seletores *CSS* não são usados unicamente para definir estilos para páginas *Web*. Eles também são usados na navegação pela árvore *DOM* para que atributos e conteúdos dos nós possam ser recuperados, modificados ou excluídos de forma mais dinâmica. Por exemplo, na árvore *DOM* exibida na FIGURA 2 o seletor "#conteudo" retornaria o nó " " mais à esquerda.

2.4 HTML Parsers

Um *HTML Parser* é uma ferramenta que cria uma árvore *DOM*, a partir da análise de um código HTML (NYEIN, 2011). O analisador é capaz de detectar e corrigir vários erros de sintaxe *HTML* em documentos mal formatados (YANG; ZHANG; TSAI, 2008).

Padrão	Significado
*	Corresponde a qualquer elemento.
E	Corresponde a qualquer elemento E. (isto é, um elemento do tipo E)
E F	Corresponde a qualquer elemento F que é descendente de um elemento E.
E > F	Corresponde a qualquer elemento F que é filho do elemento E.
E:first-child	Corresponde à primeira ocorrência do elemento E.
E.warning	Corresponde a qualquer elemento do tipo E que possui a classe “warning”.
E#myid	Corresponde a qualquer elemento E que possui o ID “myid”.

Quadro 1: Sintaxe de alguns seletores CSS 2.1. Fonte: Bos *et al.* (2011)

Normalmente é disponibilizada uma interface para que a árvore *DOM* possa ser navegada. Alguns *parsers* permitem que essa função seja realizada através de seletores *CSS*, além de prover métodos que permitem manipular e recuperar informações de nós.

Neste trabalho é utilizado o analisador de *HTML* **phpQuery**. É uma ferramenta *server-side*, orientado a seletores *CSS3* e com métodos para recuperação e manipulação do *Document Object Model* (CUDNIK, 2009).

3 Web Mining

Textos, vídeos, imagens e sons são publicados e compartilhados a todo momento e em todas as partes do mundo através da *World Wide Web*. O acompanhamento, absorção e compreensão dessa imensurável quantidade de informação é uma tarefa impraticável, ainda que em um nível micro. Por isso o conceito de *Web Mining* foi criado por pesquisadores.

Web Mining, derivado da expressão *Data Mining*, pode ser amplamente definido como a descoberta e análise de informação útil da *World Wide Web* (COOLEY; MOBASHER; SRIVASTAVA, 1997).

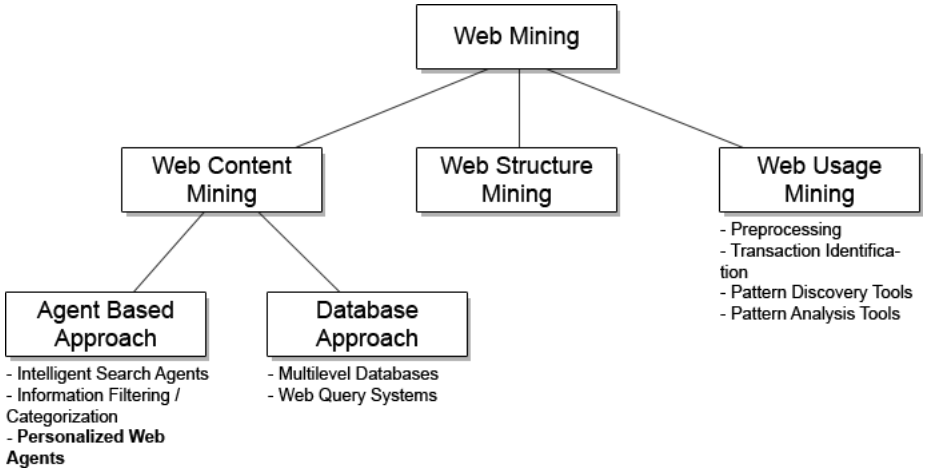

Figura 3: Taxonomia de Web Mining. Fonte: Cooley, Mobasher e Srivastava (1997), Júnior (2007)

Uma taxonomia da *Web Mining* é apresentado na FIGURA 3. Nos parágrafos seguintes tem-se uma breve explicação sobre cada subtópico nela apresentada, sem se estender até os nós do terceiro nível.

Web Structure Mining (*Mineração da Estrutura da Web*) se refere ao processo de descoberta de conhecimento através da relação entre os documentos contidos na Web.

Web Usage Mining (*Mineração do Uso da Web*) analisa o padrão de comportamento de usuários na Web. Como exemplo simples, pode ser citado a descoberta de padrões no acesso de usuários a um determinado site em um *Web server*.

Por fim, *Web Content Mining* é tratado no subcapítulo seguinte.

3.1 Web Content Mining

Web Content Mining (*Mineração do Conteúdo da Web*) é o processo de extração de informação do conteúdo de documentos e seus metadados (palavras-chave, informações sobre produtos, imagens, etc). Apesar deste enfoque abranger principalmente documentos HTML, ele se estende a formatos não-textuais como imagens e vídeos (SANTOS, 2009). Segundo Cooley, Mobasher e Srivastava (1997) *Web Content Mining* pode ser dividido em três abordagens, explicadas a seguir.

A primeira delas, *Agent Based Approach*, em português *Abordagem Baseada em Agentes*, pode ser dividida nas seguintes categorias:

- *Intelligent Search Agents* (*Agentes Inteligentes de Busca*): sistemas que buscam por dados relevantes, organizando e interpretando a informação descoberta. FAQ-Finder (HAMMOND *et al.*, 1995) e BargainFinder (WAN; PENG, 2010) são dois exemplos clássicos dessa categoria.

- *Information Filtering/Categorization* (*Filtragem/Categorização de Informação*): agentes que usam uma variedade de técnicas de recuperação de informação e características de documentos da *Web* para automaticamente obter, filtrar e organizar a informação. BO (Bookmark Organizer) (MAAREK; SHAUL, 1996) é um exemplo desse tipo de aplicação, tendo em vista que

ele organiza coleções de documentos da *Web* baseados na sua informação conceitual.

- *Personalized Web Agents* (*Agentes Web Personalizados*): são agentes que aprendem as preferências do usuário e descobrem informações da *Web* baseada nessas preferências (COOLEY; MOBASHER; SRIVASTAVA, 1997).

A segunda abordagem, *Database Approach* (*Abordagem de Database*) foca em técnicas para organização da informação semi-estruturada da *Web* em dados mais estruturados, usando mecanismos padrões de consulta à databases e técnicas de *data mining* para análise. Pode ser dividido em duas categorias:

- *Multilevel Databases* (*Databases Multiníveis*): a principal ideia dessa abordagem é que o nível mais baixo da database contenha informação semi-estruturada armazenada em vários repositórios *Web*. Nos níveis mais altos, meta dados ou generalizações são extraídos dos níveis mais baixos e organizados em coleções estruturadas, como databases relacionais orientadas a objetos (COOLEY; MOBASHER; SRIVASTAVA, 1997).

- *Web Query Systems* (*Sistemas de Consultas Web*): sistemas que utilizam padrões de consultas a databases, ou a informações estruturais sobre documentos *Web*, e até processadores de linguagens naturais para processar as consultas que são realizadas nas buscas na Internet (COOLEY; MOBASHER; SRIVASTAVA, 1997).

Este trabalho está situado na área de *Web Content Mining*, especificamente na *Abordagem Baseada em Agentes* na categoria de *Agentes Inteligentes de Busca*, visto que o conteúdo presente no HTML de páginas de *e-commerce* serão analisados por um robô de busca inteligente que coletará e organizará a informação relevante.

3.2 Wrapper

A *World Wide Web* não é estruturada de forma a apresentar informações que possam ser lidas por máquinas. A maioria da imensurável quantidade de dados presente na Web está apresentada em estruturas fracamente semânticas que não podem ser compreendidas por agentes (DOORENBOS; ETZIONI; WELD, 1997). Faz-se necessário então uma regra de tradução para cada fonte de dados. A esse "tradutor" é dado o nome de *wrapper* (KUSHMERICK, 1997; COWIE; LEHNERT, 1996).

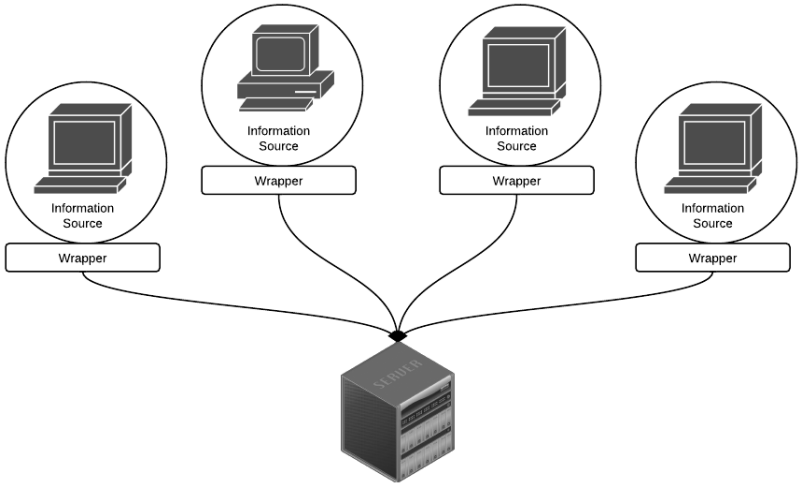

Figura 4: Um sistema de integração de informações usando wrappers. Fonte: Yang *et al.* (2000).

Formalmente, um *wrapper* é um programa, agente ou conjunto de regras que entende a informação de uma fonte e a traduz em uma forma regular e compreensível que pode ser usada por outros agentes. Um *wrapper* é especializado apenas em uma única fonte de informações (YANG *et al.*, 2000). A FIGURA 4 mostra um sistema de integração de informações usando vários wrappers, cada um para uma fonte específica de informação.

Segundo Firat (2003), *wrappers* podem ser divididos em três tipos baseado no seu nível de automação:

- **Manual**: codificar manualmente um *wrapper* normalmente é uma tarefa tediosa e, em alguns casos, impraticável. O número de fontes de dados pode ser muito grande ou crescer rapidamente e a estrutura e o conteúdo da fonte pode frequentemente mudar. Todos esses fatores tornam essa tarefa manual altamente custosa e demorada (TATBUL *et al.*, 2001).

- **Semi-automático (interativo)**: em alguns casos as regras de um *wrapper* que lida com os detalhes específicos de uma fonte de dados é relativamente pequena. A outra parte é igual em todos os *wrappers* ou pode ser gerada semi-automaticamente baseada em uma descrição declarativa dada pelo usuário (TATBUL *et al.*, 2001).

- **Automático**: a geração automática de um *wrapper* significa que não há intervenção humana nesse processo. Ferramentas de geração automática podem ser específicas pra um site ou genéricas; normalmente necessitam de um estágio de treinamento e são baseados em *algoritmos de aprendizado supervisionado* (TATBUL *et al.*, 2001).

Neste trabalho, o *wrapper* gerado será um conjunto de *seletores CSS* que especificam quais dados de páginas *Web* devem ser extraídos, gerando informações relevantes a partir de documentos semi-estruturados.

4 Shopping Comparison

Shopping Comparison, ou em português *Comparador de Preços*, foi introduzido na *World Wide Web* em 1995 como a terceira forma de B2C (*Business-to-consumer*), após a venda e o leilão online. Eles evoluíram de simples comparadores de preço ao oferecimento e combinação de *reviews* de produtos e marcas, avaliações de funções, dentre outros (WAN, 2009).

4.1 Serviços de Shopping Comparison

Em 30 de Junho de 1995 um grupo de pesquisadores do *Andersen Consulting and Smart Store Center* disponibilizaram um serviço inteligente de comparação de preços, conhecido como *BargainFinder*. O projeto foi desenvolvido apenas como demonstração e compreendia apenas a busca de *CDs* de música (WAN; PENG, 2010).

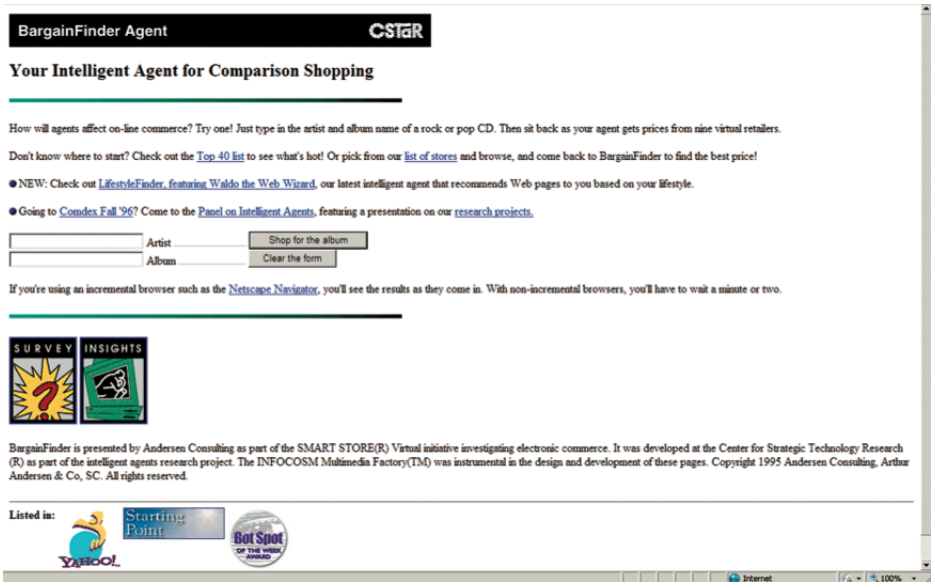

Figura 5: Interface de busca do BargainFinder em 1995. Fonte: Wan e Peng (2010)

A interface inicial do serviço continha dois campos de texto que o usuário podia preencher com o nome de um artista e álbum, como a FIGURA 5 mostra. Ao clicar em *"Shop for the album"* o serviço pesquisava oito lojas online de vendas de *CD* de músicas, extraía os preços e exibia o resultado em uma única página.

Apesar de sua interface parecer bastante primitiva comparada com os serviços existentes atualmente, o *BargainFinder* foi o primeiro serviço de *Shopping Comparison* a receber atenção pública e exposição na mídia, ainda que não fosse o primeiro (WAN, 2009).

O sucesso do *BargainFinder* foi imediato: em apenas um mês o serviço já era usado mais de 2,000 vezes diariamente. Entretanto, as oito lojas que compunham os resultados do comparador não foram notificadas de que suas páginas estavam sendo pesquisadas, seus preços coletados e comparados com os praticados pelas suas concorrentes. As reações foram adversas: após 8 meses de funcionamento três lojas bloqueram o acesso do BargainFinder a seus servidores, uma cooperou ativamente, três permaneceram neutras e uma cessou as operações. Apesar disso, algumas lojas visionárias enxergaram o potencial do *BargainFinder* e seis delas pediram para que fossem incluídas no catálogo. Esse comportamento indicava um panorama altamente favorável aos serviços de *Shopping Comparison*, que hoje em dia são tão populares a ponto de lojas online pagarem para ser listadas nos serviços mais populares (WAN; PENG, 2010).

4.2 Shopbot

Segundo Zhang e Jing (2011), *shopbots*, também chamados de *agentes de comparação de preços*, são sistemas que automaticamente buscam uma grande variedade de lojas online coletando e agrupando informações de produtos ou serviços especificados.

Uma definição mais ampla, que é adotada neste trabalho, é a de que *shopbots* são ferramentas automáticas que buscam diversos sites de e-commerce, como lojas online, recuperando informações sobre os produtos que são analisadas e extraídas para ajudar os consumidores a fazerem decisões de compra. Estes agentes empregam um processo automático de construção de *wrappers* para extração e análise de páginas *Web* semi-estruturadas (HUANG; TSAI, 2011).

A recuperação de dados por esses agentes se divide em dois métodos distintos: *data wrapping* e *data feeding*. No primeiro, o agente acessa os sites de tempos em tempos e atualiza o banco de dados, enquanto no segundo as lojas online disponibilizam os dados em um formato previamente definido pelos serviços de *Shopping Comparison*. Ainda que o segundo método seja mais flexível e estável, os serviços de e-commerce hesitam em disponibilizar os dados dessa forma para não competir com seus pares meramente no preço (WAN, 2009).

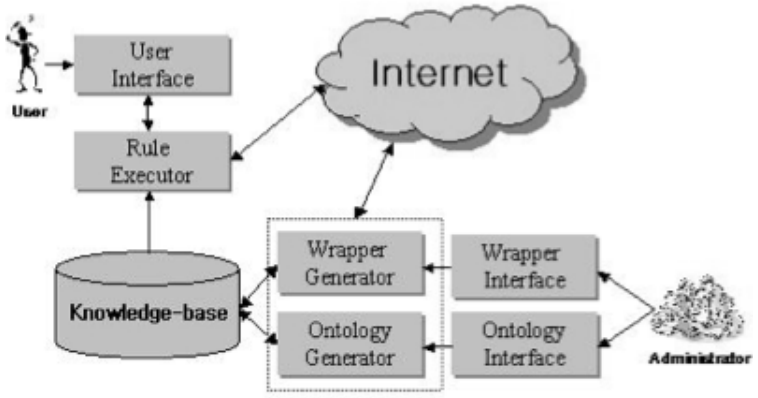

Figura 6: Arquitetura de um agente de comparação de preços. Fonte: Yang *et al.* (2000)

Como exemplo, é apresentada na FIGURA 6 uma arquitetura de um *agente de comparação de preços* proposto por Yang *et al.* (2000) que se utiliza do método de *data wrapping* para recuperação de dados. Os módulos pertinentes à este traba-

lho e que compõem o ponto de vista administrativo, ou seja, aqueles que podem ser acessados apenas pelo *system designer* (administrador) do serviço, são compostos por:

- *Wrapper Generator*: módulo responsável pela geração automática de *wrappers* para cada fonte de dados.
- *Wrapper Interface*: é o módulo no qual o *system designer* define as fontes de dados para que seja realizada a geração de *wrappers*.

Os módulos que compreendem o ponto de vista do usuário são compostos por:

- *User Interface*: é a interface do serviço de comparação de preços. Através dela o usuário digita o *termo de busca*.
- *Rule Executor*: quando um usuário realiza uma busca esse módulo é ativado para que cada *wrapper* previamente gerado seja utilizado nas fontes de dados. As informações recuperadas são então combinadas e exibidas para o usuário.

A título de exemplo, considere um usuário que deseja buscar por *smartphone* em um serviço com a arquitetura acima especificada. Para que a busca ofereça resultados não-nulos, válidos e coerentes é previamente necessário que o *system designer* tenha adicionado lojas de *e-commerce* através do módulo *Wrapper Interface*, e que os *wrappers* tenham sido corretamente gerados pelo *Wrapper Generator*. O usuário então acessa o serviço através da *User Interface* e digita em um *input de texto* o termo de busca *smartphone*. O módulo *Rule Executor* é ativado para cada loja online adicionada pelo *system designer* e executa as regras de extração contidas no *wrapper* correspondente. Os dados obtidos são então validados,

agrupados e exibidos na *User Interface* para que o usuário possa tomar sua *decisão de compra*.

Existem várias outras arquiteturas de *shopbots*, algumas mais simples, outras mais complexas e eficientes. Essas propostas podem ser encontradas em Huang e Tsai (2011), Wan (2009), Doorenbos, Etzioni e Weld (1997) e Prasad, Kumari e Raju (2009). Neste trabalho será adotado como base a arquitetura proposta por Yang *et al.* (2000) e acima explicada.

4.3 Impactos no Comércio Eletrônico

O mercado brasileiro de Comércio Eletrônico encerrou o ano de 2011 com R$ 18,7 bilhões em faturamento, como mostrado na FIGURA 7. Foram 31,9 milhões de consumidores que efetuaram 53,7 milhões de pedidos via web. (EBIT, 2011) São números animadores, mas qual a influência dos serviços de *Shopping Comparison* no comércio eletrônico? Consumidores realmente utilizam serviços de comparação de preços? A comparação de preços é benéfica para as lojas de e-commerce?

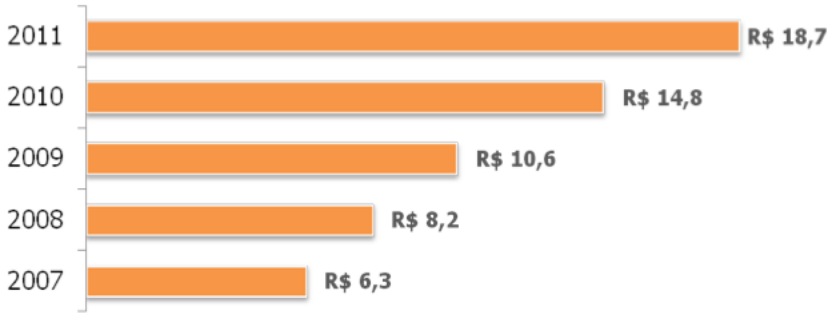

Figura 7: Evolução do faturamento do setor de comércio eletrônico (em bilhões). Fonte: ebit (2011)

Várias pesquisas têm sido realizadas na área para tentar elucidar tais questões. Montgomery *et al.* (2004) cita que a adoção de serviços de comparação de preços sofreu um aumento de 0,1% em Junho de 1997 para 5,7% em Maio de 2002. O tráfego do *shopping.com*, o principal líder em serviços de comparação de preços, está logo atrás do *eBay* e da *Amazon*, dois gigantes do comércio online, desde 2003 (WAN, 2009). Esse substancial crescimento parece trazer um panorama aparentemente desfarovável às lojas de *e-commerce*: a presença de serviços de *Shopping Comparison* colocam uma pressão sobre as margens de preço das lojas online.

Ainda que vários estudos analíticos tenham assumido que os consumidores buscam por um produto perfeitamente homogêneo, levando em conta apenas a comparação de preços e comprando da loja que oferece o menor custo (CHEN; SUDHIR, 2001; KEPHART; GREENWALD, 1999) pesquisas recentes demonstram que essa suposição não é sustentável. Ao usar serviços de comparação de preços os consumidores não consideram apenas o preço, mas também fatores como a forma de envio, políticas de ressarcimento, proteção à privacidade e a reputação da marca (MONTGOMERY *et al.*, 2004; SMITH, 2002; SMITH; BRYNJOLFSSON, 2001). Os serviços de *Shopping Comparison* realmente exercem uma significante pressão nas margens de preço dos vendedores onlines, entretanto eles possuem várias opções estratégicas para mitigar essa pressão, incluindo precificação estratégica, ofuscação de buscas, dentre outros (SMITH, 2002).

Estudos demonstrados por Zhang e Jing (2011) demonstram que os serviços de *Shopping Comparison* se tornam mais atraentes quando a dispersão do preço dos produtos pesquisados é alta. Por exemplo, computadores tem uma alta variação de preços quando comparados a livros, isso encoraja o uso da comparação de preços já que o consumidor obtém os valores em segundos.

Fica evidente que os serviços de comparação de preços têm crescido fortemente nos últimos anos. No entanto, ainda que exista um impacto sobre as lojas online e seus preços, tais impactos não podem ser generalizados e devem ser estudados especificamente para cada tipo de produto ou serviço.

5 Metodologia

Neste capítulo são apresentados as ferramentas que foram empregadas no desenvolvimento deste trabalho e as bases de dados usadas nos experimentos.

5.1 Métodos

Os métodos empregados neste trabalho consistem nas etapas necessárias para a geração automática de um *wrapper* de uma loja de *e-commerce*, são elas: Submissão da URL de busca; Criação do *Document Object Model*; Detecção de Padrões e Geração do Wrapper.

Para os testes e elaboração do *shopbot* foi determinado um conjunto inicial de *fonte de dados* composto por 13 lojas de *e-commerce*, exibidas no Quadro 2.

Loja	URL
Dafiti	http://www.dafiti.com.br/
Shoptime	http://www.shoptime.com.br/
Submarino	http://www.submarino.com.br/
Nethoes	http://www.netshoes.com.br/
WalMart	http://www.walmart.com.br/
CompraFácil	http://www.comprafacil.com.br/
RicardoEletro	http://www.ricardoeletro.com.br/
Americanas	http://www.americanas.com.br/
Saraiva	http://www.livrariasaraiva.com.br/
Fnac	http://www.fnac.com.br/
Extra	http://www.extra.com.br/
PontoFrio	http://www.pontofrio.com.br/
MagazineLuiza	http://www.magazineluiza.com.br/

Quadro 2: Lojas de *e-commerce* utilizadas para elaboração e testes do protótipo.

No entanto, para realizar mensurações mais precisas e garantir que a aplicação seja genérica e não direcionada para o conjunto inicial, estabeleceu-se outro

conjunto, listado no Quadro 3, composto por mais 10 lojas de *e-commerce* para testes após a conclusão do protótipo de *shopbot*.

Faz-se necessário ressaltar que as lojas que compõem o último conjunto foram escolhidas aleatoriamente em um serviço de *Shopping Comparison* já estabelecido.

Loja	URL
KaBuM!	http://www.kabum.com.br/
Colombo	http://www.colombo.com.br/
Insinuante.com	http://www.insinuante.com.br/
Taqi	http://www.taqi.com.br/
Loja Easy.com.br	http://www.easy.com.br/
Girafa.com.br	http://www.girafa.com.br/
Shopfato.com	http://www.shopfato.com.br/
Centauro.com.br	http://www.centauro.com.br/
Classic Tennis	http://www.classictennis.com.br/
Leader	http://www.leader.com.br/

Quadro 3: Lojas de *e-commerce* utilizadas para testes e validação do protótipo.

5.1.1 Entendimento do Problema

Este trabalho realiza uma pesquisa de natureza tecnológica visando o desenvolvimento de um protótipo de *shopbot* capaz de gerar *wrappers* automáticos baseados em uma série de padrões previamente detectados em lojas de *e-commerce* brasileiras.

Como mencionado no Capítulo 1, cada loja possui seu próprio layout, organização estrutural, tecnologias, etc. O HTML é uma linguagem de marcação semi-estruturada e não semântica, como explicado no Capítulo 2, faz-se necessário então um *wrapper* que possa traduzir um documento *HTML* em informações úteis sobre os produtos.

O algoritmo deverá receber como entrada uma *string* não-nula, válida e formatada; realizar o processamento necessário para identificar anúncios e seus atributos e, em caso da não-ocorrência de erros, retornar um *wrapper* válido que possa ser utilizado por outras aplicações.

5.2 Material

Abaixo encontra-se a descrição das ferramentas utilizadas no desenvolvimento deste trabalho, desde a concepção do *shopbot* até a interface para comparação de preços.

5.2.1 PHP

O *PHP* (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) (PHP.NET, 2012) é uma linguagem de script, *server-side* e *open-source*, elaborada para a *World Wide Web* pra que páginas *Web* dinâmicas possam ser produzidas. O código é interpretado no servidor *Web* através de um módulo processador de *PHP* e o resultado é enviado para o cliente.

Essa linguagem foi escolhida em detrimento às demais por ser uma tecnologia *Open Source*, além de ser facilmente instalado e configurado.

5.2.2 phpQuery

phpQuery é um analisador *HTML* (*Parser HTML*) *server-side*, orientado à seletores *CSS3* e com métodos para recuperação e manipulação do *Document Object Model* (CUDNIK, 2009). É uma biblioteca para que documentos *HTML* possam ser manipulados através de sua árvore *DOM* em scripts desenvolvidos através da linguagem *PHP*.

Essa ferramenta foi escolhida após uma análise entre os *parsers HTML* em *PHP* mais utilizados, sendo o *phpQuery* a ferramenta com o maior número de recursos e a melhor *API*.

5.2.3 Ambiente Computacional

O ambiente computacional utilizado para realização dos experimentos foi um *MacBook Pro* com 4GB de RAM e processador *Intel Core i5* de 2.3 GHz.

6 Arquitetura do Shopbot proposto

Este capítulo descreve com detalhes a construção do *shopbot* e do algoritmo responsável por identificar os atributos de produtos gerando um *wrapper* reaproveitável.

As etapas propostas para o *shopbot* deste trabalho foram adaptadas de Yang *et al.* (2000): 1) determinar quais os *termos de busca* para realizar uma requisição teste a fim de encontrar informações relevantes de produtos; 2) Extrair apenas informações dos produtos da página de resultados, ignorando fragmentos desnecessários ou redundantes; 3) Ser capaz de reconhecer os atributos de um produto, tais como preço, título, imagem, etc; 4) Gerar um *wrapper* que possa ser utilizado na fonte de dados fornecida, ou seja, na loja submetida.

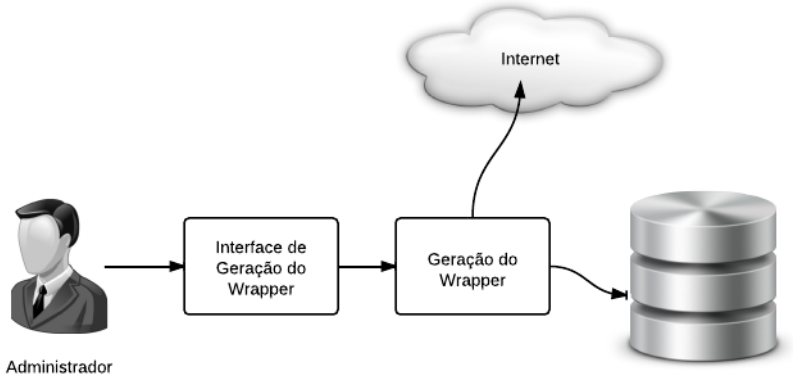

Figura 8: Arquitetura do shopbot proposto.

A FIGURA 8 mostra a arquitetura proposta para cumprir esses requerimentos:

- **Interface de Geração do *Wrapper***: é a interface pelo qual o administrador adiciona as fontes de dados;

- **Geração do *Wrapper***: através da submissão de uma fonte de dados pelo administrador, os algoritmos para detecção de padrões e geração do *wrapper* são ativados. Ao fim desse processo, caso não haja erros e o reconhecimento tenha sido realizado, o *wrapper* gerado é então salvo na database.

6.1 Determinação dos Termos de Busca adequados

6.1.1 Detecção de Padrões

Apesar de diversas abordagens exigirem uma etapa de treinamento (YANG *et al.*, 2000; DOORENBOS; ETZIONI; WELD, 1997) o *shopbot* proposto neste trabalho se baseia em padrões estruturais e sintáticos que são amplamente utilizados na grande maioria das lojas de *e-commerce*, dispensando a fase de aprendizado. Sendo, portanto, classificado como de *Aprendizado Não Supervisionado*.

Algoritmos Não Supervisionados trabalham com um conjunto de n exemplos $\{x_i\}_{i=1}^{n}$ e realizam tarefas de *clustering* ("agrupamento"), redução de dimensionalidade, etc (ZHU; GOLDBERG, 2009).

Os padrões utilizados em cada fase do algoritmo serão abordados no decorrer da explicação de cada uma.

6.1.2 Submissão da URL e Termos de Busca

Para que o processo se inicie, e um *wrapper* possa ser automaticamente gerado, é necessário que o *administrador* submeta uma *URL* de busca não-nula, válida e previamente formatada de uma fonte de dados. Nesta *URL*, o *parâmetro* que irá conter o termo de busca deve ser explicitado através do uso da *string* "<term>".

Na Quadro 4 é exemplificado a formataçao das *URLs* de busca do conjunto inicial de testes composto por 13 lojas de *e-commerce*. Pode-se notar que o

Loja	URL de Busca
Dafiti	http://www.dafiti.com.br/catalog/?q=<term>
Shoptime	http://www.shoptime.com.br/busca/<term>
Submarino	http://www.submarino.com.br/busca?q=<term>
Nethoes	http://www.netshoes.com.br/busca/?q=<term>
WalMart	http://www.walmart.com.br/busca?ft=<term>
CompraFácil	http://www.comprafacil.com.br/comprafacil/Busca .jsf?Q=<term>&token=jUUM8Byxg58%3D
RicardoEletro	http://www.ricardoeletro.com.br/Busca/Resultado/ ?loja=&q=<term>
Americanas	http://www.americanas.com.br/busca/<term>
Saraiva	http://www.livrariasaraiva.com.br/pesquisaweb/ pesquisaweb.dll/pesquisa?ORDEMN2=E& ESTRUTN1=&PALAVRASN1=<term>
Fnac	http://www.fnac.com.br/pesquisa/- 1/fnac.html?q=<term>
Extra	http://buscando.extra.com.br/search?w=<term>
PontoFrio	http://search.pontofrio.com.br/search?w=<term>
MagazineLuiza	http://www.magazineluiza.com.br/search/index.asp ?q=<term>

Quadro 4: Descrição das lojas e suas URLs de busca utilizadas como entrada para o shopbot.

parâmetro que recebe o *termo de busca* foi formatado com a string "<term>" que será automaticamente substituído pelos termos de busca adequados.

Por termos de busca adequados sub-entende-se: termos que retornam ao menos cinco resultados válidos nas lojas em que são aplicados. Exemplos de termos não adequados são apresentados no Quadro 5: nota-se que são termos de busca "restritivos", ou seja, retornam poucos resultados por estarem especificamente associados à apenas uma marca ou produto.

Ainda que um termo possa ser considerado adequado, ele pode se tornar inadequado se aplicado em um domínio não compatível. Como exemplo, consi-

Termos Não Adequados
iphone 4s
nike shox delivery
nokia lumia 800

Quadro 5: Exemplos de termos de busca não adequados.

dere o termo de busca "tv lcd". É um termo genérico que, normalmente, obtem muitos resultados, sendo classificado como adequado. No entanto, ele não obterá resultados satisfatórios caso seja aplicado a uma loja de artigos esportivos. Da mesma forma, um termo "tênis de corrida" não irá obter resultados em uma loja de eletrônicos.

Portanto, a aplicação de um termo está intrinsecamente ligado aos produtos que são oferecidos pela loja de *e-commerce*. Termos de busca adequados, e utilizados neste trabalho, são exemplificados no Quadro 6.

Termos Adequados
tv lcd
tênis nike
notebook
celular

Quadro 6: Exemplos de termos de busca adequados.

Como o algoritmo desconhece qual termo é o mais adequado, ele testa um a um até que o *wrapper* seja gerado ou os termos acabem.

Um pseudocódigo, que contempla o teste exaustivo até que o *wrapper* seja gerado ou os termos acabem, é descrito a seguir:

```
var url_submetida;
var termos = ['tv lcd', 'tenis nike', 'notebook', 'celular'];
var pos = 0;

while (!wrapperValido()) do {
```

```
    gerarWrapper(substitua('<term>', termos[pos], url_submetida));
    pos++;
}
```

6.1.3 Análise do HTML e Conversão em Document Object Model

Uma vez que a *URL* tenha sido formatada com algum termo de busca, é necessário que o algoritmo construa o *Document Object Model*, ou árvore *DOM*, a partir do conteúdo *HTML*.

A *URL* é acessada através da biblioteca *cURL* do *PHP* e seu conteúdo é analisado e convertido em uma árvore *DOM* pela ferramenta *phpQuery* (CUDNIK, 2009).

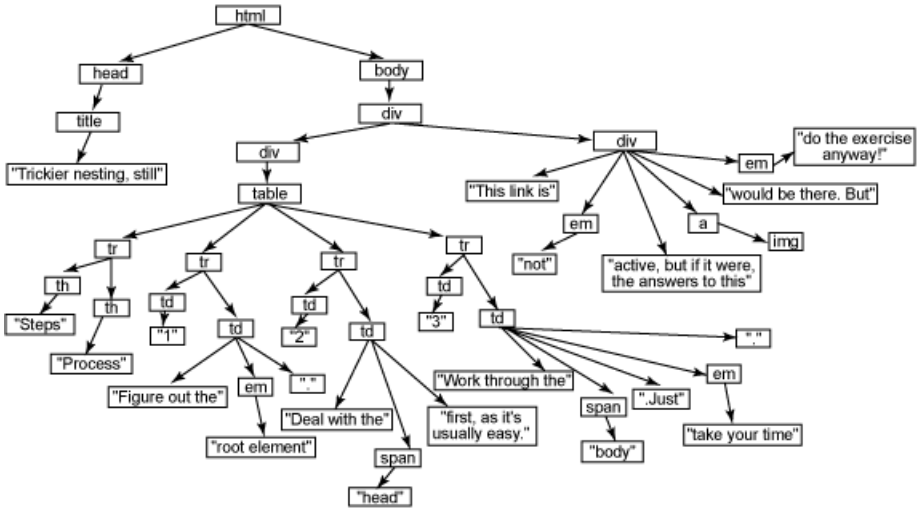

Figura 9: Exemplo de uma árvore *DOM* com diferentes elementos. Fonte: http://vinaytech.wordpress.com/2008/11/24/ajax-and-dom/

A FIGURA 9 mostra um exemplo de uma estrutura de árvore *DOM*. Não foi reproduzido neste trabalho uma árvore completa porque páginas *web* mais com-

plexas, como resultados de busca de lojas online, normalmente possuem uma árvore com muitos mais nós e ramificações, inviável de ser reproduzida.

É através dessa árvore que o algoritmo irá detectar onde estão os produtos e gerar o *wrapper*.

6.2 Eliminação de Ruídos

Em geral, as lojas de *e-commerce* apresentam os resultados em forma de lista, como exibido na FIGURA 10, ou em grade, como exibido na FIGURA 11.

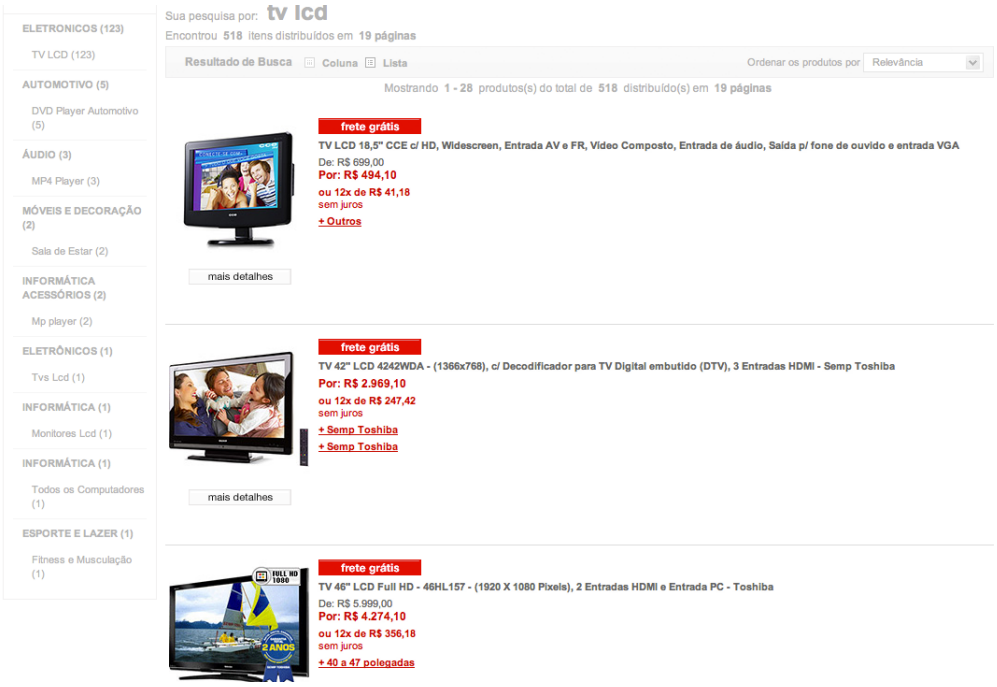

Figura 10: Exemplo de uma página com apresentação de resultados em forma de lista.

Como pode-se notar, a listagem dos resultados segue um padrão estrutural: os títulos, preços, parcelas e links são exibidos na mesma posição para cada resultado encontrado. Isso quer dizer que esses elementos seguem, com leves al-

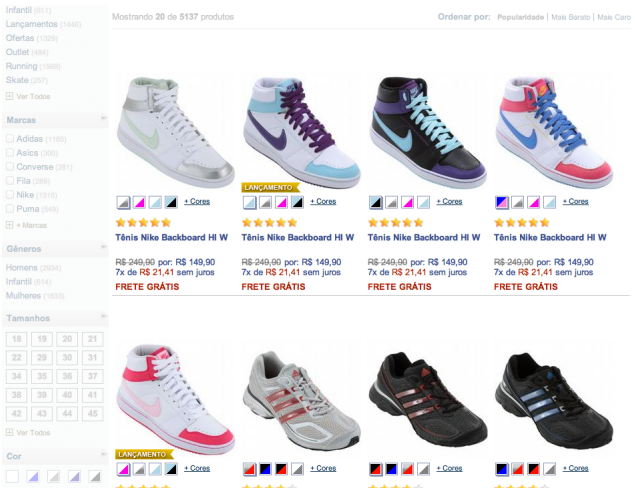

Figura 11: Exemplo de uma página com apresentação de resultados em forma de grade.

terações, um mesmo padrão na árvore *DOM*. Em resumo: o objetivo inicial do algoritmo é identificar os padrões estruturais na árvore, isolando-os dos fragmentos desnecessários. Essa identificação considera apenas o tipo de cada nó, desconsiderando o conteúdo textual.

Para que os padrões possam ser identificados é necessário estabelecer quais os elementos fazem parte da sub-árvore de cada nó. Isso é realizado através de um algoritmo pós-ordem como exemplificado a seguir:

```
function posordem(arv) {
   for (cada elemento filho da arvore arv) do {
      nodes = posordem(arv->childrens);
      elementos[] = elemento->nodeName;
      setPadrao(elemento, elementos + nodes);
   }
   return elementos + nodes;
}
```

O algoritmo gera uma tabela contendo o identificador de cada elemento, quais seus nós-filhos e um ponteiro para cada nó que possui aquela estrutura. A FIGURA 12 reproduz um trecho de uma árvore *DOM*, o resultado da aplicação do algoritmo é demonstrado no Quadro 7.

Elemento	Nós-filhos	Número de Ponteiros
div	img, span	10
li	a	9
ul	li, a	9
li	a, a, div, img, span, ul, li, a	9
li	a, a, div, img, span, ul	1
ul	li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul	1

Quadro 7: Resultado da aplicação do algoritmo pós-ordem na árvore exemplificada na FIGURA 12

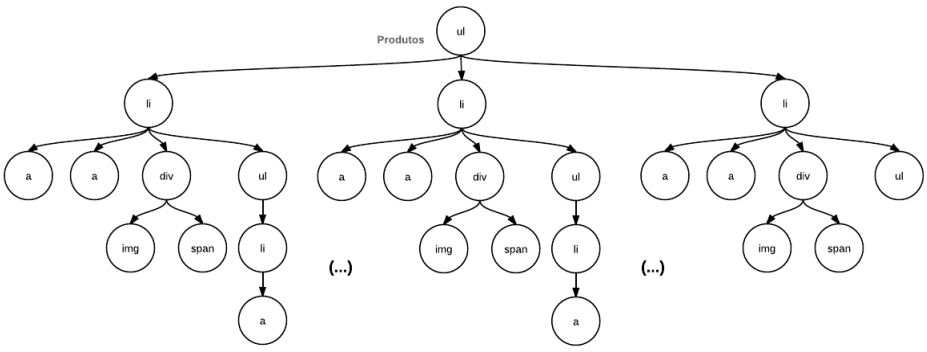

Figura 12: Uma árvore DOM simplificada para demonstração da aplicação do algoritmo pós-ordem.

É importante ressaltar que a Quadro 7 apresenta um resultado da aplicação do algoritmo em uma árvore muito reduzida para facilitação da compreensão. A demonstração da aplicação do algoritmo em uma árvore *DOM* real produziria um resultado extremamente grande, iniviável de ser reproduzido. Ainda assim, acredita-se que o exemplo fornecido pela FIGURA 12 é capaz de fornecer informações suficientes para a compreensão da aplicação.

O segundo passo, após a geração da tabela, é a remoção de ruídos para que apenas os padrões relevantes continuem listados. Os ruídos identificados são listados abaixo:

1. Padrões que se repetem mais de 40 vezes são removidos. No conjunto inicial de lojas foi identificado que nenhuma exibe uma quantidade de resultados superior a 30 produtos, o que significa que padrões que aparecem mais de 40 vezes são ruídos.

2. Padrões que tenham um número de elementos filho superior a 200 são removidos. Normalmente esses padrões pertencem a elementos do tipo "container", que englobam boa parte do conteúdo da página. Não sendo, portanto, alvo de análise do algoritmo.

3. Por fim, são removidos os *sub-padrões*. Sub-padrões são os padrões que estão contidos dentro de padrões maiores. Por exemplo, na FIGURA 12 fica claro que queremos encontrar o padrão referente a cada sub-árvore do segundo nível. No entanto, existem sub-padrões como as sub-árvores "img -> span", "li -> a" e "ul -> li -> a" que constam no Quadro 7 nas linhas 2 e 3. O algoritmo remove essas entradas desde que elas contenham um número de ponteiros menor ou igual a quantidade de vezes que o "padrão-pai" possui.

O código que realiza a remoção de ruído é exibido abaixo:

```
for (cada entrada pattern na tabela) do {
   if (pattern['qtd'] > 40) {
      remove(pattern);
   } else if (qtdElementos(pattern) > 200) {
      remove(pattern);
   } else if (isSubPattern(pattern)
         && patternPai(pattern)['qtd'] <= pattern['qtd']) {
      remove(pattern);
   }
}
```

O resultado da remoção de ruídos dos dados no Quadro 7 é demonstrado no Quadro 8. Como esperado, os sub-padrões são removidos.

Elemento	Nós-filhos	Número de Ponteiros
li	a, a, div, img, span, ul, li, a	9
li	a, a, div, img, span, ul	1
ul	li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul	1

Quadro 8: Resultado da remoção de ruídos.

6.3 Ordenação dos Resultados

Como mencionado anteriormente, a exibição dos produtos na árvore *DOM* segue o mesmo padrão com leves alterações. Por exemplo, na FIGURA 12 podemos ver claramente que a última sub-árvore pertence à mesma classe das outras, ainda que possua uma estrutura levemente modificada. Faz-se necessário então

que o algoritmo saiba "agrupar" padrões que provavelmente pertençam à mesma classe, mas que possuem estruturas levemente diferentes.

Inicialmente, necessita-se calcular o quão diferente são os padrões. Portanto, dados dois padrões com quantidades de elementos α e β, a equação 1 calcula o valor γ, que representa em valores numéricos a diferença entre eles.

$$\gamma = \left| \frac{(\alpha - \beta)}{(\alpha + \beta)} \right| \tag{1}$$

O valor retornado por γ varia de 0 a 1. Quando o valor retornado se aproxima de 0, significa que o número de elementos analisados são bem semelhantes. Opositivamente, quanto mais próximo de 1, menos semelhante são as quantidades. Ou seja, γ fornece um medidor quantitativo para a diferença de quantidade de elementos de cada padrão.

Foi efetuado então um calibramento do valor ideal de γ, através de análises do conjunto inicial de 13 lojas. Padrões só são considerados possíveis de junção entre si caso o valor de γ seja menor ou igual a *0.2*.

No entanto, apenas a análise da diferença numérica dos elementos não é o bastante. É preciso verificar qual a diferença estrutural entre eles. Isso é feito através da equação 2.

$$\delta = \frac{\{\{\alpha\} - \{\beta\}\}}{(\alpha + \beta)} \tag{2}$$

O valor fornecido por δ mensura de 0 a 1 quão diferente é as estruturas dos elementos analisados. Se esse valor se aproxima de 0, significa que as estruturas são bastante similares. Entretando, se δ se aproxima de 1, as estruturas são muito diferentes.

Novamente foi efetuado um calibramento do limite aceitável para o valor de δ, que foi estabelecido como menor ou igual a *0.15*.

Portanto, para que dois elementos diferentes sejam unidos é necessário que a quantidade de ponteiros que o sub-padrão contêm seja menor ou igual à quantidade de ponteiros do "padrão-pai" **e** que γ seja menor ou igual a *0.20* **e** que δ seja menor ou igual a *0.15*:

```
for (cada entrada pattern na tabela) do {
   for (cada entrada subpattern na tabela) do {

      if (pattern != subpattern
         && subpattern['qtd'] <= pattern['qtd']
         && gama <= 0.2
         && delta <= 0.15) {
            join(pattern, subpattern);
         }

   }
}
```

O resultado da aplicação do algoritmo acima nos dados da Quadro 8 é exibido no Quadro 9.

Por fim, é necessário que os padrões remanescentes sejam classificados em uma ordem de prioridade decrescente de acordo com a probabilidade de ser o padrão de produtos, o qual se objetiva encontrar.

Isso é realizado através da equação 3, aonde μ é a quantidade de vezes que o padrão se repete e θ é a quantidade de elementos únicos (não-repetidos).

$$\rho = \mu * \theta \tag{3}$$

Elemento	Nós-filhos	Número de Ponteiros
li	a, a, div, img, span, ul, li, a	10
ul	li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul	1

Quadro 9: Resultado da junção de padrões semelhantes.

Os padrões são organizados em ordem decrescente de acordo com o valor de ρ, garantindo que padrões com muitos elementos e poucas repetições ou padrões com poucos elementos e muitas repetições não recebam um valor discriminante alto, ao contrário de elementos mais complexos e que se repetem por mais vezes, que normalmente são as listagens de produtos.

O resultado da aplicação da equação 3 no Quadro 9 é exibido no Quadro 10, e como esperado o padrão que contém os produtos é o que recebe a maior prioridade.

6.4 Geração do Wrapper

Cada padrão detectado na etapa anterior é submetido para a fase de geração do *wrapper*, os elementos contidos em cada um são analisados em busca dos seguintes atributos:

1. Título: o nome do produto;

2. Link: o link para a página do produto;

Elemento	Nós-filhos	Número de Ponteiros	ρ
li	a, a, div, img, span, ul, li, a	10	60
ul	li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul, li, a, li, a, a, div, img, span, ul	1	6

Quadro 10: Resultado da organização de acordo com a prioridade.

3. Imagem: uma imagem que represente o produto;

4. Preço: qual o preço praticado para o produto;

5. Parcelas: qual é o parcelamento oferecido pela loja.

O algoritmo irá executar as operações necessárias até que um *wrapper* válido seja gerado ou os padrões terminem:

```
for (cada entrada pattern na tabela) do {
   // processos para gerar o wrapper
   if (wrapperValido()) break;
}
```

Um *wrapper* é considerado válido quando ele possui no mínimo as regras para extração de *título*, *link* e *preço*. Ou seja, regras para extração das *parcelas* e *imagens* não são obrigatórias.

6.4.1 Formato do Wrapper

O *wrapper* gerado por esta aplicação fornece a *URL* na qual um *termo de busca* deve ser submetido e um conjunto de *seletores CSS* que especifica aonde os dados devem ser buscados em um documento *HTML*. O formato é exemplificado abaixo:

```
http://www.americanas.com.br/busca/<term>
.hproduct >  a
.n
a
.parcel
.sale
.photo
```

A primeira linha fornece a *URL* que deve ser formatada e requisitada para realizar uma busca na fonte de dados. A aplicação que faz uso do *wrapper* simplesmente substitui a string "<term>" pelo termo de busca desejado, efetua uma requisição e transforma o documento em uma árvore *DOM*.

A segunda linha fornece um seletor que retorna todos os anúncios de produtos. A aplicação deverá iterar sobre cada elemento retornado por esse seletor usando os seletores descritos nas linhas seguintes para recuperar os atributos.

Da terceira linha em diante são especificados os seletores para *título do produto*, *link*, *parcelas*, *preço* e *imagem*, respectivamente. Nos seletores título, parcelas e preço o *parser HTML* deve simplesmente recuperar a informação textual contida dentro do nó retornado, enquanto que no seletor "imagem" ele deve recuperar o dado contido no atributo *HTML* "src" e no seletor link ele deve recuperar o dado contido no atributo "href".

As informações retornadas podem então ser manipuladas da maneira mais conveniente pela aplicação: armazenadas em banco de dados, exibidas para o usuário, etc.

6.4.2 Definindo Seletores CSS para os Anúncios

Como comentado anteriormente, a entrada para o algoritmo de geração de *wrapper* é um padrão previamente detectado. Cada um, por sua vez, possui um conjunto de elementos que *provavelmente* são os anúncios (referenciados pelos ponteiros).

O primeiro *seletor CSS* que se faz necessário é o que retornará todos os anúncios. Portanto, o algoritmo percorre cada elemento do padrão armazenando seu seletor em um vetor de ocorrências. Ao fim deste processo o seletor que mais se repete no vetor é definido como o *seletor CSS* dos produtos.

```
ocorrencias = [];
for (cada elemento element em pattern) do {
   ocorrencias[] = seletorCSS(element);
}

return maisRepetido(ocorrencias);
```

6.4.3 Extração do Título do Produto

Para que o título do produto seja extraído, um algoritmo percorre todos os nós do elemento verificando se o *termo de busca* está contido na informação textual do nó. Na maior parte dos casos, o título do anúncio contém os *termos de busca*.

Assim que o algoritmo encontra um nó que obedece a essa regra, o *seletor CSS* que permite identificá-lo é armazenado em um vetor de ocorrências. O processo é repetido até que todos os nós de todos elementos tenham sido analisados.

O *seletor CSS* que mais se repete no vetor de ocorrências é definido como o seletor para título de produto.

```
for (cada elemento element em pattern) do {
   ocorrencias = [];
   for (cada elemento no em element) {
      if (estaContido(termoDeBusca, no->text)) {
         ocorrencias[] = seletorCSS(no);
         break;
      }
   }
   return maisRepetido(ocorrencias);
}
```

6.4.4 Extração do Link do Produto

O algoritmo inicia percorrendo todos os nós de um elemento em busca do elemento “a”, que é a tag *HTML* para links. Todas as ocorrências tem seu atributo “href” armazenados, assim como o *seletor CSS* que permitem identificá-los. Essa fase é realizada para todos os nós de todos os elementos.

Com o vetor de ocorrências computado, o algoritmo remove todos os links que apontam para o mesmo documento. Isso porque o link para um produto não pode se repetir, cada anúncio é heterogêneo e possui sua própria página única. Links que referenciam a mesma página em diferentes anúncios são ruídos e, portanto, são removidos.

Nos dados remanescentes, o *seletor CSS* que mais se repete é identificado como seletor para link de produto.

```
for (cada elemento element em pattern) do {
   ocorrencias = [];
   for (cada elemento no em element) {
      if (no->nodeName == a) {
         ocorrencias[] = [no->href, seletorCSS(no)];
      }
   }

   for (cada elemento ocorrencia em ocorrencias) {
      for (cada elemento ocorrencia2 em ocorrencias) {
         if (ocorrencia != ocorrencia2) {
            if (ocorrencia[0] == ocorrencia2[0]) {
               remove(ocorrencia);
            }
         }
      }
   }

   return maisRepetido(ocorrencias);
}
```

6.4.5 Extração do Parcelamento do Produto

A extração das parcelas é realizada de forma bastante trivial: o algoritmo verifica se o conteúdo textual dos nós de cada elemento obedece à seguinte *expressão regular (Regex)*:

```
/[0-9]*(x|X) (de )?R\$( )?[0-9]+(.[0-9]*)?(,[0-9]*)?/
```

Qualquer texto com os formatos “12x de R$50”, “12X R$50”, etc. será reconhecido pela *regex* acima. Quando isso acontecer, o *seletor CSS* que identifica o elemento é armazenado no vetor de ocorrências.

Por fim, o seletor que mais se repete no vetor é definido como a regra de tradução para o atributo de parcelamento.

```
for (cada elemento element em pattern) do {
   ocorrencias = [];
   regex = '/[0-9]*(x|X) (de )?R\$( )?[0-9]+(.[0-9]*)?(,[0-9]*)?/';
   for (cada elemento no em element) {
      if (preg_match(regex, no->text)) {
         ocorrencias[] = seletorCSS(no);
         break;
      }
   }
   return maisRepetido(ocorrencias);
}
```

6.4.6 Extração do Preço do Produto

Assim como nas etapas anteriormente realizadas, o algoritmo inicia percorrendo todos os nós de cada elemento verificando se a string "R$" está contida na informação textual. Em caso positivo, é necessário que ela não obedeça à *regex* exibida na etapa anterior, ou seja, que a informação não seja de valor de parcelas. Caso a informação respeite às duas regras, o valor numérico da string e o *seletor CSS* é armazenado no vetor de ocorrências.

Entretanto, os anúncios de produtos costumam exibir mais de um valor de preço, como demonstrado na FIGURA 13. O algoritmo necessita então escolher o elemento mais provável de ser o preço correto.

Após uma análise no conjunto inicial de lojas, verificou-se que o comportamente mais comum é a exibição do preço praticado anteriormente e o novo preço. Como pode ser visto na FIGURA 13, todas as lojas exibem algo do tipo: "De R$ 1.000,00 por R$ 800,00". O algoritmo então faz a seguinte verificação:

Figura 13: Demonstração dos variados atributos de preço que são exibidos em um mesmo anúncio.

1. O valor numérico e o seletor *CSS* são guardados no vetor de ocorrências quando o nó contém a string "R$".

2. Se o algoritmo encontra outro nó com a string "R$" dentro do mesmo elemento, ele faz uma simples verificação: se o preço verificado for menor que o anteriormente encontrado e é maior que 20% dele, o último registro armazenado no vetor de referências é removido e os valores pertinentes ao nó atual são armazenados.

```
for (cada elemento element em pattern) do {
   ocorrencias = [];
   regex = '/[0-9]*(x|X) (de )?R\$( )?[0-9]+(.[0-9]*)?(,[0-9]*)?/';
```

```
    precoEncontrado = 0;
    for (cada elemento no em element) {
       if (estaContido('R$', no->text)
          && !preg_match(regex, no->text)) {
          precoAt = valorNumerico(no->text);
          if (precoEncontrado == 1) {
             precoAnt = ultimoRegistro(ocorrencias)[0];
             if (precoAnt < precoAt)
                && precoAt/precoAnt > 0.2) {
                   remove(ultimoRegistro(ocorrencias));
                   ocorrencias[] = [precoAt, seletorCSS(no)];
                }
          } else {
             ocorrencias[] = [precoAt, seletorCSS(no)];
          }
          precoEncontrado++;
       }
    }
    return maisRepetido(ocorrencias);
}
```

Após todos os elementos terem sido analisados, o seletor que mais vezes ocorre no vetor de ocorrências é definido como o seletor para o preço dos produtos.

6.4.7 Extração da Imagem do Produto

Na etapa de extração de imagens o algoritmo inicia percorrendo todos os nós de cada elemento até que ele encontre um nó do tipo "img". Quando isso acontece, a localização da imagem (atributo "src") e o *seletor css* são armazenados no vetor de ocorrências.

No entanto, a cada nova inserção o algoritmo verifica se a mesma imagem já não foi inserida no vetor, checando se a localização da imagem é única. Isso é

feito porque todo produto heterogêneo possui sua própria imagem, portanto as que se repetem são ruídos, como mostrado na FIGURA 14.

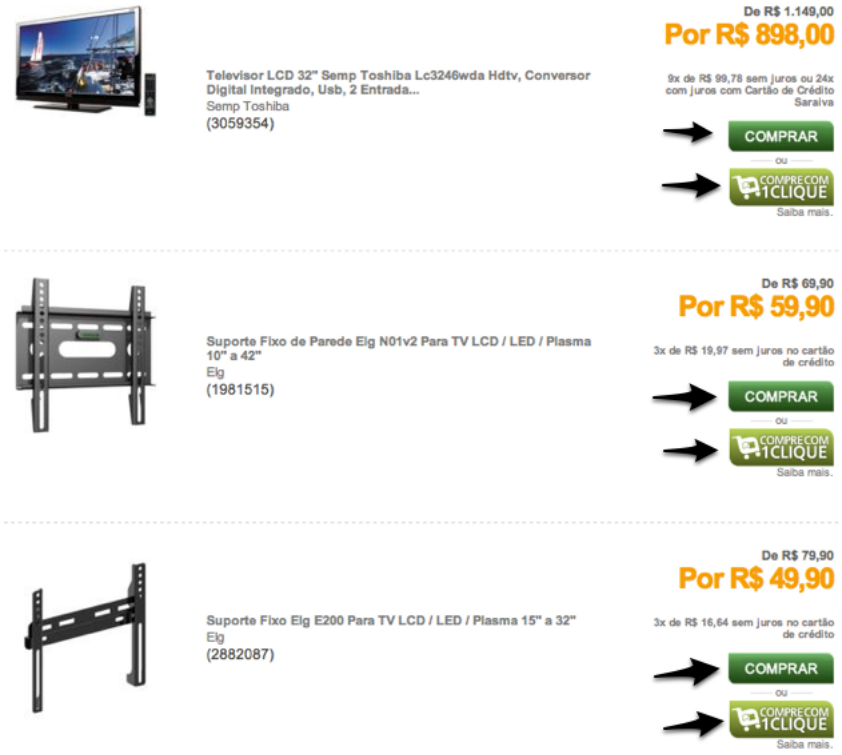

Figura 14: Demonstração da repetição de imagens em diversos anúncios diferentes.

Por fim, o seletor que mais se repete no vetor de ocorrências é definido como o tradutor para o atributo de imagens de produto.

```
for (cada elemento element em pattern) do {
   ocorrencias = [];
   for (cada elemento no em element) {
      if (no->nodeName == img) {
         flag = false;
         for (cada elemento ocorrencia em ocorrencias) {
            if (ocorrencia[0] == no->src) {
```

```
                flag = true;
                break;
            }
        }

        if (!flag) {
            ocorrencias[] = [no->src, seletorCSS(no)];
        }
    }
}

return maisRepetido(ocorrencias);
}
```

6.5 Implementação do Protótipo

O protótipo foi implementado em *PHP* usando programação orientada a objetos (*OOP - Object-Oriented Programming*), para possibilitar maior reuso e facilidade de manutenção do código-fonte.

Como mencionado anteriormente, o *parser HTML* utilizado para converter documentos *HTML* em árvores *DOM* foi o *phpQuery*.

Os *wrappers* que compõem o protótipo são previamente gerados pelo *system designer* e salvos em database *MySQL* no formato *JSON*.

6.5.1 Interface do Protótipo

Para que os experimentos pudessem ser realizados com os *wrappers* gerados, foi implementada uma interface para submissão de termos de busca e exibição dos resultados encontrados.

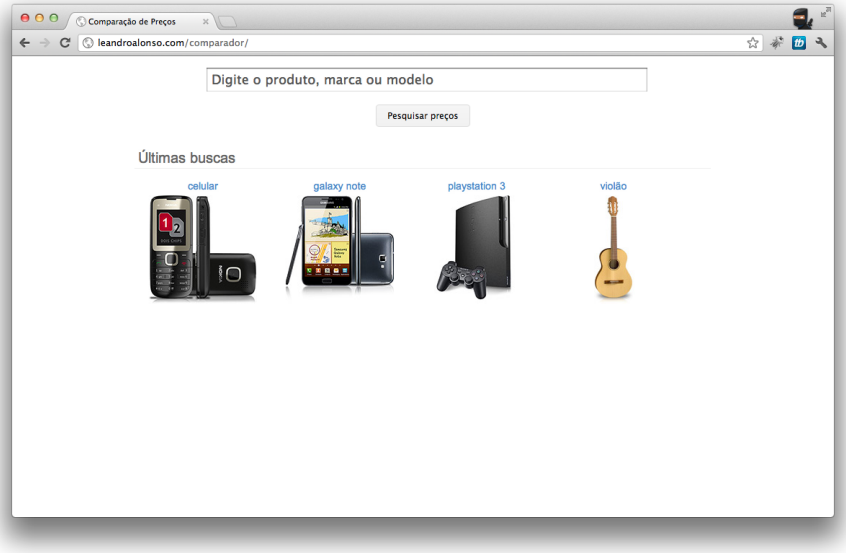

Figura 15: Interface inicial do protótipo.

A primeira interface com que o usuário depara-se, exibida na FIGURA 15, exibe as últimas buscas realizadas e um campo aonde um produto, marca ou modelo pode ser pesquisado.

Ao digitar um termo e submeter o formulário, cada loja de *e-commerce* é requisitada e o *wrapper* correspondente, que foi previamente gerado e está na *database*, traduz o documento *HTML* em informações relevantes. Os resultados são então exibidos para o usuário, como mostrado na FIGURA 16.

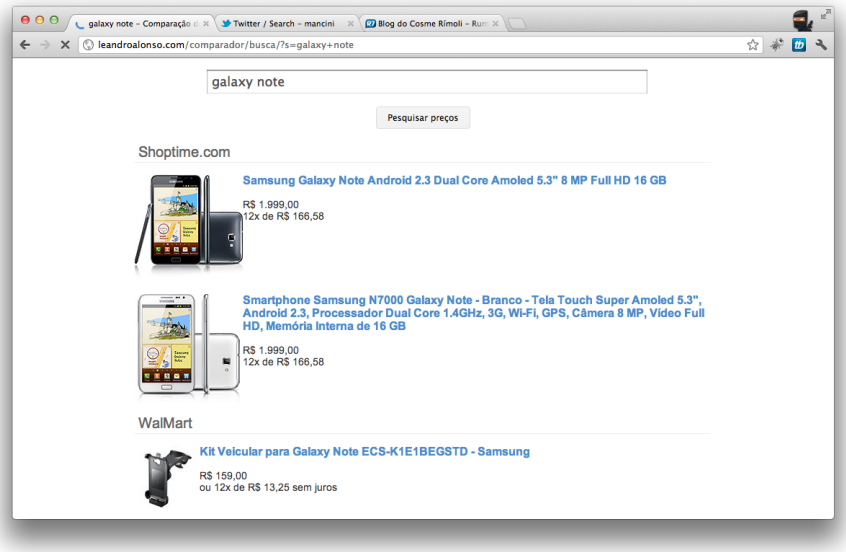

Figura 16: Interface do protótipo exibindo resultados para a busca por “galaxy note”.

7 Resultados e Discussão

Realizou-se dois tipos de experimentos para a validação dos resultados do *shopbot*:

1. **Verificação da Detecção de Padrões**: para cada fonte de dados foi verificado se o conjunto de padrões contém o padrão referente aos produtos. Para realizar tal verificação, todos os padrões detectados foram imprimidos na tela.

2. **Geração de um wrapper válido**: foi verificado se o *shopbot* retorna um *wrapper* válido para cada fonte de dados.

Para ambos os experimentos acima existem dois resultados possíveis: quando a detecção de padrões ocorre com sucesso e quando um *wrapper* é corretamente gerado o resultado é obtido com 100% de sucesso. Caso contrário, o resultado não é obtido e há 0% de sucesso. Portanto, o resultado dos experimentos é descrito por duas polaridades: *sim*, quando o resultado é obtido, e *não*, quando o experimento falha.

7.1 Resultados da Detecção de Padrões

A etapa de detecção de padrões foi inicialmente testada no conjunto inicial de 13 lojas de *e-commerce*. Os resultados são descritos na Tabela 11.

Estes resultados mostram que o desempenho da etapa de detecção de padrões foi completamente satisfatória, alcançando 100% de sucesso. O padrão que contêm os anúncios de produtos foram encontrados em todas as 13 lojas de *e-commerce* estabelecidas no conjunto inicial de testes.

Foi realizado então um novo experimento em um conjunto com mais 10 fontes de dados para validar o algoritmo e garantir que ele não seja direcionado

Loja	Padrão de Produtos Detectado?
Dafiti	Sim
Shoptime	Sim
Submarino	Sim
Nethoes	Sim
WalMart	Sim
CompraFácil	Sim
RicardoEletro	Sim
Americanas	Sim
Saraiva	Sim
Fnac	Sim
Extra	Sim
PontoFrio	Sim
MagazineLuiza	Sim

Quadro 11: Resultado da detecção de padrões no conjunto de fonte de dados inicialmente estabelecido.

para as lojas de *e-commerce* contidas no conjunto inicial. Os resultados são descritos na Tabela 12.

Como pode-se verificar, o algoritmo de detecção de padrões alcançou 70% de sucesso no conjunto adicional, os problemas verificados foram:

1. *Taqi*: O algoritmo foi incapaz de localizar o padrão.

2. *Loja Easy.com.br*: Houve problema com a codificação dos caracteres e o parser não foi executado, impossibilitando o prosseguimento da execução do algoritmo.

3. *Leader*: Houve problemas na execução do *parser HTML*, impossibilitando a manipulação do documento e a execução do algoritmo.

Excluindo-se problemas técnicos, o algoritmo falhou em apenas uma loja de *e-commerce*, o que garante um resultado satisfatório.

Loja	Padrão de Produtos Detectado?
KaBuM!	Sim
Colombo	Sim
Insinuante.com	Sim
Taqi	Não
Loja Easy.com.br	Não
Girafa.com.br	Sim
Shopfato.com	Sim
Centauro.com.br	Sim
Classic Tennis	Sim
Leader	Não

Quadro 12: Resultado da detecção de padrões no conjunto adicional de 10 lojas.

7.2 Resultados da Geração de Wrapper

Ainda que os resultados dos experimentos na etapa de detecção de padrões sejam animadores, sabe-se que o objetivo do *shopbot* é a geração de um *wrapper*. Portanto, a detecção de padrões é essencial para que o algoritmo possa prosseguir, mas a mensuração da qualidade do *shopbot* deve ser realizada através da verificação da geração correta de *wrappers*.

Assim como nos experimentos anteriores, os testes foram realizados primeiramente com o conjunto inicial de lojas. Os resultados estão descritos na Tabela 13.

Faz-se necessário novamente ressaltar que um *wrapper* é considerado válido quando possui no mínimo as regras para extração de *título*, *link* e *preço*. Portanto, como pode-se observar na Tabela 13, três *wrappers* não foram gerados.

O motivo para essa falha é que as lojas *Submarino*, *Extra* e *PontoFrio* não possuem os preços dos anúncios no documento *HTML*. Ele é dinâmicamente exibido através da execução de um *script*, que é realizada pelos *motores JavaScript* dos navegadores.

Loja	Wrapper Corretamente Gerado?
Dafiti	Sim
Shoptime	Sim
Submarino	Não
Nethoes	Sim
WalMart	Sim
CompraFácil	Sim
RicardoEletro	Sim
Americanas	Sim
Saraiva	Sim
Fnac	Sim
Extra	Não
PontoFrio	Não
MagazineLuiza	Sim

Quadro 13: Resultado da geração de *wrapper* no conjunto de fonte de dados inicialmente estabelecido.

O *parser HTML* não possui a capacidade de interpretar scripts e, portanto, o valor do produto não está contido na árvore *DOM*, fazendo com que o *shopbot* não consiga encontrar esse atributo. Consequentemente o *wrapper* não é corretamente gerado, mesmo que ele consiga todos os outros atributos.

Portanto, para o conjunto inicial, a taxa de sucesso do *shopbot* foi de 76%: das 13 lojas de *e-commerce* foram gerados 10 *wrappers* corretamente.

Foi então realizado um novo experimento com o conjunto adicional de 10 lojas de *e-commerce*, os resultados estão descritos na Tabela 14.

Como pode ser verificado, foram gerados 6 *wrappers* corretamente, garantindo uma taxa de sucesso de 60% no conjunto adicional. Os problemas encontrados foram:

1. *Taqi, Loja Easy.com.br e Leader*: A etapa de detecção de padrões havia falhado para essas 3 lojas, portanto era esperado que a etapa de geração de *wrapper* também falhasse.

Loja	Wrapper Corretamente Gerado?
KaBuM!	Não
Colombo	Sim
Insinuante.com	Sim
Taqi	Não
Loja Easy.com.br	Não
Girafa.com.br	Sim
Shopfato.com	Sim
Centauro.com.br	Sim
Classic Tennis	Sim
Leader	Não

Quadro 14: Resultado da geração de *wrapper* em um conjunto de 10 lojas.

2. *KaBuM!*: O algoritmo foi incapaz de detectar corretamente todos os atributos, retornando um *wrapper* inválido.

Levando-se em conta todas as lojas de *e-commerce* analisadas, O *shopbot* possui uma taxa de sucesso de 73%, sendo que a maior causa das falhas são atributos dinâmicamente exibidos por *scripts* ou problemas na execução do *parser HTML*.

8 Conclusões e Trabalhos Futuros

Serviços de *Shopping Comparison* tem sido amplamente utilizados, como demonstrado por Wan (2009). Devido a isto, se faz necessário soluções que sejam capazes de identificar atributos de produtos (título, preço, imagem, etc). Esses agentes inteligentes que realizam tais tarefas são chamados de *shopbots* (agentes inteligentes de *Shopping Comparison*).

Portanto, o objetivo deste trabalho foi elaborar um conjunto de procedimentos e regras que compõem a arquitetura de um *shopbot*. Como demonstrado no capítulo 7, o algoritmo proposto conseguiu gerar um *wrapper* reaproveitável e correto para 73% das lojas de *e-commerce* testadas. Ainda que o número de fontes de dados seja limitado, a amostra cobriu as principais lojas de varejo on-line do Brasil, o que demonstra uma boa taxa de eficácia.

Conclui-se então que o protótipo proposto tem potencial para ser empregado em uma aplicação real de *Shopping Comparison*, coletando dados de múltiplas fontes de dados e apresentando ao consumidor qual loja de *e-commerce* oferece o melhor preço.

No entanto, ainda que resultados satisfatórios tenham sido alcançados, eles podem ser melhorados através do uso de ferramentas de *HTML parser* mais robustas e atualizadas. Como mencionado no capítulo anterior, lojas de *e-commerce* que exibem o preço dinamicamente através de *JavaScript* constituem uma limitação ao protótipo proposto, visto que a ferramenta utilizada (*phpQuery*) não consegue identificar tais atributos.

Além disso, o *shopbot* pode ser incrementado através da adição de regras e procedimentos que identifiquem outros atributos, tais como: frete grátis, meios de pagamento, localização do produto, etc.

Por fim, propõe-se um estudo mais profundo dos casos em que o algoritmo falhou para que as técnicas sejam refinadas e, consequentemente, o algoritmo seja melhorado. Dessa forma, gerando resultados aperfeiçoados.

REFERÊNCIAS

BOS, B.; ÇELIK, T.; HICKSON, I.; LIE, H. W. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. [S.l.], jun. 2011. Http://www.w3.org/TR/2011/REC-CSS2-20110607/.

CHEN, Y.; SUDHIR, K. When shopbots meet emails: Implications for price competition on the internet. In: *Working Paper Series MK Marketing*. [S.l.: s.n.], 2001.

CONSORTIUM, W. W. W. *Introduction to HTML 4*. December 1999. [Online; acessado em 14 de Abril de 2012]. Disponível em: <http://www.w3.org/TR-/html401/intro/intro.htmlh-2.2>.

CONSORTIUM, W. W. W. *XHTML2 Working Group Home Page*. December 2010. [Online; acessado em 14 de Abril de 2012]. Disponível em: <http://www-.w3.org/MarkUp/>.

CONSORTIUM, W. W. W. *HTML & CSS*. 2011. [Online; acessado em 15 de Abril de 2012]. Disponível em: <http://www.w3.org/standards/webdesign-/htmlcsswhatcss>.

COOLEY, R.; MOBASHER, B.; SRIVASTAVA, J. Web mining: information and pattern discovery on the world wide web. In: *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*. [S.l.: s.n.], 1997. p. 558 –567.

COWIE, J.; LEHNERT, W. Information extraction. *Commun. ACM*, ACM, New York, NY, USA, v. 39, n. 1, p. 80–91, jan. 1996. ISSN 0001-0782.

CUDNIK, T. *phpQuery - jQuery port to PHP*. 2009. [Online; acessado em 15 de Abril de 2012]. Disponível em: <http://code.google.com/p/phpquery/>.

DOORENBOS, R. B.; ETZIONI, O.; WELD, D. S. A scalable comparison-shopping agent for the world-wide web. In: *In Proceedings of the First International Conference on Autonomous Agents*. [S.l.]: ACM Press, 1997. p. 39–48.

EBIT. *Relatório Web Shoppers*. 2011. Acessado em 9 de Abril de 2012. Disponível em: <http://www.webshoppers.com.br/>.

FIRAT, A. *Information integration using contextual knowledge and ontology merging*. Tese (Doutorado), 2003. AAI0805734.

HAMMOND, K.; BURKE, R.; MARTIN, C.; LYTINEN, S. Faq finder: a case-based approach to knowledge navigation. In: *Artificial Intelligence for Applications, 1995. Proceedings., 11th Conference on*. [S.l.: s.n.], 1995. p. 80 –86.

HUANG, S.-L.; TSAI, Y.-H. Designing a cross-language comparison-shopping agent. *Decis. Support Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 50, n. 2, p. 428–438, jan. 2011. ISSN 0167-9236.

JúNIOR, J. R. C. *Desenvolvimento de uma Metodologia para Mineração de Textos.* Disserta (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro-PUC-RIO, 2007.

KEPHART, J. O.; GREENWALD, A. R. Shopbot economics. In: *Proceedings of the third annual conference on Autonomous Agents*. New York, NY, USA: ACM, 1999. (AGENTS '99), p. 378–379. ISBN 1-58113-066-X.

KUSHMERICK, N. *Wrapper induction for information extraction*. Tese (Doutorado), 1997. AAI9819266.

MAAREK, Y. S.; SHAUL, I. Z. B. Automatically organizing bookmarks per contents. *Computer Networks and ISDN Systems*, v. 28, n. 7,Äì11, p. 1321 – 1333, 1996. ISSN 0169-7552. <ce:title>Proceedings of the Fifth International World Wide Web Conference 6-10 May 1996</ce:title>. Disponível em: <http://www.sciencedirect.com/science/article/pii/0169755296000244>.

MCKENNA, R. Creating value in the network economy. In: TAPSCOTT, D. (Ed.). Boston, MA, USA: Harvard Business School Press, 1999. cap. Real-time marketing, p. 145–158. ISBN 0-87584-911-3.

MONTGOMERY, A. L.; HOSANAGAR, K.; KRISHNAN, R.; CLAY, K. B.; (EMAIL, S. R. K.; W, I. T. W.; PROFESSOR, R. F. C. Designing a better shopbot. *Management Science*, v. 50, p. 189–206, 2004.

NYEIN, S. S. Mining contents in web page using cosine similarity. In: *Computer Research and Development (ICCRD), 2011 3rd International Conference on*. [S.l.: s.n.], 2011. v. 2, p. 472 –475.

PHP.NET. *O que é PHP?* 2012. [Online; acessado em 15 de Abril de 2012]. Disponível em: <http://www.php.net/manual/pt_BR/intro-whatis.php>.

PRASAD, R.; KUMARI, V.; RAJU, K. Comparison shopping agents: the essential characteristics and challenges to be met. In: *Intelligent Agent Multi-Agent Systems, 2009. IAMA 2009. International Conference on*. [S.l.: s.n.], 2009. p. 1 –2.

ROBIE, J.; HORS, A. L.; NICOL, G.; HéGARET, P. L.; CHAMPION, M.; WOOD, L.; BYRNE, S. *Document Object Model (DOM) Level 2 Core*

Specification. [S.l.], nov. 2000. Http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.

SANTOS, R. Conceitos de mineração de dados na web. In: TEIXEIRA, M. M.; TEIXEIRA, C. A. C.; TRINTA, F. A. M.; FARIAS, P. P. M. (Ed.). *XV Simpósio Brasileiro de Sistemas Multimídia e Web, VI Simpósio Brasileiro de Sistemas Colaborativos – Anais*. [S.l.: s.n.], 2009. p. 81–124.

SMITH, M. The impact of shopbots on electronic markets. *Journal of the Academy of Marketing Science*, Springer Netherlands, v. 30, p. 446–454, 2002. ISSN 0092-0703. 10.1177/009207002236916. Disponível em: <http://dx.doi.org/10.1177/009207002236916>.

SMITH, M. D.; BRYNJOLFSSON, E. Consumer decision-making at an internet shopbot: Brand still matters. *The Journal of Industrial Economics*, Blackwell Publishers Ltd, v. 49, n. 4, p. 541–558, 2001. ISSN 1467-6451. Disponível em: <http://dx.doi.org/10.1111/1467-6451.00162>.

TATBUL, N.; KARPENKO, O.; CONVEY, C.; YAN, J. *Data Integration Services*. [S.l.], May 2001.

WAN, Y. *Comparison-Shopping Services and Agent Designs*. [S.l.]: IGI Global, 2009. 1–336 p.

WAN, Y.; PENG, G. What's next for shopbots? *Computer*, v. 43, n. 5, p. 20 –26, may 2010. ISSN 0018-9162.

YANG, J.; CHOI, J.; KIM, J.; HAM, H.; LEE, K. A more scalable comparision shopping agent. In: *In Proc' EIS2000 Engineering of Intelligent Systems*. [S.l.: s.n.], 2000. p. 766–772.

YANG, S. J. H.; ZHANG, J.; TSAI, S. T. C. An automatic semantic segment detection service for html documents. In: *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 1*. Washington, DC, USA: IEEE Computer Society, 2008. (SCC '08), p. 210–217. ISBN 978-0-7695-3283-7-01. Disponível em: <http://dx.doi.org/10.1109/SCC.2008.155>.

ZHANG, J.; JING, B. The impacts of shopbots on online consumer search. In: *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. [S.l.: s.n.], 2011. p. 1 –10. ISSN 1530-1605.

ZHU, X.; GOLDBERG, A. B. *Introduction to Semi-Supervised Learning*. [S.l.]: Morgan & Claypool Publishers, 2009. (Synthesis Lectures on Artificial Intelligence and Machine Learning).