UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Roberto Panerai Velloso

ALGORITMO NÃO SUPERVISIONADO PARA SEGMENTAÇÃO E REMOÇÃO DE RUÍDO DE PÁGINAS *WEB* UTILIZANDO *TAG PATHS*

Florianópolis

2014

Roberto Panerai Velloso

ALGORITMO NÃO SUPERVISIONADO PARA SEGMENTAÇÃO E REMOÇÃO DE RUÍDO DE PÁGINAS *WEB* UTILIZANDO *TAG PATHS*

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientadora: Profa. Dra. Carina Friedrich Dorneles

Florianópolis

2014

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Velloso, Roberto Panerai
Algoritmo não supervisionado para segmentação e remoção de ruído de páginas web utilizando tag paths / Roberto Panerai Velloso ; orientadora, Carina Friedrich Dorneles - Florianópolis, SC, 2014.
67 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. remoção de ruído. 3. segmentação de página. 4. extração estruturada. 5. mineração de dados da web. I. Dorneles, Carina Friedrich. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Titulo.

Roberto Panerai Velloso

ALGORITMO NÃO SUPERVISIONADO PARA SEGMENTAÇÃO E REMOÇÃO DE RUÍDO DE PÁGINAS *WEB* UTILIZANDO *TAG PATHS*

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 3 de setembro 2014.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Programa

Banca Examinadora:

Profa. Dra. Carina Friedrich Dorneles
Orientadora
Universidade Federal de Santa Catarina

Profa. Dra. Renata Galante
Universidade Federal do Rio Grande do Sul

Prof. Dr. Renato Fileto
Universidade Federal de Santa Catarina

Prof. Dr. Ronaldo dos Santos Mello
Universidade Federal de Santa Catarina

À minha família.

AGRADECIMENTOS

Agradeço o apoio dos familiares e a atenção dispensada pelos professores do curso, em especial pela orientação que recebi para realização e publicação deste trabalho.

Está instaurada a dúvida.
A metódica dúvida epistemológica.
Neste mundo a terra não está no centro
nenhum saber é saber completo.
Seja bem-vinda era da razão.
Não há que se temer a revisão.
Nada que se diga ou que foi dito
merece estatuto de dogma irrestrito.
Cuidado com a verdade
que se pretende
maior que a realidade,
pois, os fatos são os fatos
e fluem diante de nós
que estupefatos
assistimos ao espetáculo.

Galileu Galilei

RESUMO

Segmentação e remoção de ruído de páginas *web* são etapas essenciais no processo de extração de dados estruturados. Identificar a região principal da página, eliminando o que não é importante (menus, anúncios, etc.), pode melhorar significativamente o desempenho do processo de extração. Para essa tarefa é proposto um novo algoritmo, totalmente automático, que utiliza uma sequência de *tag paths* (TPS) como representação da página *web*. A TPS é composta por uma sequência de símbolos (*string*), cada um representando um *tag path* diferente. O algoritmo proposto procura por posições na TPS onde é possível dividi-la em duas regiões de tal forma que seus alfabetos não se intersectem, o que significa que as regiões têm conjuntos de *tag paths* completamente distintos e, portanto, são regiões diferentes da página. Os resultados mostram que o algoritmo é muito efetivo em identificar o conteúdo principal de vários *sites*, e melhora a precisão da extração, removendo resultados irrelevantes.

Palavras-chave: remoção ruído. segmentação página. extração estruturada. mineração dados *web*.

ABSTRACT

Web page segmentation and data cleaning are essential steps in structured web data extraction. Identifying a web page main content region, removing what is not important (menus, ads, etc.), can greatly improve the performance of the extraction process. We propose, for this task, a novel and fully automatic algorithm that uses a tag path sequence (TPS) representation of the web page. The TPS consists of a sequence of symbols (string), each one representing a different tag path. The proposed technique searches for positions in the TPS where it is possible to split it in two regions where each region's alphabet do not intersect, which means that they have completely different sets of tag paths and, thus, are different regions. The results show that the algorithm is very effective in identifying the main content block of several major web sites, and improves the precision of the extraction step by removing irrelevant results.

Keywords: noise removal. page segmentation. structured extraction. web mining

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Um passo fundamental da extração de dados estruturados da *web*, é a fase de limpeza que deve ocorrer antes da extração da informação. Não se pode esperar bons resultados na fase de extração sem efetuar a limpeza da página, removendo o ruído indesejado. Em Yi, Liu e Li (2003), é mencionado que apesar da importância desta tarefa, existem relativamente poucos trabalhos nesta área e, de acordo com Liu e Chen-Chuan-Chang (2004), o ruído existente nas páginas pode prejudicar consideravelmente a extração de dados na *web*.

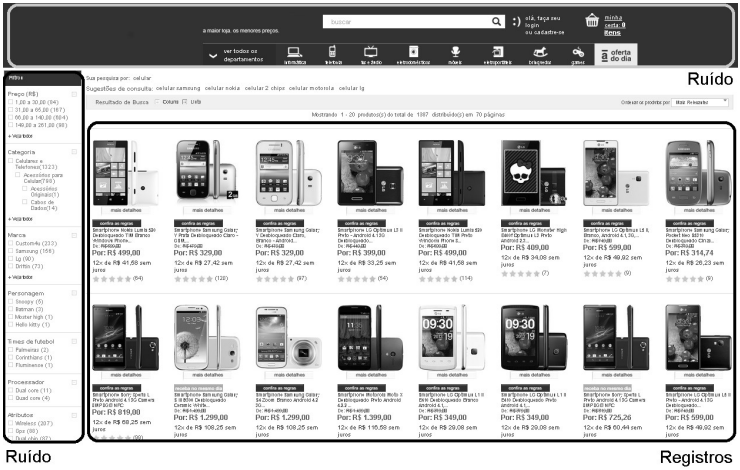

Figura 1 – Exemplo de uma página de comércio eletrônico com os registros e ruído identificados.

A extração de dados estruturados de páginas *web*, na maioria das propostas existentes, consiste na identificação e localização de padrões estruturais no documento HTML, os quais podem representar registros. Em Liu, Grossman e Zhai (2003), é dada uma definição, embora um tanto informal, para o termo "registro", neste contexto: "objetos regularmente estruturados, organizados em listas que, frequentemente, descrevem objetos similares". O problema é que, em geral, apenas os registros da região principal da página são de interesse, porém outras regiões do documento (como menus, anúncios, etc.) frequentemente contêm padrões em sua estrutura, fazendo com que sejam identificados como registros e, consequentemente, extraídos prejudicando a precisão

da extração dos dados. Daí surge a importância de se efetuar a limpeza das páginas *web* antes de se extrair seu conteúdo estruturado. A Figura 1 apresenta uma página típica de comércio eletrônico, onde os registros de interesse são os produtos apresentados no corpo do documento e não os menus (que também seguem um padrão estrutural) na barra lateral esquerda e no cabeçalho da página.

Alguns dos trabalhos na área de remoção de ruído e segmentação são elaborados para aplicação em IR (*Information Retrieval*) sendo direcionados para indexação e *clustering* de páginas (*i.e.* é assumido, nestes trabalhos, que a região principal é textual/não estruturada) como em Fernandes et al. (2011), Yi, Liu e Li (2003) e, devido às diferenças intrínsecas entre conteúdo estruturado e não estruturado, estas propostas não podem ser utilizadas para extração estruturada. As propostas existentes, que podem ser utilizadas para este fim, na sua maioria ou dependem do domínio em que são aplicadas (REIS et al., 2004), ou requerem definições *a priori* de regras estáticas e/ou bases de dados (CAI et al., 2003), ou são supervisionadas (não automáticas) e necessitam de uma amostragem das páginas que serão objeto da extração (CHAKRABARTI; KUMAR; PUNERA, 2008), ou então dependem de aspectos específicos da linguagem HTML para funcionar, como em Cho, Lin e Kao (2009).

O presente trabalho descreve um novo algoritmo para segmentação e remoção de ruído de páginas *web* que utiliza, como representação do documento HTML, a sequência de *tag paths* da árvore DOM. Conforme a análise apresentada na Seção 3.5, o algoritmo é computacionalmente eficiente, com complexidade média linear. Trata-se, em sua forma mais ampla, de um algoritmo de segmentação das regiões de um documento HTML que é utilizado aqui no contexto de extração estruturada e que leva em consideração o estilo (CSS) e a estrutura da página. As principais contribuições da proposta deste trabalho são as seguintes:

- Algoritmo totalmente automático: o algoritmo proposto não necessita de treinamento nem de intervenção humana;
- Independência de domínio: nenhuma suposição é feita com relação ao domínio do conteúdo das páginas de entrada do algoritmo;
- Independência da sintaxe HTML: não existe associação de regras, no algoritmo, a *tags* HTML específicas;
- Operação sobre uma única página: como entrada, é necessária

apenas uma única página, e não um conjunto de amostras de páginas de um mesmo *site*, o que constitui uma grande vantagem desta proposta, como discutido em Liu e Chen-Chuan-Chang (2004).

Durante a discussão dos resultados, são apresentadas e discutidas, também, as situações em que o algoritmo não obteve sucesso (total ou parcial). São apontadas as causas e sugeridas maneiras de se atacar as limitações identificadas durante a realização dos experimentos.

Para avaliar o quão efetivo é o algoritmo proposto neste trabalho, foi comparado o ruído de saída do MDR (LIU; GROSSMAN; ZHAI, 2003), uma conhecida técnica de extração estruturada, contra o ruído de saída do MDR combinado com o algoritmo proposto. O conjunto de testes consiste em uma coleção de 51 páginas de *sites* com conteúdo estruturado em diversos domínios como: vídeos, buscas, comércio eletrônico de vários segmentos, notícias, periódicos científicos, bancos, entre outros. O resultado obtido foi uma média de 88,86% de ruído removido das páginas do conjunto de teste.

Esta dissertação está organizada da seguinte maneira: no Capítulo 2 é feita uma revisão dos trabalhos relacionados, observando as limitações das propostas existentes para solução do problema; no Capítulo 3 são enunciadas as definições necessárias ao entendimento da proposta e são detalhados e ilustrados os algoritmos desenvolvidos a partir da formulação do problema e das hipóteses; no Capítulo 4 o método de avaliação dos resultados é detalhado e também são apresentados os resultados obtidos e as limitações do algoritmo proposto e, finalmente; no Capítulo 5 uma breve conclusão e sugestões de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Existem muitos trabalhos propondo formas de segmentar páginas *web* e identificar o que é conteúdo e o que é ruído nelas. Elas podem ser agrupadas em três categorias: as que utilizam o conteúdo textual da página e *tags* HTML (porém sem considerar a estrutura da árvore DOM), as que utilizam a árvore DOM e as que utilizam informação visual (o documento renderizado). O presente trabalho não utiliza nenhuma destas representações, embora necessite da árvore DOM para gerar a TPS do documento HTML. Nesta capítulo apresentamos, também, uma breve descrição do MDR, a técnica de extração utilizada neste trabalho como *baseline*.

2.1 PROPOSTAS BASEADAS NO TEXTO

Nos trabalhos detalhados nesta seção (KOHLSCHÜTTER; NEJDL, 2008; KOHLSCHÜTTER; FANKHAUSER; NEJDL, 2010; WENINGER; HSU; HAN, 2010) a segmentação das partes de uma página e subsequente identificação da região principal é realizada utilizando o conteúdo textual do documento. O foco principal destes trabalhos, entretanto, não é em extração estruturada, mas em indexação e *clustering* de páginas. A maioria dos trabalhos existentes sobre segmentação e remoção de ruído, que atuam sobre o texto, é direcionada para este tipo de aplicação, cujo objetivo principal é identificar o "assunto" da página.

Em Kohlschütter e Nejdl (2008) a página é modelada como uma sequência de blocos de texto delimitados por *tags* de abertura e fechamento (chamados de *gaps*). Os blocos de texto são analisados segundo sua densidade textual e os blocos contíguos, com aproximadamente a mesma densidade, são agrupados para formar blocos maiores e a região principal da página seria a de maior densidade textual. A densidade de texto é calculada segundo a Equação 2.1, onde $p(b_x)$ é a densidade do bloco b_x, n_{tokens_x} é o número de palavras do bloco e n_{lines_x} é o número de linhas do bloco, sendo que cada linha tem um tamanho arbitrário de 80 caracteres.

$$p(b_x) = \frac{n_{tokens_x}}{n_{lines_x}} \tag{2.1}$$

Em Kohlschütter, Fankhauser e Nejdl (2010), a mesma medida de densidade textual proposta em Kohlschütter e Nejdl (2008)

(Equação 2.1) é utilizada, porém uma série de outras características são agregadas ao problema de identificação da região principal como: o tamanho das palavras, tamanho das sentenças, número de palavras e a densidade de *links*, além de um conjunto de regras bem específicas para tratar textos e *tags* do documento. Estas outras informações agregadas são utilizadas para gerar uma árvore de decisão utilizada para identificar o conteúdo principal do documento.

Em Weninger, Hsu e Han (2010) é mencionada a desvantagem de se utilizar regras vinculadas a *tags* HTML específicas, o que pode invalidar uma abordagem no caso de uma eventual mudança na sintaxe HTML ou nas práticas de desenvolvimento atuais. Neste trabalho, em cada linha do código fonte HTML do documento, é calculada a taxa de *tags*, que consiste na relação entre o número de caracteres que não são *tag* e o número de *tags* existentes em uma determinada linha do código fonte do documento. O histograma resultante é suavizado com uma curva gaussiana e a região com o conteúdo principal (com taxa de *tags* elevada) é identificada por *clustering* ou pela escolha de um limiar. No caso da identificação por limiar, este tem que ser encontrado por experimentação.

2.2 PROPOSTAS BASEADAS NA ÁRVORE DOM

Nos trabalhos detalhados nesta seção (YI; LIU; LI, 2003; CHAKRABARTI; KUMAR; PUNERA, 2008; CHO; LIN; KAO, 2009; FERNANDES et al., 2011; ZHENG; GU; LI, 2012; LUTZ; HEUSER, 2013; SOUZA; MELLO, 2013) a segmentação, e subsequente identificação da região principal, é realizada utilizando a árvore DOM da página. Portanto, estas técnicas levam em consideração a estrutura da página *web* e, consequentemente, podem ser aplicadas no contexto de extração estruturada.

Em Souza e Mello (2013) a região principal é identificada através da localização das estruturas da árvore DOM que mais se repetem na página com auxílio de uma base de conhecimento previamente existente e, também, de regras heurísticas vinculadas à *tags* HTML. A proposta deste trabalho, além de identificar a região principal, também realiza a extração dos registro.

Em Yi, Liu e Li (2003) é construída uma árvore de estilos (com base nas definições CSS) para um determinado *site*, a partir de várias amostras de suas páginas, para determinar o que seria ruído e o que seria conteúdo nas páginas deste *site*, especificamente. Após a construção da árvore de estilos do *site* o ruído é determinado com base na seguinte

observação feita pelos autores: os blocos de ruído possuem formatação e conteúdo semelhantes, enquanto que o conteúdo é diverso e possui formatação diversa.

Em Chakrabarti, Kumar e Punera (2008) o problema de segmentar uma página é modelado como um problema de minimização de energia[1] em um grafo ponderado construído a partir de amostras de páginas anotadas manualmente com relação à sua segmentação. Os nós da árvore DOM são os nós do grafo e os pesos das arestas representam o custo de se segmentar os dois nós, que estão interligados no grafo, juntos ou separados. O custo é calculado a partir das características de cada nó (formato, cor de fundo, fonte, tamanho, etc.) e o peso de cada característica é determinado a partir do conjunto de treinamento. Apenas a questão da segmentação é abordada, sem a preocupação em identificar quais partes da página contêm conteúdo relevante e quais contêm ruído.

Em Cho, Lin e Kao (2009), uma árvore "visual" (não é, de fato, renderizada como nas técnicas visuais propriamente ditas) é gerada a partir da árvore DOM, redefinindo a árvore original com o auxílio de um dicionário de *tags* pré-definido que indica quais *tags* são relevantes na estrutura do documento. Cada nó da árvore "visual" tem sua entropia calculada com relação ao atributo *CLASS*. O trabalho propõe três metodologias distintas de avaliação da entropia: entropia dos nós filhos (*child entropy*), entropia do caminho (*path entropy*) e entropia do caminho parcial (*partial path entropy*). Após o cálculo da entropia, estes valores são analisados, comparados a um limiar estipulado e cada bloco da página é, então, classificado como: CB (*container block*), PB (*presentation block*), FB (*functional block*) ou IB (*informative block* (conteúdo relevante).

Em Fernandes et al. (2011) é adotada uma abordagem orientada a *site*, assim como Yi, Liu e Li (2003), onde um conjunto de árvores DOM de páginas de um mesmo *site* são alinhadas para identificar os segmentos de uma página a partir das semelhanças entre elas. O resultado do alinhamento é, então, refinado, de acordo com duas regras heurísticas (agrupamento de nós textuais próximos e agrupamento dos nós filhos com o nó pai), e o resultado final representa a subdivisão/organização/*template* do *site* analisado pelo algoritmo. Este trabalho apenas segmenta a página, sem identificar quais blocos/segmentos seriam ruído ou conteúdo. Apesar de necessitar de treinamento, este é realizado de maneira não supervisionada.

[1] encontrar cortes no grafo que minimizem o custo de uma determinada função de energia

Em Zheng, Gu e Li (2012) é utilizada uma base de dados de termos associados a "papéis semânticos" para encontrar, na árvore DOM, os nós folha que desempenham estes papéis, atribuindo, a cada um, uma medida de "entropia semântica-estrutural" de acordo com o papel desempenhado por aquele nó folha. A entropia dos demais nós é calculada recursivamente, a partir da entropia dos nós descendentes. Quanto maior for esta medida de entropia, maior a probabilidade de se tratar de uma região de interesse. A entropia é calculada conforme a Equação 2.2, onde n é um nó da árvore DOM e p_i é a proporção de nós folha, descendentes de n, que desempenham o papel i.

$$H_n = \sum_{i=1}^{m} p_i log(p_i) \tag{2.2}$$

Em Lutz e Heuser (2013) de forma semelhante à Kohlschütter, Fankhauser e Nejdl (2010), são utilizadas as densidades de texto e de *links*, porém, neste trabalho, o documento é tratado como uma árvore DOM e não como uma sequência de blocos. Além destas informações (densidade e texto e *links*), são utilizados os dados dos formulários da *web* oculta para formar mais um parâmetro (*QueryScore*) de treinamento para uma árvore de decisão gerada com o algoritmo C4.5 (QUINLAN, 1993). A árvore de decisão é, então, utilizada para identificar quais nós da árvore DOM são ruído e quais são conteúdo relevante na resposta à submissão de um formulário.

2.3 PROPOSTAS BASEADAS EM INFORMAÇÃO VISUAL

Além das técnicas baseadas no texto e na árvore DOM da página, existem também algumas que utilizam informação visual, como Cai et al. (2003), Simon e Lausen (2005), Fernandes et al. (2007), Liu, Meng e Meng (2010). Em geral utilizam a árvore DOM em conjunto com as características visuais do documento, as quais são obtidas por intermédio de um navegador *web*, utilizado para renderizar a página e fornecer as informações necessárias (*e.g.* coordenadas de tela, definições de estilo, etc.). Não existe um consenso, na comunidade científica, com relação ao custo-benefício (complexidade computacional *versus* ganho na precisão) de se utilizar um renderizador para esta finalidade. Em Kohlschütter e Nejdl (2008), é mencionada a desvantagem destas abordagens visuais, devido a grande complexidade computacional das mesmas, pois precisam renderizar a página para obter as informações de *layout*.

Em Cai et al. (2003) é utilizado um renderizador de um navegador *web* para obter informação visual (principalmente coordenadas e formatação) de uma página e, assim, construir uma estrutura de árvore que representa a hierarquia dos blocos visuais do documento. Além da informação visual, também é utilizado um extenso conjunto de regras que servem para dar tratamento diferenciado a *tags* HTML específicas com objetivo principal de identificar separadores visuais. Neste trabalho, assim como em Chakrabarti, Kumar e Punera (2008), não há preocupação em identificar quais blocos visuais da página contêm conteúdo relevante e quais contêm ruído.

Em Simon e Lausen (2005) é proposta uma técnica de extração estruturada, com identificação do conteúdo/ruído que utiliza o renderizador de um navegador *web* para corrigir (no caso de HTML mal formado) e processar o HTML e para obter as informações visuais necessárias ao algoritmo. A etapa de extração é uma adaptação, ou melhoramento, do algoritmo MDR (LIU; GROSSMAN; ZHAI, 2003) que incorpora o uso das informações visuais. A etapa de identificação do conteúdo principal ocorre após a identificação dos padrões estruturais (ou registros), onde recebem um peso, inversamente proporcional à distância que se encontram do centro da página. Outra melhoria proposta, relativa à identificação de padrões dos registros, é o tratamento de repetições consecutivas (*tandem repeats*) na comparação da estrutura dos registros. Esta abordagem identifica o ruído apenas após a etapa de extração, sem evitar o custo computacional de procurar por padrões no ruído.

Em Fernandes et al. (2007), é utilizada a abordagem visual de Cai et al. (2003) para segmentar a página em blocos. Um conjunto de páginas são analisadas e suas estruturas são comparadas para verificar se são equivalentes. Páginas com estruturas equivalentes são então analisadas com respeito às informações estatísticas dos termos contidos em seus blocos (subdivisões) e um *score* de importância é atribuído a cada bloco. Apesar de necessitar de um conjunto de páginas para treinamento, toda a análise do conjunto é realizada de forma não supervisionada.

Em Liu, Meng e Meng (2010) é proposta uma abordagem para extração estruturada que, assim como no trabalho de Fernandes et al. (2007), se baseia em Cai et al. (2003) para criar a representação visual do documento e, então, utilizar as características visuais das páginas da *web* oculta para localizar e extrair os registros contidos no documento. As características visuais utilizadas para identificação do conteúdo principal são: posição (centralização horizontal), tamanho (conteúdo rele-

vante é maior que demais blocos), *layout* (registros possuem arranjos semelhantes), semelhança visual (registros possuem formatação semelhante). Estas características são então utilizadas para filtrar blocos de ruído e para agrupar os demais blocos da página.

2.4 ANÁLISE COMPARATIVA DOS TRABALHOS

Na Tabela 1 é feito um comparativo entre as técnicas que foram encontradas na literatura e a proposta apresentada, com objetivo de explicitar as diferenças entre estes trabalhos. As diversas técnicas de remoção de ruído são comparadas com relação às seguintes características:

- Tipo de entrada do algoritmo: conteúdo textual da página, árvore DOM, informações visuais ou TPS;
- Única página: se o algoritmo consegue operar sobre apenas uma página de entrada ou se necessita de um conjunto de páginas;
- Treinamento: se o algoritmo necessita de treinamento e/ou supervisão;
- Informação *a priori*: se o algoritmo faz uso, ou necessita, de informações pré-definidas (*e.g.* dicionário de *tags*); e
- Independência da sintaxe HTML: técnicas que não tratam *tags* HTML de forma específica são consideradas independentes.

Tabela 1 – Comparativo entre técnicas de segmentação de página

#	Técnica	Entrada	Única página	Treinamento	Informação a priori	Indep. HTML	Abordagem
1	Kohlschütter e Nejdl (2008)	Texto	-	-	-	-	Densid. Textual
2	Kohlschütter, Fankhauser e Nejdl (2010)	Texto	-	-	-	-	*Decision tree*
3	Weninger, Hsu e Han (2010)	Texto	-	-	-	-	*Clustering* e limiar
4	Souza e Mello (2013)	DOM	Sim	Não	Sim	Não	Heurística
5	Yi, Liu e Li (2003)	DOM	Não	Sim	Não	Sim	Heurística
6	Chakrabarti, Kumar e Punera (2008)	DOM	Não	Sim	Não	Sim	Grafos
7	Cho, Lin e Kao (2009)	DOM	Sim	Não	Sim	Não	Entropia
8	Fernandes et al. (2011)	DOM	Não	Sim	Não	Sim	Heurística
9	Zheng, Gu e Li (2012)	DOM	Sim	Não	Sim	Sim	Entropia Semântica
10	Lutz e Heuser (2013)	DOM	Não	Sim	Não	Sim	*Decision tree*
11	Cai et al. (2003)	Visual	Sim	Não	Sim	Não	Conjunto de regras
12	Simon e Lausen (2005)	Visual	Sim	Não	Não	Sim	Distância
13	Fernandes et al. (2007)	Visual	Não	Sim	Não	Não	Estatística
14	Liu, Meng e Meng (2010)	Visual	Sim	Não	Sim	Não	*Clustering*
15	Velloso e Dorneles (2013)	TPS	Sim	Não	Não	Sim	Segment. TPS

Como é possível observar, na Tabela 1, as técnicas existentes, que foram analisadas, para eliminação de ruído, aplicáveis à extração estruturada (Linhas #4 a #14), com exceção de Simon e Lausen (2005) (Linha #12), não atendem a todos os requisitos que o algoritmo proposto atende (Linha #15). As técnicas que não dependem da sintaxe HTML ou necessitam de treinamento e/ou de definições *a priori*. A abordagem adotada em Simon e Lausen (2005) atende a todos os requisitos estipulados, no entanto, utiliza um renderizador (que é computacionalmente dispendioso) e só identifica o ruído posteriormente à identificação dos registros. Ao realizar a remoção do ruído apenas após a extração dos registros, perde-se a oportunidade de otimizar o processo de extração, onde seria possível identificar e eliminar o ruído antes de submeter a página ao algoritmo de extração.

2.5 MDR

O algoritmo MDR realiza a tarefa de extração de registros percorrendo a árvore DOM em largura e realizando comparações entre as subárvores adjacentes, verificando o quão semelhantes são. Ao encontrar ramificações que atendem ao limiar de semelhança estipulado, os nós das subárvores são alinhados (utilizando *partial tree alignment* (ZHAI; LIU, 2005)) e os registros são, então, extraídos. O algoritmo foi elaborado com base na seguinte observação: "os registros existentes em um documento HTML são contíguos/adjacentes e estão debaixo do mesmo nó pai".

Os motivos pelos quais foi escolhida esta técnica, e não outras mais recentes ou com melhores resultados, são os seguintes:

- a existência de grande quantidade de documentação e artigos publicados, o que possibilita implementação independente e validação dos resultados;
- a disponibilidade de implementações funcionais, de terceiros;
- o alto impacto das publicações originais, que são referenciadas até o presente momento;
- o fato de que apesar da idade do trabalho, este permanece relevante, funcional e apresenta resultados satisfatórios.

3 ALGORITMO PARA REMOÇÃO DE RUÍDO

O objetivo geral do algoritmo proposto neste trabalho é localizar a região principal de uma página *web* com conteúdo estruturado, eliminando o restante da página (considerado ruído) e, consequentemente, elevar a precisão da extração dos dados.

Neste capítulo, é apresentada, inicialmente, na Seção 3.1, a forma de representação dos documentos HTML utilizada neste trabalho (TPS). Na Seção 3.2, as definições necessárias para formular o problema. Na Seção 3.3, o problema é enunciado a partir das definições e das hipóteses levantadas. Na Seção 3.4, são apresentados os algoritmos desenvolvidos a partir do enunciado do problema e na Seção 3.5, é feita a análise da complexidade destes algoritmos.

3.1 REPRESENTAÇÃO TPS

A representação da página *web* utilizada no presente trabalho foi anteriormente empregada em Miao et al. (2009) e Xie et al. (2012). Em ambos os casos, esta representação foi usada para a extração e não para segmentação e remoção de ruído. Estes dois trabalhos são citados aqui apenas para ilustrar que, de acordo com seus resultados, esta representação, assim como a árvore DOM, também é capaz de expor a estrutura da página *web* e, portanto, serve para os propósitos dos algoritmos discutidos no presente trabalho. Esta representação da página *web* permite visualizar e manipular a estrutura do documento como uma *string* e não mais como uma árvore, o que facilita certas manipulações [1], em especial as realizadas neste trabalho (segmentação de regiões).

A Figura 2 ilustra como o código HTML de uma página *web* é convertido para representação de *tag paths*.

Na Figura 2, no passo "1" a árvore DOM é construída através do *parsing* do código HTML. No passo "2" a árvore DOM é então percorrida da raiz até cada um de seus nós para construir cada *tag path*, atribuindo um código (ou símbolo) para cada caminho diferente encontrado, reutilizando o mesmo código para *tag paths* iguais. No passo "3" é construída uma *string* apenas com os códigos dos *tag paths*, chamada de TPS (sequência de *tag paths*) e seu respectivo alfabeto

[1] *e.g.* Xie et al. (2012) utiliza um algoritmo de *strings* (árvore de sufixos) para encontrar padrões na TPS

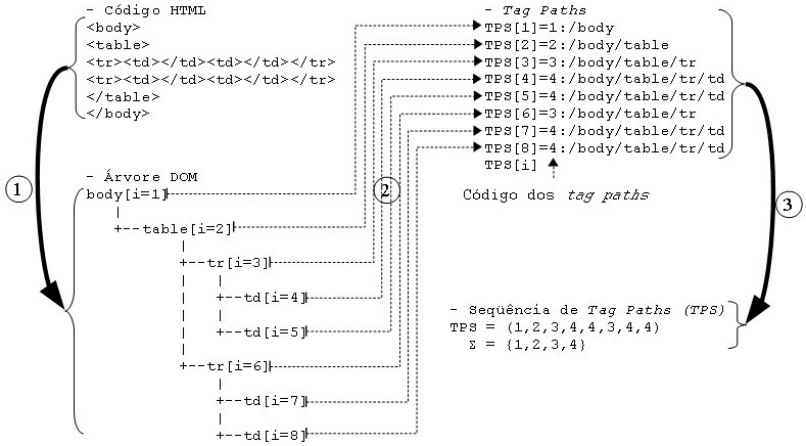

Figura 2 – Exemplo de uma sequência de *tag paths* construída a partir do código HTML.

(conjunto de símbolos da sequência).

3.2 DEFINIÇÕES BÁSICAS

Nesta seção são apresentados os conceitos e definições necessários para o entendimento do problema e do algoritmo proposto. As definições são exemplificadas com o auxílio da Figura 2.

Definição 1 (Árvore DOM) . *A árvore DOM (Document Object Model) é uma estrutura hierárquica, derivada do código HTML, que representa uma página web. Cada nó desta árvore representa um "objeto" do documento HTML. A especificação de referência é mantida pelo consórcio W3C (W3C, 2005) que é o órgão encarregado de manter os padrões e especificações da web.*

Exemplo. Na Figura 2 um pequeno trecho de código HTML é utilizado para ilustrar como uma árvore DOM é construída a partir do código.

Definição 2 (*tag paths*) . *Um tag path (TP) é uma string descrevendo o caminho absoluto, a partir da raiz da árvore DOM, até um dado nó da árvore. Seja "n_i" um determinado nó da árvore DOM, onde "i" representa a posição/ordem, em profundidade, do nó "n_i" na árvore, então é dito que TP_i é a string descrevendo o caminho, a partir da raiz da árvore DOM, até o nó "n_i".*

Exemplo. Na Figura 2, o caminho absoluto TP_4 do nó *body* até a célula da tabela td_4 é TP_4 = "*body/table/tr/td*".

Definição 3 (Sequência de *tag paths*) . *Uma sequência de tag paths (TPS) de uma árvore DOM com "n" nós é definida como sendo a sequência ordenada* $TPS[1..n] = (TP_1, TP_2, TP_3, ..., TP_{n-1}, TP_n)$ *onde dois tag paths* TP_i *e* TP_j*, com* $i \neq j$*, são considerados iguais apenas se seus caminhos e suas definições de estilo são iguais, caso contrário eles são considerados diferentes.*

Exemplo. Na Figura 2 é mostrada a TPS de um dado trecho de código HTML, onde cada TP recebe um código, produzindo $TPS = (1, 2, 3, 4, 4, 3, 4, 4)$. Na sequência de *tag paths* os *tag paths* não são representados pelas suas *strings*, mas sim por códigos (ou símbolos), com objetivo de facilitar sua manipulação, onde *tag paths* iguais recebem o mesmo código. Esta definição é similar à utilizada em Xie et al. (2012), onde cada *tag path* é representado na sequência por um símbolo, exceto que neste trabalho foram incorporadas as definições de estilo das *tags*. Como as definições CSS são utilizadas para diferenciar e caracterizar as diversas regiões de uma página, é muito importante levar em consideração essas definições na momento de segmentar a página.

Definição 4 (Alfabeto da TPS) . *Seja* Σ *o conjunto contendo todos os símbolos de uma dada sequência* TPS *de tamanho "n",* Σ *é dito o alfabeto de* TPS *definido como* $\Sigma = \{\alpha | TPS[i] = \alpha \wedge 1 \leq i \leq n\}$*, onde* α *é um símbolo do alfabeto.*

Exemplo. Na Figura 2, a TPS é formada apenas pelos símbolos "1","2", "3" e "4", então seu alfabeto é $\Sigma = \{1, 2, 3, 4\}$. Informalmente falando, o alfabeto contém todos os símbolos distintos de uma determinada sequência.

Definição 5 (Conjunto de frequências de *tag paths*) . *Seja* (s, f) *um par ordenado onde "s" é um símbolo de um alfabeto de uma determinada TPS e "f" é o número de vezes que o símbolo "s" ocorre nesta TPS, então o conjunto de frequências* FS *de uma TPS é definido como sendo o conjunto de todos os pares* (s, f) *existentes nesta TPS, ou seja* $FS = \{(s_1, f_{s1}), (s_2, f_{s2}), (s_3, f_{s3}), \ldots, (s_{n-1}, f_{sn-1}), (s_n, f_{sn})\}$*, onde "n"é o tamanho do alfabeto da TPS.*

Exemplo. Na Figura 2, o símbolo "1" ocorre uma vez na sequência, o símbolo "2" ocorre uma vez também, o símbolo "3" duas vezes e o símbolo "4" quatro vezes, então para esta sequência FS =

$\{(1,1),(2,1),(3,2),(4,4)\}$. O conjunto FS é o mapeamento de todos os símbolos de uma sequência para suas respectivas frequências.

Definição 6 (Limiares de frequência) . *Dada uma TPS com alfabeto Σ e conjunto de frequências FS, os limiares de frequência FT são definidos como sendo a sequência ordenada contendo apenas as frequências distintas de FS. Assim, temos que $FT = \{f|\exists(s,f) \wedge (s,f) \in FS \wedge s \in \Sigma\}$, onde "s" é um símbolo do alfabeto Σ e "f" é sua respectiva frequência.*

Exemplo. Na TPS exibida na Figura 2, o conjunto de frequências dos *tag paths* é $FS = \{(1,1),(2,1),(3,2),(4,4)\}$. Neste caso, o conjunto dos limiares de frequência é igual a $FT = \{1,2,4\}$, pois os símbolos "1" e "2" tem frequência igual a **1**, o símbolo "3" tem frequência igual a *2* e o símbolo "4" tem frequência igual a *4*. O conjunto FT é necessário para filtrar os símbolos da TPS. Se $FT = \{1,2,4\}$, então não faria sentido filtrar símbolos com frequência igual a *3*, pois não existem símbolos na sequência com esta frequência.

Definição 7 (Região) . *Seja uma TPS a concatenação de duas outras sequências $TPS = TPS_a.TPS_b$, então é dito que TPS_a e TPS_b são regiões de TPS, sse $\Sigma_a \cap \Sigma_b = \emptyset$.*

Exemplo. Na Figura 2 a TPS pode ser dividida em duas subsequências $TPS_a = TPS[1..2] = (1,2)$ e $TPS_b = TPS[3..8] = (3,4,4,3,4,4)$, com alfabetos $\Sigma_a = \{1,2\}$ e $\Sigma_b = \{3,4\}$, então TPS_a a TPS_b são regiões distintas de TPS, pois a intersecção de seus respectivos alfabetos é vazia.

3.3 PROPOSTA

No presente trabalho é proposto um novo algoritmo para segmentação das regiões de uma página *web* e identificação da região principal. O algoritmo proposto leva em consideração a estrutura e o estilo de formatação das páginas e é computacionalmente eficiente pois, conforme demonstrado mais adiante, na Seção 3.5, sua complexidade é, na média, linear em relação ao tamanho da página. A página *web*, que é submetida ao algoritmo, é representada como uma sequência de *tag paths*. O algoritmo proposto é formulado a partir de duas hipóteses:

1. diferentes regiões de uma página *web* são descritas por diferentes *tag paths*;

   A formulação da hipótese 1 vem da observação de que as várias regiões de uma página *web* são diferentes ramificações da árvore DOM e essas regiões são descritas ou utilizando *tags* distintas para cada região ou, se as *tags* forem semelhantes, com diferentes estilos, para que cada região possa ser facilmente distinguida pelo usuário/leitor da página. Se todas as regiões de uma página se parecessem, seria mais difícil, para o usuário, distingui-las. Então, pela Definição 3, pode-se notar que os símbolos utilizados em cada região de uma página *web* deveriam ser diferentes, e então deve ser possível segmentar a página como apresentado na Definição 7. Esta hipótese, como é possível observar, é geral o suficiente para poder ser reutilizada em outros contextos, com objetivo de segmentar páginas web, que não o de extração estruturada.

2. em *sites* com conteúdo estruturado, a região principal é mais densa/maior que as demais regiões (menus, anúncios, texto, etc.).

   A hipótese 2 vem do contexto no qual a segmentação proposta neste trabalho é aplicada (*i.e.* extração estruturada). Como se está considerando apenas páginas com registros, e para descrever a estrutura desses registros são necessários nós da árvore DOM, então são necessários mais nós para descrever conteúdo estruturado do que é necessário para conteúdo não estruturado (*e.g.* texto). Portanto, é razoável assumir que, para páginas que contêm registros, a região principal da página é a maior de todas (*i.e.* a que contém mais nós).

Utilizando as definições da Seção 3.2 e as hipóteses acima, podemos formular o problema de segmentação de páginas e identificação da região principal como: **"encontrar a maior *região* da TPS de uma página *web* que possua um *alfabeto* que não se intersecte com o *alfabeto* de outras *regiões* menores"**.

Um detalhe **crucial** que deve ser levado em consideração, é que podem existir *tag paths* em um documento que representam divisões estruturais da página (*i.e.* formatação visual da página). Estes *tag paths*, se forem divisões, ocorrerão ao longo de toda a TPS, impedindo a identificação das regiões, pois sempre haverá uma intersecção dos alfabetos. Para remover este "ruído" da sequência, podemos filtrar os símbolos com frequências mais baixas, sem prejudicar o processo de segmentação, pois os símbolos com frequências mais altas continuam

sendo considerados. Para ilustrar este procedimento, é apresentado um exemplo de uma página *web* na Listagem 3.1 que inicia e termina com o mesmo *tag path* ("/body/br", Linhas 2 e 18) e tem três regiões delimitadas, também, pelo mesmo *tag path* ("/body/div", Linhas 3, 8 e 13). Partindo da hipótese 1, em que diferentes *tag paths* são utilizados para descrever cada região, sem filtrar os símbolos de baixa frequência da TPS, não seria possível identificar essas regiões utilizando o algoritmo proposto.

Listagem 3.1 – Código HTML

```
1  <body>
2    <br> <!-- cabeçalho -->
3    <div> <!-- menu -->
4      <span class='region1'>...</span>
5      ...
6      <span class='region1'>...</span>
7    </div>
8    <div> <!-- região principal -->
9      <span class='region2'>...</span>
10     ...
11     <span class='region2'>...</span>
12   </div>
13   <div> <!-- publicidade -->
14     <span class='region3'>...</span>
15     ...
16     <span class='region3'>...</span>
17   </div>
18   <br> <!-- rodapé -->
19 </body>
```

A TPS, do código HTML apresentado na Listagem 3.1, é $TPS = (1, 2, 3, 4, ..., 4, 3, 5, ..., 5, 3, 6, ..., 6, 2)$. Como os símbolos 2 e 3 ocorrem ao longo de toda a sequência, os alfabetos das regiões de menus, publicidade e da região principal, sempre terão uma intersecção. Porém, considerando agora, para fins ilustrativos, apenas os símbolos com frequência maior que 3, durante o processo de segmentação é possível encontrar as regiões com alfabetos distintos (sem intersecções) e dividir a TPS da seguinte maneira:

$TPSfiltrada = (, , , 4, ..., 4, , 5, ..., 5, , 6, ..., 6,)$
$menus = (4, ..., 4)$
$principal = (5, ..., 5)$
$publicidade = (6, ..., 6)$

Neste exemplo, estipulamos uma determinada frequência apenas para ilustrar qual o objetivo deste procedimento. Entretanto, no algoritmo *searchRegion*(), a filtragem, de fato, é realizada iterativamente. Cada frequência é "testada" até que seja possível localizar um ponto para dividir a sequência.

3.4 ALGORITMOS

Nesta seção, são apresentados os algoritmos desenvolvidos para tratar o problema definido na Seção 3.3. Eles são os seguintes:

- ***tagPathSequenceFilter*()** é o algoritmo de nível mais alto, que recebe um arquivo HTML como entrada, chama os demais algoritmos e retorna uma árvore DOM, podada, apenas com o conteúdo da região principal;
- ***convertToSeq*()** converte a árvore DOM de uma página *web* para sua representação TPS;
- ***searchRegion*()** é o algoritmo principal, que realiza a busca, propriamente dita, pela região principal da TPS;
- ***filterAlphabet*()** filtra um alfabeto, removendo os símbolos de menor frequência, tornando o algoritmo de busca mais robusto e resistente ao ruído;
- ***pruneDOMTree*()** poda a árvore DOM original, deixando apenas o conteúdo da região principal encontrado com *searchRegion*, mantendo a estrutura do documento original.

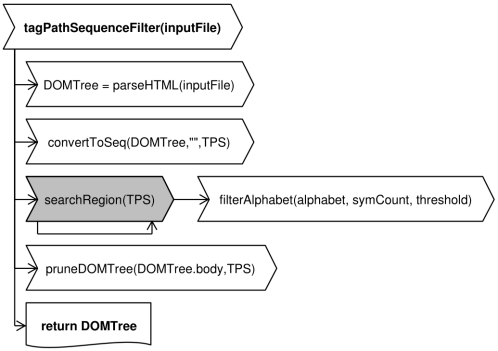

Figura 3 – Visão de alto nível da interação entre os algoritmos.

A Figura 3 apresenta uma visão de alto nível da interação existente entre os algoritmos desenvolvidos no presente trabalho, com o algoritmo principal (*searchRegion*) destacado.

3.4.1 Algoritmo de *tagPathSequenceFilter*()

Algoritmo 1 Filtra o ruído de uma página *web*

Input: *inputFile* - um arquivo HTML
Output: árvore DOM de *inputFile* podada

```
1: procedure tagPathSequenceFilter(inputFile)
2:     DOMTree ← parseHTML(inputFile)
3:     convertToSeq(DOMTree.body, " ", TPS) ▷ " " = string vazia
4:     searchRegion(TPS)
5:     pruneDOMTree(DOMTree.body, TPS)
6:     return DOMTree
7: end procedure
```

O procedimento *tagPathSequenceFilter*() no Algoritmo 1 retorna a região principal de *inputFile*. O procedimento *parseHTML*(), na Linha 2, converte o código HTML em uma árvore DOM; *convertToSeq()*, na Linha 3, converte a árvore DOM em uma TPS; o procedimento *searchRegion*(), na Linha 4, procura recursivamente pela maior região da TPS que possui um alfabeto único e, finalmente; *pruneDOMTree*(), na Linha 5, retira da árvore DOM (poda) todos os nós que não fazem parte da TPS resultante, preservando a estrutura do documento retornado na Linha 6.

Em seguida são detalhados os algoritmos *convertToSeq*(), *searchRegion*(), *filterAlphabet*() e *pruneDOMTree*(). O algoritmo *parseHTML*(), que consiste em gerar a árvore DOM a partir do código HTML, não faz parte do escopo deste trabalho e, portanto, não é discutido aqui.

3.4.2 Algoritmo de *convertToSeq*()

O procedimento *convertToSeq*() no Algoritmo 2 converte uma página *web* de sua representação em árvore DOM para uma representação em formato TPS, percorrendo a árvore DOM em profundidade. Inicialmente é chamado no Algoritmo 1 com um *tag path* vazio como parâmetro, que representa a *string* do *tag path* anterior (*i.e.* da

Algoritmo 2 Converte uma árvore DOM para a representação de sequência de *tag path*

Input: *node* - um nó da árvore DOM, inicialmente a raiz da árvore
Input: *tp* - o *tag path* anterior, inicialmente vazio
Input: TPS - a TPS de uma árvore DOM, inicialmente vazia
Output: a TPS de uma árvore DOM armazenada em TPS

```
1:  procedure convertToSeq(node,tp,TPS by ref.)
2:      tp ← concat(tp,"/", node.tag, node.style)
3:      if tp ∋ tagPathMap then
4:          tagPathMap ← tagPathMap + {tp}
5:          tagPathMap[tp].code ← tagPathMap.size
6:      end if
7:      TPS ← concat(TPS, tagPathMap[tp].code);
8:      for each child of node do
9:          convertToSeq(child, tp, TPS)
10:     end for
11: end procedure
```

chamada recursiva anterior). Na Linha 2, o *tag path* anterior é concatenado com a *tag* atual e com suas definições de estilo, com objetivo de distinguir caminhos repetidos com estilos diferentes; na Linha 3, é verificado se o *tag path* atual já foi visto antes pelo algoritmo ou não ($tagPathMap$ é utilizado para este propósito) e, se não, na Linha 4, ele é inserido no conjunto $tagPathMap$ e um novo código é atribuído a ele na Linha 5, como mostrado na Definição 3; na Linha 7, o código do *tag path* é concatenado ao final da sequência e, finalmente, o procedimento é invocado recursivamente na Linha 9 para cara filho de *node*.

3.4.3 Algoritmo de *searchRegion*()

Este é o algoritmo central, já que é responsável por encontrar a região principal e, para facilitar seu entendimento, foi incluída uma ilustração do seu funcionamento (a Figura 4). Ela apresenta uma ilustração da execução do algoritmo excluindo o procedimento de filtragem dos símbolos com baixas frequências para maior clareza, pois a intenção é passar a idéia principal. Na Figura 4 estão representadas as várias iterações feitas em cima da sequência até o ponto em que a região principal é localizada. Em cada iteração, o cursor percorre as posições da sequência e verifica, em cada posição, se não há intersecção entre os

alfabetos das subsequências “a” e “b” e, neste caso, inicia uma nova iteração em cima da maior subsequência resultante, até que esta não possa mais ser subdividida, situação em que a região principal foi localizada. Uma descrição detalhada é apresentada para a listagem do algoritmo.

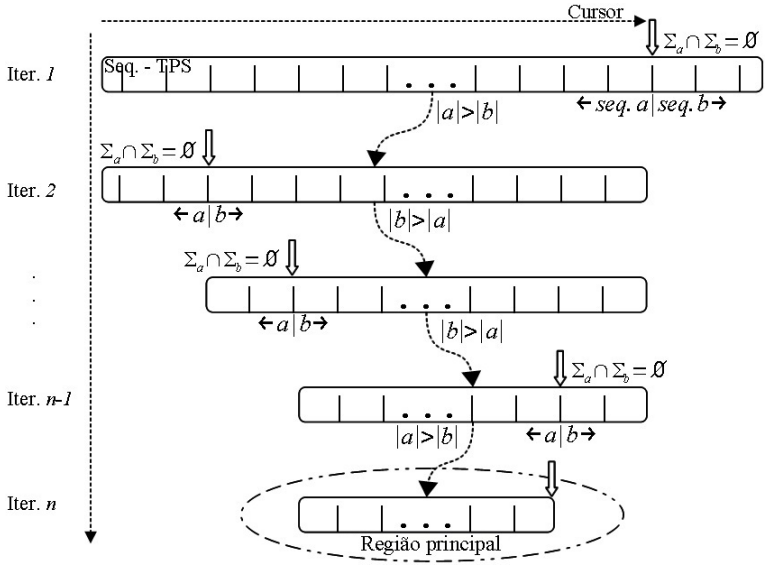

Figura 4 – Ilustração da execução do algoritmo.

Algoritmo 3 Busca regiões da TPS com alfabetos diferentes

Entrada: *tagPathSequence* - a TPS de uma determinada página *web*
Saída: a região principal da TPS, armazenada em *tagPathSequence*

```
1: procedure searchRegion(tagPathSequence[1..n] by reference)
2:     Σ ← ∅                                          ▷ alfabeto da sequência
3:     t ← 0                                          ▷ índice dos limiares de frequência
4:     for i ← 1..n do
5:         sym ← tagPathSequence[i]
6:         if sym ∋ Σ then
7:             Σ ← Σ ∪ {sym}
8:             symCount[sym] ← 0
9:         end if
10:        symCount[sym] ← symCount[sym] + 1
11:    end for
12:    thresholds ← OrderedSetOfFrequencies(symCount)
13:    regionFound ← false
14:    while not regionFound do
15:        t ← t + 1
16:        currΣ ← filterAlphabet(Σ, symCount, thresholds[t])
17:        if currΣ.size < 2 then
18:            break
19:        end if
20:        currSymCount ← symCount
21:        regionΣ ← ∅
22:        for i ← 1..n do
23:            sym ← tagPathSequence[i]
24:            if sym ∈ currΣ then
25:                regionΣ ← regionΣ ∪ {sym}
26:                currSymCount[sym] ← currSymCount[sym] − 1
27:                if currSymCount[sym] = 0 then
28:                    currΣ ← currΣ − {sym}
29:                    if currΣ ∩ regionΣ = ∅ then
30:                        if (currΣ ≠ ∅) ∧ (n − 2i)/n > 0.20 then
31:                            regionFound ← true
32:                        end if
33:                        break
34:                    end if
35:                end if
36:            end if
37:        end for
38:    end while
39:    if regionFound then
40:        if i < n/2 then
41:            tagPathSequence ← tagPathSequence[i + 1..n]
42:        else
43:            tagPathSequence ← tagPathSequence[1..i]
44:        end if
45:        searchRegion(tagPathSequence)
46:    end if
47: end procedure
```

O procedimento *searchRegion*() no Algoritmo 3 calcula o alfabeto da TPS e as frequências correspondentes de cada símbolo nas Linhas 4 a 11; na Linha 12, os limiares de frequência, como especificado na Definição 6, são calculados; nas Linhas 14 a 38 é realizada a busca por uma posição na sequência onde é possível dividi-la (*i.e.* onde existe uma região); na Linha 15 os limiares de frequência são iterados; na Linha 16 o alfabeto da TPS, da Definição 4, é filtrado, como descrito na Subseção 3.4.4; na Linha 22 a TPS é iterada; na Linha 25 o alfabeto da região é calculado e; nas Linhas 27 a 35 é verificado se existe uma intersecção entre os alfabetos das duas partes da TPS (uma intersecção vazia indica que uma possível região foi encontrada, como na Definição 7). A região encontrada só é considerada válida se for ao menos 20% maior que o restante da sequência. Caso contrário, o filtro de frequência é iterado e a busca prossegue. Este percentual é, na verdade, um parâmetro com o propósito de evitar relatar uma região sob condições ambíguas (nos experimentos foi utilizado o valor de 20%); finalmente nas Linhas 39 a 46 a TPS é dividida se uma região foi encontrada, chamando o procedimento *searchRegion*() recursivamente na Linha 45.

3.4.4 Algoritmo de *filterAlphabet*()

Algoritmo 4 Filtra símbolos de baixa frequência no alfabeto

Input: Σ - alfabeto a ser filtrado
Input: *symCount* - conjunto das frequências (FS) dos símbolos de Σ
Input: *threshold* - limiar de frequência
Output: alfabeto filtrado

```
1: procedure FILTERALPHABET(Σ, symCount, threshold)
2:     filteredΣ ← ∅
3:     for i ← 1..n do
4:         if symCount[Σ[i]] ≥ threshold then
5:             filteredΣ ← filteredΣ ∪ {Σ[i]}
6:         end if
7:     end for
8:     return filteredΣ
9: end procedure
```

O procedimento $filterAlphabet()$ no Algoritmo 4 remove do alfabeto Σ, todo símbolo com frequência inferior a *threshold*. Nas Linhas 3 to 7 apenas os símbolos com frequência maior ou igual a *threshold*

são inseridos no conjunto resultante. O resultado de $filterAlphabet()$ é utilizado no Algoritmo 3, Linha 24, onde apenas os símbolos em $curr\Sigma$ são considerados durante a busca pela região principal.

3.4.5 Algoritmo de *pruneDOMTree*()

Algoritmo 5 Poda da árvore DOM (nós que não pertencem à sequência)

Input: *node* - um nó da árvore DOM, inicialmente a raiz
Input: *sequence* - a TPS que deve permanecer na árvore DOM
Output: árvore DOM podada, referenciada em *node*

```
 1: procedure PRUNEDOMTREE(node by ref.,sequence)
 2:     for each child of node do
 3:         if pruneDOMTree(child, sequence) = true then
 4:             remove child from node
 5:         end if
 6:     end for
 7:     if node ∋ sequence and node.childCount = 0 then
 8:         return true
 9:     end if
10:     return false
11: end procedure
```

O procedimento $pruneDOMTree()$ no Algoritmo 5 percorre a árvore DOM em profundidade removendo os nós que não pertencem a *sequence*. Na Linha 3 a árvore DOM é percorrida; nas Linhas 7 a 9 é decidido se o nó *node* deve ser removido ou não.

Um nó é removido da árvore apenas se ele não está em *sequence* **e** não tem nenhum filho. Ao evitar a exclusão de nós internos da árvore que ainda possuem filhos, mesmo quando estes nós internos não pertencem à sequência, é possível manter a estrutura da árvore remanescente intacta, evitando afetar, negativamente, a etapa subsequente de extração estruturada.

3.5 COMPLEXIDADE DO ALGORITMO

Nas Equações 3.1, 3.2 e 3.3 de análise de complexidade foi utilizada a notação padrão encontrada na literatura de análise assintótica,

onde $T(.)$ representa a recorrência, $\Theta(.)$ representa o limite justo (superior e inferior) e $O(.)$ representa o limite superior não justo.

A complexidade analisada refere-se à Linha 29 (cálculo da intersecção dos alfabetos) do procedimento *searchRegion*(), pois esta operação de intersecção é a que possui maior custo computacional associada ao tamanho da entrada do algoritmo. Portanto, é suficiente analisar apenas este procedimento. Na análise de complexidade foi considerado que a operação de intersecção é realizada em cada posição da sequência. Na prática, entretanto, esta operação é realizada apenas quando ocorre uma mudança no alfabetos. Observando as Linhas 14 e 22 do procedimento *searchRegion*(), pode-se ver que o laço de repetição da Linha 14 itera os limiares de frequência até encontrar uma região e na Linha 22 a TPS, filtrada em uma determina frequência, é iterada, também, até encontrar uma região e, se encontrada, a região é processada recursivamente.

No pior caso, quando a intersecção dos alfabetos é vazia apenas na última posição da TPS, a complexidade é, no máximo, $O(n^2 f)$, onde n é o tamanho da TPS e f é o tamanho do conjunto *thresholds*. Na prática, o tamanho do conjunto *thresholds* é bem menor do que o tamanho da TPS, então é possível afirmar que a complexidade de tempo se aproxima de $O(n^2)$ como mostrado na Equação 3.1.

$$T(n) = T(n-1) + \Theta(n) \Longrightarrow \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2) \tag{3.1}$$

Na média, se a TPS é dividida na metade, a complexidade é $O(n)$, como mostrado na Equação 3.2.

$$T(n) = T(n/2) + \Theta(n) \Longrightarrow \sum_{i=1}^{log_2^n} \frac{n}{2^i} = n - 1 = O(n) \tag{3.2}$$

No melhor caso, a TPS é dividida na primeira posição, resultando em uma complexidade de $O(n)$ como mostrado na Equação 3.3.

$$T(n) = T(n-1) + \Theta(1) \Longrightarrow \sum_{i=1}^{n} 1 = n = O(n) \tag{3.3}$$

Em situações reais, como as que ocorreram na avaliação empírica do algoritmo, as sequências são divididas aproximadamente quatro ou cinco vezes até que não seja mais possível dividi-las. Portanto, para situações reais, é possível afirmar que o algoritmo possui complexidade

de tempo $O(in)$, onde n é o tamanho da TPS e i é o número de vezes que a sequência é dividida, o qual podemos considerar como sendo uma constante (sem impacto assintótico) e, sendo assim, afirmar que a complexidade de tempo é $O(n)$.

4 RESULTADOS EXPERIMENTAIS

Neste capítulo, são descritos e discutidos os resultados dos experimentos e sua realização. As Seções 4.1 e 4.2 descrevem o conjunto de dados utilizado nos testes e a maneira como estes testes foram realizados e medidos; a Seção 4.3 utiliza um dos casos de teste para exemplificar e esclarecer como os dados obtidos, através dos experimentos, foram compilados na Tabela 2; a Seção 4.4 apresenta os dados propriamente ditos e; na Seção 4.5 são discutidos os resultados obtidos.

4.1 *SETUP* DOS EXPERIMENTOS

Os experimentos foram realizados em páginas da *web*, em HTML, de *sites* que apresentam conteúdo estruturado. São páginas de diversos domínios, de empresas/instituições que são referência em seus segmentos de atuação. O conjunto de dados foi previamente salvo em disco para acesso *offline*.

Para medir a eficácia do algoritmo foram utilizadas as medidas padrão de avaliação de qualidade em *IR*: precisão (*precision*), revocação (*recall*) e *F-measure*. A medida *F-measure* é importante para evidenciar que, mesmo em situações com diminuição de *recall*, ainda assim a qualidade total da extração apresenta ganho, devido aumento maior na precisão.

Além das medidas padrão, os experimentos foram avaliados, também, quanto ao percentual de ruído removido, pois o algoritmo proposto pode ser combinado em série com outros algoritmos de extração, servindo como etapa de otimização, pois evitaria o processamento do ruído. Essas métricas são particularmente relevantes em situações em que o algoritmo de extração possua complexidade quadrática ou superior.

O ruído removido é apresentado sob duas perspectivas: percentual de ruído podado da árvore DOM (coluna 'Red.' da Tabela 2), que é simplesmente a quantidade do nós retirados da árvore DOM original, e percentual de registros espúrios removidos (coluna 'Ruído rem.' da Tabela 2), calculado conforme a Equação 4.1. As duas métricas são detalhadas na Seção 4.3.

$$NoiseRemoved = 1 - \frac{\#Rec_{totalTPS} - \#Rec_{targetTPS}}{\#Rec_{total} - \#Rec_{target}} \tag{4.1}$$

$\#Rec_{total}$ e $\#Rec_{target}$ são o total de registros e o total de registros de interesse, respectivamente, da página original, e $\#Rec_{totalTPS}$ e $\#Rec_{targetTPS}$ são o total de registros e o total de registros de interesse, respectivamente, da página *web* filtrada.

4.2 METODOLOGIA DE AVALIAÇÃO DOS RESULTADOS

O principal objetivo do algoritmo proposto é eliminar ruídos em páginas *web*, a fim de evitar que algoritmos que precisem processar estas páginas possam trabalhar apenas com conteúdo útil. Assim, para testar a efetiva eliminação de ruídos, foi utilizado o algoritmo MDR (LIU; GROSSMAN; ZHAI, 2003), que é uma conhecida técnica de extração estruturada. O resultado da extração pura (apenas MDR) será considerado como *baseline* para comparação com o resultado da extração combinada com a técnica proposta de remoção de ruído, como ilustrado na Figura 5.

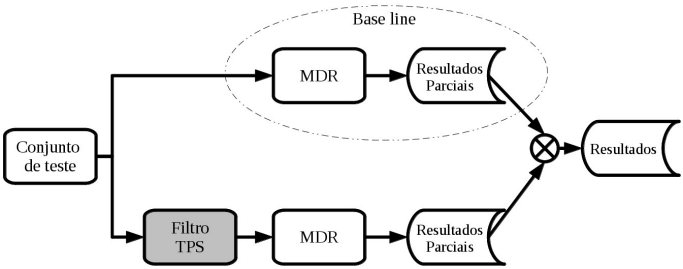

Figura 5 – Método de avaliação adotado.

A avaliação dos resultados é constituída das seguintes etapas:

- escolha de uma técnica de extração automática de dados estruturados;
- definição de um conjunto de páginas para teste;
- aplicação da técnica de extração no conjunto de teste, documentando os resultados obtidos como *target* (registro de interesse) ou *noise* (ruído);
- nova aplicação da técnica de extração no mesmo conjunto de teste, mas desta vez filtrando-o antes, utilizando a técnica proposta neste trabalho, documentando os resultados da mesma maneira;

- comparação de ambos os resultados e medição do aumento/-redução da precisão (remoção de ruído);
- verificação da hipótese nula H_0, que seria: *"a utilização do algoritmo de eliminação de ruído não implica em aumento na qualidade da extração"*. E da hipótese alternativa H_1, que é a negação de H_0.

Para primeira etapa foi escolhido o MDR para realizar a extração, pois esta técnica possui diversas implementações disponíveis, vários trabalhos publicados a respeito, funciona em uma única página e os resultados da extração são razoáveis para avaliar a técnica proposta neste trabalho. Outras técnicas poderiam ser utilizadas, se estivessem disponíveis, e não necessitem de várias páginas HTML como entrada (*e.g. RoadRunner* (CRESCENZI et al., 2001)).

4.3 EXEMPLIFICAÇÃO DOS EXPERIMENTOS

A seguir um dos experimentos realizado é detalhado para exemplificar como os resultados estão compilados na Tabela 2.

Quando ambas abordagens (MDR e filtragem TPS + MDR) são aplicadas a uma página de busca com 20 resultados do *web site* "YouTube" (#51), os seguintes resultados são obtidos:

- página *web* não filtrada (*i.e.* página original)
  - árvore DOM processada: 1424 nós;
  - resultados do MDR: 82 registros no total (62 ruído/20 alvo).
- página *web* filtrada (*i.e.* página após aplicação do filtro da TPS)
  - árvore DOM processada: 674 nós, tamanho $47,33\%$ da página original, redução de $-52,67\%$;
  - resultados do MDR: 20 registros no total (0 ruído/20 alvo), ruído removido 100%.

Neste resultado observa-se uma melhora na extração dos registros, assim como uma redução considerável do tamanho da árvore DOM que precisou ser processada. Um percentual de 52.67% da árvore DOM foi podado sem que ocorresse a perda dos registros de interesse durante o processo. Tudo o que foi podado da árvore, neste caso, era ruído. A Figura 6 ilustra a região principal da página e a Figura 7 mostra o

gráfico da respectiva TPS e a região principal detectada pelo algoritmo proposto.

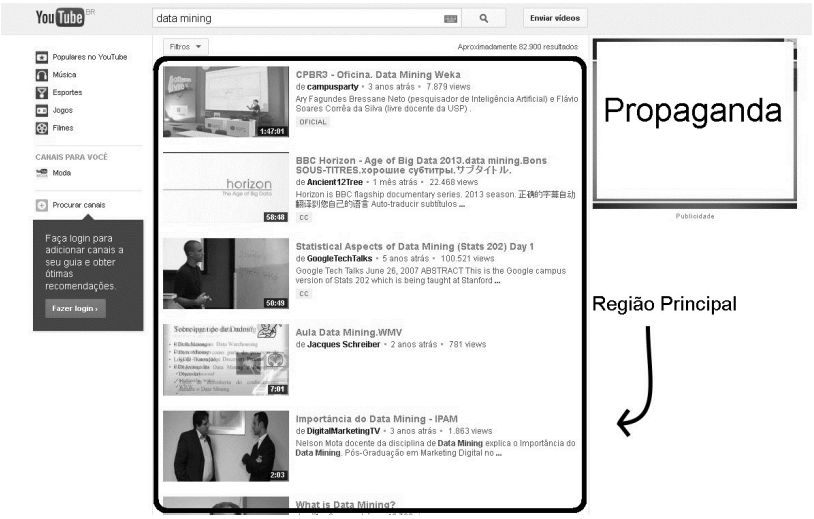

Figura 6 – Um página de resultados do *site* YouTube com sua região principal delimitada.

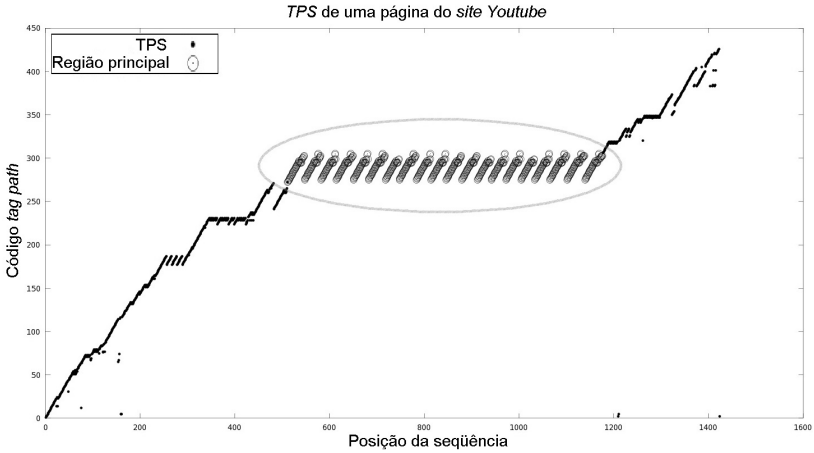

Figura 7 – Gráfico da TPS "código TP vs posição" de uma página do *site* YouTube com a região principal que foi detectada pelo algoritmo.

Sem aplicar o filtro na TPS, são obtidos 82 registros no total e, como sabe-se de antemão que existem 20 registros de interesse nesta página, considera-se que a quantidade de 62 registros representam 100% do ruído presente nesta página e que deve ser removido. Quando o filtro é utilizado, desta vez são obtidos apenas os 20 registros de interesse na fase de extração, resultando em uma precisão de 100%, o que significa que todo o ruído da página foi removido, neste caso.

4.4 RESULTADOS

Na Tabela 2, são apresentados, nas três primeiras colunas, o tamanho da árvore DOM processada pelo MDR (*i.e.* original) e a redução obtida após a remoção do ruído (*i.e.* podado). A coluna "Região principal" indica se o processo de remoção de ruído preservou a região principal ou não. As próximas quatro colunas são os resultados obtidos com o MDR sem e com remoção de ruído respectivamente, exibindo o total geral de registros extraídos e o total de registros de interesse (*target*) extraídos para ambas abordagens. A última coluna mostra o percentual de ruído removido, calculado conforme a Equação 4.1.

Como se pode observar na Tabela 2, o total da coluna "Conteúdo principal" indica que o algoritmo funcionou em 82,35% dos casos e removeu, com este conjunto de teste, uma média de 88,86% de todo o ruído presente nos dados, como mostra a média da coluna "Ruído rem.".

A média de 50,18% de redução no tamanho da árvore DOM é um resultado interessante. Primeiro, por que significa que praticamente metade da árvore DOM é ruído, na média. Segundo, por que este valor ficou bem próximo do valor relatado, de forma independente, em Gibson, Punera e Tomkins (2005) como sendo o tamanho do "*template*" das páginas (em torno de 50%).

Uma situação que chama a atenção na Tabela 2 é o resultado obtido para a página do *site* "*Build*" (#10). Sem filtrar o ruído, o MDR retornou um total de 117 registros, incluindo os 12 registros de interesse. Após aplicar o filtro de ruído, um total de 69 registros são retornados, nenhum deles de interesse, todos ruído. Então, após a filtragem, se a árvore retornada fosse, na realidade, o seu complemento, o resultado seria de 48 registros no total ($117 - 69 = 48$), incluindo os 12 registros de interesse, o que seria um bom resultado já que significa uma remoção de 65,71% do ruído. O que se pode deduzir disto é que, a segmentação da página ocorreu corretamente, apenas a identificação da

região principal falhou, já que neste caso, era relativamente pequena.

Tabela 2 – Resultados

						MDR (# registros)				(eq. 4.1)
		Tam. DOM (# nós)			Região	Orig.		Podado		Ruído
#	*Site*	Orig.	Podado	Red.	principal	Tot	Tgt	Tot	Tgt	rem.
1	ACM	601	340	-43,43%	Sim	61	10	16	10	88,24%
2	Amazon	3499	1069	-69,45%	Sim	344	16	41	16	92,38%
3	Apple	1422	962	-32,35%	Sim	63	15	23	15	83,33%
4	Avon	911	575	-36,88%	Sim	69	20	29	20	81,63%
5	Barnes & Noble	1242	778	-37,36%	Sim	147	30	54	30	79,49%
6	Bestbuy	3632	1425	-60,77%	Sim	299	15	15	15	100,00%
7	Blockbuster	2176	1381	-36,53%	Sim	84	25	32	25	88,14%
8	Bondfaro	3897	1820	-53,30%	Sim	160	28	30	28	98,48%
9	Bradesco	1913	1113	-41,82%	Sim	164	10	93	10	46,10%
10	Build	2712	746	-72,49%	Não	117	12	69	0	N/A
11	Buscapé	3608	1607	-55,46%	Sim	223	24	24	24	100,00%
12	Costco	4504	326	-92,76%	Não	246	96	8	8	N/A
13	Dell	1737	881	-49,28%	Sim	177	12	64	12	68,48%
14	Disney	2778	2006	-27,79%	Sim	193	96	116	96	79,38%
15	Drugstore	1774	850	-52,09%	Sim	156	18	52	18	75,36%
16	eBay	2623	1801	-31,34%	Sim	162	50	50	50	100,00%
17	Elsevier	906	160	-82,34%	Sim	120	10	32	10	80,00%
18	Footlocker	2440	1106	-54,67%	Sim	238	60	60	60	100,00%
19	Gamestop	1947	935	-51,98%	Não	86	12	6	0	N/A
20	Gap	2249	1365	-39,31%	Sim	235	124	126	124	98,20%
21	Globo	400	193	-51,75%	Sim	80	10	20	10	85,71%
22	Globo G1	900	619	-31,22%	Não	225	10	202	0	N/A
23	Google	1421	474	-66,64%	Sim	118	11	21	11	90,65%
24	Home Depot	5199	1304	-74,92%	Sim	325	24	24	24	100,00%
25	HP	1783	1258	-29,44%	Sim	71	15	15	15	100,00%
26	Itaú	1111	410	-63,10%	Não	77	10	11	0	N/A
27	Lojas Americanas	2660	710	-73,31%	Sim	211	20	20	20	100,00%
28	Macy's	5676	2158	-61,98%	Não	164	40	43	0	N/A
29	Magazine Luiza	3167	1115	-64,79%	Sim	314	40	44	40	98,54%
30	Mercadolivre	2401	1771	-26,24%	Sim	136	50	52	50	97,67%
31	Microsoft	871	272	-68,77%	Sim	57	16	16	16	100,00%
32	Newegg	8481	3419	-59,69%	Não	965	20	307	0	N/A
33	Nike	4082	1829	-55,19%	Sim	329	23	23	23	100,00%
34	Office Depot	3363	1111	-66,96%	Sim	108	24	38	24	83,33%
35	PC Mall	4285	2548	-40,54%	Sim	216	25	53	25	85,34%
36	Rakuten	3386	2768	-18,25%	Sim	112	35	61	35	66,23%
37	Ralph Lauren	995	564	-43,32%	Sim	46	12	12	12	100,00%
38	Reuters	1202	198	-83,53%	Sim	136	10	10	10	100,00%
39	Scopus	4929	4688	-4,89%	Sim	114	20	75	20	41,49%
40	Sears	5726	3890	-32,06%	Sim	397	50	75	50	92,80%
41	Sephora	3022	1440	-52,35%	Sim	365	60	60	60	100,00%
42	Sony	2200	1316	-40,18%	Sim	98	15	18	15	96,39%
43	Staples	2959	1611	-45,56%	Sim	178	24	24	24	100,00%
44	Submarino	2389	1268	-46,92%	Sim	116	20	22	20	97,92%
45	Terra	869	588	-32,34%	Sim	122	50	76	50	63,89%
46	Tiffany	3899	2753	-29,39%	Não	183	12	67	0	N/A
47	Valor Econômico	514	126	-75,49%	Não	55	10	2	0	N/A
48	Wal-Mart	1576	808	-48,73%	Sim	110	20	40	20	77,78%
49	Webmotors	2119	1361	-35,77%	Sim	113	14	19	14	94,95%
50	Yahoo!	760	290	-61,84%	Sim	67	10	10	10	100,00%
51	YouTube	1424	674	-52,67%	Sim	82	20	20	20	100,00%
	Média/Total			-50,18%	82,35%					88,86%
					17,65%					

Como consequência da remoção de ruído, podemos observar um aumento considerável na precisão média da extração, como relatado na Tabela 3. A redução no *recall* corresponde às situações onde o filtro falhou, as quais são discutidas e tratadas na Seção 4.5, e, apesar desta redução, a medida F apresenta melhora significativa, o que significa que a hipótese nula H_0 foi refutada pelos resultados obtidos.

Tabela 3 – *Precision, recall e F-measure.*

	MDR (a)	MDR+filtro (b)	Variação ($b - a$)
Precision	18,93%	75,08%	56,15%
Recall	100,00%	82,35%	-17,65%
F-measure	31,83%	78,55%	46,72%

4.5 DISCUSSÃO DOS RESULTADOS

Existem três situações principais onde o algoritmo poderia ser melhorado e, destas, apenas duas podem levar a perda de conteúdo de interesse. Na Tabela 2, coluna "Conteúdo principal", estas duas situações são responsáveis por 17,65% dos casos, onde o conteúdo principal foi removido durante o processo de filtragem.

1. **páginas com *templates* muito homogêneos**. São páginas com pouca diferença entre as regiões. Neste caso, utilizando a técnica proposta, não resta muito a fazer, pois simplesmente não há informação suficiente para uma diferenciação entre as regiões da página, já que todas se parecem. Não são perdidos registros de interesse, pois como a página inteira se parece (*i.e.* toda sua TPS é composta pelos mesmo símbolos), não será possível segmentar a sequência. Entretanto, a quantidade total de ruído removido é baixa;

2. **páginas com *templates* muito heterogêneos**. São páginas onde o conteúdo principal é subdividido em mais de uma região. Neste caso, a região principal acaba por ser dividida várias vezes durante o processo de filtragem e apenas a parte maior acaba passando pelo filtro (e ainda assim, pode ser que seja ruído). Mais a frente é proposta uma forma de contornar este problema;

3. **páginas com região principal menor que o restante**. Este problema é uma consequência da segunda hipótese levantada na Seção 3.3: "a região principal é mais densa, ou maior, que as demais regiões". Neste caso, ruído sempre será retornado no lugar do conteúdo principal. A mesma proposta apresentada para situação anterior também pode ser utilizada neste caso.

No caso de *templates* heterogêneos, a filtragem da TPS ainda pode ser utilizada realizando algumas modificações no algoritmo. Um exemplo de *template* heterogêneo são os *sites* de notícias, onde cada registro de interesse tem uma estrutura diferente, mas todos pertencem ao mesmo domínio (*i.e.* pertencem à mesma entidade). Nesta situação específica, a segmentação da TPS ainda pode ser utilizada para segmentar a página em diferentes regiões, e uma abordagem semântica pode ser usada para combinar as várias regiões de interesse, retornando um conjunto de regiões como resultado e não apenas uma única região.

Para a situação descrita para o *site* "*Build*" (que ocorreu, também, em outros oito *sites*, ou seja, em 17,65% dos casos), quando

a região principal é menor que as demais, uma abordagem semântica também pode ser utilizada para validar se a região encontrada, de fato, possui conteúdo de interesse e, se não possuir deve ser descartada e a árvore complementar (*i.e.* inverter a poda) deve ser retornada em seu lugar. Neste caso não é necessária nenhuma alteração no algoritmos apresentados, apenas no Algoritmo 1 entre as Linhas 4 e 5 (depois de encontrar a região, mas antes de podar a árvore DOM), bastaria verificar se a região resultante é, de fato, a região principal e, caso contrário, realizar a poda da árvore DOM com o complemento da sequência, garantindo assim a presença da região principal na árvore DOM resultante. Técnicas como a apresentada em Gibson, Wellner e Lubar (2007), que classifica uma sequência de blocos como sendo conteúdo ou não, pode ser utilizada para implementar esta verificação.

5 CONCLUSÕES E TRABALHOS FUTUROS

Como mostrado nos resultados, o algoritmo proposto para segmentação da página e remoção do ruído foi bastante efetivo em várias páginas comerciais/institucionais. Na maioria dos casos, uma grande quantidade de ruído foi removido sem comprometer os registros de interesse. Também, quando aplicada em conjunto com o MDR, foi observado um incremento considerável na precisão da extração e na qualidade geral da extração (*e.g.* aumento do *F-measure* de 31,83% para 78,55%).

Nas situações em que o algoritmo proposto atua aquém das expectativas, outras técnicas podem ser combinadas dependendo da aplicação. Em casos extremos, onde a página possuir estrutura muito homogênea (de forma que não seja possível encontrar as diferentes regiões) ou muito heterogênea (onde o conteúdo principal está espalhado em diversas regiões diferentes), o conteúdo principal poderia, talvez, ser detectado através do uso de uma técnica semântica (*e.g.* Gibson, Wellner e Lubar (2007)).

O algoritmo mostrou excelente desempenho, pois apresentou resultados satisfatórios para a maioria dos *sites* comerciais nos quais foi testado. Ele também supera as limitações (necessidade de treinamento, dependência de *tags* HTML, anotações manuais, entre outras) de trabalhos anteriores na área de limpeza, remoção de ruído e segmentação de página, como mencionado no Capítulo 2.

Uma sugestão para trabalho futuro seria encontrar maneiras de adaptar o algoritmo para melhorar seus resultados nas seguintes situações:

- páginas que apresentem estrutura muito heterogênea: encontrar uma forma de agrupar as várias regiões principais em uma só;
- páginas que apresentem estrutura muito homogênea: encontrar uma forma de diferenciar o ruído da região principal;
- páginas onde a região principal tem tamanho inferior ao ruído total: identificar a região principal através de outros atributos que não o tamanho.

Além disto, seria interessante realizar uma quantidade maior de testes, para consolidar os resultados já obtidos. Inclusive utilizando técnicas diversas de extração estruturada.

Este trabalho também foi publicado no *JIDM - Journal of Information and Data Management* no ano de 2013, como artigo completo (VELLOSO; DORNELES, 2013), e os algoritmos descritos neste trabalho foram implementados nas linguagens *C++* e *M* (MatLab/Octave) e estão disponíveis para acesso.

REFERÊNCIAS

CAI, D. et al. Extracting content structure for web pages based on visual representation. In: **Web Technologies and Applications**. [S.l.]: Springer, 2003. p. 406–417.

CHAKRABARTI, D.; KUMAR, R.; PUNERA, K. A graph-theoretic approach to webpage segmentation. In: ACM. **Proceedings of the 17th international conference on World Wide Web**. [S.l.], 2008. p. 377–386.

CHO, W.-T.; LIN, Y.-M.; KAO, H.-Y. Entropy-based visual tree evaluation on block extraction. In: IEEE COMPUTER SOCIETY. **Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01**. [S.l.], 2009. p. 580–583.

CRESCENZI, V. et al. Roadrunner: Towards automatic data extraction from large web sites. In: **VLDB**. [S.l.: s.n.], 2001. v. 1, p. 109–118.

FERNANDES, D. et al. Computing block importance for searching on web sites. In: ACM. **Proceedings of the sixteenth ACM conference on Conference on information and knowledge management**. [S.l.], 2007. p. 165–174.

FERNANDES, D. et al. A site oriented method for segmenting web pages. In: ACM. **Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval**. [S.l.], 2011. p. 215–224.

GIBSON, D.; PUNERA, K.; TOMKINS, A. The volume and evolution of web page templates. In: ACM. **Special interest tracks and posters of the 14th international conference on World Wide Web**. [S.l.], 2005. p. 830–839.

GIBSON, J.; WELLNER, B.; LUBAR, S. Adaptive web-page content identification. In: ACM. **Proceedings of the 9th annual ACM international workshop on Web information and data management**. [S.l.], 2007. p. 105–112.

KOHLSCHÜTTER, C.; FANKHAUSER, P.; NEJDL, W. Boilerplate detection using shallow text features. In: ACM. **Proceedings of the**

third ACM international conference on Web search and data mining. [S.l.], 2010. p. 441–450.

KOHLSCHÜTTER, C.; NEJDL, W. A densitometric approach to web page segmentation. In: ACM. **Proceedings of the 17th ACM conference on Information and knowledge management**. [S.l.], 2008. p. 1173–1182.

LIU, B.; CHEN-CHUAN-CHANG, K. Editorial: special issue on web content mining. **Acm Sigkdd explorations newsletter**, ACM, v. 6, n. 2, p. 1–4, 2004.

LIU, B.; GROSSMAN, R.; ZHAI, Y. Mining data records in web pages. In: ACM. **Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2003. p. 601–606.

LIU, W.; MENG, X.; MENG, W. Vide: A vision-based approach for deep web data extraction. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 22, n. 3, p. 447–460, 2010.

LUTZ, J. A.; HEUSER, C. A. Descoberta de ruído em páginas da web oculta através de uma abordagem de aprendizagem supervisionada. **Journal of Information and Data Management**, 2013.

MIAO, G. et al. Extracting data records from the web using tag path clustering. In: ACM. **Proceedings of the 18th international conference on World wide web**. [S.l.], 2009. p. 981–990.

QUINLAN, J. R. **C4.5: programs for machine learning**. [S.l.]: Morgan kaufmann, 1993.

REIS, D. d. C. et al. 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.

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.

SOUZA, A. F.; MELLO, R. S. **DeepEC: uma abordagem para extração e catalogação de conteúdo presenta na deep web**. Dissertação (Mestrado) — INE, Departamento de Informática e Estatística, UFSC, 2013.

VELLOSO, R. P.; DORNELES, C. F. Automatic web page segmentation and noise removal for structured extraction using tag path sequences. **Journal of Information and Data Management**, v. 4, n. 3, p. 173, 2013.

W3C. **Document Object Model (DOM)**. 2005. `http://www.w3.org/DOM/`. [Online; acesso em 27 de agosto de 2014].

WENINGER, T.; HSU, W. H.; HAN, J. Cetr: content extraction via tag ratios. In: ACM. **Proceedings of the 19th international conference on World wide web**. [S.l.], 2010. p. 971–980.

XIE, X. et al. Extracting data records from web using suffix tree. In: ACM. **Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics**. [S.l.], 2012. p. 12.

YI, L.; LIU, B.; LI, X. Eliminating noisy information in web pages for data mining. In: ACM. **Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2003. p. 296–305.

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.

ZHENG, X.; GU, Y.; LI, Y. Data extraction from web pages based on structural-semantic entropy. In: ACM. **Proceedings of the 21st international conference companion on World Wide Web**. [S.l.], 2012. p. 93–102.