EXTRAÇÃO DE DADOS NA WEB: TRANSFORMANDO LISTAS HTML EM FORMATO TABULAR

WILLIAM FELIPE MARX

EXTRAÇÃO DE DADOS NA WEB: TRANSFORMANDO LISTAS HTML EM FORMATO TABULAR

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Sergio Mergen

Alegrete

Marcco de 2013

Marx, William Felipe
Extração de dados na Web: Transformando listas HTML em formato tabular / William Felipe Marx. — Alegrete, 2013
xxii, 41 f. : il. ; 29cm

Trabalho de Conclusão de Curso — Universidade Federal do Pampa
Orientador: Sergio Mergen

1. Wrapper. 2. Integração de Dados. 3. Extração de Informação. I. Ttulo.

WILLIAM FELIPE MARX

EXTRAÇÃO DE DADOS NA WEB: TRANSFORMANDO LISTAS HTML EM FORMATO TABULAR

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 04 de março de 2013

Sergio Mergen
Orientador

Cleo Zanella Billa
UNIPAMPA

Fabio Kepler
UNIPAMPA

TCC.pdf

Agradecimentos

Ao término deste trabalho, deixo aqui meus sinceros agradecimentos:

* a Deus por tudo;
* a minha família, pelo incentivo e segurança que me passaram durante todo esse período;
* ao Prof. Dr. Sergio Mergen, por toda paciência, dedicação, em sua orientação;
* aos amigos da Ciência da Computação pelo agradável convívio durante o curso.

"Pensar é o trabalho mais difícil que existe. Talvez por isso tão poucos se dediquem a ele."

(Henry Ford)

Resumo

Dentro da linguagem HTML, existem construtores que, embora voltados primariamente para formatação visual, servem também para estruturar a informação. Um desses construtores são as listas. Verificando blocos de texto contidos em listas, é possível perceber uma divisão inicial que organiza a informação como uma coleção de registros. Dada essas características das listas HTML, surgiram trabalhos acadêmicos que visam transformar os blocos de dados contidos nas listas em tabelas de dados, compostas por linhas e colunas. Um dos trabalhos mais conceituados, chamado ListExtractor, depende de bases de informação preexistentes para realizar a transformação. Esse tipo de estruturação de fontes de dados é útil em pesquisas relacionadas à *dataspaces*, e tem aplicação direta em áreas como integração de dados, extração de conhecimento e recuperação de informação. Dentro deste contexto, o objetivo deste trabalho é a criação de uma séerie de regras estatísticas que visam transformar listas em tabelas. De modo geral, as regras se valem da presença e frequência no texto de caracteres especiais que costumam ser usados para realizar a separação de conteúdo, e não dependem de bases de conhecimento preexistentes. As regras propostas correspondem a extratores de dados, que podem ser incorporadas em arquiteturas de dataspaces. Os experimentos mostram o desempenho dos extratores criados na transformação de listas HTML reais recuperadas da Web. Também é feita uma comparação entre os extratores propostos e o ListExtractor.

Abstract

Some HTML constructs are used not only for visualization purposes but also to structure information. One of such structures is used to represent lists. Looking at the markup of HTML lists, it is possible to see that the information is organized as a collection of records. Given this, some works aim at transforming the text records into tables, composed by records and columns. One of the most known works, called ListExtractor, depends on existent knowledge bases in order to perform the transformation. This kind of data source structuring is useful in researches related to *dataspaces*, and have direct application in areas like data integration, knowledge extraction and information retrieval. In this context, the goal of this work is creation a series of statistical rules whose goal is to transform lists into tables. Generally speaking, the rules explore the presence and frequency of special characters inside the text to perform the content segmentation, and they do not rely on existent knowledge bases. The proposed rules correspond to data extractors, which can be incorporated in dataspaces architectures. Experiments show the extractors performance when applied to real HTML lists found on the Web. Additionally, a comparison is made between the extractors and the ListExtractor approach.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A produção de dados na *Web* vem crescendo exponencialmente, desde simples páginas HTML até formatos mais sofisticados como *feeds* RSS e *Web Services*. De certa forma, essas informações podem ser vistas como fazendo parte de uma gigantesca base de dados heterogênea e sem uma autoridade central que regula a estrutura dos dados e tampouco as políticas para adição e remoção de informações.

Indo um pouco mais além, é possível criar uma linha imaginária que divida essa gigantesca base de dados em bases menores, para os diferentes tipos de domínios existentes. Dessas bases menores surge um conceito que está em voga nos dias atuais: *dataspaces*. Em poucas palavras, um *dataspace* pode ser descrito como um conjunto de fontes de dados heterogêneas que atendem a um propósito comum [Franklin et al., 2005]. A possibilidade de acessar dados de um *dataspace* aumenta a qualidade das pesquisas, e tem aplicação direta em áreas como integração de dados, extração de conhecimento e recuperação de informação. Por exemplo, em [Mergen, 2011] é proposta uma arquitetura de dataspaces chamada EIDOS, cujo propósito é facilitar a recuperação de dados a partir de fontes heterogêneas que possuam sobreposição de conteúdo.

Um dos desafios a ser vencido para que *dataspaces* saiam do mundo das ideias e se tornem elementos tangíveis envolve reconhecer a estrutura presente nas fontes de dados, para que mais adiante um processo de integração possa consolidar esses dados em uma base unificada, seja ela virtual ou materializada. Esse problema se torna ainda mais desafiante quando a estrutura é implícita, oculta por detrás de padrões de documentação difíceis de ser identificados.

Esse é o caso, por exemplo, dos dados publicados nas páginas HTML. Em boa parte dos casos, o conteúdo das páginas HTML são textos escritos em linguagem natural, que por vezes são envoltos em *tags* de marcação, que mais tem o fim de definir a formatação visual do documento do que criar uma estrutura que organize o texto

em conceitos semânticos. Ou seja, informações descritas em páginas HTML tendem a ser pobres em estrutura. Ironicamente, esses documentos são a fonte mais rica de informações da *Web*, visto que a linguagem HTML é o padrão de facto para a publicação de dados na *Web*.

Dentro da linguagem HTML, existem construtores que, embora voltados primariamente para formatação visual, servem também para estruturar a informação. Um desses construtores são as listas. Verificando blocos de texto contidos em listas, é possível perceber uma divisão inicial que organiza a informação como uma coleção de registros. No entanto, cada registro por si só pode ser estruturado, dividindo o conteúdo em campos que representem informações distintas, mas que estejam associadas.

Dada essas características das listas HTML, surgiram trabalhos acadêmicos que visam transformar os blocos de dados contidos nas listas em tabelas de dados, compostas por linhas e colunas. Um desses trabalhos[Elmeleegy et al., 2009] em especial define processos a serem executados sobre uma lista, para que ao fim seja possível realizar a transformação. No entanto, os processos requerem o uso de bases de conhecimento para auxiliar na identificação das colunas contidas em cada linha.

Dentro deste contexto, este trabalho propõe uma série de regras estatísticas que visam transformar listas em tabelas. De modo geral, as regras se valem da frequência do número de ocorrências de caracteres que costuma realizar a separação de conteúdo nas listas. Com isso as regras independem da existência de bases de conhecimento.

A transformação de listas em tabelas também tem como motivação o fato de a arquitetura EIDOS suportar tabelas como fonte de dados interna, e permitir que sejam definidos módulos de extração capazes de transformar fontes de dados em algum formato qualquer em um formato tabular. Ou seja, pretende-se que as regras de extração propostas neste trabalho sejam aproveitadas para aumentar o número de fontes de dados disponibilizadas pelo dataspace gerenciado pela arquitetura EIDOS.

Este texto está organizado da seguinte forma:

No capítulo 2 são descritos conceitos relacionados ao trabalho, como mediadores, extratores e *wrappers*. Os conceitos são relevantes ao trabalho, uma vez que o mecanismo a ser criado pode ser tratado como um extrator ou até mesmo como um *wrapper*, e ser empregado dentro de um mediador. Além disso, a arquitetura EIDOS é apresentada, pois se trata de uma proposta de mediador em que o mecanismo de extração criado neste trabalho pode ser encaixado.

No capítulo 3 são descritos trabalhos relacionados à extração tabular a partir de listas. Dos trabalhos apresentados, é dada ênfase ao método proposto por[Elmeleegy et al., 2009], uma vez que uma versão reduzida e adaptada desse método será usada para fins de fazer uma comparação aproximada com os métodos propostos

neste trabalho. O estudo das características deste método ajuda a entender os casos em que seu uso é recomendado, e serve de motivação para a criação de novos mecanismos de extração que sejam complementares.

No capítulo 4 são apresentados os métodos de extração criados. Neste capítulo são explicados alguns conceitos necessários para a realização do trabalho, como a estrutura de listas HTML e o uso de caracteres de separação. No tocante aos métodos de extração criados, cada um deles é explicado através de exemplos de extração baseado em listas HTML ilustrativas.

No capítulo 5 são demonstrados os resultados obtidos na extração de dados a partir de listas HTML encontradas na *Web*. Os experimentos utilizam três coleções de listas distintas, que ajudam a medir o desempenho dos métodos de extração em circunstancias distintas. Também é realizada uma comparação com o método proposto por [Elmeleegy et al., 2009], de modo a identificar as vantagens de desvantagens de cada abordagem, e como elas podem ser melhoradas para que trabalhem em conjunto.

Por fim, o capítulo 6 traz algumas conclusões que se chegou neste trabalho sobre as regras propostas.

Capítulo 2

Fundamentos

2.1 Integração de Dados

A *Web* é uma enorme fonte de informação, mas ainda não totalmente explorada. Para obter um melhor aproveitamento das informações, e fazer aplicações mais completas, interessantes e interativas, é preciso integrar a informação de diversas fontes de dados. A dificuldade é saber a onde e quais são as informações mais relevantes, como agrupá-las satisfatoriamente para cruzar com informações relacionadas.

Para tentar amenizar o problema do mapeamento das informações da *Web* existe o conceito de Metadados, que "*são dados sobre dados*", ou seja informações sobre os dados, como assunto, tipo de formatação etc.

Então dentre os fatores que dificultam a integração de dados da *Web*, pode se ressaltar o grande número de fonte de dados, irregularidades na estrutura dos dados, alto grau de autonomia das páginas além da falta de metadados[Salgado & Lóscio, 2001]. Além de que informações na internet, ainda são mais voltadas para navegação por humanos, e não por sistemas automatizados computacionalmente [Crescenzi et al., 2001b].

A integração de dados e suas arquiteturas clássicas servem como um norte para ajudar na escolha da arquitetura para determinada solução. Cada solução depende muito das características do problema.

Em uma aplicação onde os dados precisam ficar disponíveis somente para leitura, com conteúdos de fontes diversas. Uma solução possível é, coletar todo o conteúdo das fontes e armazenar em um repositório específico. Então de tempos em tempos, essa informação precisa ser atualizada.

Neste exemplo já se utilizou alguns conceitos de integração de dados, como:

- Tipos de acesso: quando se diz que a informação fica disponível somente para leitura;

- Tipos de implementação: quando se opta por manter um repositório com cópias das informações originais.

Depende do domínio e de determinadas especificidades do problema, da fase inicial até o amadurecimento da solução, pode-se mudar bastante a arquitetura, como por exemplo, se a frequência de modificação dos dados originais for grande. Ou até pode existir várias soluções distintas, dependo da escolha do administrador.

2.1.1 Arquiteturas Clássicas

Existem na literatura algumas arquiteturas clássicas para integração de dados, abaixo iremos comentar sucintamente a arquiteturas de Mediadores.

Antes de começar é necessário entender como funcionam os esquemas de dados das fontes, que podem ser definidos da seguinte forma:

- Local: Esquema de dados da fonte local.

- De Exportação: Esquema que a fonte disponibiliza para camada superior acessar.

- Global: Faz a integração entre as fontes, possui um mapeamento entre os atributos globais e os de exportação das fontes locais.

2.1.1.1 Mediadores

Mediador 2.1 possui uma visão global sobre as fontes distribuídas. Ele possui um esquema global que faz um mapeamento pré-definido para os esquemas locais. Os mediadores podem ser classificados em horizontais ou verticais dependendo do intervalo do domínio. Os verticais são para um único domínio, com um número limitado de fontes para as quais é especializado. Já os horizontais abrangem muitos domínios diferentes, os quais acarretam um largo número de fontes [Mergen, 2011].

As estratégias de mapeamentos podem ser: *Global-As-View (GAV):* O esquema global possui uma visão sobre os esquemas de exportação das fontes ou *Local-As-View (LAV):* Os esquemas locais são definidos como visões sobre o sistema global depois que este é definido[Mergen, 2011].

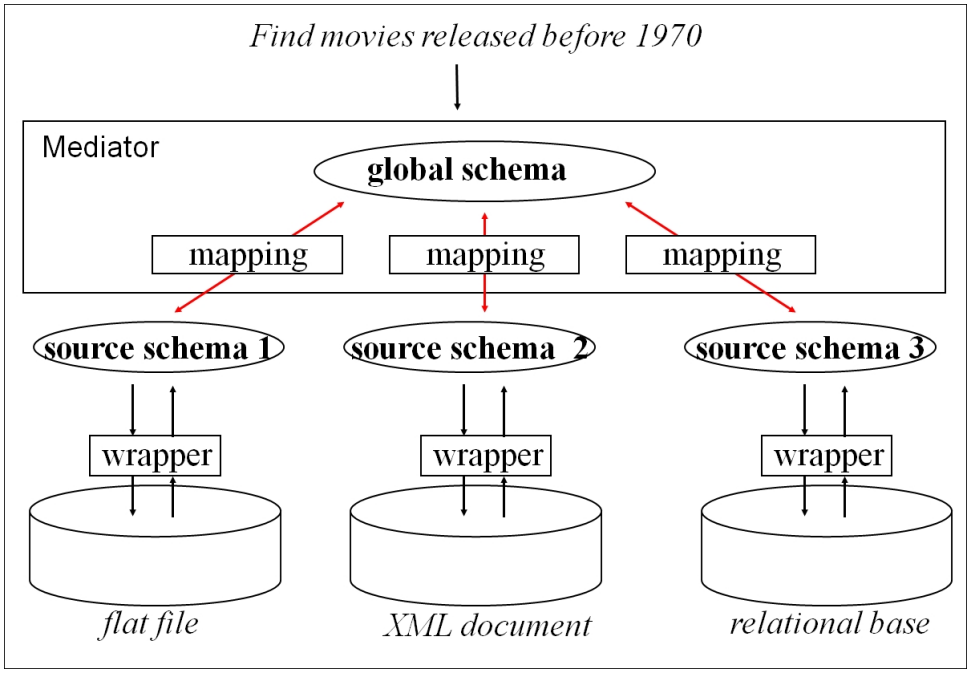


Figura 2.1: Exemplo de arquitetura usando mediador [Mergen, 2011]

2.1.1.2 Wrappers

Wrapper(invólucro) no contexto de banco de dados, basicamente começou a ser usado com a função de transformar dados de sistemas legados para os formatos atualizados. *Wrapper* é um componente de integração de sistemas, que é responsável por extrair informações em algum formato específico de determinada fonte de dados, e devolver estas informações em um formato conhecido. O *Wrapper* geralmente acessa os dados diretamente devido a isso se encontra em uma camada mais baixa do *Software*. Ele provê uma interface de acesso uniforme permitindo retornar as informações que ele é designado a extrair.

Wrapper atua como uma camada de abstração entre o esquema global e a fonte. Permite integrar sistemas modelados para diferentes estruturas.

Então um sistema pode possuir vários *Wrappers*, cada um para extrair algum tipo de estrutura de informação. Assim os resultados de diversas fontes de dados heterogêneas podem se juntados e comparados etc.

Porém construir *Wrappers* manualmente é uma tarefa difícil e trabalhosa [Machanavajjhala et al., 2011], são frágeis e de difícil manutenção[Crescenzi et al., 2001b]. A construção deles pode ser tanto totalmente manual ou automática, não são fáceis de construir de forma genérica, devem ser robustos para identificar estruturas que não estão em total conformidade com seu

modelo.

A estrutura e ou a forma como uma informação se encontra em determinado momento pode mudar, sendo necessário criar um novo Wrapper para ela [Kayed et al., 2006]. Para amenizar um pouco esse problema utiliza-se os *Wrappers* de indução, os quais se propõem a serem mais genéricos e se adaptarem melhor a pequenas mudanças[[Kayed et al., 2006], [Cuenca et al.,]].

Wrappers de indução são projetados para gerar *Wrappers* que possuem vários graus de dependências como tipo de texto, domínio, cenário. Podem usar técnicas de aprendizado de máquina, mineração de dados, compreenção de mensagens, essas técnicas podem ajudar a maximizar a reusabilidade e diminuir os custos de manutenção[Kayed et al., 2006].

2.1.2 Heterogeneidade dos Dados

A diversidade dos dados da *Web* é enorme, [Mergen, 2011] lista algumas características que podem contribuir para a heterogeneidade dos dados:

- **Formato/modelo:** as informações podem estar codificadas em diferentes formatos ou modelos como JPEG, MPEG, WMV, PDF, DOC, ODT, CSV, XML etc.

- **Estrutura:** Os dados de um mesmo formato podem apresentar estruturas diferentes, por exemplo, em uma página HTML as informações dispostas em um parágrafo estão estruturadas diferentemente das de uma tabela.

- **Semântica:**

  - **Nomeação Heterogênea:** nomes homogêneos e sinônimos, onde um nome pode representar diferentes conceitos, ou diferentes nomes representarem o mesmo conceito.

  - **Estrutura Heterogênea:** relações entre conceitos que se modificam de uma fonte de dados para outra, como dois conceitos de diferentes esquemas com diferentes pais.

  - **Domínio Heterogêneo:** diferentes valores com possibilidade de instanciar um dado conceito.

2.1.3 Estrutura dos Dados

De acordo com a estrutura os dados pode ser classificados em 3 tipos [Barbosa et al., 2001]:

- **Dados Estruturados:** possuem estrutura ou esquemas de representação pré-definidos, como por exemplo, banco de dados relacionais.
- **Dados Semi-Estruturados:** possuem uma estrutura que pode variar, como páginas HTML e XML.
- **Dados Não-Estruturados:** sem estrutura pré-definida como imagens, vídeos e texto livre.

2.2 Arquitetura EIDOS

A arquitetura EIDOS foi feita para fazer consulta em dados estruturados, trata cada fonte de dados individual como uma relação de 3 entidades: relações, atributos e valores. As relações entre essas entidades são capturadas em índices especializados. Assim o mecanismo de reescrita consegue construir consultas com predicados baseados nos índices. Ela permite fazer consultas complexas em sua base de dados com base no conhecimento do usuário sobre o assunto, permitindo que ele refine os resultados a medida que vai conhecendo melhor o conteúdo das fontes.

Esta arquitetura não possui um mapeamento global sobre as fontes, as consultas são reescritas com base nos atributos da consulta e das fontes, com mecanismo de indexação *on-the-fly* permitindo integração em uma escala sem precedentes.

EIDOS também suporta a mescla de consultas estruturadas com palavras-chave. Por exemplo, se procurar por *select year from movie* junto com uma palavra chave. Somente os resultados que contenham a palavra chave serão devolvidos.

A arquitetura do EIDOS pode ser vista na figura 2.2 e está dividida em duas principais camadas. A primeira é a camada do mediador, é a onde são feitas as reescritas para a consulta do usuário, e também é responsável pela indexação do conteúdo extraído pelos *Wrappers*. A segunda é a camada de dados onde é feita a extração da informação pelos *Wrappers* diretamente nas fontes.

A camada do mediador contêm os seguintes Componentes:

- *Crawler*: procura por estruturas relevantes na *Web* e indexa no repositório EIDOS, e frequentemente procura por novas estruturas e atualiza os índices.

- *Query Processor*: Recebe consultas relacionais da camada de Reescrita, onde cada consulta é composta por um conjunto de relações, que podem provir de fontes diferentes. De acordo com as fontes de dados a consulta pode ser quebrada em sub-consultas e mais tarde juntadas para formar um resultado, similarmente a pesquisas em múltiplos banco de dados.

- *Publisher*: Semelhante ao *Crawler* a diferença é que ao invés de percorrer sites atrás de *links* o *Publissher* retorna conteúdo de serviços disponibilizados na internet, como repositórios etc.

- Camada de reescrita: O *Query Parser* recebe consultas e transforma para formato que o *Rewriter* possa ler, o qual é responsável por encontrar as consultas adequadas para executar nas fontes. Já o *Indexer* decide como gravar no *Index* as informações que recebe do *Crawler* e do *Publisher*.

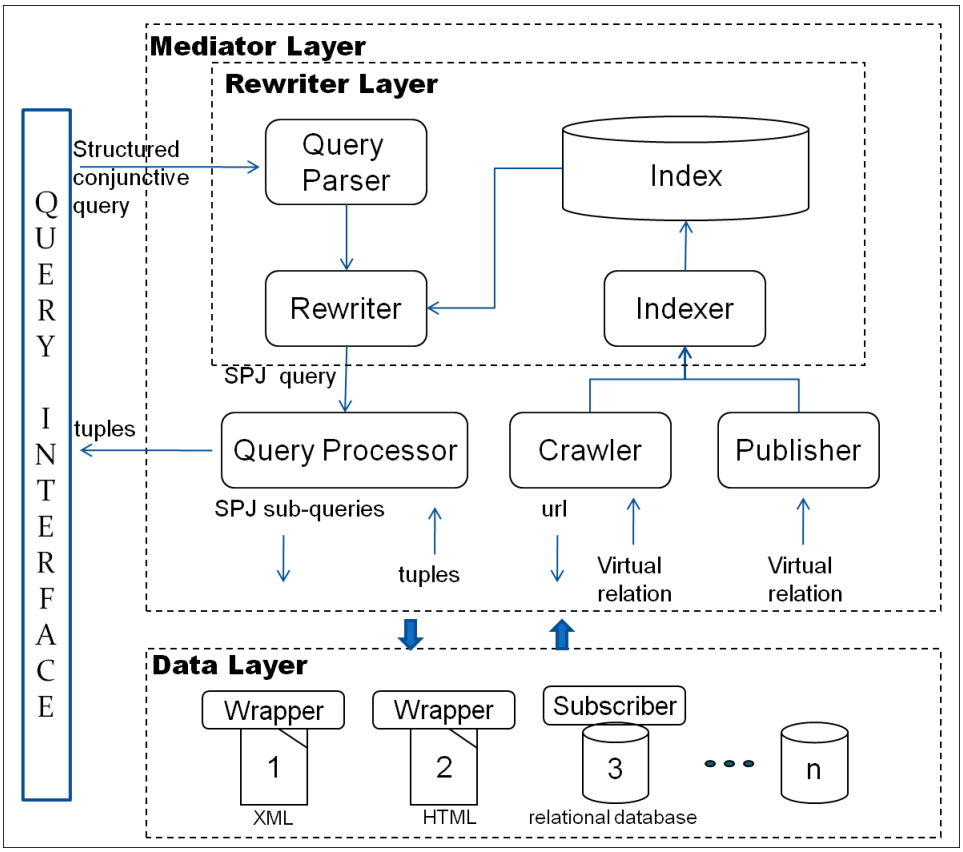


Figura 2.2: Arquitetura EIDOS [Mergen, 2011]

Capítulo 3

Trabalhos Relacionados

Existem diversos estudos a respeito de abordagens automatizadas que extraem listas de objetos a partir da *Web* [Arasu & Garcia-Molina, 2003, Crescenzi et al., 2001a]. O processo típico de extração consiste em três passos: extração de registros, extração de atributos e rotulação.

O primeiro passo envolve identificar os registros presentes no documento em análise. Em alguns tipos de documentos essa identificação é trivial, como em listas HTML, onde as próprias linhas já separam os registros. Já em páginas na *Web*, esses registros seriam segmentos no texto que encapsulam dados referentes a um objeto específico. Entre as técnicas usadas para realizar essa busca pode-se citar o uso de algoritmos de casamento de *strings* [Liu et al., 2003] e processamento de linguagem natural.

O segundo passo envolve extrair atributos do objeto identificado. Considerando objetos contendo dados de filme, esses atributos poderiam ser 'titulo' e 'ano', por exemplo. O último passo envolve interpretar os atributos identificados, ou o próprio objeto que os agrega, afim de rotulá-los com nomes apropriados. Normalmente, as soluções propostas para esse problema são baseadas modelos probabilísticos e aprendizado de máquina [Wang & Lochovsky, 2003, Zhu et al., 2006].

Os trabalhos mais próximos ao proposto nesta monografia são aqueles que realizam o segundo passo. Existem diversos trabalhos nesta categoria, como aqueles que analisam páginas de resposta de formulários *Web* para realizar a extração de atributos. Por exemplo, em [Zhao et al., 2007] é usado um método estatístico para minerar páginas de resposta na busca de dados que obedeçam *templates* pré-definidos. Já o trabalho de [Zhai & Liu, 2005] utiliza informação visual de como os dados seriam renderizados pelo navegador de *Internet* para inferir como esses dados estão estruturados.

Existem também estudos mais específicos, e ainda mais próximos a este trabalho,

cujo foco é a extração de atributos de listas estruturadas como em 3.1. As próximas seções descrevem essas abordagens, sendo que será dada ênfase ao trabalho proposto por [Elmeleegy et al., 2009], que se traduz em uma abordagem completa para extração tabular a partir de listas HTML.

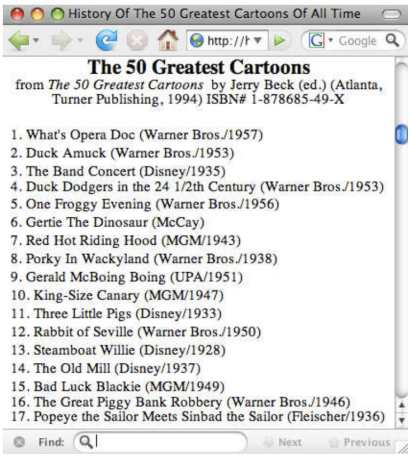


Figura 3.1: Aplicação da técnica do ListEsxtractor em uma lista de desenhos [Elmeleegy et al., 2009]

3.1 Collie

Collie[Machanavajjhala et al., 2011] é uma técnica de extração de listas coletivas, que busca entidades em comum em vários sites diferentes incrementando seu conteúdo. Ela extrai, entidades como livros, filmes, escolas etc. Estas entidades são retiradas de pequenos sites da internet. Nesta abordagem é partido do princípio que o conteúdo de uma lista pode complementar o de outra, e inclusive eliminar possíveis atributos com ruídos, visto que diferentes listas podem conter informações adicionais distintas, e inclusive dispor o conteúdo de formas diferentes possibilitando eliminar ruídos de etiquetagem e alinhamento de atributos.

A abordagem possui flexibilidade para trabalhar tanto com conjunto de dados HTML e não HTML, trabalhando também com dados com baixo sinal de separação dos atributos. Para quebrar e etiquetar os dados de um conjunto de dados ela usa um método não supervisionado[Álvarez et al., 2008] que cria pedaços de informação etiquetados, com etiquetas do tipo: nome, telefone, cidade, estado etc. Então com esses pedaços busca casa-los com *templates* do conjunto de sementes de entidades, já as sementes de entidades precisa ser constantemente supervisionada.

3.2 Automatic Data Extraction from List and Tables in Web Tables

Em [Lerman et al., 2001] é apresentada uma técnica de extração de lista e tabelas completamente automática que agrupa linhas e colunas. Para isso eles desenvolveram uma suíte de algoritmos de aprendizado não supervisionados os quais induzem a estrutura da lista explorando regularidades.

Basicamente o algoritmo funciona extaindo listas de um conjunto da páginas Web não etiquetadas, procurando encontrar um *template* em uma página e verificando se ele se repete nas outras páginas, para então identificar as listas em cada página.

Uma limitação da técnica é que ela necessita analisar muitas páginas antes de realizar a extração de uma única lista.

3.3 WWT

WWT [Gupta & Sarawagi, 2009] é uma abordagem que constrói tabelas a partir de várias listas com poucos exemplos de linhas. Aproveitando um enorme corpus de listas como subsídio para a extração, o método extrai as linhas usando modelos estatísticos, permitindo trabalhar com dados com bastante ruído.

O método trabalha sem necessidade de nenhuma intervenção humana, usando Campo Condicional Randômico (CRF), que usa muitas características de *templates* como ocorrências de palavras, dicionário de casamentos e padrões de expressão regular.

3.4 SoftMealy

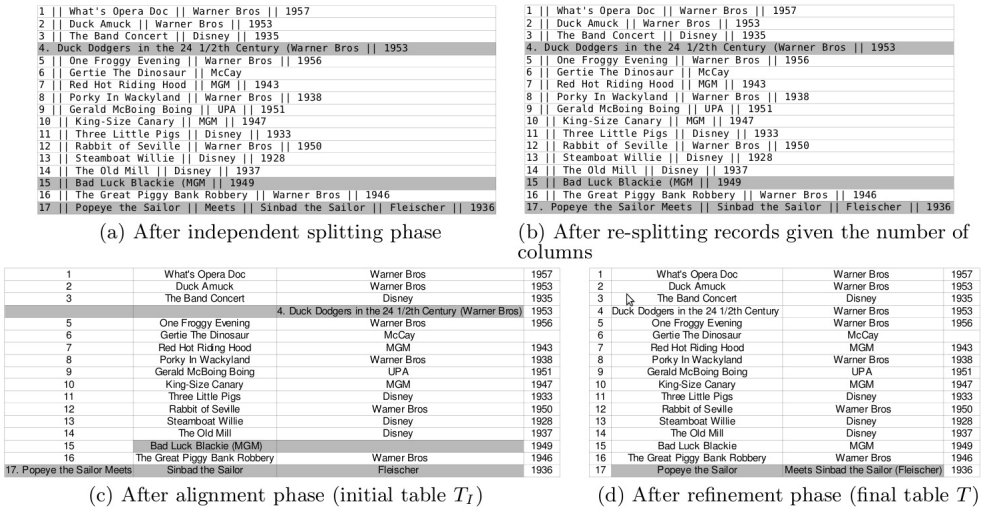
A abordagem do SoftMealy[Hsu & Dung, 1998] é um *wrapper* que usa tradução de estados finais que em inglês significa *Finite-State Transducers* (FST), o qual casa o contexto dos *tokens* de entrada separados por regras contextuais para determinada transição dos estados, assim codificando diferentes permutações de atributos. Para extrair um atributo o *wrapper* reconhece o separador ao redor do atributo. Um separador é uma linha de borda invisível entre dois *tokens* adjacentes.

As regras de contexto disjuntivo permitem ao *wrapper* caracterizar o contexto da separação. A entrada do algoritmo de generalização é um conjunto de tuplas etiquetadas que provêm da posição dos separadores e da permutação dos atributos. O algoritmo de aprendizagem suporta taxonomias de árvores de *tokens* que induzem maior precisão nas regras contextuais do que muitos exemplos manuais.

A limitação da abordagem é que ela não permite fazer generalização nos casos em que não existam separadores nas listas analisadas.

3.5 ListExtractor

O ListExtractor [Elmeleegy et al., 2009] é uma técnica para extração de listas da *Web* que separa linhas em colunas com domínio independente e completamente não-supervisionada, Uma aplicação do método pode ser vista na lista da Figura 3.2(a) com resultado final na Figura 3.2(d).



(a) After independent splitting phase

(b) After re-splitting records given the number of columns

(c) After alignment phase (initial table T_I)

(d) After refinement phase (final table T)

Figura 3.2: Aplicação da técnica do ListEsxtractor em uma lista de desenhos [Elmeleegy et al., 2009]

A técnica utiliza duas funções de pontuação, uma para mensurar a qualidade individual do campo, chamada *Field Quality Score* ($FQ(f)$) e uma função para mensurar a consistência dos campos vizinhos, chamada *Field-to-Field Consistence Score* ($F2FC(f1, f2)$).

A técnica possui três fases, que podem ser vistas na parte direita da Figura 3.3. O básico de cada uma dessas fases será explicada nas seções seguintes.

3.5.1 Splitting Independente (Independent Splitting Phase)

Cada linha de entrada é separada em múltiplos campos de registros, sendo que a quantidade de colunas de uma linha não precisa ser a mesma das demais linhas. Para fazer a melhor separação possível o artigo considera três métodos alternativos, sendo que o utilizado é o terceiro que apresentou melhores resultados em suas separações:

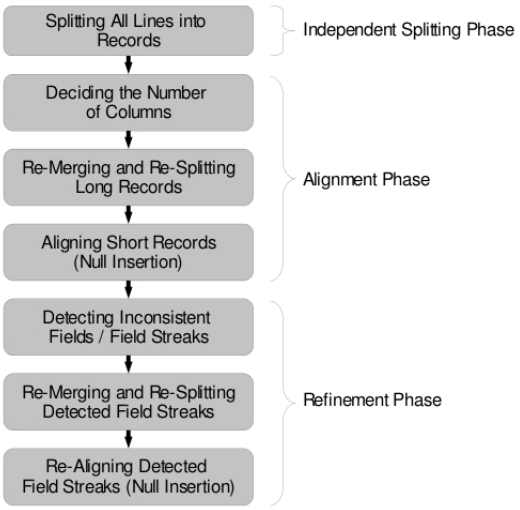


Figura 3.3: Sequência de Operações [Elmeleegy et al., 2009]

1. Maximização do somatório da pontuação FQ: geralmente faz separações agressivas já que aumentando número de campos conduz a um somatório maior. Pode ser implementado algoritmo de programação dinâmica. Em outros termos pode ser expressa como a soma das somas.

2. Maximização da Média FQ: evita divisões agressivas. Não pode usar programação dinâmica por que a média não pode se decompor na função objetivo. Não pode ser expressado na média das médias.

3. Método Guloso: Não é computacionalmente dispendioso e não resulta em separações agressivas. O método cria um *rank* C_f onde a lista de todos os candidatos aos campos são ordenados com *Field Quality Score* FQ decrescente. A cada interação o topo é removido e transformado em um segmento.

Field Quality Score: Dado um campo candidato f, a pontuação é computada baseada em três fontes de informação.

Suas pontuações são as seguintes:

1. ***Type Score*** $S_{ts}(f)$***:*** Reflete se um candidato pode ser reconhecido como um campo individual de uma coluna de uma tabela, reconhecendo valores numéricos como datas, urls, email e telefones.

2. ***Language Model Score*** $S_{lms}(f)$***:*** registra a probabilidade de ocorrer uma sequência de palavras, utilizando a probabilidade condicional de uma palavra w_i

seguir uma sequência $w_1, ..., w_{i-1}$. Ou seja, é calculado $P_r(w_i|w_1, ..., w_{i-1})$ através de um corpus de páginas *web*.

Para maximizar a probabilidade de uma sequência dentro de uma célula, eles utilizam as seguintes pontuações.

- *Pontuação de Coesão Interna* S_{ic}3.1: computa a probabilidade da média condicional de cada palavra em relação a anterior.

$$\mathrm{S_{ic}}(f) = \frac{\sum_{h=1}^{m-1} P_r(w_{i+h}|w_i, ..., w_{i+h-1})}{m-1} \tag{3.1}$$

- *Pontuação de Coesão Externa* S_{ei}3.2: Computa a média inversa da probabilidade dos limites dos campos. Na equação, $P_r(w_i|w_{i-1})$ é probabilidade da primeira palavra de f seguir a última palavra do campo anterior, e $P_r(w_{i+h+1}|w_{i+h})$ é probabilidade da primeira palavra do campo seguinte seguir a última palavra do campo f.

$$\mathrm{S_{ei}}(f) = \frac{2}{P_r(w_i|w_{i-1}) + P_r(w_{i+h+1}|w_{i+h})} \tag{3.2}$$

3. ***Table corpus Suport Score*** $S_{tcs}(f)$***:*** Reflete a pontuação do quão bem a tabela é suportada pelo corpus.

Para calcular FQ e S_{lms} a escala dos componentes S_{ts}, S_{ic}, S_{ei}, e S_{tcs} é normalizada, conforme a equação 3.3. Onde a_{ts}, a_{lms}, a_{tcs} são os pesos das fontes de informação.

$$FQ(f) = a_{ts} \times S_{ts}(f) + a_{lms} \times S_{lms}(f) + a_{tcs} \times S_{tcs}(f) \tag{3.3}$$

Um exemplo onde pode ser visto o resultado da fase de *Spliting* da lista 3.1 é na Figura 3.2(a).

3.5.2 Alinhamento (Alignment Phase)

É definido uma tabela inicial ($T1$) com o resultado da separação da primeira fase. Esta tabela é construída com um número mais comum K de colunas. Caso uma linha tiver mais colunas que a definida na tabela está linha será reconstruída e separada novamente pelo *SplitLine*, forçando a nova linha a ter k colunas, isso pode ser visto na Figura 3.2(b) onde a linha 17 foi forçada a possuir 4 campos.

O número k de colunas é determinado somente uma vez, e as linhas com mais do que k campos deverão ser separadas novamente, como mostrado na seção **??**. Já

as linhas com menos do que k campos deverão incluir espaços nulos, como mostrado a seção 3.5.2.2.

3.5.2.1 Alinhamento dos Registros Longos

Quando a linha tem mais que k campos, é necessário separá-las novamente. Para isso é empregada uma modificação do algoritmo *SplitLine* chamada *BoundedSplitLine* o qual tem um limiar k_{max}. Para ser incluído na linha, cada candidato f_{top} deve apresentar $min\_fields(r, f_{top})$ menor que k_{max}, onde $min\_fields(r, f_{top})$ é a soma dos campos já adicionados na linha r.

3.5.2.2 Alinhando Registros Curtos

Depois da nova separação dos registros com mais de k campos, restam ainda os que possuem menos do que k campos. Para esses casos, é necessário encontrar o campo faltante e inserir um valor nulo no lugar.

Para isso é utilizada uma adaptação do algoritmo de programação dinâmica de Needleman-Wunsch para computar a função objetivo, para o custo do melhor alinhamento possível.

$A[i, j]$ pode ser construída com o melhor alinhamento das subsequência de F_i e C_j. Pois $A[i, j]$ pode ser construído recursivamente considerando três possibilidades:

- f_i alinhado com c_j e o resto seguindo a mesma ordem $A[_{i-1,j-1}]$
- f_i é incomparável e o resto do alinhamento segue a ordem $A[_{i-1,j}]$
- c_j é incomparável e o resto do alinhamento segue a ordem $A[_{i,j-1}]$
- $UnMatched(c_j)$ é o custo atribuído para coluna não correspondente c_j com qualquer campo.
- $UnMatched(f_i)$ é o custo atribuído para o campo não correspondente f_i com qualquer coluna.
- $Matched(f_i, c_j)$ é o custo atribuído para alinhar f_i com a coluna c_j.

Depois dessas etapas pode-se então construir a tabela T presente na Figura 3.2(c) onde pode se perceber que na linha 6 desta figura no campo referente ao ano foi colocado um campo vazio, já nas linhas 4, 15, 17 embora foram mal separadas os anos que ficaram em campos individuais foram devidamente alinhados na ultima coluna.

3.5.3 Refinamento (Refinement Phase)

Analisa os campos e tenta detectar campos incorretos de acordo com os outros campos da mesma coluna. Separações inconsistentes não ocorrem isoladamente, elas ocorrem afetando os campos vizinhos. Quando é detectado um campo inconsistente essa linha é reagrupada e dividida novamente, já tomando em consideração as características da coluna na qual ela se encontra. Na Figura 3.2(d) podemos ver os resultados depois dessa fase onde as linhas 4 e 15 foram corrigidas e a linha 17 ainda apresentou erro no nome do desenho e no da produtora.

Capítulo 4

Extração de Listas Web

Este capítulo apresenta a abordagem utilizada para realizar a extração de dados a partir de listas HTML. As seções seguintes apresentam definições úteis para o entendimento do trabalho. A seguir, os métodos de extração propostos serão apresentados.

4.1 Listas HTML

As listas reconhecidas pelo módulo de extração implementado estão em formato HTML(*HiperText Markup Language*), que é um padrão de publicação da *Web* é usada para enriquecer conteúdo textual, inserindo marcações e atributos que indicam como o conteúdo está estruturado para ser lido e interpretado em um navegador.

As listas HTML podem ser ordenadas, não ordenadas, ou de definições. Além de poder ser aninhadas umas dentro das outras. Para cada uma dessas variações, são usados conjuntos de *tags* diferentes.

As *tags* que definem uma lista ordenada são <ol> e <li>, onde <ol> ... delimitam o início e o fim da lista, e seus itens são definidos pela *tag* . Já nas listas não ordenadas seus itens não possuem uma ordem pré-definida. As *tags* que definem as listas não ordenadas são <ul> e <li>, onde <ul> ... delimitam o início e o fim da lista, e seus itens são definidos pela *tag* . Um exemplo de uma lista não ordenada pode ser visto na Figura 4.1.

Nas listas ordenadas seus itens aparecem numerados por um marcador, esse marcador pode ser alterado podendo ser tanto numérico, alfanumérico além da possibilidade de inverter a ordem de crescente para decrescente.

Neste trabalho serão consideradas apenas listas ordenadas com a *tag* "$< ol >$"e não ordenadas com a *tag* "$< ul >$". Os possíveis aninhamentos são ignorados.

Figura 4.1: Lista com atuação do Pele pela Seleção Brasileira.

4.2 Parsing HTML

Para analisar o conteúdo de uma página *Web*, é conveniente decompor sua estrutura e separar o que se quer utilizar. Para isso é convencional que se use um componente funcional de *software* denominado *parser*. Um *parser* serve para analisar os componentes de uma estrutura e decompô-los em estruturas menores, seguindo algumas regras gramaticais. Essa análise é feita a partir da classificação léxica dos lexemas em *tokens*(palavras chaves, *tags*) que servem para determinar a estrutura da página. Para fazer essa análise o *parser* geralmente cria uma árvore com um mapeamento dos símbolos da gramática.

Algumas vezes as páginas *Web* contém alguns erros de codificação que requerem que os *parsers* a serem mais flexíveis e robustos identificando possíveis erros. Existem muitos *parsers* para verificar a validade dessas estruturas corrigindo possíveis erros de codificação. Neste trabalho usamos o HTMLCleaner para acessar os elementos da página.

HtmlCleaner é um parser HTML, open-source, escrito em Java. Ele disponibiliza um conjunto de regras para filtragem e extração de dados das página. Como casualmente existem páginas HTML malformadas inadequadas para uso direto, por isso é necessário limpar o código HTML, encontrando a correspondência ente as tags. O HtmlCleaner reordena as elementos individuais da página e produz XML bem-formado.

Para a manipulação dos elementos da página, o HtmlCleaner cria uma estrutura de árvore DOM (*Document Object Model*). DOM é um padrão definido pela *W3C* que representa documentos HTML, XHTML, e XML em uma estrutura com objetos organizados de forma hierárquica. Um exemplo de árvore DOM é apresentado na figura 4.2.

A plataforma DOM possui três níveis: o Core DOM para qualquer documento, o HTML DOM para documentos HTML e o XML DOM para documentos XML. Para o

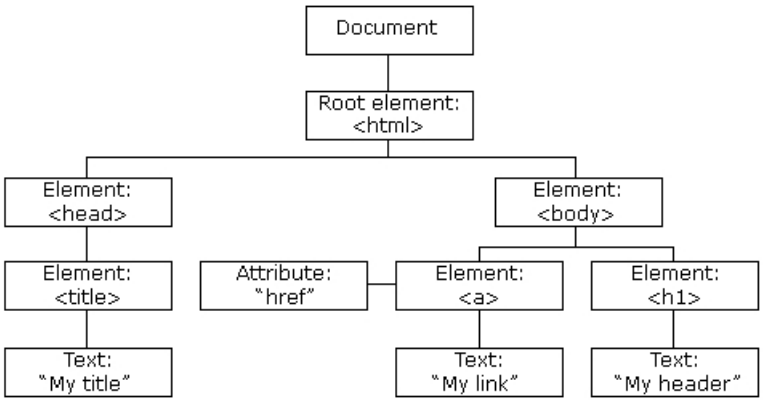


Figura 4.2: Exemplo Árvore DOM

DOM, tudo no documento são nós, como comentários, elementos e atributos. A partir de uma árvore DOM é possível acessar os nós internos, navegando pela estrutura do documento e recuperando os nós de interesse. Para este trabalho de conclusão de curso, os nós de interesse são os elementos cujo nome seja ul, ol e li.

4.2.0.1 Conjunto de Caracteres Separadores

As regras de extração propostas neste trabalho se baseiam na noção de que existem caracteres específicos que costumam ser utilizados para fazer a estruturação de conteúdo em páginas *Web*. A Figura 4.3 apresenta um exemplo de lista HTML[1] com caracteres separadores bem definidos, os quais garantem a estruturação da informação.

Ao analisar o exemplo se percebe o uso do ponto para separar a ordem do nome do clube, dos parênteses para separar a nacionalidade do clube e o hífen para separar o percentual de votos obtidos por cada clube. O conjunto de caracteres usados para estruturar o texto de uma lista HTML é chamado de caracteres de separação. Neste trabalho, para fins de experimentos, são considerados como potenciais caracteres de separação todos os caracteres especiais, ou seja, os caracteres que não sejam alfanuméricos.

4.3 Métodos

Os métodos desenvolvidos neste trabalho buscam identificar delimitadores de informação, fazendo uso de dados estatísticos a respeito dos candidatos a caracteres separadores pertencentes a lista a ser analisada.

[1] http://pt.wikipedia.org/wiki/Anexo:Lista_dos_Clubes_do_S%C3%A9culo_da_FIFA

Figura 4.3: Exemplo de lista HTML com separadores bem definidos.

Como os métodos utilizam apenas caracteres separadores, não são analisadas as subestruturas embutidas na tag "$< li >$"como divs, hiperlinks ou modificadores de caixa, como "$< strong >$". Sendo assim, somente é considerado o texto visível para os usuários no navegador, deixando ainda de lado os efeitos em sua apresentação.

As próximas seções descrevem cada um dos métodos de extração criados.

4.3.1 Método Caractere Sempre Presente

O método Caractere Sempre Presente (CSP) faz a segmentação utilizando apenas um candidato a caractere separador como delimitador de informação para todas as linhas da lista.

O método busca pelo candidato a caractere separador com maior número de ocorrências que necessariamente está presente em todas as linhas da lista. Um exemplo de lista com essas características pode ser vista na Figura 4.4. A cada ocorrência desse caractere é criado um novo segmento.

Este método não é eficaz nas seguintes circunstâncias:

- quando existe mais de um caractere separador.
- quando existe uma linha onde o separador não ocorra, por ele estar faltando, por outro caractere estar em seu lugar.
- quando existir ao menos uma linha com apenas uma coluna.

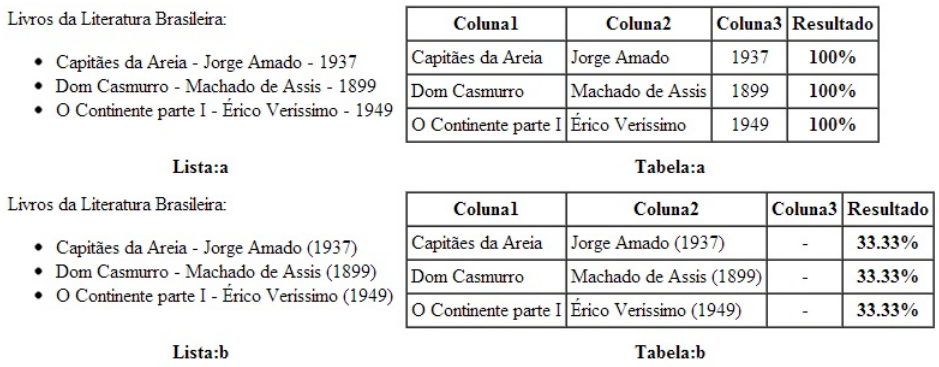


Figura 4.4: Exemplo de Segmentação Método Caractere Sempre Presente

- quando o caractere com maior número de ocorrências não for separador.

Quando existe mais de um separador, pode ser que uma coluna seja segmentada corretamente, como ocorre na **Lista b** da figura 4.4, onde os elementos da primeira coluna estão corretos, e os da segunda não, pois estão aglutinados com os da terceira que se encontra vazia. Isso explica a baixa taxa de acerto dos segmentos de cada linha (33%).

4.3.2 Método Melhor Caractere

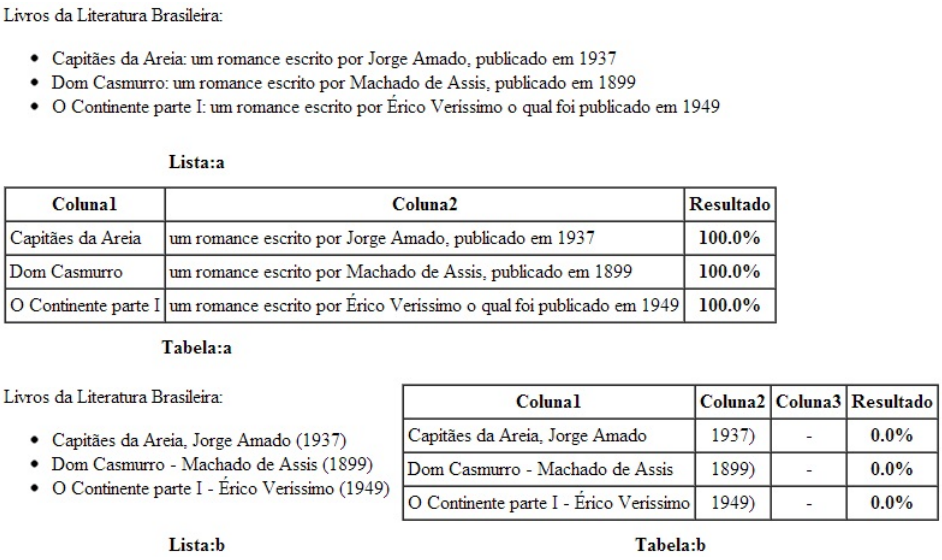
O método Melhor Caractere (MC) também faz a segmentação utilizando apenas um caracter como delimitador de informação para todas as linhas da lista. O método busca pelo candidato a caractere separador com maior número de ocorrências em toda lista, e usa-o como separador para todas as linhas da lista. Logo independentemente de determinada lista possuir n linhas, o método escolhe somente 1 caractere como separador, e este não precisa estar necessariamente em todas as linhas da lista. Caso alguma linha tenha algum caractere separador diferente da maioria, apenas está linha segmentará de forma errada.

Diferentemente do método CSP este permite a escolha de um caractere separador mesmo que ele não exista em todas as linhas da lista.

Este método não é eficaz nas seguintes circunstâncias:

- quando existe mais de um caracatere separador, como na Lista b da Figura 4.5.

- quando o caractere com maior número de ocorrências não for separador.



Livros da Literatura Brasileira:

- Capitães da Areia: um romance escrito por Jorge Amado, publicado em 1937
- Dom Casmurro: um romance escrito por Machado de Assis, publicado em 1899
- O Continente parte I: um romance escrito por Érico Veríssimo o qual foi publicado em 1949

Lista:a

Coluna1	Coluna2	Resultado
Capitães da Areia	um romance escrito por Jorge Amado, publicado em 1937	**100.0%**
Dom Casmurro	um romance escrito por Machado de Assis, publicado em 1899	**100.0%**
O Continente parte I	um romance escrito por Érico Veríssimo o qual foi publicado em 1949	**100.0%**

Tabela:a

Livros da Literatura Brasileira:

- Capitães da Areia, Jorge Amado (1937)
- Dom Casmurro - Machado de Assis (1899)
- O Continente parte I - Érico Veríssimo (1949)

Coluna1	Coluna2	Coluna3	Resultado
Capitães da Areia, Jorge Amado	1937)	-	**0.0%**
Dom Casmurro - Machado de Assis	1899)	-	**0.0%**
O Continente parte I - Érico Veríssimo	1949)	-	**0.0%**

Lista:b **Tabela:b**

Figura 4.5: Exemplo de Segmentação Método Melhor Caractere

4.3.3 Método de Um Caractere

O método Um Caractere (UC) faz a segmentação utilizando apenas um caractere separador como delimitador da informação em cada linha da lista. O método busca pelo caractere separador com maior número de ocorrências em cada linha, e usa-o para separar a informação da linha analizada.

Também é importante deixar claro que a escolha do caractere separador de uma linha é totalmente independente das outras linhas, visto que a cada linha é feita uma nova escolha do caractere que irá fazer a separação, logo se determinada lista possui n linhas o método pode escolher até n caracteres diferentes como separadores.

Diferentemente dos métodos anteriores, este método pode escolher mais de um caractere separador por lista.

Este método não é eficaz nas seguintes circunstâncias:

- quando existe mais de um caracatere separador por linha como na Lista b da Figura 4.6 onde algumas linhas possuem três caracteres de separação (como o hífen e os parênteses). .
- quando o caractere com maior número de ocorrências na linha não for separador.

4.3.4 Método Caractere com a Menor Variação

O método Caractere com a Menor Variação (MCV) faz a segmentação utilizando apenas um caractere separador como delimitador da informação para todas as linhas da lista.

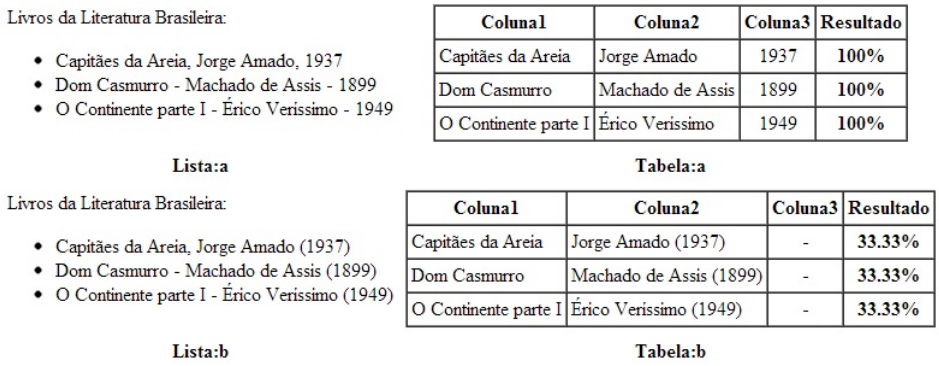


Figura 4.6: Exemplo de Segmentação Método Um Caractere

O método busca pelo caractere separador com menor variação do número de ocorrências em relação a média de ocorrências de todos os caracteres encontrados em toda a lista.

Este método tem os resultados muito parecidos com o método MC, se saindo melhor em poucos casos onde existe apenas um caractere separador e outros caracteres sem função de separação ocorrendo com muita frequência, um exemplo de segmentação onde ele se sai bem é o da *Lista a* da Figura4.7.

Este método não é eficaz nas seguintes circunstâncias:

- quando existe mais de um caractere separador, como na Lista b da Figura 4.7..

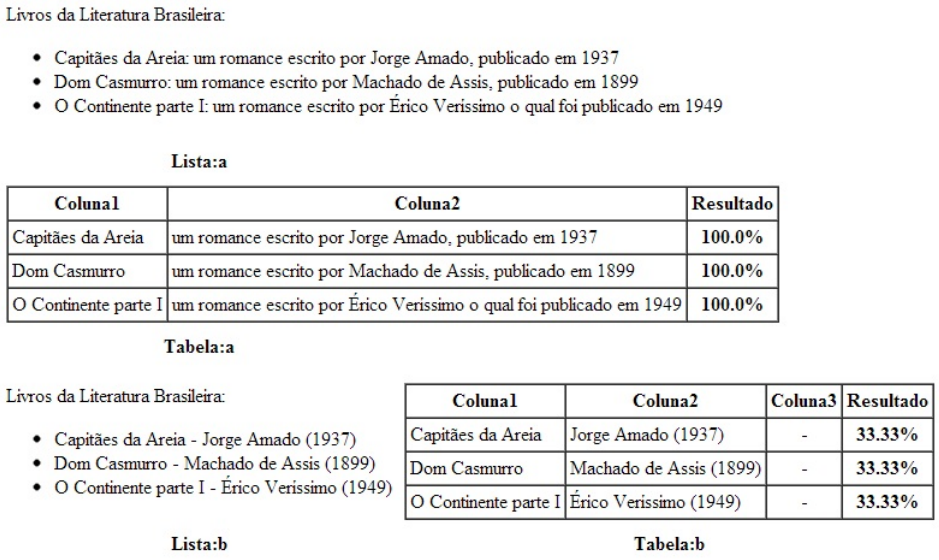


Figura 4.7: Exemplo de Segmentação Método Caractere com a Menor Variação

4.3.5 Método Conjunto de Caracteres Sempre Presente

O método Conjunto de Caracteres Sempre Presente (CCSP) faz a segmentação utilizando um conjunto de caracteres separadores para todas as linhas da lista. O método busca por todos os caracteres especiais que estão presentes necessariamente em todas as linhas da lista, independentemente do número de ocorrências, o qual o mínimo é será igual ao número de linhas da lista.

Este método possibilita a segmentação considerando mais de um caractere como pode ser visto na Lista a da Figura 4.8, diferentemente dos métodos anteriores.

Este método não é eficaz nas seguintes circunstâncias:

- quando existe uma linha onde o separador não ocorra, por ele estar faltando, por outro caractere estar em seu lugar como na Lista b da figura 4.8, ou por existir ao menos uma linha com apenas uma coluna, acarretando a falta de um caractere.

Livros da Literatura Brasileira:

- Capitães da Areia - Jorge Amado (1937)
- Dom Casmurro - Machado de Assis (1899)
- O Continente parte I - Érico Veríssimo (1949)

Lista:a

Coluna1	Coluna2	Coluna3	Resultado
Capitães da Areia	Jorge Amado	1937	**100.0%**
Dom Casmurro	Machado de Assis	1899	**100.0%**
O Continente parte I	Érico Veríssimo	1949	**100.0%**

Tabela:a

Livros da Literatura Brasileira:

- Capitães da Areia, Jorge Amado (1937)
- Dom Casmurro - Machado de Assis (1899)
- O Continente parte I - Érico Veríssimo - 1949

Lista:b

Coluna1	Coluna2	Coluna3	Resultado
Capitães da Areia, Jorge Amado (1937)	-	-	**0.0%**
Dom Casmurro - Machado de Assis (1899)	-	-	**0.0%**
O Continente parte I - Érico Veríssimo - 1949	-	-	**0.0%**

Tabela:b

Figura 4.8: Exemplo de Segmentação Método Conjunto de Caracteres Sempre Presente

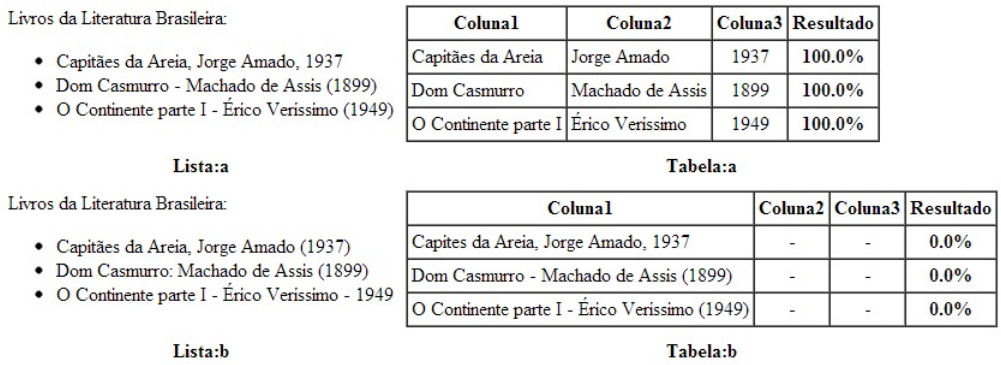
4.3.6 Método Conjunto de Melhores Caracteres por Linha

O método Conjunto de Melhores Caracteres (CMCL) faz a segmentação utilizando um conjunto de caracteres separadores este conjunto é o mesmo para todas as linhas da lista. O conjunto tem tamanho variável, possuindo todos os caracteres separadores cujo número de ocorrências é maior que a média subtraída do desvio padrão.

Este método permite mais de um separador por lista, e diferentemente do método CCSP ele permite que falte algum caractere entre as linhas da lista, ou tenha algum caractere trocado em alguma linha, não acarretando em erro além do local onde ele estiver faltando. O caso de haver alguma divergência de uma pequena porcentagem de caracteres em algumas linhas da lista pode ser visto na Lista a da Figura4.9.

Este método não é eficaz nas seguintes circunstâncias:

- quando a variação da ocorrência de um caractere for maior que a sua média perante o número de linhas totais subtraído pelo desvio padrão, o método pode não segmentar direito como pode ser observado na Lista b da Figura4.9.


Livros da Literatura Brasileira:

- Capitães da Areia, Jorge Amado, 1937
- Dom Casmurro - Machado de Assis (1899)
- O Continente parte I - Érico Verissimo (1949)

Lista:a

Coluna1	Coluna2	Coluna3	Resultado
Capitães da Areia	Jorge Amado	1937	100.0%
Dom Casmurro	Machado de Assis	1899	100.0%
O Continente parte I	Érico Verissimo	1949	100.0%

Tabela:a

Livros da Literatura Brasileira:

- Capitães da Areia, Jorge Amado (1937)
- Dom Casmurro: Machado de Assis (1899)
- O Continente parte I - Érico Verissimo - 1949

Lista:b

Coluna1	Coluna2	Coluna3	Resultado
Capites da Areia, Jorge Amado, 1937	-	-	0.0%
Dom Casmurro - Machado de Assis (1899)	-	-	0.0%
O Continente parte I - Érico Verissimo (1949)	-	-	0.0%

Tabela:b

Figura 4.9: Exemplo de Segmentação Método Conjunto de Melhores Caracteres

4.3.7 Método Conjunto de Melhores Caracteres por Tipo

O método Conjunto de Melhores Caracteres (CMCT) também faz a segmentação utilizando um conjunto de caracteres separador como delimitadores de informação, este conjunto também é o mesmo para todas as linhas da lista, tendo tamanho variável.

Este método faz a segmentação utilizando todos os caracteres separadores, cujo o número de ocorrências é maior que a média dos caracteres separadores em relação ao seu tipo subtraída do desvio padrão, onde neste caso considera-se cada caractere não alfanumérico diferente como um símbolo.

Este método como o CMCL também permite mais de um separador por lista, e alguns caracteres trocados ou faltantes como pode ser visto na Lista a da Figura4.10.

Este método não é eficaz nas seguintes circunstâncias:

- quando a variação da ocorência de um caractere for maior que a sua média perante o número de linhas totais subtraído pelo desvio padrão pode não segmentar direito como pode ser observado na Lista b da Figura4.10.

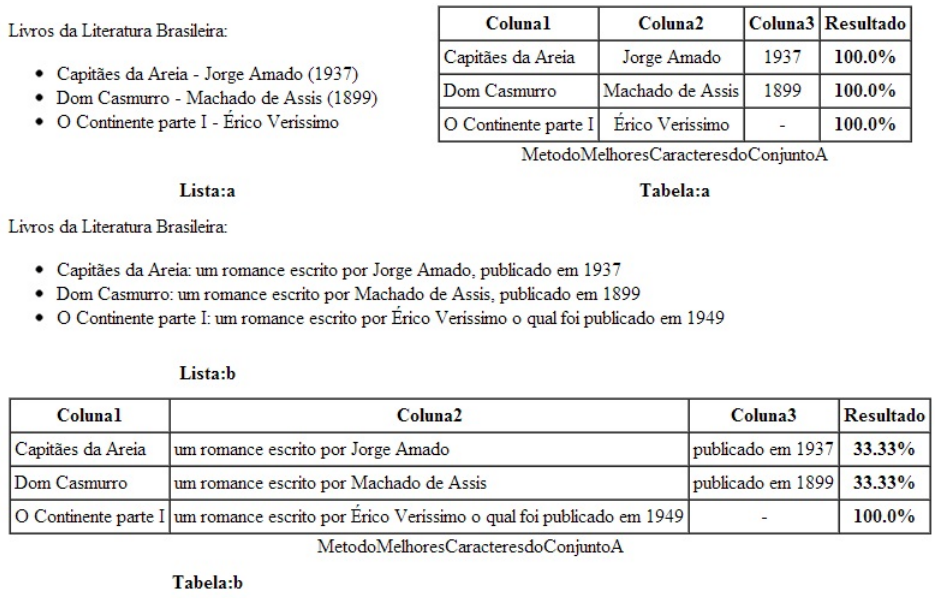


Figura 4.10: Exemplo de Segmentação Método Conjunto de Melhores Caracteres por Tipo

Capítulo 5

Resultados

Os resultados dos métodos criados serão mostrados e analizados neste capítulo. As seções a seguir detalham aspectos relacionados aos experimentos, como as coleções de listas utilizadas, a linha de base escolhida para efeitos de comparação, os indicadores de desempenho e as análises dos experimentos realizados.

5.0.8 Coleções de Listas

O conjunto de listas usado nas baterias de extração foi gerado a partir de listas reais encontradas na *Web*. Cada conjunto possui 50 listas coletadas manualmente a partir dos seguintes critérios:

- as linhas não são parte de listas de *menu*;
- as linhas são compostas por textos com no máximo 500 caracteres;
- as linhas possuem caracteres de separação.

A partir desses critérios foram elaborados três conjuntos distintos, conforme descrito a seguir:

Wikipédia

Esta coleção possui listas aleatórias recuperadas a partir de páginas da Wikipédia[1]. A geração da coleção foi realizada através de um recurso da Wikipédia que direciona o usuário a uma página aleatória. Nesta página, foram coletadas todas as listas que satisfizeram os critérios definidos acima. Em seguida, uma nova página aleatória foi acessada, e o processo se repetiu até que 50 listas tivessem sido coletadas.

[1] http://pt.wikipedia.org/wiki/Wikip%C3%A9dia:P%C3%A1gina_principal

Listas 10

Esta coleção possui listas aleatórias existentes em sites que informam os 10 tópicos mais relevantes a respeito de um determinada categoria. Foram utilizadas aproximadamente 50 páginas do site Listas10[2] durante a coleta. Dentre as categorias coletadas encontram-se Cinema, Músicas e Esportes.

WT10G

Esta coleção possui listas aleatórias recuperadas a partir da *Web*. A geração da coleção foi realizada através de um *snapshot* da *Web* de 1997, contida na *corpus* WT10G[3]. As páginas da coleção estão divididas em múltiplas pastas numeradas, sendo cada pasta dividida em múltiplos documentos XML numerados. Cada documentos XML também é dividido em blocos numerados, referentes a uma página específica. Navegando aleatoriamente por essa estrutura, foi recuperada uma página a partir de onde foram coletadas as listas que satisfizeram os critérios definidos acima. Em seguida, uma nova página aleatória foi acessada, e o processo se repetiu até que 50 listas tivessem sido coletadas.

5.0.9 Critérios de Marcação

As listas foram manualmente marcadas com a divisão considerada correta. Essa marcação serve como linha de base para analisar a efetividade dos métodos propostos. Esse procedimento foi realizado dividindo as linhas das listas em colunas que pudessem ser rotuladas de modo simples e intuitivo.

Por exemplo, uma linha "Batman(1987)" seria separada em duas colunas, pois é natural pensar que a primeira coluna se refere ao nome do filme, enquanto a segunda se refere ao ano do filme. Já uma linha "Batman bateu recorde de público (e de arrecadação também)" não seria separada, pois faz mais sentido manter toda a informação em apenas uma coluna sobre notícias do filme.

Outro critério adotado diz respeito aos campos multivalorados, os quais foram mantidos em uma única divisão. Por exemplo, a linha "Batman: 1989, 1992, 1995, 1997" seria separada em dois segmentos, um para o nome do filme e outro para os anos de lançamento. Caso contrário, seria necessário criar rótulos separados para cada ano, o que não é visto como uma boa prática de modelagem, uma vez que o número de valores de um atributo multivalorado é dinâmico.

Na tabela da figura 5.1 podemos ver algumas características das coleções criadas, quando realizada essa segmentação manual. Logo podemos perceber que as coleções

[2] http://lista10.org/
[3] http://ir.dcs.gla.ac.uk/test_collections/wt10g.html

variam bastante em número de linhas, segmentos e colunas, com exceção da coleção Listas10 que possui sempre o mesmo número de linhas. O número de colunas de uma lista é definido como o maior número de segmentos encontrados em alguma das linhas da lista. Na maioria das listas o número de colunas ficou de 2 a 3, porém esse número apresentou variação como pode ser visto no desvio padrão.

Tabela Geral das Coleções

	Wikipedia	Listas10	WT10G
Média de linhas da lista	9,08	10,00	9,64
Desvio do Número de Linhas das Listas	9,53	0,00	10,88
Mínimo Número de Linhas da Lista	1	10,00	1,00
Máximo Número de Linhas da Lista	54	10,00	56,00
Média de segmentos por linha	2,48	2,38	2,34
Desvio da Média de Segmentos por Linha	2,66	2,45	2,40

Figura 5.1: Informações das coleções criadas

5.0.10 Indicadores para Avaliação

A efetividade dos métodos apresentados neste trabalho foi avaliada a partir de medidas de cobertura e precisão. A medida de cobertura foi calculada de acordo com a Equação 5.1.

$$Cobertura = \frac{itens\ corretos\ encontrados}{itens\ corretos} \tag{5.1}$$

Os itens destacados na equação podem referir-se aos segmentos das linhas, às linhas propriamente ditas ou às listas. Ou seja, a cobertura pode ser calculada em três níveis distintos. A *Cobertura dos Segmentos* verifica quantos dos segmentos que deveriam ser separados foram realmente identificados. A *Cobertura das Linhas* verifica quantas linhas foram devidamente segmentadas. Já a *Cobertura das Listas* verifica quantas listas tiveram todas suas linhas devidamente segmentadas.

A cobertura das linhas e listas é mais restrita do que a cobertura dos segmentos. Um único segmento não identificado na linha torna essa linha inválida, para fins de medição. Da mesma forma, uma única linha inválida em uma lista torna essa lista inválida.

De modo semelhante, a medida de precisão foi calculada de acordo com a Equação 5.2.

$$Precisao = \frac{itens\ corretos\ encontrados}{itens\ encontrados} \tag{5.2}$$

Observe que neste caso, considerando que os itens sendo avaliados sejam as linhas ou as listas, os valores de precisão seriam equivalentes aos de cobertura. Isso ocorre porque o número de linhas e listas já é conhecido, e não algo que precisa ser descoberto. Dessa forma, a precisão é calculada apenas sobre os segmentos encontrados. Nesse caso, a *Precisão dos Segmentos* verifica quantos dos segmentos encontrados são realmente segmentos corretos.

5.0.11 Análise dos Resultados

A Figura 5.2 apresenta a medida de cobertura encontrada pelos métodos sobre a coleção Wikipédia. Conforme se pode ver, a cobertura de todos os métodos foi homogênea. No entanto, o método CMCT teve um desempenho levemente superior aos demais no tocante aos segmentos e às listas, enquanto o método CCSP se saiu melhor na recuperação de listas totalmente corretas.

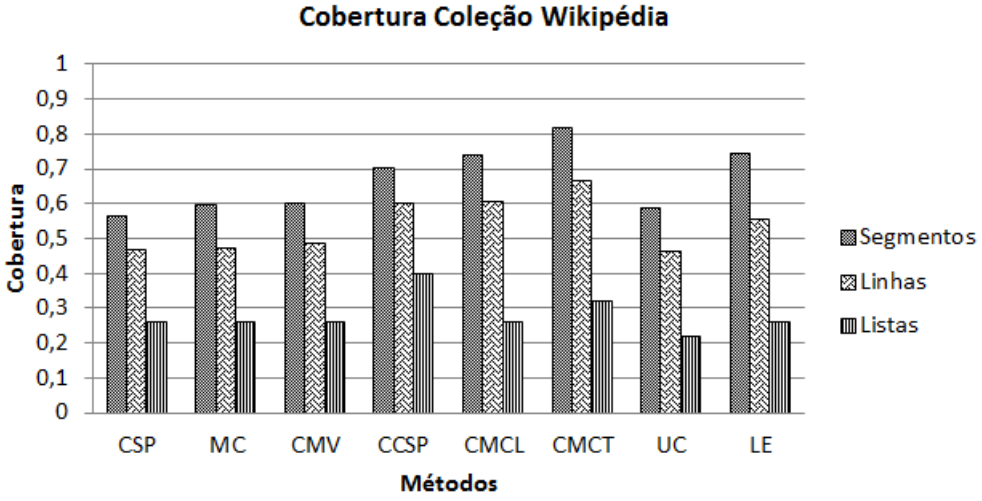


Figura 5.2: Medida de cobertura encontrada sobre a coleção Wikipédia

A Figura 5.3 apresenta a medida de cobertura encontrada pelos métodos sobre a coleção Listas 10. O método CCSP se saiu melhor em todos os critérios. Destaca-se também o fato de que o LE não atingiu bons níveis de cobertura com relação ao número de linhas e listas totalmente corretas. Esse desempenho pode estar associado a um número insuficiente de expressões regulares que foram implementadas. Por exemplo, nenhuma das expressões regulares usadas identifica valores monetários, sendo que esse tipo de texto ocorre com uma frequência maior nesta coleção.

A Figura 5.4 apresenta a medida de cobertura encontrada pelos métodos sobre a coleção WT10G. Percebe-se que o desempenho de todos os métodos foi insatisfatório. De certa forma isto está associado a natureza heterogênea dos dados dessa coleção.

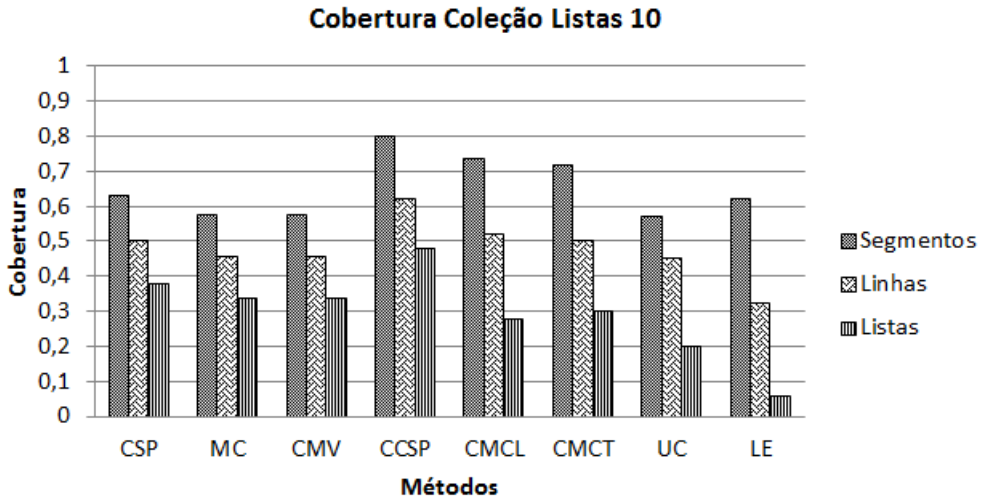


Figura 5.3: Medida de cobertura encontrada sobre a coleção Listas 10

Pode-se deduzir que as listas desta coleção tenham sido criadas sem uma atenção especial ao uso de padrões de estruturação de conteúdo, como costuma ocorrer com listas criadas na Wikipédia por exemplo. Além disso, os próprios dados são de domínios diversos, o que pode ter dificultado a distinção entre caracteres usados como conteúdo e caracteres usados como separadores. Essa dificuldade se tornou anda maior pelo fato de esta coleção possuir linhas longas, onde a probabilidade de ocorrência de caracteres especiais aumenta, sem que eles de fato separem conteúdo.

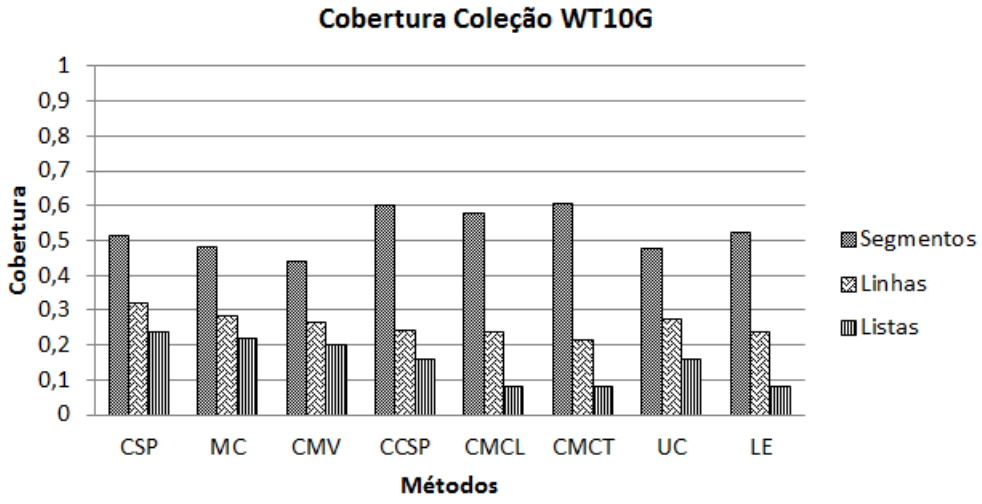


Figura 5.4: Medida de cobertura encontrada sobre a coleção WT10G

De modo geral, percebe-se que os métodos que utilizam conjuntos de caracteres são mais propensos a identificar segmentos corretos. Isso leva a crer que seja comum listas utilizarem mais de um caractere para a separação de conteúdo. Curiosamente,

dos métodos que utilizam um único caractere por linha, aqueles que usam o mesmo caractere para a lista inteira (CSP, MC, CMV) se saem melhor do que o método que pode escolher caracteres diferentes para cada linha (UC). Apesar de serem mais restritivos, os métodos CSP, MC, CMV se sobressaem pelo fato de que normalmente as linhas de uma lista usam o mesmo caractere como separador. Assim, sua presença nas linhas reforça o fato de que ele seja o caractere de separação, mesmo que em uma dada linha sua frequência seja inferior a algum outro caractere especial.

Na Tabela 5.1 está presente os valores de precisão e cobertura encontrados.

	Coleções		
	Wikipédia	Listas 10	WT10G
CSP	60,33% / 56,33%	66,41% / 66,15%	47,46% / 51,55%
MC	62,95% / 59,86%	59,47% / 57,39%	42,85% / 48,10%
UC	61,10% / 58,92%	58,71% / 57,18%	40,28% / 47,74%
CMV	62,99% / 60,33%	57,47% / 57,39%	39,76% / 44,10%
CCSP	70,75% / 74,14%	75,14% / 80,22%	47,29% / 60,27%
CMCL	68,42% / 74,03%	64,66% / 73,81%	39,61% / 57,78%
CMCT	74,82% / 81,87%	63,67% / 77,73%	39,75% / 62,80%
LE	66,64% / 74,28%	53,20% / 71,81%	37,37% / 60,47%

Tabela 5.1: Precisão e cobertura dos segmentos, sendo o primeiro item da célula a precisão e o segundo a cobertura.

A Figura 5.5 apresenta o gráfico de medida F encontrada pelos métodos referente a sua capacidade de segmentação em todas as coleções. Os resultados corroboram a constatação de que os métodos baseados em coleções de caracteres tem um desempenho maior no reconhecimento dos segmentos, tanto com relação a precisão quanto com relação a cobertura. Esse resultado demonstra que métodos desse tipo são relevantes, senão como uma técnica própria de extração, pelo menos como parte de uma etapa inicial de segmentação, como proposto por [Elmeleegy et al., 2009].

Além disso, em listas compostas por caracteres de separação, como as analisadas neste artigo, alguns dos métodos de divisão apresentados obtiveram um desempenho considerado bom principalmente se levado em conta a simplicidades de se utilizar apenas caracteres separadores. Ainda, as regras independem da existência de bases de conhecimento, o que torna o mecanismo de extração útil em situações onde não existe uma base que possa ser usada ou a base esteja indisponível.

Para finalizar os experimentos, a Figura 5.6 mostra que o método que segmentou menos em todos os repositórios foi o Método CSP, pois uma simples linha sem o caractere escolhido basta para o método não fazer a segmentação.

Já o método que segmentou mais foi o CMCT nas coleções da Wikipédia e no WT10G. Na coleção do Listas10, o melhor método foi o CMCL.

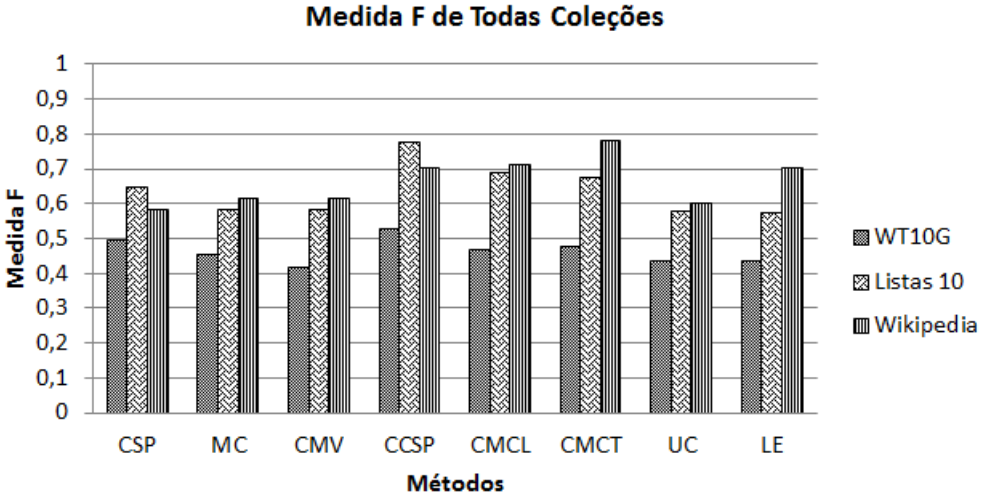


Figura 5.5: Medida F encontrada sobre a três coleções

	Método que Segmento Menos	Método que Segmento Mais	Média de Segmentos da Solução
Wikipedia	Caractere Sempre Presente = 20	Conjunto de Melhores Caracteres por Tipo = 30	22,7
Listas10	Caractere Sempre Presente = 22	Conjunto de Melhores Caracteres = 31,4	23,8
WT10G	Caractere Sempre Presente = 26,86	Conjunto de Melhores Caracteres por Tipo = 47,6	22,6

Figura 5.6: Métodos que Segmentaram Mais e Menos por Coleção

Capítulo 6

Conclusão

Este trabalho apresentou uma série de regras para segmentação de registros baseadas na presença e frequência de caracteres especiais que costumam ser empregados na delimitação de conteúdo. De modo geral, as regras verificam caracteres mais frequentes ou mais bem distribuídos pelas linhas de uma lista. Como as regras utilizam apenas a frequências de caracteres separadores as regras independem da existência de bases de conhecimento, o que torna o mecanismo de extração útil em situações onde não existe uma base que possa ser usada ou a base esteja indisponível.

Os experimentos mostraram que algumas das regras são mais abrangentes do que outras, sendo capazes de fazer uma melhor segmentação em listas HTML de modo geral. Outras regras são mais específicas para listas bem comportadas, que seguem algum padrão pré-definido. Também foi realizada uma comparação das regras propostas com o método ListExtractor. A comparação mostra que, quando as listas são segmentadas por delimitadores de conteúdo, algumas das regras baseadas em caracteres especiais se sobressaem ao ListExtractor.

Os resultados servem como motivação para que se investigue uma forma de combinar as regras propostas com a ideia de segmentação usada pelo ListExtractor. Dessa forma, se abre uma linha de pesquisa voltada ao aprimoramento das soluções mecionadas no texto atragem do uso de abordagens híbridas.

Outra possibilidade de trabalho futuro envolve partir para o rotulamento das colunas. Ou seja, após realizada a segmentação, derivar o nome de cada coluna a partir dos respectivos valores encontrados nos registros. Essa etapa é necessária para que se crie uma solução que possa ser aproveitada como módulo de extração em arquiteturas de dataspaces, como a arquitetura EIDOS citadas neste trabalho.

Cabe ressaltar que um rotulamento preciso de colunas depende de um bom alinhamento, pois é a partir dos valores das colunas que se determina o nome mais ade-

quado. Muito embora as regras propostas não realizem propriamente um alinhamento, assim como é feito pelo ListExtractor, algumas delas obtiveram uma boa precisão na segmentação dos registros. Assim, pretende-se verificar como os algoritmos de rotulamento se comportam ao usar como entrada as estruturas tabulares geradas pelas regras propostas neste trabalho.

Referências Bibliográficas

[Álvarez et al., 2008] Álvarez, M.; Pan, A.; Raposo, J.; Bellas, F. & Cacheda, F. (2008). Extracting lists of data records from semi-structured web pages. *Data & Knowledge Engineering*, 64(2):491--509. ISSN 0169023X.

[Arasu & Garcia-Molina, 2003] Arasu, A. & Garcia-Molina, H. (2003). Extracting structured data from Web pages. Em *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pp. 337--348, New York, NY, USA. ACM.

[Barbosa et al., 2001] Barbosa, A. C. P.; Biancardi, C. & Silvestre, L. J. (2001). Integracao de Dados Heterogêos em Ambiente Web. *World Wide Web Internet And Web Information Systems*.

[Crescenzi et al., 2001a] Crescenzi, V.; Mecca, G. & Merialdo, P. (2001a). RoadRunner: Towards Automatic Data Extraction from Large Web Sites. Em *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pp. 109--118, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Crescenzi et al., 2001b] Crescenzi, V.; Mecca, G.; Merialdo, P. & Others (2001b). Roadrunner: Towards automatic data extraction from large web sites. Em *Proceedings of the international conference on very large data bases*, pp. 109--118.

[Cuenca et al.,] Cuenca, A.; Torres, H.; Distribuidos, S. & Warehouse, D. Extracción de información de fuentes de datos heterogéneas e incorporación al data warehouse. *International Journal*.

[Elmeleegy et al., 2009] Elmeleegy, H.; Madhavan, J. & Halevy, A. (2009). Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078--1089.

[Franklin et al., 2005] Franklin, M.; Halevy, A. & Maier, D. (2005). From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4):27--33. ISSN 0163-5808.

[Gupta & Sarawagi, 2009] Gupta, R. & Sarawagi, S. (2009). Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2(1):289--300.

[Hsu & Dung, 1998] Hsu, C.-N. & Dung, M.-T. (1998). Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*, 23(8):521--538. ISSN 03064379.

[Kayed et al., 2006] Kayed, M.; Girgis, M. & Shaalan, K. (2006). A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411--1428. ISSN 1041-4347.

[Lerman et al., 2001] Lerman, K.; Knoblock, C. & Minton, S. (2001). Automatic data extraction from lists and tables in web sources. Em *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, volume 98. Citeseer.

[Liu et al., 2003] Liu, B.; Grossman, R. & Zhai, Y. (2003). Mining data records in Web pages. Em *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pp. 601--606, New York, NY, USA. ACM.

[Machanavajjhala et al., 2011] Machanavajjhala, A.; Iyer, A. S.; Bohannon, P. & Merugu, S. (2011). *Collective extraction from heterogeneous web lists*. ACM Press. ISBN 9781450304931.

[Mergen, 2011] Mergen, S. (2011). *Indexing and Querying Dataspaces*. Tese de doutorado.

[Salgado & Lóscio, 2001] Salgado, A. C. & Lóscio, B. F. (2001). Integração de Dados na Web. pp. 157--174.

[Wang & Lochovsky, 2003] Wang, J. & Lochovsky, F. H. (2003). Data extraction and label assignment for web databases. Em *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pp. 187--196, New York, NY, USA. ACM.

[Zhai & Liu, 2005] Zhai, Y. & Liu, B. (2005). Web data extraction based on partial tree alignment. Em *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pp. 76--85, New York, NY, USA. ACM.

[Zhao et al., 2007] Zhao, H.; Meng, W. & Yu, C. (2007). Mining templates from search result records of search engines. Em *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pp. 884--893, New York, NY, USA. ACM.

[Zhu et al., 2006] Zhu, J.; Nie, Z.; Wen, J.-R.; Zhang, B. & Ma, W.-Y. (2006). Simultaneous record detection and attribute labeling in web data extraction. Em *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pp. 494--503, New York, NY, USA. ACM.