UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rodrigo Gonçalves

Algoritmos de Pré-Processamento para Uniformização de Instâncias XML Heterogêneas

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Ronaldo dos Santos Mello (UFSC)

Florianópolis, Fevereiro de 2008

Algoritmos de Pré-Processamento para Uniformização de Instâncias XML Heterogêneas

Rodrigo Gonçalves

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Mario Antônio Ribeiro Dantas

Banca Examinadora

Ronaldo dos Santos Mello (UFSC)

Renata de Mattos Galante (UFRGS)

Carina Friedrich Dorneles (UPF)

Patricia Vilain (UFSC)

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.
(Albert Einstein)

À minha família, sem a qual nunca teria chegado aqui.
A todas as pessoas que, de uma forma ou outra, ajudaram a
tornar este trabalho uma realidade.

Agradecimentos

Inicialmente agradeço à Universidade Federal de Santa Catarina, pela possibilidade de desenvolver este trabalho.

Agradeço ao meu orientador, Ronaldo, pela possibilidade de realizar este trabalho sob sua orientação. Suas sugestões e dicas foram valiosas para a confecção do mesmo, assim como sua paciência para revisá-lo, com *"olhos de águia"*.

Um agradecimento muito especial a minha família, sem a qual nada disto seria possível. Em especial um agradecimento a sua compreensão quanto às semanas e fins de semanas ausentes frutos da dedicação a este trabalho.

Quanto aos amigos, não citarei nomes para não esquecer ninguém. Basta dizer que todos, de uma forma ou outra, contribuíram para que este trabalho se concretizasse.

Agradeço ao pessoal da Manager Systems, cuja compreensão quanto às ausências advindas da dedicação a este trabalho foram valiosas para que o mesmo pudesse ser realizado.

Em resumo, agradeço a todos que de qualquer forma contribuíram para que este trabalho se tornasse uma realidade.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O aumento no volume de informações disponíveis na *Web* torna necessário sistemas cada vez mais práticos e eficientes na coleta e integração destas informações, para fins de consulta. Um dos formatos mais utilizados para disponibilizar as informações na *Web* é O XML. O XML, dada a sua natureza dinâmica, permite representações completas e adequadas dos mais diferentes domínios de dados. Ao mesmo tempo, esta natureza dinâmica lhe confere aspectos que tornam complexa a integração de dados neste formato. Este trabalho vem ao encontro deste problema, provendo um conjunto de *técnicas de pré-processamento* para uniformizar as estruturas de dados no formato XML. Esta uniformização, que busca respeitar a semântica dos dados, visa facilitar a comparação e posterior integração por abordagens já existentes para comparação e integração de dados. Através de estudos de caso e experimentos, demonstra-se como os pré-processamentos sugeridos influem positivamente nos resultados de trabalhos existentes.

Abstract

The increasing availability of data on the *Web* creates the need for more pratical and efficient systems for collecting and integrating these data, in order to provide queries over them. One of the most used formats to represent information in the *Web* is XML. XML, given its dynamic nature, allows complete and adequate representation of data from different domains. But, at the same time, such dynamic nature makes integration activities complex. This work focus on reducing such complexity, providing a set of preprocessing techniques to compatibilize the structures of XML instances. This compatibilization, which seeks to respect data semantics, tries to facilitate the comparison and further integration of these data by already existing approaches for XML data comparison and integration. Through case studies and experiments, we demonstrate how the suggested preprocessings provide better results for the existing related work.

Capítulo 1

Introdução

Fontes de dados são repositórios de informações relacionadas a uma mesma área ou tópico. Armazenar dados sobre uma área do conhecimento, de forma que se possa recuperá-los eficientemente é uma preocupação antiga. Desde épocas distantes mantém-se, em bibliotecas, informações sobre diversos temas, através de livros, revistas, textos, etc. Em tempos atuais, com o avanço da tecnologia e a redução no custo de armazenamento dos dados em meio eletrônico, cada vez mais se mantém informações a respeito de diferentes áreas e domínios do conhecimento. Associada a este custo reduzido está a facilidade de criar, manter e consultar tais dados [CAR 03, ELM 99, MEL 02a].

Com as redes de comunicações e o surgimento da *Web*[1], têm-se tornado uma prática comum a disponibilização de dados na *Web*, que podem ser consultados e utilizados a partir de qualquer lugar [CAR 03]. Desta forma, é possível o acesso instantâneo à informação de qualquer ponto do mundo. O custo reduzido para publicar e manter disponível uma fonte de dados na *Web* também é um dos motivos para sua popularização [DAU 02, MEL 02a].

Dada a natureza diversa das informações contidas nestas fontes e dos responsáveis por sua criação e manutenção, não há uma forma padrão sob a qual elas disponibilizam seus dados [MEL 02a]. Alguns formatos genéricos, porém, têm sido utilizados por muitas delas para publicar os dados. Dentre os formatos, destacam-se aqueles

[1] Sinônimo para WWW - *Word Wide Web*

que mantém as informações em forma semi-estruturada, em especial o formato XML[2] [CON 00, MEL 02a, FLE 05].

A forma semi-estruturada de representação da informação, diferente da estruturada, que restringe a estrutura e o conteúdo dos dados armazenados, e da forma não-estruturada, que dificulta, muitas vezes, a descoberta e o acesso a informações, apresenta-se como uma solução adequada. Uma vez que permite a organização dos dados com um formato menos rígido e, ao mesmo tempo, com uma formatação que possibilita a compreensão e o acesso a eles [CON 00, MEL 02a].

O XML têm-se firmado como um padrão para a troca de informação, especialmente na Web. Sua natureza dinâmica e capacidade de auto-descrever dados facilitam a publicação de informações. Por ser auto-descritiva, não exige informações adicionais para ser compreendida. Por ser dinâmica, permite que domínios diversos sejam representados de forma adequada, sem limitar as informações contidas e o acesso a elas [MEL 02a, CAR 03].

Apesar dos aspectos positivos e ganhos que as fontes de dados na *Web* trouxeram, elas apresentam alguns problemas. Dentre eles, pode-se citar a existência de informações duplicadas dentro de uma fonte ou entre fontes sobre um mesmo tema. A inconsistência entre informações advindas de várias fontes é outro problema. Pode-se encontrar tanto informações conflitantes como também incompletas - uma fonte pode ter mais dados a respeito de uma determinada entidade que outra fonte. Além destes problemas, a facilidade de publicar e manter fontes de dados na *Web* permite que a quantidade destas aumente rapidamente. Em função disto, o gerenciamento de consultas sobre as fontes torna-se complexo, dada a necessidade de recuperar dados em muitos lugares em busca de um conhecimento desejado [MIL 06, MEL 02a, CAR 03].

Para auxiliar nesta problemática de consulta a fontes de dados heterogêneas, surgiram os sistemas para integração de dados [BER 01]. Estes sistemas buscam realizar consultas sobre diversas fontes sobre um tema e apresentar, a quem requisitou a consulta (seja um usuário ou outro sistema), uma visão integrada dos resultados. Para tal, eles acessam uma ou mais fontes, realizando um processo de integração dos dados obtidos. Esta

[2]XML - *eXtended Markup Language*

integração visa eliminar informações redundantes ou inúteis, com o objetivo de melhorar a qualidade do resultado da consulta [MEL 02a, MIL 06, WEI 04, NIE 02].

No processo de integração das informações de uma consulta feita sobre várias fontes de dados, um ponto chave é determinar quais informações devem ser integradas. Para tal, é preciso que se estabeleçam critérios para determinar quais dados são equivalentes e, por conseqüente, podem vir a ser integrados. Quando se lida com fontes semi-estruturadas, este processo de análise e integração é complexo, dada a natureza dinâmica dos dados.

Em dados semi-estruturados, determinados elementos podem ou não estar presentes, assim como podem apresentar-se de uma ou várias formas diferentes, conforme a intenção semântica e a informação disponível. Tradicionalmente, para se determinar a equivalência entre instâncias de dados semi-estruturadas, trabalha-se a idéia de estabelecer um grau de similaridade entre os dados contidos nelas. Este grau indica a quantidade de dados que as instâncias têm em comum. Quanto maior o grau, mais similares elas serão [CAR 03, DOR 04, MIL 06].

O problema da *determinação da similaridade entre instâncias de dados semi-estruturados com vistas a uma integração dos dados* é a motivação para este trabalho. A intenção aqui é propor uma abordagem para melhorar o resultado do cálculo da similaridade entre dados semi-estruturados, oriundos de fontes de dados heterogêneas. Foca-se neste trabalho em instâncias semi-estruturadas no padrão XML, dada sua ampla adoção pela indústria [DAU 02]. A abordagem a ser apresentada busca evitar que as instâncias a serem comparadas necessitem compartilhar um esquema de estrutura comum. Para realizar o processo que permite isto, ela não necessita que haja definições de esquema (*DTD*[3], *XML Schema*) [CON 00, DAU 02] associadas às instâncias. Tal decisão foi tomada objetivando tornar o método o mais abrangente possível, ou seja, ele não lida somente com instâncias válidas perante um esquema.

Durante a pesquisa sobre similaridade entre dados semi-estruturados, que levou ao desenvolvimento deste trabalho, um problema comum detectado quando se tenta integrar fontes de dados XML é a existência de pequenas diferenças de cunho estrutural entre as instâncias de dados sendo integradas. Tais diferenças, apesar de não gerarem diferen-

[3]DTD - *Document Type Definition*

ças semânticas, acabavam prejudicando o processo de comparação das instâncias. Com vistas a contribuir com sistemas de comparação de dados semi-estruturados existentes, estabeleceu-se um conjunto de *técnicas de pré-processamento*, a serem aplicadas às instâncias de dados semi-estruturados antes que a similaridade entre elas seja estabelecida. Elas reduzem as diferenças estruturais entre as instâncias de dados, buscando não influir negativamente na semântica. Estas técnicas, em conjunto com uma taxionomia sobre os trabalhos relacionados à comparação entre dados semi-estruturados, fruto de uma revisão da literatura sobre o tema, são as principais contribuições deste trabalho.

Este trabalho inicia pelo capítulo 2, que apresenta uma revisão sobre o problema da similaridade entre dados, desde níveis mais básicos, como cadeias de caracteres, até a similaridade entre dados semi-estruturados. Este capítulo define também uma taxionomia para classificar os trabalhos relacionados à similaridade entre dados. A partir desta taxionomia, no capítulo 3, uma análise dos trabalhos relacionados à similaridade entre instâncias de dados semi-estruturadas, em especial no formato XML, é feita. Esta análise busca identificar os problemas sobre os quais a abordagem proposta pretende contribuir.

No capítulo 4 introduz-se as técnicas de pré-processamento propostas, identificando os problemas que elas buscam sanar e como a sua execução é realizada. Para cada técnica, um algoritmo de alto nível é apresentado e sua complexidade é brevemente analisada. Uma vez definidas as técnicas, no capítulo 5 faz-se uma análise detalhada sobre elas, com relação ao impacto sobre os trabalhos relacionados identificados no capítulo 3, dependências entre as técnicas sugeridas e impacto da execução de cada uma delas com relação às demais. A partir desta análise, define-se uma possível ordem para sua execução, assim como formas de combinar sua execução, reduzindo a complexidade de sua aplicação e melhorando o resultado do processo.

Com o objetivo de validar as técnicas de pré-processamento e o método de comparação, elaborou-se um conjunto de testes sobre algumas massas de dados, apresentados no capítulo 6. Eles analisam o impacto da abordagem nos resultados dos cálculos de similaridade entre documentos XML, considerando os trabalhos existentes. Por fim, algumas considerações finais sobre o trabalho desenvolvido e sugestões para futuros trabalhos são apresentadas no capítulo 7.

Capítulo 2

Similaridade entre Dados Semi-Estruturados

2.1 Introdução

Cálculo da similaridade entre dados trata de sua comparação com o objetivo de avaliar quão semelhantes eles são. Estes dados podem ser não estruturados (sons, imagens, vídeos, cadeias de caracteres, etc.), semi-estruturados (documentos XML, blocos de texto, etc.) ou ainda estruturados (tuplas de um banco de dados relacional) [DOR 04, CAR 03, FLE 05, NAV 01].

A forma de cálculo da *similaridade* varia conforme o contexto e os tipos de dados. Para os vários tipos de dados, existem diferentes métricas para determinar a similaridade, com maior ou menor sucesso. Para *strings*, a quantidade de caracteres iguais pode ser uma medida de similaridade, assim como o esforço para transformar uma *string* em outra (comumente chamada *distância de edição*) [NAV 01]. Para blocos de texto, a comparação com base nas freqüências dos termos contidos é uma das métricas adotadas [BRO 98]. Com dados complexos, abordagens que representam os dados como vetores no espaço e calculam a similaridade destes vetores são utilizadas [CAR 03, DOR 04].

Dentre os dados complexos, incluem-se os dados estruturados e semi-estruturados. Dados *estruturados* são aqueles que seguem um formato rígido para organizar suas in-

formações. Como exemplo temos as tuplas em um banco de dados relacional [DAT 03]. Já os dados *semi-estruturados* seguem um formato para organizar suas informações tal como os estruturados, porém este formato não é tão rígido.

Apresenta-se neste capítulo o conceito de cálculo de similaridade em dados semi-estruturados. Inicia-se com uma justificava para a pesquisa na área através da citação de algumas das possíveis aplicações do cálculo de similaridade entre dados. Em seguida faz-se uma breve revisão histórica, a fim de situar o momento atual em que a pesquisa sobre o tema encontra-se. Introduz-se, a seguir, alguns conceitos básicos (não exaustivos) relacionados ao cálculo da similaridade entre dados. Por último, sugere-se uma taxionomia para classificar os trabalhos relacionados à similaridade entre dados semi-estruturados.

2.2 Aplicações

As aplicações da análise de similaridade entre dados são inúmeras. Uma delas é a determinação de elementos equivalentes, ou muito próximos, em processos de integração de esquemas de dados, onde precisa-se determinar os elementos que são equivalentes entre os diversos esquemas. Neste caso é comum o uso de dicionários (*thesaurus*) e técnicas de *similaridade de strings*. Para alguns casos (como dados semi-estruturados), é necessária uma comparação estrutural entre os elementos [MEL 02b, MEL 02a].

A limpeza de dados (*data cleasing*) é outra aplicação importante. A existência de informações duplicadas devido a pequenas inconsistências nas mesmas (erros de grafia ou omissão de informações) é muito comum em bancos de dados, especialmente no contexto de *data warehouses*. Uma limpeza dos dados resultantes de consultas sobre eles faz-se necessária, visando melhorar sua consistência [WEI 04, ANA 02].

A pesquisa por um dado, tendo-se apenas uma vaga definição do que se deseja, é outra aplicação do cálculo de similaridade entre dados. Neste caso, fornece-se uma representação aproximada do dado desejado e o motor de busca encarrega-se de determinar os elementos mais equivalentes ao informado, retornando-os em um "*ranking*" conforme a similaridade. Um exemplo desta aplicação seria uma pesquisa em uma biblioteca digital, onde a partir de informações aproximadas pode-se buscar os dados de uma determinada

obra [DOR 04, PAR 05].

A clusterização de dados também é um processo que pode aproveitar as métricas de similaridade. Neste caso, determinam-se os elementos similares, para posteriormente agrupá-los em classes. Para este tipo de aplicação, métricas baseadas em espaços vetorais geralmente são utilizadas [YAN 05, MA 05, CAR 03].

A integração de dados beneficia-se também dos avanços no cálculo da similaridade entre dados. Como a integração de dados deve detectar a existência de dados duplicados entre fontes de dados, é necessária uma maneira de determinar a semelhança entre os dados destas fontes e em caso de dados considerados equivalentes, realizar a integração dos mesmos [MEL 02a, CAR 03, LEN 02, ANA 02].

Um último exemplo de uso para as métricas de similaridade seria a determinação de textos semelhantes (ou seja, que tratam do mesmo assunto) e textos contidos (determinação de textos que sejam partes de outros textos). Dentre as abordagens existentes, uma comum é a geração de extratos dos textos sendo comparados e a análise da quantidade de extratos em comum que eles possuem, através de métricas de similaridade entre conjuntos [BRO 98].

2.3 Histórico

A pesquisa sobre a similaridade entre dados semi-estruturados têm sua origem em outra pesquisa, referente a dados estruturados. Nesta área, encontram-se trabalhos que se preocupam em estabelecer métricas para determinar as diferenças entre dados organizados em árvore [TAI 79, LU 79]. Esta pesquisa evoluiu para o que hoje entende-se como a *distância de edição entre árvores*, uma das formas mais utilizadas para determinar a semelhança entre árvores (quanto menor a distância, mais similares). Trabalhos relacionados a essa abordagem introduzem formas de determinar seqüências de passos aplicáveis sobre uma árvore com o objetivo de transformá-la em outra. Desta forma, pode-se calcular a distância pela quantidade de passos necessários para transformar uma árvore na outra [SHA 90, WAN 94, SHA 97].

Com adoção do XML, a necessidade da comparação de documentos XML levou a

busca por uma adaptação das métricas de similaridade para dados estruturados de forma que pudessem ser aplicadas a dados semi-estruturados [CHE 00, WAN 03, MAR 01]. Vale observar que a distância de edição entre árvores começou como uma métrica para determinar as alterações entre versões de uma mesma árvore e acabou sendo adotada para a comparação entre árvores. Muitos dos trabalhos existentes hoje em relação à similaridade entre árvores são baseados em trabalhos prévios referentes à detecção de alterações [FLE 05, CHA 97, CHA 96, MAR 01].

A pesquisa em similaridade entre dados semi-estruturados recente vem sendo trabalhada sob diversos enfoques, não ficando restrita às pesquisas originadas da similaridade entre árvores. Trabalhos recentes utilizam conceitos como a freqüência dos termos contidos nos dados como indicador de similaridade, assim como outras abordagens como *séries temporais* [FLE 05] e *espaços vetoriais* [CAR 03].

2.4 Métricas

A maioria dos trabalhos existentes na literatura calcula a equivalência entre dados determinando uma distância entre eles dentro de um espaço métrico. Quanto menor esta distância, mais similares eles serão. Para estabelecer essa distância, utilizam-se funções de comparação. Estas funções devem apresentar quatro características básicas [DOR 04]:

1. Auto-similaridade: $disim(A, A) = disim(B, B)$
2. Minimialidade: $disim(A, B) \geq disim(A, A)$
3. Simetria: $disim(A, B) = disim(B, A)$
4. Inequalidade triangular: $disim(A, C) + disim(C, B) \geq disim(A, B)$

disim é uma função que retorna a distância no espaço entre dois elementos sendo comparados. *Auto-similaridade* refere-se ao fato que a distância entre dois elementos iguais deve ser sempre uma constante, independente dos elementos considerados. *Minimialidade* define que a distância entre dois elementos iguais deve ser sempre igual ou

menor que a distância entre dois elementos possivelmente diferentes. *Simetria* afirma que a distância entre um elemento A e outro elemento B é igual a distância entre B e A. Por último, através da *inequalidade triangular* garante-se que a distância entre dois elementos A e B nunca pode ser maior que a distância entre os dois elementos considerando um elemento intermediário C.

Qualquer função que respeite as propriedades de *auto-similaridade*, *minimialidade* e *inequalidade triangular* pode ser considerada uma função métrica. Dentre as funções métricas utilizadas para a comparação de documento semi-estruturados, identificou-se cinco abordagens mais comuns: espaço vetorial, comparação de *strings*, comparação de árvores, séries temporais (*transformadas de Fourier*) e freqüência de elementos.

2.4.1 Espaço Vetorial

Nesta abordagem, utilizada para comparar objetos complexos, os objetos são transformados em vetores segundo o valor de seus atributos e então, através de cálculos utilizando estes vetores (como a *distância Euclediana* ou o cosseno entre os vetores dos objetos), estipula-se a similaridade entre os objetos pela distância ou ângulo entre seus vetores [CAR 03, REG 93]. Quanto menores eles forem, mais similares os objetos serão.

Os vetores são gerados usualmente da seguinte forma: para cada atributo é calculado um valor numérico (através de fórmulas pré-definidas) que o representa, sendo este valor utilizado como uma componente na construção do vetor. Outra possibilidade [CAR 03] é gerar vários vetores para um objeto e fazer uma composição (como, por exemplo, uma soma) entre eles para ter uma representação final do objeto no espaço.

Apesar de adotada para comparação de dados semi-estruturados, esta abordagem apresenta problemas. Dado o alto número de dimensões que o espaço acaba necessitando para comportar uma representação dos dados, a comparação fica prejudicada. Isto porque, considerando que o dado é um vetor no espaço, o número de dimensões do mesmo precisa ser compatível com o tamanho e quantidade de elementos que o compõe [GIO 99].

2.4.2 Comparações entre *strings*

Dentre as diversas formas para identificar *strings* similares no contexto de integração de dados, a que merece destaque é a distância de edição (*edit distance*). Nesta abordagem, operações são realizadas sobre uma *string* (como inserção, remoção, troca e movimentação de caracteres) visando tranformá-la na *string* com a qual está sendo comparada. O custo mínimo em termos de operações necessárias para tal determina a distâncias entre elas e por conseqüente sua similaridade [NAV 01, COH 03]. A Figura 2.1 exemplifica em linhas gerais a distância de edição entre *strings*. Nesta calcula-se o custo de transformar a palavra *qaurema* na palavra *quaresma*.

Figura 2.1: Exemplo de distância de edição entre *strings*

Palavra 1	**Palavra 2**
`qaurema`	`quaresma`
Possíveis operações: *(i)* Duas exclusões (**a** e **u** e três inclusões (**u**, **a**, **s**) *(ii)* Duas trocas (**a** por **u** e **u** por **a**) e uma inclusão (**s**) *(iii)* Uma inversão (**a** por **u**) e uma inclusão (**s**)	

Caso existam pesos diferentes associados às operações (por exemplo, o custo de substituir um caracter é maior que inserir um novo caracter ou excluir) denomina-se a técnica de *distância geral de edição*. Caso os pesos sejam únicos para todas as operações, classifica-se a técnica como *distância de edição simples*, ou simplesmente *distância de edição* [NAV 01].

Diversas técnicas baseadas na distância de edição surgiram através de limitações impostas sobre as operações permitidas na transformação de uma *string* na outra. Dentre elas, tem-se *Levensthein*, que permite inserções, exclusões e substituições; *Hamming*, que permite apenas substituições (usada para detectar palavras escritas de forma errada); *Episode*, que permite apenas inserções (usada em buscas); e *Longest common subsequence*, que permite apenas inserções e exclusões, sendo utilizada para determinar a maior seqüência de caracteres comum entre as *strings* comparadas [NAV 01, COH 03].

Outra métrica de comparação entre *strings* é a *Jaccard Similarity*, que propõe tranformar cada *string* em um conjunto de *tokens* e estabelecer a semelhança entre eles através da razão entre o tamanho dos conjuntos de intersecção e união dos conjuntos de *tokens* das *strings* [NAV 01]. A Fórmula 2.1 apresenta a proposta de Jaccard para comparar os conjuntos de *tokens* A e B de duas *strings s* e *t*.

$$sim(s,t) = \frac{A \cap B}{A \cup B} \tag{2.1}$$

Monge e Elkan [MON 96] propõe uma métrica conhecida como *Esquema de Reconhecimento Recursivo*. Ela consiste em dividir as *strings* a serem comparadas em *tokens* e, para cada dupla de *tokens*, aplicar uma função de *distância de edição* e ao final ponderar as *n*-comparações em um único valor que será a semelhança entre as *strings*. A Fórmula 2.2 descreve a abordagem de Monge e Elkan.

$$sim(s,t) = \frac{1}{K} \sum_{i=1}^{K} \max_{j=1}^{L} \; sim'(A_i, B_j) \tag{2.2}$$

Um ponto importante, sobre estas métricas baseadas em *tokens*, é que elas buscam determinar *strings* ou textos que tenham aspectos em comum. Elas podem acabar considerando semelhantes cadeias de *tokens* distintas, desde que contenham elementos em comum.

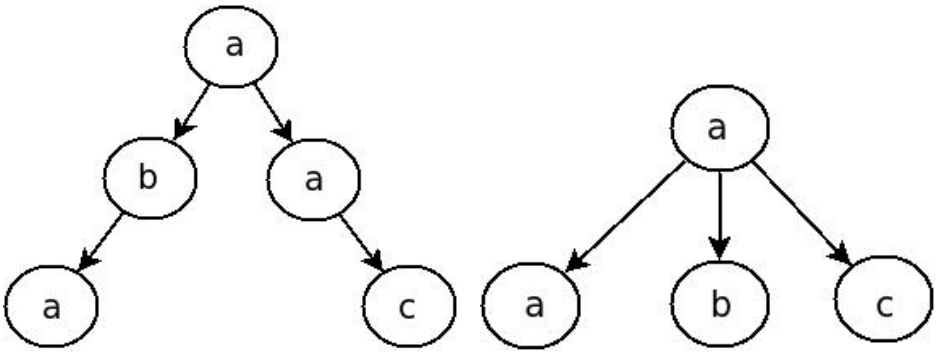
2.4.3 Comparações entre árvores

Uma vez que, em geral, documentos estruturados podem ser representados através de uma árvore, uma comparação importante para determinar a similaridade entre eles é determinar quão similares são as árvores que os representam. Para a comparação estrutural entre árvores, buscando determinar sua similaridade, uma das métricas mais utilizadas é a *distância de edição de árvores*, originada a partir da *distância de edição para strings*. A aplicação da métrica de distância de edição para árvores vem de sua aplicação para grafos, que são sua forma generalizada [NIE 02].

Esta abordagem segue o mesmo princípio da distância de edição para *strings*, determinando um conjunto de operações necessárias para converter uma árvore na outra. Cada

operação, tal como no caso das *strings*, pode ter um peso associado, sendo o custo total da conversão o grau de similaridade entre as árvores. A Figura 2.2 demonstra um possível cálculo de distância de edição, ao transformar a árvore à esquerda na árvore à direita.

Figura 2.2: Exemplo do cálculo de distância de edição entre árvores



Possíveis custos:
(i) Remoção do elemento **(b)** e do elemento **(a)**, e inclusão do elemento **(b)** sob **a**. Remoção do elemento **(c)** e inclusão do mesmo sob **(a)**. *(ii)* Remoção dos elementos **(a)** e **(c)**. Renomeação do elemento **(b)** para **(a)** e do elemento **(a)** para **(b)**. Adicão de um elemento **(c)** sob o elemento **a**.

O cálculo da distância de edição entre árvores, se não for limitado em termos de operações e seqüência de operações possíveis, é um problema NP-Completo [MES 03]. Portanto, é comum restringir-se o conjunto e a seqüência de operações que podem ser aplicadas, para evitar que a determinação da seqüência de operações de menor custo, que indica a similaridade entre as árvores, tenha um custo muito alto de cálculo. Busca-se em geral um custo mínimo não-ótimo quando se compara árvores usando a distância de edição [NIE 02].

2.4.4 Séries temporais e *Transformada de Fourier*

Nas abordagens que usam *Transformada de Fourier* [FLE 05], a idéia básica é converter a estrutura dos documentos sendo comparados em séries de números segundo uma fórmula comum[1]. Feito isto, um sinal (uma onda) é gerado com bases nestes números. Por exemplo, no caso de documentos XML, os números representam as tags que iniciam e terminam um elemento, contribuindo para um aumento ou diminuição do sinal gerado [FLE 05].

Uma vez gerado o sinal, é aplicada sobre ele uma *Transformada Discreta de Fourier* que converte os valores que o compõe para o domínio da freqüência das estruturas. A similaridade entre os documentos é obtida a partir da diferença entre os sinais dos documentos sendo comparados [FLE 05].

2.4.5 Freqüência de elementos

Nesta técnica, a similaridade da freqüência dos elementos que compõem dois dados que estão sendo comparados funciona como um indicador da proximidade do conteúdo dos mesmos. Um dos tipos mais utilizados de comparação com base na freqüência de elementos é a *IDF* (*Inverse Document Frequency*). Na métrica *IDF*, os elementos considerados mais raros (baseando-se na ocorrência deles na massa de dados sendo considerada) têm um peso maior na similaridade dos dados que os contém.

A idéia por trás da métrica *IDF* fica mais clara se for considerada, por exemplo, uma busca por textos similares a um texto que contém os termos *Indústria* e *Souza*. Se os textos sendo pesquisados contêm com muita freqüência o termo *Indústria*, isto não deve contribuir muito para indicar sua semelhança com o conteúdo desejado, pois é um termo comum. Porém, se contêm com pouca freqüência o termo *Souza*, isto significa que a possibilidade de uma relação com o conteúdo desejado é alta, haja vista que *Souza* é um termo incomum.

[1] Um exemplo desta conversão é apresentado no Capítulo 3.

2.5 Taxionomia

A partir das abordagens de similaridade apresentadas, e as possíveis formas de aplicá-las na comparação de dados semi-estruturados, pode-se definir uma taxionomia para classificar tanto estas abordagens quanto trabalhos relacionados à similaridade entre dados semi-estruturados em geral, a partir do tipo de comparação feita. A taxionomia sugerida tem quatro categorias: comparação *de conteúdo*, comparação *de elementos complexos*, comparação *de estrutura* e comparação *completa*.

A categoria relacionada à comparação de ***conteúdo*** refere-se às métricas que tratam de valores atômicos (*strings*, números, etc.). Como exemplo, temos a *distância de edição de string*, *Jaccard Similarity*, etc.

A categoria de ***dados complexos*** refere-se aos trabalhos que comparam elementos complexos, compostos por um ou mais elementos simples, através da combinação de métricas relacionadas à comparação de conteúdo. Incluem-se aqui tuplas de um banco de dados relacional, estruturas de lista, etc. As métricas relacionadas à comparação através de vetores no espaço são um exemplo desta categoria.

Na categoria de ***comparação de estrutura*** incluem-se trabalhos que procuram por similaridades entre objetos estruturados ou semi-estruturados, analisando apenas a forma como os dados estão estruturados. Eles podem considerar apenas a estrutura dos dados, como também os próprios dados como parte da estrutura. Estas abordagens, porém, não respeitam a semântica relacionada aos valores, por não tratar de forma diferenciada a estrutura dos dados. Um exemplo desta categoria é a *distância de edição entre árvores* [NIE 02].

A última categoria é a dos trabalhos que realizam uma ***comparação completa*** considerando, além da estrutura, o conteúdo dos valores contidos nos objetos comparados, respeitando sua semântica. As abordagens apresentadas no próximo capítulo, referente aos trabalhos relacionados à similaridade entre dados no formato XML, em sua maioria fazem parte desta categoria.

Tabela 2.1: Taxionomia da similaridade entre dados semi-estruturados

Categoria	Considera	Exemplo
Conteúdo	*Strings*, números, etc.	Distância de edição
Dados complexos	Elementos complexos	Comparação vetorial
Estrutura	Dados semi-estruturados	Distância de edição
Completa	Dados semi-estruturados	Diversos[a]

[a] Apresentados no Capítulo 3.

2.6 Conclusão

Neste capítulo foi feita uma breve revisão do conceito de similaridade entre dados. Através das possíveis aplicações, demonstrou-se sua importância no contexto atual dos sistemas de informação. Analisando trabalhos relacionados e literatura clássica sobre o tema, identificou-se as abordagens mais comuns na comparação entre dados aplicadas para dados semi-estruturados. Para cada tipo de comparação, uma análise resumida foi feita.

A partir destas análises, foi estabelecida uma taxionomia, resumida pela Tabela 2.1, para classificar as abordagens encontradas e também classificar os trabalhos relacionados em relação ao uso da similaridade entre dados semi-estruturados, que é o tema do próximo capítulo.

Capítulo 3

Trabalhos relacionados

O Capítulo 2 apresentou um estudo sobre as diferentes formas de detecção de similaridade entre dados de uma maneira geral, analisando as abordagens mais comuns existentes e que servem como suporte para a comparação de dados semi-estruturados. A partir deste estudo, definiu-se uma taxionomia para classificar as abordagens de acordo com o tipo de informação que elas manipulam, sob a ótica da comparação de dados semi-estruturados.

Neste capítulo, a partir desta taxionomia, faz-se uma revisão dos trabalhos relacionados à detecção de similaridade entre dados semi-estruturados, em especial documentos XML. Ela está estruturada de acordo com a taxionomia proposta e visa estabelecer o estado da arte do tema. A partir desta revisão, uma análise comparativa entre os trabalhos é feita sob três enfoques distintos: relacionado a características gerais, relacionado ao tratamento da estrutura e relacionado ao respeito à semântica das informações. A partir desta análise, algumas necessidades dos trabalhos, para que produzam um bom resultado, são identificadas.

3.1 Comparação por conteúdo e de dados complexos

Na proposta de **A. Broder [BRO 98]**, introduz-se uma forma de determinar a *contenção* e a *similaridade* entre textos. Por *contenção* entende-se o percentual do conteúdo

de um texto que está contido dentro de outro texto. Por *similaridade* entende-se o percentual de conteúdo em comum que dois textos possuem.

Para calcular a contenção e a similaridade, a abordagem introduzida sugere transformar os documentos em sequências de *tokens* (palavras ou partes de frases) de um tamanho específico chamadas *shingles*. Para gerar estes *shingles*, uma das abordagens sugeridas é deslizar uma *janela* do tamanho de um *shingle* sobre o texto, gerando *shingles* únicos à medida que a mesma desliza. Por exemplo, dada a frase "Um processo de comparação de dados", os shingles com comprimento de duas palavras, ao deslizarmos a *janela* sobre este texto, são os seguintes: (Um processo, processo de, de comparação, comparação de, de dados).

Uma vez definidos os conjuntos de *shingles* dos textos, calcula-se a contenção e similaridade entre eles. Estabelece-se assim os valores de contenção e similaridade entre os textos. A Figura 3.1 apresenta duas formas para contenção e similaridade entre os conjuntos de tokens *A* e *B* de dois textos, apresentadas por Broder et al. [BRO 98].

Figura 3.1: Contenção e similaridade entre dois conjuntos A e B

Contenção: $\frac{\|A \cap B\|}{\|B\|}$	Similaridade: $\frac{\|A \cap B\|}{\|A \cup B\|}$

Em relação a dados semi-estruturados, se for abstraída a estrutura, pode-se aplicar estas métricas de contenção e similaridade. Tal forma de comparação, porém, não seria ideal, pois apresentaria alguns problemas: *(i)* ela ignora a semântica dos dados, estabelecida por sua estrutura; *(ii)* o fato de dados semi-estruturados não serem necessariamente ordenados permite que dois documentos possuam a mesma informação em ordens diferentes, o que prejudicaria o resultado da comparação.

A abordagem de A. Broder [BRO 98] pode ser útil, porém, na comparação de elementos textuais longos em dados semi-estruturados. Exemplos podem ser a crítica a um livro, a biografia de um autor, etc.

No trabalho de **Carvalho et al. [CAR 03]** propõe-se uma abordagem para identificar objetos similares usando um espaço vetorial. Quatro métodos para construir e com-

parar os objetos são introduzidos. No primeiro método, o vetor representando o objeto é construído levando em conta todos os valores dos atributos do objeto - faz-se uma abstração de sua estrutura, considerando apenas o conteúdo. A similaridade é calculada pelo *cosseno* entre os dois vetores. No segundo método, apenas um subconjunto dos atributos é utilizado para construir o vetor abstraído dos objetos. Este subconjunto é especificado por um usuário especialista através da identificação de atributos que sejam *semanticamente equivalentes e relevantes*.

O terceiro método constrói um vetor independente para cada atributo a ser considerado na comparação. Os atributos utilizados são escolhidos da mesma forma que no segundo método. A similaridade é calculada pela soma das similaridades dos atributos. O último método trabalha da mesma forma que o terceiro, porém considera a definição de *pesos* para cada um dos atributos considerados, de forma a estabelecer o grau de importância dos mesmos na comparação.

Em relação à aplicação do trabalho no domínio de documentos semi-estruturados, o trabalho não contempla o fato de que determinadas propriedades utilizadas na comparação entre as instâncias XML podem não estar presentes em todas as instâncias. Também não considera-se a hierarquia complexa dos documentos XML (haja vista que não é seu foco principal), não estabelecendo uma forma de compor a similaridade na hierarquia dos documentos.

Dorneles et al. [DOR 04] propõe um conjunto de métricas para comparar coleções de dados em documentos XML. O objetivo destas métricas é possibilitar consultas complexas em coleções de documentos a partir da especificação de uma consulta através de um objeto complexo - uma árvore de dados.

Para realizar estas consultas, o problema é dividido em duas partes: a comparação de valores atômicos e de valores complexos. Para a comparação de valores atômicos (*strings*, números, datas, etc.) são propostas as *MAVs - Metrics for Atomic Values*. Estas métricas são responsáveis por estabelecer um grau de similaridade entre valores atômicos e são utilizadas pelas métricas para tratar valores complexos: as *MCVs - Metrics for Complex Values*. As *MCVs* possibilitam a comparação entre estruturas de dados complexas - definem-se no trabalho métricas para tratar tuplas, listas e coleções.

Para tuplas, a métrica *tupleSim* é estabelecida. Ela define a similaridade como a média da similaridade entre os atributos de mesmo nome entre as tuplas. A média é feita com base no dado complexo com o maior número de propriedades. Portanto, se a comparação ocorre entre elementos com 5 e 4 propriedades, a média será feita sobre 5 propriedades - no caso, assume-se que para a propriedade que não existe a similaridade para a mesma será nula - valor zero. As propriedades consideradas na comparação podem ou não ter *pesos* associados, que indicam o *grau de importância* que as mesmas possuem na comparação.

Para listas, a métrica *listSim* é definida. A similaridade é dada pela média da similaridade entre os elementos das duas listas, considerando que elementos em mesma posição devem ser iguais. Desta forma, respeita-se a ordem dos elementos no processo de comparação. A média é feita sempre pela lista com maior número de elementos.

De forma semelhante, a métrica *setSim* é definida para comparar conjuntos. Nesta, cada elemento de um dos conjuntos é comparado com todos os elementos do outro conjunto. O elemento que possuir a maior similaridade com ele é adotado como seu similar - a similaridade entre os dois é a utilizada para estabelecer a média posterior de similaridade entre os elementos dos conjuntos. Da mesma forma que em *listSim*, a média é feita pelo número de elementos do maior conjunto.

Para cada uma das *MCVs*, variações delas são estabelecidas para considerar casos onde o usuário quer a similaridade de parte dos dados complexos sendo pesquisados com relação ao objeto complexo de pesquisa. Estas se denominam *subTupleSim*, *subListSim* e *subSetSim*. A diferença fica no fator de divisão para calcular a similaridade média, definido pelo tamanho do objeto referência da pesquisa.

As métricas estabelecidades por Dorneles et al. [DOR 04], apesar de definidas no contexto de pesquisas em bancos de dados, podem ser utilizadas para o cálculo de similaridade entre documentos XML.

3.2 Comparação por similaridade estrutural

Chawathe et al. [CHA 97] apresenta uma técnica para comparar a estrutura de dois documentos, estruturados como um grafo bipartido, através da técnica conhecida como "*Edge Cover*". Sua abordagem consiste em estabelecer um grafo bipartido, onde cada metade representa a árvore de um dos documentos sendo comparados. Para cada nodo dos grafos é definida uma operação para torná-lo um nodo do outro grafo. Cada uma destas operações gera uma aresta - *edge*. As operações de transformação no grafo incluem inserções, exclusões, atualizações e movimentações dos nodos. Para cada operação, há um custo associado.

Uma vez estabelecidas todas as possíveis arestas para transformar os nodos de um grafo nos nodos do outro grafo, a técnica de "*Edge Cover*" é utilizada para determinar o menor custo para tornar as duas metades do grafo iguais. Para tal, o algoritmo proposto busca o conjunto de arestas que liga todos os nodos entre os dois grafos ao menor custo possível. Este custo é a similaridade entre os dois documentos.

A abordagem de Chawathe et al. [CHA 97] pode ser aplicada no contexto de documentos XML, porém tende a não apresentar bons resultados, pois não considera aspectos como a hierarquia dos dados (os nodos podem ser movidos de forma a chegar a um custo mínimo sem respeito à hierarquia) e a semântica definida tanto pelo nome dos elementos quanto pela própria hierarquia. Ela não diferencia dados de estrutura, podendo realizar transformações semânticas incorretas por terem um custo menor que uma transformação semanticamente correta.

Chawathe et al. [CHA 96] apresenta um algoritmo heurístico de detecção de alterações para estabelecer um *script* que define a transformação de um documento estruturado em outro, podendo ser utilizado como uma forma de estabelecer a similaridade entre eles. Este trabalho, porém, tal como no anterior [CHA 97], não considera a semântica dos dados durante a geração do *script* de transformação.

Flesca et al. [FLE 05] apresenta um abordagem que utiliza o conceito de Séries Temporais para a comparação da estrutura de dois documentos XML. Para realizar a comparação, uma *onda* é gerada para cada documento com base na sua estrutura e no

seu conjunto de *tags*. A comparação entre as ondas é feita através da aplicação da *Transformada Discreta de Fourier*, que as leva para o domínio da freqüência.

Para gerar a onda de cada documento, o processo sugerido trabalha da seguinte forma: através de uma varredura pelas estruturas dos documentos, uma seqüência de valores é estabelecida. Algumas propostas são adotadas para estabelecer esta seqüência de valores. Uma das propostas é associar a cada tipo de *tag* encontrada um valor e gerar as seqüências de valores seguindo esta lógica, considerando apenas as *tags* de abertura dos elementos. Outra proposta é também associar um valor a cada *tag*, sendo definidos valores positivos ou negativos conforme encontra-se a *tag* de abertura ou de encerramento de um elemento, respectivamente.

A Figura 3.2 demonstra exemplos destas codificações sugeridas. Na primeira codificação, cada *tag* encontrada tem um valor atribuído, ocorrendo isto apenas nas *tags* de abertura dos elementos. A segunda codificação associa a cada *tag* um valor, onde este valor é positivo ou negativo conforme a *tag* esteja iniciando ou terminando um elemento.

Figura 3.2: Codificações propostas por Flesca et al. [FLE 05]

<autor>	autor - 10
<nomeCompleto>	nomeCompleto - 20
<nome></nome>	nome - 30
<sobreNome></sobreNome>	sobreNome - 40
</nome>	
</autor>	Codificação 1: 10, 20, 30, 40
	Codificação 2: 10, 20, 30, -30, 40, -40, -20, -10

Uma vez definida as seqüências de valores - *ondas* - representando a estrutura dos documentos, sobre elas são aplicadas *Transformadas Discretas de Fourier* que as transportam para o domínio da freqüência de estruturas, onde podem então ser comparadas e um grau de similaridade estabelecido.

Como o foco está na similaridade estrutural, a abordagem de Flesca et al. [FLE 05] tem utilidade, por exemplo, em contextos de clusterização de documentos que contêm es-

truturas próximas. Entretanto, para a comparação *completa* de documentos XML ela não se apresenta como uma solução adequada, pois não considera os dados dos documentos.

Nierman et al. [NIE 02] usa o conceito de distância de edição entre árvores representando documentos XML como uma métrica para a similaridade estrutural entre os mesmos. Em relação à distância de edição tradicional, sua abordagem inova ao considerar aspectos como a movimentação de sub-árvores.

Para realizar a comparação, define-se um conjunto básico de operações que podem ser realizadas. Este conjunto limitado busca reduzir o custo de cálculo na distância de edição entre as árvores sem provocar um grande redução na qualidade da comparação. As operações permitidas incluem renomeação, inserção e remoção de nodos, assim como inserção e exclusão de sub-árvores.

Segundo Nierman et al. [NIE 02], as operações de exclusão e inclusão de sub-árvores indicam melhor a similaridade entre as estruturas dinâmicas de documentos XML, pois são tratadas a um *custo* menor que a remoção e inclusão de todos os elementos das mesmas. Tais operações, porém, são limitadas a alguns critérios definidos que impedem, por exemplo, que se exclua toda a árvore de um documento e se inclua a árvore do outro documento. Se tal operação fosse permitida, o custo para transformar um documento no outro poderia ser extremanente reduzido e não representaria corretamente a distância entre os documentos.

Da mesma forma que nos trabalhos citados anteriormente, características particulares de documentos XML, como o fato de não serem necessariamente árvores ordenadas (quando vêm de esquemas diferentes ou não tem esquemas) não são consideradas por Nierman et al. [NIE 02]. Além disso, trabalha-se apenas a similaridade estrutural entre os documentos, que no caso da comparação *completa* entre documentos XML não é adequada, pois não representa a similaridade efetiva, em termos semânticos, entre os documentos.

O trabalho de **Melnik et al. [MEL 02b]** define um algoritmo para gerar um mapeamento entre os nodos de dois grafos, baseado nas suas similaridades. Considera-se que dois nodos são similares se seus vizinhos correspondentes (ligados por arestas de mesmo rótulo) são similares. O algoritmo propõe uma propagação da similaridade entre

os elementos para seus vizinhos, em uma abordagem chamada *"similarity flooding"*.

O princípio deste trabalho pode ser entendido pelo seguinte exemplo: se dois elementos a_1 e b_1 estão conectados por arestas a a_2 e b_2, e as arestas têm o mesmo rótulo l_1, e a_2 e b_2 são similares, então presume-se que a_1 e b_1 também são similares.

Esta abordagem, no contexto de documentos XML, apresenta o problema de não considerar a hierarquia dos elementos (afinal, é destinada a grafos) durante o processo de comparação, assim como a semântica dos nomes dos elementos de documentos XML sobre o valor contido nos mesmos. Mesmo assim, ela pode ser aplicada para auxiliar um usuário no processo de *schema matching* de documentos XML, ao identificar possíveis equivalências de forma automática.

D. Buttler [BUT 04] usa o conceito de *shingles* definido por A. Broder [BRO 98] para comparar a estrutura de documentos XML. Para tal, define-se os *shingles* representando a estrutura de um documento da seguinte forma: para cada nodo de um documento XML, seu *path* é convertido em um valor através de uma função de *hash* e este valor representa um *shingle*. Tal processo é repetido em todos os documentos que se deseja comparar. Uma vez estabelecidos os *shingles* da estrutura de cada documento, a similaridade entre os mesmos é definida através das operações estabelecidas em [BRO 98].

Conforme demonstrado por D. Buttler [BUT 04], esta abordagem apresenta bons resultados para estimar a similaridade entre a estrutura de documentos XML, possibilitando um processo de clusterização dos mesmos. Entretanto, pelas mesmas razões citadas para Flesca et al. [FLE 05], ela não é adequada para a comparação entre documentos XML.

Kailing et al. [KAI 04] apresenta um conjunto de filtros que permite determinar, a um custo reduzido, se é vantajoso comparar duas árvores de acordo com um valor mínimo de similaridade baseado na distância de edição entre árvores. Tais filtros baseiam-se em aspectos como altura das árvores, graus e nomes dos nodos, etc. Além de apresentar tais filtros, introduz-se uma maneira combinada de aplicá-los. Desta forma, consegue-se otimizar o processo de comparação de grandes massas de árvores a partir da aplicações dos filtros, que evitam a comparação custosa via distância de edição. Esta abordagem não tem aplicação direta para a comparação de documentos XML, mas pode auxiliar o processo de comparação de grandes massas de dados, evitando que instâncias muito

distintas estruturalmente sejam comparadas.

3.3 Comparação completa

Na proposta de **Weis et al. [WEI 04]**, trata-se o problema de detectar objetos duplicados dentro de um documento XML. Sua solução é baseada em uma análise iterativa *top-down* na hierarquia dos elementos (objetos) do documento, identificando duplicatas em cada nível.

Dois objetos (elementos) são considerados possíveis duplicatas se eles possuem a mesma *tag*, têm elementos antecessores similares, conteúdo similar e os elementos filhos de seus elementos filhos diretos também possuem conteúdo similar. O conteúdo de um elemento é entendido como seu conteúdo textual e o conteúdo de seus atributos, assim como o conteúdo textual de seus filhos diretos.

Para comparar os objetos, um processo de abstração de sua estrutura é feito, mantendo apenas o conteúdo. Este conteúdo é dividido então em *tokens* e estes *tokens* são comparados por métricas de similaridade para *strings*. Com o objetivo de acelerar o processo, Weis et al. [WEI 04] sugere três filtros para evitar a comparação, por distância de edição, entre *tokens* que não têm possibilidade de serem similares. Estes filtros, detalhados em Weis et al. [WEI 04], analisam os conjuntos de caracteres dos termos para identificar aqueles que não têm a possibilidade de serem similares, conforme um valor máximo de distância de edição adotado.

Para comparar dois objetos, o método proposto utiliza o conceito de *IDF - Inverse Document Frequency* - para estimar a similaridade entre dois objetos a partir do percentual em comum de dados (*strings*) que eles contém em relação ao conjunto de dados não-comuns entre eles. Quanto maior este percentual, mais similares os objetos serão. A Figura 3.3 apresenta as fórmulas sugeridas por Weis et al. [WEI 04] para estimar a similaridade *sim* entre dois conjuntos de tokens S e S', a partir da *IDF* de cada um de seus elementos. A primeira fórmula define que a *IDF* de um conjunto de termos é a soma de suas *IDF*. A segunda fórmula estabelece que a relação entre a *IDF* dos conjuntos de termos em comum e distintos entre dois elementos comparados define a similaridade entre

eles.

Figura 3.3: Similaridade de conteúdo segundo Weist et al. [WEI 04]

$IDF_S(o) := \sum_{o \in S} IDF(o)$	$sim(S, S') := \frac{IDF(S \cap S')}{IDF((S \cup S') \setminus (S \cap S'))}$

A similaridade entre os objetos é calculada e os objetos considerados similares são *agrupados*. Cada *grupo* origina um objeto (eleito aleatoriamente dentro do grupo) que substitui, no documento original, todos os outros objetos do *grupo*.

Tal abordagem apresenta bons resultados, especialmente pelo uso da *IDF* para estimar a similaridade. Porém, ao realizar a abstração dos atributos do elementos e também dos seus elementos filhos, ela perde a semântica representada pelas *tags*. Em domínios onde a semântica associada tem importância na definição da similaridade entre os dados, assim como onde há a possibilidade destas *tags* identificarem valores similares textualmente, porém distintos semanticamente, esta abordagem pode gerar resultados incorretos.

Outra questão não tratada em [WEI 04] é que um dos objetos sendo comparados pode conter muito mais informação que outro, como, por exemplo, um bloco textual longo, e este bloco impede que seja identificada corretamente a similaridade entre os objetos. Outro aspecto em relação à abordagem é o fato que, apesar de dois objetos apresentarem uma estrutura diferente em termos de nomes de elementos e estrutura, eles podem a vir ser casados. Entretanto, um casamento deste tipo apresentará problemas em um posterior processo de integração. A abstração da estrutura também impede, por exemplo, que se defina pesos às propriedades dos objetos sendo comparados, que poderiam ser utilizados como indicadores semânticos da relevância das propriedades no processo de comparação.

Weis et al. [WEI 04], apesar da abstração da estrutura realizada, a faz de forma parcial, conseguindo assim respeitar em parte a estrutura e semântica das informações. Tal aspecto leva sua abordagem a ser classificada, na taxionomia apresenta, como um exemplo de *Comparação completa*.

Em **Weis et al. [WEI 05]** introduz-se um *framework* para detecção de objetos duplicados em documentos XML. Durante o processo de análise considera-se a estrutura,

contexto e relações dos objetos. Este processo, porém, baseia-se no provimento, por um especialista, dos tipos de objetos duplicados que devem ser identificados dentro do documento, assim como a definição do que compõe a *identificação* do objeto. Esta definição dos objetos é utilizada para compará-los, utilizando uma métrica de similaridade adequada (como a apresentada em [WEI 04]), para identificar as duplicatas.

Para que um especialista defina tais informações, utiliza-se o conceito de OD - *Object Descriptor*, que representa as propriedades a serem consideradas nos objetos a serem comparados. Supõe-se, portanto, que as instâncias estão sob um ou mais esquemas de conhecimento de um usuário especialista, de forma que ele possa definir, a partir de seus esquemas, os *Object Descriptors* necessários.

Ma et al. [MA 05] introduz um método para determinar a similaridade assimétrica entre documentos XML, que utiliza a estrutura como guia para determinar quais dados devem ser comparados. A abordagem sugerida compara documentos sob um mesmo esquema, estabelecendo um grau de similaridade a partir dos dados compatíveis (que estejam sob a mesma hierarquia de elementos), definindo a similaridade entre dois elementos pela média da similaridade de seus atributos e conteúdo textual.

Esta abordagem diferencia-se de outros trabalhos ao tratar de forma diferenciada a comparação entre coleções de valores, estabelecendo uma forma assimétrica de definir a similaridade destas coleções. Esta forma segue a seguinte lógica: quanto maior a quantidade relativa de elementos de uma coleção contida em outra, mais similar ela será. Esta comparação assimétrica, porém, pode não prover bons resultados ao comparar instâncias mais complexas. Se uma das instâncias possuir mais dados em determinados atributos que a outra, o resultado desta abordagem poderá não ser o mais adequado, pois ela poderá considerá-las distintas, mesmo não o sendo.

Park et al. [PAR 05] introduz um sistema de busca em coleções de documentos XML que contêm dados similares a um determinado dado em um contexto. Para definir este contexto, o dado a ser pesquisado tem associado um *path* em XML. Durante o processo de pesquisa, diferenças entre o *path* informado na pesquisa e os *paths* presentes nos documentos pesquisados são analisadas, de forma a detectar *paths* similares ao da pesquisa, levando em conta sua semântica.

Apesar de ser uma abordagem destinada a pesquisa em documentos XML por dados pontuais, a abordagem introduzida por Park et al. [PAR 05] para aproximação de *paths* em documentos XML pode ser utilizada em processos de comparação onde é necessário detectar elementos comuns em *paths* ligeiramente diferentes.

Puhlmann et al. [PUH 06] apresenta um algoritmo para detecção de elementos duplicados que faz uma adaptação do método *SNM (Sorted Neighbor Model)* para a detecção de objetos duplicados em um documento XML. O método *SNM* é composto de três passos básicos: *(i)* atributos chave (estabelecidos por um usuário especialista), que definem os elementos a serem analisados, são extraídos, gerando *chaves* para a comparação; *(ii)* as *chaves* extraídas são ordenadas alfabeticamente; e *(iii)* as *chaves* ordenadas são varridas por uma *janela*, e a comparação ocorre apenas entre *chaves* dentro da mesma janela. Através deste método, reduz-se consideravelmente o número de comparações necessárias para detectar elementos duplicados.

Tal como em Weis et al. [WEI 05], assume-se uma configuração, antes do processo de comparação, por um usuário especialista, que define os objetos a serem comparados e por quais valores eles serão comparados. Além disto, todos os objetos sendo comparados devem estar sob um mesmo esquema.

Em **Weis et al. [WEI 06]** apresenta-se um método para detectar duplicatas em documentos XML considerando relacionamentos mais complexos, envolvendo nodos pais e filhos com cardinalidades múltiplas, por exemplo. Tal abordagem, porém, requer que as instâncias sendo comparadas estejam sob um esquema integrado e que, um usuário especialista, provenha as informações necessárias à comparação e identificação dos elementos a serem analisados, assim como suas relações.

Wan et al. [WAN 06] e **Yang et al. [YAN 05]** introduzem métodos para realizar a clusterização de documentos XML com base em seus conteúdos e estruturas. Em Wan et al. [WAN 06], a similaridade entre os documentos é calculada através da *Proportional Transportation Similarity*, onde identificam-se documentos similares comparando o conteúdo dos mesmos. Neste comparação, os elementos onde estes dados estão contidos são considerados (inclusive sua hierarquia), porém permite-se que dados de elementos diferente sejam comparados desde que respeitem determinadas condições. Estas condições

visam permitir comparar apenas elementos próximos segundo os critérios estabelecidos no trabalho.

Em Yang et al. [YAN 05], a similaridade entre os documentos é calculada pela representação dos mesmo no *Structured Linked Vector Model* (SLVM). Através de uma expansão do mesmo em SLVM-LSI (*Structured Linked Vector Model Latent Semantic Indexing*, os autores incluem na comparação aspectos semânticos dos documentos. Sua proposta inova ao apresentar um método iterativo de aprendizado para calcular a similaridade entre os documentos pelo SLVM, diferente de outras abordagens que utilizam algoritmos heurísticos.

Em Yang et al. [YAN 05] e Wan et al. [WAN 06], como foca-se na clusterização de documentos, seus métodos de comparação são aproximados e não se adequam-se a uma determinação de similaridade entre documentos com foco em qualidade semântica.

Milano et al. [MIL 06] apresenta um método para calcular a distância de edição entre árvores de instâncias XML que leva em conta a semântica dos elementos. Sua abordagem pressupõe que as instâncias comparadas estejam sob esquemas compatíveis, ou seja, os dados equivalentes contidos nas instâncias devem estar sob a mesma hierarquia. Partindo desta premissa, compara-se os dados sob o mesmo *path* para identificar a similaridade entre as instâncias.

A abordagem apresenta um algoritmo que realiza a comparação e casamento *bottom-up* entre os elementos das instâncias, onde a unidade mínima de similaridade adotada é a distância de edição entre os textos dos nodos folhas das instâncias comparadas. A distância de edição entre os nodos mais complexos é calculada como a soma das distâncias de edição de seus nodos inferiores.

Com relação à esta abordagem, um ponto a comentar é que não se considera que alguns atributos podem ter relevâncias diferentes para a similaridade de um elemento e que, portanto, deveriam ter um *peso* diferenciado (como em Dorneles et al. [DOR 04]). Por exemplo, entre dois autores, o atributo *nome* deve ter uma relevância maior na similaridade que o atributo *url*. É comum um autor mudar o endereço de seu site, mas não o seu nome. Outro aspecto a comentar é que as instâncias comparadas devem estar sob um esquema compatível. No contexto de várias fontes de dados na Web, este fato não pode

ser garantido.

3.4 Análise comparativa e considerações

Nesta seção faz-se uma análise comparativa entre os trabalhos relacionados estudados, através de três tabelas que comparam aspectos gerais, estruturais e semânticos das abordagens.

Tabela 3.1: Comparativo geral entre os trabalhos relacionados

Trabalho	Foco	Categoria	Abordagem	Escalabilidade
[BRO 98]	Textos	Dados simples e complexos	*Shingles*	Pequenas e médias
[CAR 03]	Objetos complexos	Dados simples e complexos	Distância vetorial	Pequenas e médias
[DOR 04]	Objetos complexos	Dados simples e complexos	*MCV* e *MAV*	Pequenas e médias
[CHA 97]	Árvores	Similaridade estrutural	*Edge Cover*	Pequenas e médias
[CHA 96]	Árvores	Similaridade estrutural	Distância de edição	Pequenas e médias
[FLE 05]	Documentos semi-estruturados	Similaridade estrutural	Séries temporais	Pequenas a grandes
[NIE 02]	Documentos semi-estruturados	Similaridade estrutural	Distância de edição	Pequenas e médias
[MEL 02b]	Grafos	Similaridade estrutural	*Similarity Flooding*	Pequenas
[BUT 04]	Documentos semi-estruturados	Similaridade estrutural	*Shingles*	Pequenas e médias
[KAI 04]	Árvores	Similaridade estrutural	Distância de edição	Pequenas a grandes
[WEI 04]	Documentos semi-estruturados	Comparação completa	*IDF*	Pequenas a grandes
[WEI 05]	Documentos semi-estruturados	Comparação completa	*ODs*	Pequenas à grandes
[PUH 06]	Documentos semi-sstruturados	Comparação completa	*ODs*	Pequenas a grandes
[WEI 06]	Documentos semi-estruturados	Comparação completa	*ODs* e *SNMX*	Pequenas a grandes
[WAN 06]	Documentos semi-estruturados	Comparação completa	*PTS*[a]	Pequenas a grandes
[YAN 05]	Documentos semi-estruturados	Comparação completa	*SLVM*[b]	Pequenas a grandes
[MA 05]	Documentos semi-estruturados	Comparação completa	Própria	Pequenas e médias
[PAR 05]	Documentos semi-estruturados	Comparação completa	Própria	Pequenas e médias
[MIL 06]	Documentos semi-estruturados	Comparação completa	Distância de edição	Pequenas e médias

[a] *Proportional Transposition Similarity.*
[b] *Structured Link Vector Model.*

A Tabela 3.1 compara algumas características dos trabalhos relacionados. Em relação ao foco, vê-se um interesse crescente por documentos semi-estruturados (como documentos XML), o que caracteriza a importância que informações estruturadas neste formato têm tomado no contexto atual dos sistemas de informação. Em relação à categoria

dos trabalhos (conforme a taxinomia apresentada), vê-que que, os mais recentes, buscam, em sua maioria, uma comparação que leve em conta estrutura e conteúdo das instâncias, ou seja, uma *comparação completa*.

As métricas e formas de estimar a similaridade são diversas entre os trabalhos. Isto demonstra que, conforme o tema ganha importância, novas abordagens, muitas vezes advindas de outras áreas como IR - *Information Retrieval*, têm sido adaptadas para a pesquisa por similaridade em dados no padrão XML. O objetivo dos trabalhos é outro tópico com razoável diversidade, dados os diversos problemas nos quais a determinação de similaridade é necessária: remoção de duplicatas, detecção de objetos equivalentes para integração de dados, clusterização, etc.

Os trabalhos também variam quanto à escalabilidade, ou seja, ao tamanho das massas de dados que eles podem tratar. Em geral, são mais adequados a massas reduzidas e médias, mas em alguns casos podem ser aplicados a massas maiores, ou seja, tem uma boa escalabilidade, podendo tratar massas de dados com volume crescente de informações. Neste caso, em geral pressupõe alguma configuração ou compatibilização prévia entre os dados, de forma a facilitar a comparação e conseqüentemente melhorar sua performance.

Nos documentos XML, a estrutura tem um papel crucial na semântica das informações contidas, tanto em nível dos nomes dos elementos quanto da hierarquia. Considerando as informações da Tabela 3.2, vê-se que, nos trabalhos relacionados, tal fato é levado em conta e os trabalhos em sua grande maioria evitam abstrair a estrutura e, quando o fazem, buscam uma forma parcial de abstração.

Quando se considera similaridade tendo em vista um processo posterior de integração das instâncias, um aspecto importante é a possibilidade e facilidade de comparar e integrar duas instâncias de dados. No caso de documentos XML, esta facilidade está diretamente ligada a compatibilidade de seus dados (o que indica sua similaridade) e a similaridade de suas estruturas, principalmente em um processo automatizado de integração. Conforme mostra a Tabela 3.2, poucos trabalhos tem um cálculo de similaridade que indica também uma facilidade para integração posterior, ou seja, onde uma alta similaridade também indica com segurança ser possível integrar as instâncias equivalentes com

Tabela 3.2: Quadro comparativo em relação à estrutura dos dados

Trabalho	Abstrai estrutura	Indica facilidade para integração	Necessita estrutura comum entre as instâncias
[BRO 98]	Sim	Não	Não
[CAR 03]	Não	Não	Não
[DOR 04]	Não	Sim	Sim
[CHA 97]	Não	Sim	Não
[CHA 96]	Não	Sim	Não
[FLE 05]	Não	Não	Não
[NIE 02]	Não	Não	Não
[MEL 02b]	Não	Não	Não
[BUT 04]	Sim	Sim	Não
[KAI 04]	Não	Não	Não
[WEI 04]	Parcial	Não	Sim
[WEI 05]	Não	Não	Sim
[PUH 06]	Não	Não	Sim
[WEI 06]	Não	Não	Sim
[WAN 06]	Parcial	Não	Não
[YAN 05]	Parcial	Não	Não
[MA 05]	Não	Sim	Sim
[PAR 05]	Não	Não	Não
[MIL 06]	Não	Sim	Sim

sucesso.

Ainda sobre a Tabela 3.2, percebe-se que dentre os trabalhos analisados, vários assumem que diferenças estruturais entre as instâncias foram sanadas previamente, e a mesmas já possuem uma estrutura comum. Em processos automatizados de integração este fato não pode ser garantido, pois podemos ter fontes de dados dinâmicas (presentes ou não dependendo do momento), com estruturas não conhecidas previamente, que podem apresentar dificuldades para serem compatibilizadas sem um tratamento adequado.

Em relação à semântica dos dados, vê-se, conforme a Tabela 3.3, que a maioria dos trabalhos busca considerá-la, em especial os trabalhos mais recentes. Esta consideração, em geral, fica limitada à identificação dos valores associados às propriedades dos dados no padrão XML e não prevê, por exemplo, a possibilidade de definir uma relevância (através de pesos) àquelas propriedades que sejam mais significativas no processo de comparação sendo executado, dado o contexto onde o mesmo ocorre.

Tabela 3.3: Quadro comparativo em relação à consideração da semântica

Trabalho	Análise	Semântica	Possibilita aplicação de relevâncias às propriedades	Necessita configuração prévia à execução	Usuário especialista necessário	Ontologia
[BRO 98]	Dados	Não considera	Não aplicável	Não	Não	Não considera
[CAR 03]	Estrutura e dados	Considera	Sim	Sim	Sim	Não considera
[DOR 04]	Estrutura e dados	Considera	Sim	Sim	Sim	Não considera
[CHA 97]	Estrutura e dados	Não considera	Não aplicável	Não	Não	Não considera
[CHA 96]	Estrutura e dados	Não considera	Não aplicável	Não	Não	Não considera
[FLE 05]	Estrutura	Não considera	Não aplicável	Não	Não	Não considera
[NIE 02]	Estrutura	Não considera	Não aplicável	Não	Não	Não considera
[MEL 02b]	Estrutura	Não considera	Não aplicável	Não	Não	Não considera
[BUT 04]	Estrutura	Não considera	Não aplicável	Não	Não	Não considera
[KAI 04]	Estrutura e dados	Não considera	Não aplicável	Não	Não	Não considera
[WEI 04]	Estrutura e dados	Não considera	Não aplicável	Não	Não	Não considera
[WEI 05]	Estrutura e dados	Considera	Não	Sim	Sim	Não considera
[PUH 06]	Estrutura e dados	Considera	Não	Sim	Sim	Não considera
[WEI 06]	Estrutura e dados	Considera	Não	Sim	Sim	Não considera
[WAN 06]	Estrutura e dados	Considera	Não	Não	Não	Não considera
[YAN 05]	Estrutura e dados	Considera	Não	Não	Não	Não considera
[MA 05]	Estrutura e dados	Considera	Não	Não	Não	Não considera
[PAR 05]	Estrutura e dados	Considera	Não	Não	Não	Não considera
[MIL 06]	Estrutura e dados	Considera	Não	Não	Não	Não considera

A necessidade de um usuário especialista para pré-configurar e prover informações ao método de comparação é uma característica na qual os trabalhos também diferenciam-se. Entende-se que, uma vez que exista um usuário especialista e um processo de *setup* inicial, os resultados tendem a ser mais eficientes e com maior qualidade. Entretanto, contar com a existência de tal usuário limita a aplicação dos trabalhos em ambientes automatizados de pesquisa e integração de dados.

Um último aspecto a analisar é a possibilidade dos trabalhos relacionados em utilizar uma ontologia durante o seu processo de comparação. Um ontologia, como forma de representação do conhecimento, pode prover algumas informações úteis a um processo de comparação, como uma estimativa da importância de um atributo de um dado.

3.5 Conclusão

Este capítulo apresentou uma revisão da literatura referente à trabalhos relacionados à detecção de similaridade entre dados semi-estruturados, em especial em relação a dados no padrão XML. A partir da taxionomia apresentada no Capítulo 2, fêz-se um estudo dos trabalhos relacionados, descrevendo suas abordagens e destacando alguns aspectos particulares de cada um, assim como algumas de suas limitações. Este estudo originou três tabelas comparativas, que apresentam uma visão geral sobre o estado da arte.

A partir da análise destes trabalhos, e das limitações e necessidades que eles possuem, identificou-se que um dos principais aspectos para garantir sua correta execução é a compatibilidade entre as estruturas das instâncias de dados no padrão XML que estão sendo comparadas. Esta constatação motivou a definição de um conjunto de algoritmos de pré-processamento, apresentados no próximo capítulo, que buscam sanar algumas das possíveis diferenças estruturais entre as instâncias. Estas diferenças, apesar de semanticamente não significativas, prejudicam a qualidade do resultado dos trabalhos estudados.

O estudo dos trabalhos relacionados apresentado neste capítulo, em conjunto com a taxionomia apresentada no Capítulo 2, produziram um artigo apresentado na segunda edição da Escola Regional de Banco de Dados (ERBD 2006) [GON 06].

Capítulo 4

Pré-processamentos

4.1 Introdução

A revisão feita no Capítulo 3 demonstrou que a grande maioria dos trabalhos relacionados à detecção de similaridade entre dados semi-estruturados toma como guia [CAR 03, MIL 06, MA 05] ou utiliza a estrutura das instâncias para orientar [NIE 02, FLE 05, DOR 04] seus processos. Nestes trabalhos, a estrutura pode ser analisada tanto do ponto de vista semântico, respeitando-a e não promovendo alterações [CAR 03, MIL 06, MA 05], como podem também promover alterações na estrutura, ignorando [WEI 04] ou desrespeitando [NIE 02] a semântica dos dados. As diferentes formas de tratar a estrutura das instâncias devem-se ao enfoque e tipo de massas de dados tratados pelos trabalhos.

Em face da necessidade de utilizar a estrutura das instâncias como guia, a maior parte dos trabalhos relacionados apresenta problemas em comparar instâncias caso suas estruturas não estejam compatíveis. Em instâncias de dados no padrão XML a semântica é definida pelo contexto (*path*) onde os dados estão inseridos. O respeito à semântica dos dados de instâncias de dados semi-estruturadas está diretamente ligado ao respeito à hierarquia das mesmas. Os trabalhos que mais respeitam a semântica dos dados [MIL 06, MA 05, DOR 04] assumem instâncias com estruturas iguais ou compatibilizadas previamente. Quando se comparam instâncias originárias de um conjunto de fontes

conhecido e com esquemas disponíveis, a compatibilização de suas estruturas pode ser feita via abordagens tradicionais de *schema matching*. Porém, quando se considera fontes de dados diversas, sem esquemas e sem conhecimento prévio, um processo de *schema matching* torna-se impossível ou bastante prejudicado, conforme seja ou não possível extrair um esboço do esquema das instâncias a partir de suas estruturas.

Contextos como este, sem conhecimento prévio das instâncias envolvidas, tornam necessárias abordagens para a comparação de instâncias que consigam resolver diferenças entre elas relacionadas à estrutura. Este capítulo apresenta uma contribuição para o problema, buscando, através da aplicação de *pré-processamentos* sobre as instâncias, dimininuir as diferenças estruturais entre elas. Estes *pré-processamentos* buscam minimizar as limitações dos trabalhos relacionados com relação à necessidade de estruturas compatíveis entre as instâncias comparadas.

4.2 Problemas existentes

Diferenças de ordem estrutural em instâncias de dados XML, analisadas durante um processo de comparação, tendem a prejudicar a qualidade da comparação. Estas diferenças podem ser tanto na forma de escrita das *tags* que as compõem, quanto na forma como os dados estão organizados nas instâncias. Através de um estudo feito sobre diferentes formas de representar os dados do domínio literário (publicações em eventos, livros, etc.), identificou-se alguns destes problemas.

Em relação às *tags*, elas podem ser escritas em maiúsculas, minúsculas ou misturando maiúsculas e minúsculas. Podem ser compostas por duas ou mais palavras, separadas por um caracter especial ou não (um hífen, um caracter de *sublinhado*). Uma *tag* composta por duas ou mais palavras pode ou não conter preposições e/ou artigos (`dataNascimento` e `data`**`De`**`Nascimento`, por exemplo). Palavras equivalentes podem ser utilizadas para um mesmo conceito: sinônimos ou termos entre diferentes línguas; termos mais genéricos, como por exemplo `carro` e `veículo`; etc..

Em relação à estrutura, um elemento pode ser representado como simples ou complexo. Dois elementos complexos da mesma classe (*tag*) podem não compartilhar a

mesma estrutura, ou seja, um elemento pode ter propriedades [1] que o outro elemento não possui. Outro possível problema é quando a hierarquia das instâncias dadas está invertida. Isto acontece quando um elemento e_x é um antecessor de outro elemento e_y em uma instância, enquanto na outra instância e_x é um descendente de e_y.

Este trabalho apresenta um conjunto de algoritmos de *pré-processamento* (*PPs*) com o objetivo de resolver diferenças de ordem estrutural entre instâncias de dados XML, de forma que trabalhos existentes de comparação possam obter melhores resultados mesmo sem um processo prévio de *schema matching*. Para cada *PP* sugerido, um exemplo do problema que justifica sua aplicação é apresentado, assim como um algoritmo em pseudo-código descrevendo seu funcionamento. Para cada um dos algoritmos, um estudo de sua complexidade foi realizado, a partir da identificação do passo, em cada algoritmo, que apresentava o maior custo de execução. A partir desta identificação, os algoritmos foram estruturados de forma que o número de execuções de tal passo fosse o menor possível. Desta forma, buscou-se reduzir a complexidade dos algoritmos.

Tabela 4.1: Funções básicas de pré-processamento gerais

Definição	**Descrição**
$adicionaMapeamento(x, y, M)$	Adiciona em um mapa informado (*M*) um mapeamento direcionado entre os valores *x* e *y*
$remove(a, L)$	Remove um elemento (a) de um conjunto/lista/mapeamento informado (*L*)
$tamanho(L)$	Retorna o tamanho de uma lista (*L*)
$vazio(L)$	Retorna se uma lista/mapeamento (*L*) está vazia
$tag(e)$	Retorna a *tag* de um elemento (*e*)
$insere(k, L)$	Insere um elemento (*k*) no início de uma lista (*L*)
$insereFilhos(k, L)$	Insere no início de uma lista (*L*) os filhos de um elemento (*k*)
$adiciona(k, L)$	Adiciona um elemento ou lista de elementos (*k*) ao fim de uma lista (*L*)

[1] Por propriedades, entende-se tanto sub-elementos com caráter de propriedade do elemento como também seus atributos. Neste trabalho utilizar-se-á sempre *propriedade* ou *elemento* para simplificar a discussão.

4.3 Funções auxiliares

Algumas funções básicas auxiliam na definição dos algoritmos dos *PPs*. Estas funções são apresentadas nas Tabelas 4.1, 4.2 e 4.3. Esta definição em separado visa facilitar a posterior compreensão dos algoritmos.

Na Tabela 4.1, *adicionaMapeamento* é uma função que armazena ligações entre os valores passados em uma estrutura de mapeamento M indicada. Esta estrutura tem propriedades transitivas permitindo que, por exemplo, a partir de um elemento x alcançar um elemento z caso exista um relacionamento (x, y) e (y, z). Tal suporte fica mais claro no Capítulo 5, onde ele é utilizado para otimizar a execução dos *PPs*. A função *insereFilhos* busca, na instância de dados da qual o elemento *k* faz parte, seus filhos diretos e adiciona-os na lista L.

Dentre as funções apresentadas na Tabela 4.2, algumas delas (*thesEquiv, stopWord, colecaoDe*) dependem da existência de um dicionário semântico, como o *Wordnet* [MIL 95]. Apesar de tal fator poder ser entendido como limitante aos pré-processamentos que utilizam estas funções, entende-se que, como a necessidade e o uso de dicionários semânticos é algo crescente em Sistemas de Informação (haja vista a busca em automatizar processos de comparação e integração de dados, onde identificar termos equivalentes é um passo importante [MEL 02a]), a tendência é que cada vez mais existam tais recursos disponíveis e estes sejam cada vez mais completos.

Dentre as funções constantes na Tabela 4.3, convém destacar *simplificaElemento* e *trocaPai*. A primeira extrai o conteúdo de um elemento e seus descendentes, planificando o elemento e transformando-o de completo em simples. Já *trocaPai* move um elemento da posição em que se encontra no documento para uma posição como filho direto de outro elemento, juntamente com todos os seus descendentes.

4.4 *Preparação Léxica*

Tags em instâncias de dados XML podem apresentar variações na forma como são escritas. Pode-se escrevê-las com letras maiúsculas, minúsculas ou mistas. Pode-se ou

Tabela 4.2: Funções básicas de pré-processamento relacionadas a *tags*

Definição	**Descrição**
$insSublinhados(x)$	Insere, em uma *tag* (*x*), antes de toda letra em caixa alta precedida por outra letra em caixa baixa, o caracter de sublinhado (_).
$substSeps(x)$	Substitui, em uma *tag* (*x*), caracteres separadores (- e .) pelo caracter (_).
$convMinusculas(x)$	Converte as letras de uma *tag* (*x*) para minúsculas.
$simPossivel(a, b)$	Retorna VERDADEIRO se duas *strings* (*a* e *b*) podem ser similares[a]
$thesEquiv(a, b)$	Retorna VERDADEIRO se duas palavras (*a* e *b*) são equivalentes através da consulta a um *thesaurus*[b].
$palavrasEm(a)$	Dada uma *string* (a), divide a mesma em várias *strings* usando o caracter (_) como separador, retornando as *strings* geradas.
$stopWord(a)$	Retorna VERDADEIRO se a é uma *stop-word* (preposição, etc.)[c]
$juntaPalavras(A)$	Junta as *strings* contidas em uma lista (*A*) em uma única *string* utilizando o caracter (_) como separador ao uní-las.
$removePrefixo(k, p)$	Remove de uma string (*k*) o prefixo *p*, se existir, considerando o caracter (_) como separador.
$removeSufixo(k, p)$	Remove de uma string (*k*) o sufixo *p*, se existir, considerando o caracter (_) como separador.
$agrupaNodosPorTag(L)$	Agrupa um conjunto de nodos (*L*) de um documento XML pelas suas *tags*, retornando os conjuntos gerados.
$colecaoDe(a, b)$	Retorna TRUE se *a* é um termo que representa uma coleção de *b* através da consulta a um *thesaurus*.
$renomeia(n, p)$	Troca a *tag* de um elemento (*n*) para outra *tag* (*p*).

[a] Através de fórmulas de baixo custo como as apresentadas em Weis et al.[WEI 04]

[b] Ex: Wordnet [MIL 95]

[c] Através de uma estrutura auxiliar como um dicionário semântico, como por exemplo uma base de dados terminológica [SAN 06].

não separar palavras em *tags* compostas por mais de uma palavra com um separador, que pode ser um "-" (hífen) ou "_" (sublinhado). Estas diferenças na grafia, apesar de

Tabela 4.3: Funções básicas de pré-processamento relacionadas à estrutura

Definição	**Descrição**
$simples(k)$	Retorna TRUE se um elemento (*k*) é um nodo folha (sem filhos).
$tagsAntecessoras(e)$	Retorna, para um elemento (*e*) de um documento, as *tags* dos elementos que compõe o *path* do mesmo no documento.
$filhos(e)$	Retorna os elemento filhos de um elemento (*e*).
$pai(e)$	Retorna o elemento pai de um elemento (*e*).
$simplificaElemento(e)$	Retorna o conteúdo textual de um elemento (*e*), sem a estrutura associada aos mesmos.
$tagsDosFilhos(e)$	Retorna as *tags* dos elementos filhos de um elemento (*e*).
$removeFilhosComTag(e,t)$	Remove, dos filhos de um elemento (*e*), aqueles com *tag* igual a *t*.
$trocaPai(e,p)$	Move um elemento (*e*) para a posição de filho de outro elemento (*p*).
$elementosAbaixo(k)$	Retorna os elementos descendentes de um elemento (*k*).

mínimas, prejudicam algoritmos de comparação que utilizam a estrutura das instâncias como guia. Com o intuito de evitar isto, propõe-se um algoritmo de *PP* denominado *Preparação Léxica*. Este algoritmo busca, através de uma análise da grafia das *tags*, identificar possíveis pontos onde a grafia pode variar, padronizando-as. De forma geral, ele itera sobre o conjunto de *tags* em uma ou mais instâncias de dados XML e executa as seguintes transformações:

1. Insere o caracter "_" (sublinhado) antes de letras em caixa alta se o caracter anterior não é uma letra em caixa alta;

2. Susbtitui caracteres separadores de palavras pelo caracter "_";

3. Converte as letras para minúsculas.

A Figura 4.1 apresenta exemplos de *tags* sobre as quais a preparação léxica foi aplicada. Em relação aos três passos da *Preparação Léxica*, o primeiro passo vem da observação de que é comum colocar em caixa alta a primeira letra de cada palavra quando

Figura 4.1: Exemplos do *PP Preparação Léxica*

***Tags* originais**	***Tags* processadas**
dataDeNascimento, data_de_nascimento	data_de_nascimento
nome_Do_Pai, nome-do-pai	nome_do_pai
nome, Nome, NOME	nome

se tem uma *tag* composta por mais de uma palavra. Não se executa tal passo sobre letras antecedidas por outra letra em caixa alta para evitar que, em *tags* onde têm-se palavras com todas as letras em maiúsculas, coloque-se erroneamente o caracter separador antes de cada letra.

O Algoritmo 1 apresenta o pseudo-código da *Preparação Léxica*. A complexidade é determinada pelo número de vezes que a função *ProcLexica* é executada. Como ela é executada um número de vezes igual ao tamanho da entrada do algoritmo, a complexidade dele é *O(n)*. No algoritmo, para simplificar sua descrição, não se incluiu o passo de varrer as instâncias de dados XML e substituir as *tags* originais pelas *tags* processadas.

Algoritmo 1 : *Preparação Léxica* (*prepLexica - O(n)*)

Entrada: conjunto de termos a compatibilizar - A
Saída: mapeamento entre os termos originais à versão processada pelo algoritmo - M

Algoritmo:

proc $prepLexica(A)$
1. Para cada a em A faça
   $b \leftarrow convMinusculas(substSeps(insSublinhados(a)))$
   2. Se $(a \neq b)$ Então
      $adicionaMapeamento(a, b, M)$ # Armazena a *tag* original e sua versão processada
   Fim Se # para que posteriormente outro algoritmo substitua
Fim Para # as ocorrências da *tag* pela sua versão processada
Retorna M

4.5 *Uniformização da Escrita*

Uniformização da Escrita detecta, pela aplicação de métricas de similaridade entre *strings*, *tags* equivalentes cujas grafias apresentam diferenças. Ele busca resolver casos como:

- Uma das *tags* foi escrita incorretamente (letras invertidas, letras faltando, etc.);

- *Tags* compostas por duas ou mais palavras, onde em uma versão a *tag* tem uma preposição entre as palavras e a outra versão não (por exemplo, `data`**`de`**`nascimento` e `datanascimento`);

- Uma das *tags* está no singular e a outra no plural (`carros` e `carro`, por exemplo).

Não se define uma métrica específica para calcular a similaridade das *strings*. A partir de testes com termos em inglês e português, as métricas *Levensthein* [NAV 01] e *Jaro-Winkler* [NAV 01], por exemplo, obtiveram bons resultados para identificar os casos citados acima.

Uma vez identificadas *tags* equivalentes, surge o problema de como decidir qual delas adotar. Para tanto, considera-se a ajuda de um dicionário semântico, como uma base de dados terminológica ou um *thesaurus*. Através deste, utiliza-se a heurísitica de adotar a versão presente no dicionário semântico. Caso tal suporte não exista ou nenhuma das palavras esteja presente no dicionário semântico, escolhe-se aleatoriamente uma das *tags*. A Figura 4.2 demonstra um exemplo da aplicação do algoritmo.

Figura 4.2: Exemplos do *PP Uniformização da Escrita*

***Tags* originais**	***Tag* uniformizada**
carro, carros	carro
veículo, veiculo, veiulo	veiculo

O Algoritmo 2 apresenta o pseudo-código da *Uniformização da Escrita*. A operação que define sua complexidade é *procEscrita*, que é executada um número de vezes igual ao quadrado da entrada do algoritmo. Logo a complexidade do algoritmo é $O(n^2)$. Tal como no pseudo-código da *Preparação Léxica*, para simplificar sua descrição, não se incluiu o passo de varrer as instâncias de dados XML e substituir as *tags* identificadas como equivalentes.

Algoritmo 2 : *Uniformização da Escrita* (*unifEscrita* - $O(n^2)$)

Entrada: conjunto de termos a compatibilizar - A
Entrada: valor mínimo de similaridade entre strings para serem aceitas como equivalentes - t
Saída: mapeamento entre os termos equivalentes - M

```
proc unifEscrita(A)
    1. Para cada a em A faça
      2. Para cada b em A − {a} faça
        3. Se(procEscrita(a, b) >= t) então
          adicionaMapeamento(a, b, M)      # Armazena a tag original e sua versão processada
        Fim Se                             #   para que posteriormente outro algoritmo substitua
      Fim Para                             #   as tags equivalentes pela tag eleita
    Fim Para
Retorna M


proc procEscrita(a, b)
    1.Se simPossivel(a,b) então
      r ← stringSim(a, b)                  # Uma função de similaridade entre strings qualquer
    Senão
      r ← ∞
    Fim Se
Retorna r
```

4.6 *Uniformização de Termos*

A *Uniformização de Termos* uniformiza *tags* que representam o mesmo conceito, porém são definidas por termos diferentes. Por exemplo, em relação a um artigo, utilizar os termos "nome" e "título" para identificar o mesmo *conceito* - o título do artigo. Esta uniformização se dá através da consulta a um dicionário semântico (*thesaurus* [MIL 95] ou base terminológica [SAN 06]), que permite identificar termos equivalentes.

Dado um conjunto de termos identificados como equivalentes, resta definir qual deve ser adotado como o "*representante*" dos demais. Caso haja o suporte de um dicionário semântico que identifique relações de generalização, sugere-se sempre adotar o termo mais abrangente. Caso não haja tal suporte, adota-se a escolha aleatória de um dos termos. Como os termos são equivalentes, tal decisão não tende a implicar em problemas semânticos graves. A Figura 4.3 demonstra um caso onde a *Uniformização de Termos* foi aplicada.

Figura 4.3: Exemplos do *PP Uniformização de Termos*

***Tags* equivalentes**	***Tag* eleita**
carro, automovel, veiculo, vehicle	veiculo
nome, titulo, cabecalho, title	nome

Um caso particular em relação a *Uniformização de Termos* reside ao tratar instâncias cujas estruturas estão representadas em línguas distintas - por exemplo, inglês e português. Se o dicionário semântico for capaz de identificar os termos equivalentes, pode-se adotar a escolha de um termo que pertença à uma língua pré-definida.

O Algoritmo 3 apresenta o pseudo-código da *Uniformização de Termos*. Neste algoritmo, a operação de maior custo é a determinação de termos equivalentes. Como ela é executada um número de vezes igual ao quadrado da entrada do algoritmo, a complexidade do mesmo é *O(n^2)*. Idem aos algoritmos anteriores, não se inclui aqui, por questões de simplificação, a rotina responsável pela substituição, nas instâncias de dados, das *tags* identificadas como equivalentes pela *tag* eleita.

Algoritmo 3 : *Uniformização de Termos - unifTermos* ($O(n^2)$)

Entrada: conjunto de termos a compatibilizar - A
Saída: mapeamento entre os termos equivalentes - M

```
proc unifTermos(A)
    1. Para cada a em A faça
      2. Para cada b em A − {a} faça
        3. Se thesEquiv(a, b) então
            adicionaMapeamento(a, b, M)      # Armazena as tags equivalentes
        Fim Se                               #   para que posteriormente outro algoritmo substitua
      Fim Para                               #   as tags equivalentes pela tag eleita
    Fim Para
Retorna M
```

4.7 *Remoção de stop-words*

Em *tags* de elementos é comum ter-se duas ou mais palavras, especificando um conceito, como a *data de nascimento* de uma pessoa. Neste tipo de *tag* é comum utilizar palavras que deixam a *tag* mais legível. Estas palavras, porém, não agregam valor semântico às outras palavras, tornando apenas a leitura do termo mais clara. A Figura 4.4 mostra *tags* com *stop-words* e tags equivalentes sem estas *stop-words*. Vê-se que os termos na coluna à esquerda podem ter as palavras *"De", "Do", "Para"* removidas sem o risco de perda de semântica.

Para remover estas *stop-words*, é necessário que a *Preparação Léxica* tenha sido aplicado sobre as *tags*. Considerando o conjunto da Figura 4.4, têm-se como resultado a

Figura 4.4: Termos com *stop-words*

Tags* com *stop-words	***Tags* sem *stop-words***
`DataDeNascimento`	`DataNascimento`
`NomeDoPai`	`NomePai`
`EnderecoParaEntrega`	`EnderecoEntrega`

Figura 4.5. Uma vez com as *tags* processadas lexicamente, pode-se identificar (via um dicionário semântico, por exemplo) as *stop-words* existentes e removê-las. Considerando o exemplo, os dois conjuntos de *tags* ficariam iguais.

Figura 4.5: Termos com *stop-words* pré-processadas lexicamente

Tags* com *stop-words	***Tags* sem *stop-words***
`data_de_nascimento`	`data_nascimento`
`nome_do_pai`	`nome_pai`
`endereco_para_entrega`	`endereco_entrega`

O Algoritmo 4 apresenta o pseudo-código da *Remoção de stop-words*. Em sua execução, a operação mais complexa é representada por *procRemStopWords*, cuja execução ocorre um número de vezes igual ao tamanho da entrada do algoritmo. Portanto, a complexidade do mesmo é *O(n)*. Idem aos pseudo-códigos anteriores, não se inclui a rotina que faz a substituição das *tags* processadas nas instâncias de dados.

Algoritmo 4 : *Remoção de stop-words* (*remStopWords* - $O(n)$)

Entrada: Conjunto de termos a processar - ***A***
Saída: Mapeamento entre os termos originais e processados - ***M***

```
proc remStopWords(A)
    1. Para cada a em A faça
       b ← procRemStopWords(a)
       2. Se (a ≠ b) então
          adicionaMapeamento(a, b, M)      # Armazena a tag original e sua versão processada
       Fim Se                              #   para que posteriormente outro algoritmo substitua
    Fim para                               #   as ocorrências da tag pela sua versão processada
Retorna M


proc procRemStopWords(a)
    l ← palavrasEm(a)
    1. Para cada w em l faça
       2. Se (stopWord(w)) então
          Remove(l, w)
       Fim Se
    Fim Para
    r ← juntaPalavras(l)
Retorna r
```

4.8 *Remoção de Prefixos e Sufixos*

Em documentos XML, pode ocorrer a prefixação ou sufixação do nome de elementos com o nome de seu elemento pai. A Figura 4.6 demonstra este caso. As duas instâncias representam um autor, com as mesmas informações, porém os atributos chaves no processo de comparação tem *tags* diferentes.

Figura 4.6: Exemplo de instâncias com prefixação de termos

Instância 1	**Instância 2**
<Autor> <Nome>Pedro César</Nome> <Idade>26 anos</Idade> <Sexo>Masculino</Sexo> <CPF>555000555000</CPF> </Autor>	<Autor> <AutorNome>Pedro César</AutorNome> <Idade>26 anos</Idade> <Sexo>Masculino</Sexo> <AutorCPF>555000555000</AutorCPF> </Autor>

Na primeira instância, os atributos que identificam o nome e o CPF do autor estão contidos em elementos com nomes simples, `Nome` e `CPF`. Já na segunda instância, os mesmos atributos aparecem pré-fixados com o nome do elemento complexo do qual fazem parte - `Autor`. Tal prática vem dos bancos de dados relacionais, onde termos identificadores, em geral, eram prefixados desta forma para permitir a correlação - *join* - entre as tuplas de dados através destes atributos. Já no contexto de dados no padrão XML, tal prática pode impedir a correlação entre os nomes de elementos de instâncias sendo comparadas.

Para resolver este problema, aplica-se um algoritmo que itera pela árvore da instância de dados XML em um processo *bottom-up*. Para cada elemento *e*, o algoritmo compara sua *tag* com a *tag* de seu elemento pai. Caso a *tag* do elemento pai esteja contida no início ou no fim da *tag* do elemento *e* ela é removida da *tag* do elemento e. No exemplo da Figura 4.6, o termo `AutorNome` tem seu início igual ao elemento pai - `Autor`. Neste caso, o termo seria alterado para apenas `Nome`. No final do processo, as duas instâncias teriam a mesma estrutura.

O Algoritmo 5 apresenta a *Remoção de Prefixos e Sufixos*. Sua operação mais custosa é *procEleminaPrefSufix*, que pode ser executada um máximo igual a *n -1* nodos da instância XML em análise. Desta forma, a complexidade do mesmo é *O(n)*.

Algoritmo 5 : *Remoção de Prefixos e Sufixos* (*remPrefSufix* - $O(n)$)

Entrada: Nodos folha da árvore de uma instância XML a corrigir - A

proc remPrefSufix(A)
1. Para cada b em $nodosCorrigir$ faça
   $c \leftarrow pai(b)$
   2. Se $c \neq null$ então
      $procEleminaPrefSufix(b, c)$
      $adiciona(A, c)$
   Fim Se
   $remove(b, A)$
Fim Para
Retorna.

proc $procEleminaPrefSufix(a, b)$
$removePrefixo(b, a)$
$removeSufixo(b, a)$
Retorna b

4.9 *Casamento de Termos Compostos*

O *Casamento de Termos Compostos* busca identificar *tags* equivalentes compostas por mais de uma palavra, que estejam utilizando termos diferentes (sinônimos ou termos em outras línguas), ou em ordens diferentes, para definir o mesmo conceito. Alguns exemplos onde isto ocorre estão mostrados na Figura 4.7. Percebe-se que os dois conjuntos de *tags* são equivalentes, porém tradicionalmente não seria possível identificar os mesmos como equivalentes pelo simples uso de um *thesaurus* ou de *string similarity*.

Figura 4.7: Exemplos de termos compostos

Conjunto de termos 1	Conjunto de termos 2
DiretorNome	NomeDiretor
PrincipalAtor	AtorProtagonista
PaisNatal	PaisOrigem

Para executar o processo de identificação de *tags* compostas equivalentes, sugere-se inicialmente aplicar a *Preparação Léxica* sobre as *tags* a serem comparadas. Para os exemplos da Figura 4.7, teríamos como o resultado a Figura 4.8. Uma vez com as *tags* pré-processadas, comparam-se as mesmas, identificando as palavras equivalentes (via alguns dos *PPs* apresentados anteriormente) entre as que compõe as *tags*. Considerando o exemplo, se verificaria que *principal* é equivalente a *protagonista* e *natal* é equivalente a *origem* (através dos métodos utilizados na *Uniformização de Termos*). Logo, o conjunto

{diretor,nome} é igual ao conjunto *{nome,diretor}* e, portanto, *DiretorNome* e *NomeDiretor* são equivalentes. Para as outras *tags*, o processo é idêntico.

Figura 4.8: Exemplos de termos compostos processados lexicamente

Conjunto de termos 1	Conjunto de termos 2
diretor_nome	nome_diretor
principal_ator	ator_protagonista
pais_natal	pais_origem

O Algoritmo 6 apresenta o pseudo-código do *Casamento de Termos Compostos*, cuja complexidade máxima é *O(n^2)*. Isto se deve ao fato que a operação mais custosa é *compostaEquivalente*, que é executada um número de vezes igual ao quadrado do tamanho da entrada do algoritmo. Tal como nos outros *PPs* relacionados às *tags*, não se inclui aqui a rotina que substitui as *tags* equivalentes nas instâncias de dados.

Algoritmo 6 : *Casamento de Termos Compostos* - (casaCompostas - $O(n^2)$)

Entrada: Conjunto de termos a compatibilizar - A
Saída: Mapeamento entre os termos equivalentes - M

```
proc casaCompostas(A)
    1. Para cada a em A faça
      2. Para cada b em A faça
        3. Se compostaEquivalente(a, b) então
          adicionaMapeamento(a, b, M)        # Armazena as tags equivalentes
        Fim Se                               #   para que posteriormente outro algoritmo substitua
      Fim Para                               #   as tags equivalentes pela tag eleita
    Fim Para
Retorna M


proc compostaEquivalente(a, b)
    l ← palavrasEm(a)
    m ← palavrasEm(b)
    1. Se (tamanho(l) ≠ tamanho(m)) então
      retorno ← false
    2. Para cada c em l faça
      3. Para cada d em m faça
        4. Se ThesEquiv(c, d) então
          remove(c,l)
          remove(d,m)
        Fim Se
      Fim Para
    Fim Para
Retorna vazio(l) ∧ vazio(m)
```

4.10 *Reestruturação Hierárquica*

Dentre as possíveis diferenças estruturais entre instâncias de dados XML, a diferente organização hierárquica das informações é uma delas, e talvez a mais crítica. Dadas duas instâncias de dados, elas podem conter a mesma informação, porém apresentá-la com organizações hierárquicas distintas. A Figura 4.9 exemplifica tal caso. Nas duas instâncias, a informação contida é a mesma. Contudo, devido à diferente forma como os dados estão organizados, a comparação entre as instâncias é prejudicada, pois os elementos que devem ser comparados estão em pontos diferentes na hierarquia das instâncias.

Figura 4.9: Documentos XML mostrando diferenças na estrutura hierárquica

Instância 1

```
<referencia>
  <autor>
     <nome>Paulo José</nome>
     <livro>
        <nome>SimInt</nome>
        <edicao>Primeira</edicao>
     </livro>
     <pais>Brasil</pais>
  </autor>
</referencia>
```

Instância 2

```
<referencia>
  <livro>
     <nome>SimInt</nome>
     <edicao>Primeira</edicao>
     <autor>
        <nome>Paulo José</nome>
        <pais>Brasil</pais>
     </autor>
  </livro>
</referencia>
```

Trabalhos relacionados que levam em conta a hierarquia [MIL 06, MA 05] ou que analisam a similaridade estrutural [NIE 02, FLE 05] para a comparação têm dificuldade em comparar estas instâncias. Tal fato ocorre porque eles não encontram estruturas equivalentes nas posições esperadas. Para evitar este problema na comparação das instâncias, propõe-se um método de análise e transformação da hierarquia das instâncias em dois passos. Este método modifica a forma como os dados estão contidos nas mesmas, buscando otimizar a determinação da similaridade estrutural.

4.10.1 Coleta

O primeiro passo para resolver o problema é coletar informações sobre a estrutura das instâncias. Estas informações serão utilizadas para identificar em seguida os conflitos hierárquicos existentes e corrigí-los. Para tal, varre-se a árvore de uma das instâncias (escolhida aleatoriamente) e se armazena informações sobre a hierarquia dos elementos.

O Algoritmo 7 apresenta o pseudo-código que realiza este processo. Na varredura, para cada elemento e_x encontrado, ele obtém a sua *tag* t_a. Em seguida, para cada *tag* t_b dos elementos que fazem parte do path de e_x, o algoritmo verifica se existe um grupo das *tags sucessoras* à *tag* t_b, ou seja, o conjunto das *tags* dos elementos que aparecem como sucessores aos elementos que possuam a *tag* t_b. Se tal conjunto não existe, um novo conjunto é gerado e a *tag* t_a é adicionada ao mesmo. Caso o conjunto já exista, o algoritmo adiciona a *tag* t_b ao mesmo. Da mesma forma, cada *tag* t_b do *path* de t_a é adicionada ao conjunto de *tags antecessoras* de t_a.

Algoritmo 7 : Algoritmo para Coleta de Informações sobre Conflitos Hierárquicos ($coletaInfoConflHieraquicos - O(n^2)$)

Entrada: nodo raiz da instância base para coleta de informações sobre conflitos hierárquicos - ***n***
Saída: mapeamento indicando *tags* que sucedem cada *tag* na instância - ***filhos***

```
proc coletaInfoConflHieraquicos(n)
    L ← {n}
    1. Para cada l em L faça
      adicionaMapeamento(tag(l), tagsAntecessoras(l), pais)
      2. Para cada t em tagsAntecessoras(l) faça
        adicionaMapeamento(t, tag(l), filhos)
      Fim Para
      adiciona(L, NodosFilhos(l))
      remove(L, l)
    Fim Para


    3. Para cada t em pais faça
      4. Para cada p em pais(t) faça
        5. Se (p ∈ filhos(t)) então
          remove(filhos(t), p)
        Fim Se
      Fim Para
    Fim Para
Retorna(filhos)
```

Assim, ao final deste processo, têm-se todas as *tags* que são sucessoras e antecessoras de uma dada *tag* dentro de uma das instâncias. Com estes dois conjuntos, verifica-se se existem elementos que aparecem tanto como antecessor quanto como sucessor de uma dada *tag t* na instância analisada. Se houver elementos nesta situação, eles são removidos dos dois conjuntos. Esta remoção ocorre porque estes elementos representam um caso especial onde um elemento é antecessor e sucessor de uma mesma *tag* na instância. Este caso pode levar o algoritmo de detecção e correção de conflitos hierárquicos (a ser apresentado) a entrar em *loop*. Como indica uma possível complexidade estrutural alta, este caso especial não é tratado pelo *PP Reestruturação Hierárquica*. A Figura 4.10

demonstra a saída do Algoritmo 7 para a primeira instância do exemplo da Figura 4.9.

Figura 4.10: Saída do algoritmo de coleta

Tag	***Tags* sucessoras**
referencia	autor, nome, livro, edicao, pais
livro	nome, edicao
nome	
edicao	
autor	nome, livro, edicao, pais
pais	

Em relação à complexidade do algoritmo, a operação mais custosa é a determinação dos casos especiais onde uma *tag* é antecessora e sucessora de uma outra *tag* (o segundo bloco do algoritmo). Como esta operação será executada um número máximo de vezes igual ao quadrado do tamanho do conjunto de *tags* da instância, a complexidade do algoritmo fica em $O(n^2)$.

4.10.2 Detecção e correção

Uma vez aplicado o algoritmo de *coleta*, varre-se a estrutura da outra instância para identificar conflitos da sua hierarquia com a instância sobre a qual o algoritmo de *coleta* foi aplicado. Um conflito existe se, para um elemento de *tag t*, algum dos elementos que estão presentes no seu *path* possui uma *tag* existente no conjunto de *tags* sucessoras à mesma *tag t* na instância sobre a qual o algoritmo de coleta foi aplicado.

Uma vez que um elemento nesta situação seja identificado, o algoritmo realiza a inversão do mesmo, ou seja, coloca o elemento conflitante como pai do elemento ancestral que deveria estar como descendente. Desta forma, a hierarquia das instâncias torna-se mais similar. Uma vez realizada a inversão de um elemento, o processo é reiniciado a partir do elemento que foi movido para o ponto mais alto na hierarquia da instância, para manter a característica de varredura em profundidade que caracteriza o algoritmo.

O Algoritmo 8 apresenta o processo de detecção e correção das instâncias. Considerando que a operação de maior custo é a movimentação dos nodos para uma nova posição, e que esta ocorre um máximo de vezes igual ao número de nodos da instância, a complexidade do algoritmo fica definida como *O(n)*. Dadas as instâncias do exemplo

da Figura 4.9, ao final deste processo a Instância 2 apresentará a mesma hierarquia da Instância 1, através da inversão dos itens destacados na Figura 4.11.

Algoritmo 8 : Algoritmo de Reestruturacao Hierarquica ($reestruturaHierarquia - O(n)$)

Entrada: Mapeamento entre $tags$ e elementos sucessos - $filhos$
Entrada: Nodos raízes das instância a reestruturar a hierarquia - N

proc reestruturaHierarquia(N)
1. Para cada n em N faça
2. Para cada p em $path(n)$ faça
3. Se $p \in filhos(n)$ então
$remove(N, ElementosAbaixo(p))$
$trocaPai(n, Pai(p))$
$trocaPai(p, n)$
$Insere(p, N)$
AbortaLoop
Fim Se
Fim Para
remove(n, N)
insereFilhos(n, N)
Fim Para

Retorna N

Figura 4.11: Elementos invertidos nos documentos XML com diferenças na hierarquia

Instância 1	Instância 2
<referencia> <autor> <nome>Paulo José</nome> <livro> <nome>SimSearch</nome> <edicao>Primeira</edicao> </livro> <pais>Brasil</pais> </autor> </referencia>	<referencia> <livro> <nome>SimSearch</nome> <edicao>Primeira</edicao> <autor> <nome>Paulo José</nome> <pais>Brasil</pais> </autor> </livro> </referencia>

4.11 *Compatibilização de Elementos Complexos e Simples*

A mesma informação, em diferentes instâncias XML, pode estar representada em uma forma estruturada, semi-estruturada ou atômica (simples). Para comparar os elementos de uma instâncias XML sem recorrer a abstração (análise pura do conteúdo, sem levar em conta a estrutura) é preciso que os elementos sejam compatíveis quanto à representação de seus dados. A Figura 4.12 exemplifica um caso onde o mesmo elemento tem representações diferentes em duas instâncias de dados XML.

Figura 4.12: Instâncias com conflitos de representação de elementos simples e complexos

Instância 1

```
<autor>
  <nome>
     <primeiroNome>Paul</primeiroNome>
     <ultimoNome>Simon</ultimoNome>
     <iniciais>P. S.</iniciais>
  </nome>
  <livro>
     <nome>Similarity Search</nome>
     <editora>J.J. Press</editora>
  </livro>
</autor>
```

Instância 2

```
<autor>
  <nome>Paul Simon</nome>
  <livro>Similarity Search -
     J.J. Press</livro>
</autor>
```

Para resolver este problema, sugere-se um procedimento de *PP* que remova a estrutura da versão complexa do elemento e mantenha apenas seus conteúdos. O objetivo aqui é aumentar as chances de identificar uma equivalência entre os elementos, já que as diferenças estruturais não existem mais. No aspecto semântico do processo de comparação, nenhuma informação é considerada perdida, visto que sem modificação não seria possível casar os dados nas versões simples e complexa.

Caso nenhuma informação semântica em relação ao contexto das instâncias esteja disponível, a transformação do elemento complexo em simples é a abordagem mais simples a ser adotada. A conversão oposta (do elemento simples em complexo) é mais complexa, considerando as tarefas de identificação, extração e estruturação das informações do elemento simples.

O Algoritmo 9 demonstra uma abordagem *top-down* para executar o processo de compatibilização de elementos. Considerando as instâncias XML apresentadas na Figura 4.12, a execução do algoritmo sobre as mesmas deixa as duas instâncias com a mesma estrutura. A abordagem *top-down* é adotada para otimizar o processo - se um elemento a ser transformado está contido em outro elemento que também vai ser transformado, basta transformar o elemento de mais alto nível hierárquico, evitando, assim, transformações desnecessárias.

Em relação à complexidade do algoritmo, entende-se que a mesma é *O(n)*, haja vista que a operação mais complexa é a conversão de um elemento complexo em simples, cujo número máximo de vezes que pode ocorrer é *(n - 1)*.

Algoritmo 9 : *Compatibilização de Elementos Complexos e Simples* (*procCompSimples* - $O(n)$)

Entrada: Nodos raízes de instâncias a comparar - A
Saída: Nodos raízes de instâncias compatibilizadas - A

```
proc procCompSimples(A)
    L ← agrupaNodosPorTag(A)        # Agrupa os nodos em A em subgrupos pelas tags dos mesmos.
    1. Para cada l em L faça        # l é um subgrupo de L com nodos de uma mesma tag.
       simples ← CompatSimplesComplexo(l)
       2. Se ¬simples então
          3. Para cada m em l faça
             adiciona(filhos(l), N)
          Fim Para
          procCompSimples(N)
       Fim Se
    Fim Para
Retorna A

proc CompatSimplesComplexo(l)
    1. Para cada k em l faça
       2. Se (simples(k)) então
          simples ← verdadeiro
          aborta
       Fim Se
    Fim Para
    3. Se simples
       4. Para cada k em l faça
          5. Se ¬simples(k) então
             SimplificaELemento(k)
          Fim Se
       Fim Para
    Fim Se

Retorna simples
```

4.12 *Compatibilização de Complexos*

Duas instâncias de dados podem diferir em relação aos dados que contém, ou seja, ambas representam a mesma entidade, porém, com propriedades diferentes ou adicionais. Neste caso, trabalhos que se baseiam em distância de edição ou na abstração da estrutura e comparação apenas pelos dados podem apresentar problemas para identificar estas instâncias como equivalentes. A Figura 4.13 exemplifica um caso onde duas instâncias de uma mesma propriedade possuem propriedades não comuns (destacadas). Para resolver este problema, duas abordagens são possíveis:

1. Remover os subelementos não comuns;

2. Identificar os subelementos não-comuns e colocar seus conteúdos em um subelemento estruturado do tipo lista.

A primeira abordagem é útil em trabalhos onde faz-se uma comparação estrutural entre as instâncias, em que dados a mais (não comuns a ambas as instâncias) fazem com

Figura 4.13: Documentos XML com elementos não-comuns

Instância 1

```
<autor>
  <nome>
     <primeiroNome>Paulo</primeiroNome>
     <ultimoNome>José</ultimoNome>
  </nome>
  <livro>
     <nome>SimSearch</nome>
     <edicao>1st</edicao>
  </livro>
</autor>
```

Instância 2

```
<autor>
  <nome>
     <primeiroNome>Paulo</ultimoNome>
     <nomeDoMeio>J.</nomeDoMeio>
     <ultimoNome>Simon</ultimoNome>
  <nome>
  <livro>
     <nome>SimSearch</nome>
  </livro>
</autor>
```

que a comparação indique instâncias equivalentes como distintas pela existência de informações adicionais nelas. Supõe-se, para sua execução, que diferenças nos termos que identificam as propriedades das instâncias já foram sanadas, de forma que termos não comuns entre as instâncias representam efetivamente informação adicional, sobre a qual nada pode ser afirmado.

Entende-se que a segunda abordagem, ao abstrair os nomes dos elementos como *item* (de uma estrutura de lista), possibilita que muitos dos trabalhos relacionados [MIL 06, DOR 04, MA 05] consigam identificar os elementos equivalentes através da comparação destes elementos contidos dentro dos elementos compatibilizados. Como a partir da transformação feita pelo *PP* os elementos passam a ter o mesmo contexto e *tag*, os trabalhos relacionados conseguem agora compará-los.

O Algoritmo 10 demonstra a primeira solução. Vê-se que a complexidade máxima do mesmo é determinada pela etapa de remoção dos elementos não-comuns, cujo número máximo de vezes que pode ocorrer é *(n - 1)*, logo a complexidade do mesmo é *O(n)*. A Figura 4.14 ilustra o resultado da aplicação do algoritmo sobre as instâncias da Figura 4.13.

O Algoritmo 11 demonstra a segunda solução. Vê-se que a complexidade máxima do mesmo é determinada pela etapa de conversão dos elementos não-comuns em um subelemento lista, cujo número máximo de vezes que pode ocorrer é *(n - 1)*. Logo, a complexidade do mesmo também é *O(n)*. A Figura 4.15 ilustra o resultado da aplicação do algoritmo sobre as instâncias da Figura 4.13.

Aloritmo 10 : *Compatibilização de Complexos* ($compatComplexos - O(n)$)

Entrada: Conjunto de nodos raízes das instâncias a compatibilizar - **A**
Saída: Conjunto de nodos raízes das instâncias compatibilizadas - **A**

proc $compatComplexos(a)$
$L \leftarrow agrupaNodosPorTag(A)$ # Agrupa os nodos em A em subgrupos pelas *tags* dos mesmos.
1. Para cada l em L faça # l é um subgrupo de L com nodos de uma mesma *tag*.
$filhos \leftarrow \emptyset$
2. Para cada k em l faça
$filhos \leftarrow tagsDosFilhos(k) - filhos$
Fim Para
3. Para cada k em l faça
4. Se $filhos \neq \emptyset$ então
$removeFilhosComTags(k, filhos)$
Fim Se
$adiciona(filhos(k), N)$
Fim Para
$compatComplexos(N)$
Fim Para
Retorna(L)

Figura 4.14: Elementos não-comuns resolvidos - primeira abordagem

Instância 1	Instância 2
<autor> <nome> <primeiroNome>Paul</primeiroNome> <ultimoNome>Simon</ultimoNome> </nome> <livro> <nome>Similarity Search</nome> </livro> </autor>	<autor> <nome> <primeiroNome>Paul</primeiroNome> <ultimoNome>Simon</ultimoNOme> <nome> <livro> <nome>Similarity Search</nome> </livro> </autor>

Figura 4.15: Elementos não-comuns resolvidos - segunda abordagem

Instância 1	Instância 2
<autor> <nome> <primeiroNome>Paul</primeiroNome> <ultimoNome>Simon</ultimoNome> <itens> <item>J.</item> </itens> </nome> <livro> <nome>Similarity Search</nome> </livro> </autor>	<autor> <nome> <primeiroNome>Paul</primeiroNome> <ultimoNome>Simon</ultimoNOme> <nome> <livro> <nome>Similarity Search</nome> <itens> <item>1st</item> </itens> </livro> </autor>

4.13 *Remoção de Elementos Coleção*

Um tipo de diferença estrutural que pode atrapalhar a comparação entre duas instâncias de dados são elementos sem valor semântico que apenas estabelecem uma melhor organização dos dados. Considere-se a Figura 4.16. Vê-se pelo exemplo que o elemento "*clubes*" apenas melhor organiza as informações do documento, porém não contribui se-

Algoritmo 11 : *Compatibilização de Complexos* ($compatComplexos - O(n^2)$)

Entrada: Conjunto de nodos raízes das instâncias a compatibilizar - **A**
Saída: Conjunto de nodos raízes das instâncias compatibilizadas - **A**

```
proc compatComplexos(a)
    L ← agrupaNodosPorTag(A)        # Agrupa os nodos em A em subgrupos pelas tags dos mesmos.
    1. Para cada l em L faça        # l é um subgrupo de L com nodos de uma mesma tag.
       filhos ← ∅
       2. Para cada k em l faça
          filhos ← tagsDosFilhos(k) − filhos
       Fim Para
       3. Para cada k em l faça
          4. Se filhos ≠ ∅ então
             geraItensFilhosComTags(k, filhos)
          Fim Se
          adiciona(filhos(k), N)
       Fim Para
       compatComplexos(N)
    Fim Para
Retorna(L)

proc geraItensFilhosComTags(k, filhos)
    i ← novoNodo('itens')
    adicionaFilho(k, i)
    1. Para cada f em filhos faça
       itens ← filhosComTag(k, f)
       2. Para cada g em itens faça
          renomeia(g,' item')
          adicionaFilho(i, g)
       Fim Para
    Fim Para
Retorna(k)
```

manticamente (não acrescenta nova informação). Portanto, se o removermos não teremos perda de informação.

Figura 4.16: Exemplos de Elementos de Coleção

Instância 1	**Instância 2**

`<associacao>`
`  <clubes>`
`  <clube>`
`  <nome>Clube 12</nome>`
`  </clube>`
`  <clube>`
`  <nome>Corrida Rústica</nome>`
`  </clube>`
`  </clubes>`
`</associacao>`
	`<associacao>`
	`  <clube>`
	`  <nome>Corrida Rústica</nome>`
	`  </clube>`
	`  <clube>`
	`  <nome>Clube 12</nome>`
	`  </clube>`
	`</associacao>`

O procedimento de *Remoção de Elementos Coleção*, apresentado no Algoritmo 12, varre a estrutura da instância sendo analisada e identifica elementos nesta situação. Para tal, duas heurísticas são sugeridas: a primeira baseia-se no uso de um dicionário semântico para identificar um termo que é o plural de outro termo, detectando desta forma estes elementos sem valor semântico. A segunda utiliza a distância de edição de *strings* para identificar termos equivalentes no singular e plural, através do fato que o plural de palavras

em geral é feito por uma pequena alteração no final do termo no singular. Aplicando distância de edição, identifica-se os termos equivalentes nesta situação e se removem os mesmos. Considerando as instâncias da Figura 4.16, o algoritmo age sobre o elemento destacado, removendo o mesmo.

A complexidade do Algoritmo 12 é determinada pelas operações de *elementoColecao* e *trocaPai*. Como estas não podem ser executadas um número de vezes superior ao número de elementos da instância sendo analisada, a complexidade do algoritmo é *O(n)*.

Algoritmo 12 : Remoção de elementos coleção ($remElementosColecaoO(n)$)

Entrada: nodo raiz da instância a remover os elementos coleçao - n
Saída: nodo raiz da instância com os elementos coleção removidos - n

proc remElementosColecao(n)
  $tags \leftarrow TagsElementosFilhos(n)$
  1. Se ($Tamanho(tags) = 1$) $\wedge$ ($elementoColecao(n, tags[0])$) então
    $move \leftarrow verdadeiro$
  Senão
    $move \leftarrow falso$
  Fim Se
  2. Para cada f em $ElementosFilhos(n)$ faça
    3. Se $move$ então
      $trocaPai(f$, Pai(n))
    Fim Se
    $remElementosColecao(f)$
  Fim Para
Retorna n

proc $elementoColecao(nomePai, nomeFilho)$
  1. Se ($nomePai \equiv nomeFilho$) $\vee$ ($ColecaoDe(nomeFilho, nomePai)$) então
    $retorno \leftarrow verdadeiro$
  Senão
    $retorno \leftarrow falso$
  Fim Se
Retorna $retorno$

4.14 Conclusão

Este capítulo apresentou um conjunto de técnicas de pré-processamentos (*PPs)*, aplicáveis sobre instâncias de dados semi-estruturadas com o objetivo de reduzir diferenças estruturais entre elas. Estas diferenças, muitas vezes originárias de grafias diferentes para *tags* representando o mesmo conceito, prejudicam ou impedem uma correta execução dos trabalhos relacionados à detecção de similaridade em dados semi-estruturados analisados.

Para cada *PP* apresentado, um algoritmo em pseudo-código foi descrito, assim como uma breve análise de sua complexidade. Os *PPs* apresentados resolvem desde

problemas simples relacionados à grafia das *tags* até problemas complexos de organização hierárquica das instâncias de dados comparadas. Em todos os casos, a complexidade nunca ultrapassou $O(n^2)$, mantendo-se, portanto, polinomial.

No próximo capítulo, é apresentada uma análise mais detalhada de alguns aspectos particulares dos *PPs* apresentados (como a melhor forma de combiná-los, em que ordem executá-los, etc.), assim como um estudo sobre a melhor forma de combiná-los, estabelecendo uma seqüência indicada para as suas execuções.

Uma parte destes pré-processamentos originou uma artigo publicado e apresentado na conferência DEXA - *Databases and Expert Systems Applications*, edição 2007 [GON 07].

Capítulo 5

Análise e Estudos de Caso

5.1 Introdução

O objetivo dos algoritmos de pré-processamento (*PPs*) apresentados é melhorar a similaridade estrutural entre instâncias de dados XML, de forma a facilitar sua comparação e posterior integração. Para que sejam utilizados de forma que produzam os resultados desejados, faz-se neste capítulo uma análise mais detalhada sobre eles. Nesta análise identificam-se alguns aspectos que limitam sua execução, considerando tanto recursos que devem estar disponíveis quanto limitações inerentes às abordagens. A dependência entre os algoritmos de pré-processamento, estabelecendo uma ordem indicada em que eles devem ser aplicados, é outro aspecto analisado neste capítulo.

Faz-se também um estudo preliminar sobre o impacto dos *PPs* nos trabalhos relacionados estudados. Destaca-se, para cada pré-processamento, os trabalhos sobre os quais seu impacto pode ser maior e uma tabela relacionando os trabalhos relacionados e os *PPs* é apresentada. Alguns estudos de caso específicos da aplicação dos *PPs* são descritos, demonstrando como eles conseguem resolvê-los de forma automática, ou seja, sem intervenção de um usuário especialista.

5.2 Comparação *vs* Integração

Os algoritmos de *PP* podem ser divididos em duas classes: *destrutivos* e *não-destrutivos*. Esta divisão baseia-se no impacto que as transformações executadas pelos mesmos tem em um processo posterior de integração.

Os *PPs* relacionados às *tags* dos elementos são classificados como *não-destrutivos* - estes incluem *Preparação Léxica, Uniformização da Escrita, Uniformização de Termos, Remoção de stop-words, Remoção de Prefixos e Sufixos e Casamento de Termos Compostos*. Eles são definidos como não destrutivos pois não removem dados ou informação semântica que caracteriza os dados das instâncias. Por basear-se em dicionários semânticos para seus processos, as equivalências identificadas buscam respeitar a semântica dos dados.

Os *PPs* relacionados à estrutura estão classificados nas duas categorias. A *Reestruturação Hierárquica* classifica-se como não destrutivo, pois mantém os dados e a sua semântica. Já os *PPs* de *Compatibilização de Elementos Complexos e Simples* e de *Compatibilização de Complexos* caracterizam-se como destrutivos, pois removem dados e/ou informação semântica das instâncias. O *PP Remoção de Elementos Coleção* classifica-se como não destrutivo, pois baseia-se em regras que garantem que apenas elementos sem valor semântico sejam removidos.

Em um contexto de integração de instâncias, esta distinção é importante. Através dela se pode definir um ponto dentro do processo de compatibilização das instâncias a partir do qual uma *cópia* das instâncias pode ser feita para, caso sejam identificadas como similares, estas cópias não *"destruídas"* sejam integradas posteriormente, aproveitando as transformações executadas pelos pré-processamentos não destrutivos. A Figura 5.1 apresenta os *PPs* e sua classificação em *não destrutivos* e *destrutivos*.

Figura 5.1: *PPs destrutivos* e *não destrutivos*

Não destrutivos	**Destrutivos**
Preparação Léxica *Uniformização da Escrita* *Uniformização de Termos* *Remoção de stop-words* *Remoção de Prefixos e Sufixos* *Casamento de Termos Compostos* *Reestruturação Hierárquica* *Remoção de Elementos Coleção*	*Compatibilização de Elementos Complexos e Simples* *Compatibilização de Complexos*

5.3 Limitações

O ***PP Preparação Léxica*** reduz diferenças de grafia entre *tags* de instâncias de dados em XML equivalentes, uniformizando a forma como são separadas as palavras que compõem as mesmas. Esta uniformização pode não prover sempre bons resultados por não considerar casos onde *tags* compostas por mais de uma palavra tem todas as letras em maiúsculas ou minúsculas, e não possuem separadores. Nestes casos, determinar e separar as palavras que compõe a *tag* torna-se complexo: não existem indicadores confiáveis (e com custo reduzido de detecção) do ínicio e fim das palavras que compõem a *tag*. Um exemplo é a *tag* `datadenascimento`. Apesar de composta por três palavras, como todas as letras são minúsculas, não se consegue identificar as três palavras que a compõe.

Uma melhora para a *Preparação Léxica* é, antes de processar as *tags*, buscar *tags* equivalentes pela comparação das *tags* com todos os caracteres separadores removidos e letras convertidas para caixa baixa. Uma vez identificadas *tags* equivalentes por esta forma, aquela que possui condições de identificar as palavras que a compõe (que possui caracteres separadores ou letras em caixa alta isoladas) substitui a outra *tag*. Desta forma, consegue-se casar corretamente *tags* com e sem separadores. Mantém-se as *tags* com separadores por possibilitarem uma melhor tratativa semântica dos dados que representam - é mais simples identificar que `nome_pai` refere-se ao *nome* do *pai* do que `nomepai` (considerando um posterior processo de integração). Além disso, *tags* com separadores possibilitam a aplicação de outros algoritmos de pré-processamento (como o *PP Remoção de stop-words*), que dependem dos caracteres separadores para sua execução.

Em relação ao ***PP Uniformização da Escrita***, um fator limitante para a aplicação do mesmo é a definição do *threshold* (valor de distância de edição máximo) até o qual duas *tags* são consideradas equivalentes. Considera-se que a métrica adotada para a comparação deve buscar, em *tags* de tamanho reduzido, considerar apenas inversão de caracteres ou a falta de um caracter. Para *tags* de tamanho maior, pode-se considerar também a remoção de duas ou até três letras. Isto deve-se ao fato que, em *tags* pequenas, a remoção de duas letras pode realizar o casamento entre *tags* incorretamente.

Dentre as métricas estudadas, a *Weighted Damerau Distance* apresenta-se como

uma das mais adequadas. Esta métrica é uma variação da métrica de distância de edição *Levenstein*, apresentando as seguintes particularidades: estabelece pesos diferentes para cada operação e inclui uma nova operação de troca de caracteres adjacentes, com custo único. Desta forma, através do ajuste de seus pesos, conforme o tamanho das *tags*, e pela operação de troca de caracteres adjacentes, ela se torna adequada, considerando as necessidades citadas anteriormente.

O ***PP Uniformização de Termos*** tem como limitação a necessidade de informação semântica adicional para definir os termos equivalentes. Tais informações devem ser obtidas de dicionários semânticos, como por exemplo Thesauri (WordNet) e *dicionários de dados*. Tais repositórios de informação semântica são um tema atual de pesquisa, e a construção deles, a partir de fontes de dados ou outros repositórios, é uma possibilidade real. Além da existência destas estruturas, escolher o melhor termo dentre os equivalentes é outro fator que pode ser limitante não para um processo de comparação, mas para um posterior processo de integração, onde o elemento que melhor represente os elementos determinados como equivalentes deve ser escolhido. Tal decisão, porém, foge ao escopo do trabalho, que se concentra nos processos de comparação, onde a escolha do termo representante pode ser aleatória. Heurísticas para a escolha do melhor termo serão foco de futuros trabalhos. Dentre estas heurísticas, cita-se considerar o contexto (os elementos próximos por exemplo) onde os termos comparados estão localizados, de forma que evite-se assim identificar termos equivalentes erroneamente - por exemplo, *homônimos*.

O *PP Remoção de stop-words* também tem dependência de um repositório semântico para identificar as *stop-words*. Além disso, para seu correto funcionamento, as *tags* devem ter passado pelo *PP Preparação Léxica*, para que tenham a forma de separar as palavras uniformizada. Caso o *PP Preparação Léxica* não consiga uniformizá-las, o algoritmo de remoção de *stop-words* não terá como identificar as *stop-words* e removê-las. Tal situação seria a encontrada para as seguintes *tags*: *datadenascimento* e *datanascimento*.

Em relação ao ***PP Remoção de Prefixos e Sufixos***, não se identifica nenhuma limitação imediata. Seu funcionamento trata um caso especial, para o qual não há fatores que limitem sua aplicação.

Fora as limitações já apresentadas para o *PP Uniformização de Termos*, o ***PP Ca-***

samento de Termos Compostos necessita que os conjuntos de palavras dos termos sejam equivalentes para sua execução (o mesmo número de palavras e que elas possam ser identificadas como equivalentes entre si). Considerando esta limitação, entende-se que se houverem palavras adicionais, como *stop-words*, ele não será capaz de identificar os termos equivalentes. Tal fato torna a execução do *PP Remoção de stop-words* um requisito para a execução ótima do *PP Casamento de Termos Compostos*. Outro fator limitante é a identificação de casos onde mais de uma palavra em uma *tag* pode ser representada por uma única palavra na outra *tag*. Por exemplo, as *tags* `curso_alemao` e `deustchkurs`[1]. Para tal caso seria necessário o suporte de um repositório semântico capaz de identificar estas situações. No momento não se têm conhecimento de algum repositório semântico com esta capacidade.

O ***PP Reestruturação Hierárquica***, assim como todos os demais pré-processamentos relacionados à estrutura da instância, tem como fator limitante o fato de que as *tags* das instâncias devem estar compatibilizadas, pois são as guias para a execução dos mesmos. Dois casos especiais merecem destaque neste pré-processamento: inversões hierárquicas quando mais de dois elementos estão envolvidos e inversões hierárquicas com elementos simples. Ambos os casos são discutidos posteriormente neste capítulo.

O ***PP Compatibilização de Elementos Complexos e Simples*** apresenta como limitação o fato de um elemento complexo conter muito mais informações que o elemento simples e, ao se converter o elemento complexo em simples, a diferença entre os dados dos dois seja alta e dificulte o processo de comparação dos mesmos. A Figura 5.2 demonstra este caso. Vê-se no exemplo que o elemento complexo possui muito mais informações que o elemento simples, e uma comparação textual entre eles, a menos que seja feita em termos de *contenção*, dificilmente irá apontar os dois elementos como similares.

O ***PP Compatibilização de Complexos*** possui duas limitações durante sua execução. A primeira refere-se ao fato que um elemento pode apresentar sub-elementos não presentes em outros elementos de mesmo tipo e na mesma posição hierárquia na instância. Neste caso, são removidos elementos que poderiam ser utilizados em uma comparação posterior. Entende-se, mesmo assim, que este fato não implica em problemas

[1] deutsckurs significa *curso de alemão* na língua alemã.

Figura 5.2: Exemplo de elemento complexo com muitos dados

Elemento complexo	**Elemento simples**
<autor> <nome>João</nome> <cidade>Florianópolis</cidade> <data_nascimento>12/03/1982</data_nascimento> <profissao>Analista de Sistemas</profissao> </autor>	<autor>João - 12/03/1982</autor>

graves ao processo de comparação, pois os dados comuns aos vários elementos representam aqueles importantes à identificação dos objetos que representam. Portanto, para um processo de comparação baseado em dados comuns, tal limitação não se torna um fator prejudicial. A Figura 5.3 demonstra um conjunto de elementos onde este fato ocorre. No caso do exemplo, os elementos em negrito seriam removidos.

Figura 5.3: Elementos complexos com propriedades variáveis

Instância 1	**Instância 2**
<autor> <nome>João</nome> **<cidade>Florianópolis</cidade>** </autor> <autor> <nome>Pedro</nome> </autor>	<autor> <nome>João</nome> **<cidade>Florianópolis</cidade>** </autor> <autor> <nome>Pedro</nome> **<cidade>Brusque</cidade>** </autor>

A segunda limitação refere-se ao fato que, se houver *tags* de elementos que não puderam ser corrigidas pelos algoritmos de análise da escrita das *tags*, estas serão entendidas como incompatíveis e removidas. Tal limitação porém é inerente ao contexto de aplicação (automatizado), não apresentando uma solução definitiva.

No ***PP Remoção de Elementos Coleção***, dada a dificuldade em determinar elementos coleção em uma hierarquia no padrão XML de forma confiável, adotou-se uma abordagem limitada porém simples, que tenta respeitar a semântica dos elementos. Para identificar elementos inúteis mais complexos, seria necessária uma estrutura capaz de identificar casos onde um elemento apenas agrupa outros elementos, porém não lhes dá um valor semântico diferenciado. Considerando o exemplo da Figura 5.4, vê-se dois ele-

mentos contendo coleções de livros com semânticas diferenciadas. Um deles realmente representa uma coleção de livros sem uma semântica distinta, porém o outro representa livros *que foram traduzidos* (por um autor, professor, etc.). Portanto, estes elementos não poderiam ser removidos, se estivessem, por exemplo, dentro da descrição de dois autores.

Figura 5.4: Elemento sem e com valor semântico

Instância 1	**Instância 2**
<livros> <livro>Livro A</livro> <livro>Livro B</livro> <livro>Livro C</livro> <livros>	<livrosTraduzidos> <livro>Livro A</livro> <livro>Livro B</livro> <livro>Livro C</livro> <livrosTraduzidos>

A Tabela 5.1 apresenta uma visão geral das principais limitações dos *PPs*, apresentadas nesta seção.

Tabela 5.1: Limitações dos *PPs*

PP	**Principais limitações**
1 - *Preparação Léxica*	Necessidade de separadores ou letras em caixa alta
2 - *Uniformização de Termos*	Repositório semântico para identificar termos equivalentes
3 - *Uniformização da Escrita*	Métrica adequada e *threshold* de similaridade
4 - *Remoção de Prefixos e Sufixos*	-
5 - *Remoção de stop-words*	Sucesso na *Preparação Léxica* e repositório semântico para identificar *stop-words*
6 - *Casamento de Termos Compostos*	Sucesso na *Preparação Léxica* e repositório semântico para identificar termos equivalentes
7 - *Reestruturação Hierárquica*	Sucesso nos *pré-processamentos 1 a 6* e não existência de hierarquia dupla
8 - *Compatibilização de Elementos Complexos e Simples*	Sucesso na *Reestruturação Hierárquica*
9 - *Compatibilização de Complexos*	Sucesso na *Reestruturação Hierárquica*
10 - *Remoção de Elementos Coleção*	Sucesso na *Reestruturação Hierárquica*

5.4 Efeito sobre os trabalhos relacionados

Dentre os trabalhos relacionados, todos aqueles que levam em conta a estrutura e não possuem a figura de um usuário especialista beneficiam-se dos *PPs*, em menor ou maior grau. No caso de trabalhos relacionados à detecção de similaridade estrutural por distância de edição [CHA 97, CHA 96, NIE 02, KAI 04], os *PPs* diminuem as operações de edição entre as instâncias. Desta forma, a distância de edição entre elas representa mais fielmente a real diferença semântica entre elas, e não a diferença entre suas estruturas devido simplesmente à diferente organização e identificação das informações.

Em outros trabalhos, que calculam a similaridade entre as estruturas das instâncias por métodos alternativos à distância de edição [FLE 05, MEL 02b, BUT 04], a melhora originada a partir da aplicação dos *PPs* tem um impacto diferente. Sobre a abordagem de Flesca et al. [FLE 05], o impacto varia conforme a forma de codificar a estrutura do documento adotada. Na abordagem de Melnik et al. [MEL 02b], dado seu processo iterativo, é difícil estimar previamente se o impacto será alto. Em D. Buttler [BUT 04], que utiliza o conjunto de possíveis *paths* das instâncias como unidade de comparação, o impacto depende da posição na hierarquia dos elementos corrigidos pelos *PPs*. Caso estejam no topo da hierarquia, o impacto será alto e positivo, pois influencia um grande conjunto de *paths* da instância.

Nos trabalhos onde existe a figura de um usuário especialista para definir os elementos a serem comparados [CAR 03, WEI 05, PUH 06, WEI 06], os *PPs* podem auxiliar em uma possível automatização do processo de comparação quando instâncias de múltiplos esquemas estão envolvidas. Entende-se que esta automatização seja possível pela adoção de uma *instância guia*, na qual seriam definidos os elementos a serem considerados para a comparação. Nas demais instâncias, os *PPs* seriam aplicados e os elementos equivalentes aos destacados na *instância guia* seriam utilizados para a comparação.

Para trabalhos relacionados à clusterização de dados [WAN 06, YAN 05], o impacto dos pré-processamentos pode melhorar o processo ao resolver diferenças entre os nomes das propriedades das instâncias.

Para trabalhos que abstraem a estrutura, como Weis et al. [WEI 04], o benefício

é parcial. Como é realizada uma abstração da estrutura, os *PPs* terão impacto quando os nomes dos elementos das instâncias considerados na comparação (que não foram abstraídos) forem compatibilizados por eles - um dos requisitos para Weis et al. [WEI 04] considerar dois objetos (elementos) equivalentes é ambos possuírem a mesma *tag*.

Para trabalhos que se orientam pela estrutura das instâncias para a comparação, e não possuem a figura de um usuário especialista [DOR 04, MIL 06, MA 05], o impacto dos pré-processamentos é alto. Em Dorneles et al. [DOR 04], pode-se melhorar a comparação entre a árvore de pesquisa e as instâncias XML sendo analisadas através da compatibilização entre as estruturas das árvores, permitindo que mais propriedades sejam consideradas. Em Milano et al. e Mac et al. [MIL 06, MA 05], os pré-processamentos permitem que estes trabalhos, focados em instâncias sob um mesmo esquema, sejam aplicados sobre instâncias com esquemas não compatibilizados previamente, pela redução das diferenças entre suas estruturas.

5.4.1 Particularidades dos pré-processamentos

De forma geral, os pré-processamentos relacionados às *tags* das instâncias têm impacto sobre qualquer trabalho que considere, tanto de forma parcial quanto completa, a estrutura das instâncias na comparação. Eles representam a quase totalidade dos trabalhos estudados.

Para os trabalhos relacionados baseados em distância de edição entre árvores, as correções executadas pelos pré-processamentos podem reduzir a distância de edição, conseguindo destacar instâncias de dados equivalentes de outras não equivalentes. Com relação a trabalhos que abstraem a estrutura, o impacto dos pré-processamentos ligados a *tags* é reduzido, sendo considerável apenas em casos mais específicos, como em Weis et al. [WEI 04], onde terá um impacto alto caso identifique o nome de dois elementos sendo comparados como equivalentes e compatibilize os mesmos.

O *PP Reestruturação Hierárquica* tem um impacto alto em todos os trabalhos relacionados. Em trabalhos baseados em distância de edição entre árvores, ele permite a redução considerável do número de transformações necessárias nas estruturas das instâncias. Em trabalhos baseados na comparação de propriedades comuns, ele permite a comparação

de duas instâncias antes não comparáveis pela ausência de dados em posições compatíveis (qualquer um dos trabalhos baseados em vetores ou nas árvores das instâncias). Mesmo os trabalhos que realizam abstração da estrutura, como Weis et al. [WEI 04], podem beneficiar-se deste pré-processamento. Suas abstrações podem não chegar ao ponto de anular os problemas gerados pela inversão hierárquica.

Em trabalhos baseados nas estruturas das instâncias para comparar dados, o *PP Compatibilização de Elementos Complexos e Simples* terá um impacto alto caso corrija elementos importantes para identificar as instâncias como semelhantes (por exemplo, o elemento `nome` em duas instâncias de `autor`, onde um deles é complexo - `nome` e `sobrenome` - e o outro é simples). Em trabalhos baseados em distância de edição de árvores, ele permite que a distância de edição represente apenas a diferença em termos de conteúdo dos elementos compatibilizados, evitando que diferenças na sua estrutura prejudiquem a comparação.

Em trabalhos baseados em distância de edição entre árvores, o *PP Compatibilização de Complexos* tem um impacto alto. Como estes se baseiam na similaridade estrutural, sem a aplicação do *PP* a remoção dos elementos não comuns acaba gerando uma diferença entre as instâncias comparadas falsa, que não representa uma diferença semântica entre as mesmas. Da mesma forma, em trabalhos que utilizam a estrutura como guia da comparação, a segunda abordagem (que transforma elementos não-comuns em subelementos de tipo *lista*) deste *PP* permite que elementos antes incomparáveis sejam agora comparados, auxiliando a comparação entre as instâncias. Para trabalhos que realizam abstração da estrutura e consideram somente os dados, ao remover os elementos não-comuns, reduz-se a massa de dados apenas aos dados comparáveis entre as instâncias, melhorando a identificação de instâncias equivalentes pela remoção do *ruído* contido nas mesmas. Isto será exemplificado nos estudos de caso apresentados neste capítulo.

Para o *PP Remoção de Elementos Coleção*, o impacto sobre os trabalhos relacionados é alto em trabalhos que se guiam pela estrutura das instâncias. Para trabalhos baseados em distância de edição, o impacto também é alto pois evita que várias alterações sejam feitas na estrutura das instâncias para compatibilizar as mesmas.

A Tabela 5.2 apresenta o impacto dos pré-processamentos sugeridos sobre os tra-

Tabela 5.2: Impacto dos *PPs* sobre os trabalhos relacionados

Trabalho	Pré-processamentos									
	PP_a	PP_b	PP_c	PP_d	PP_e	PP_f	PP_g	PP_h	PP_i	PP_j
[BUT 04]	A	A	A	A	A	A	A	B	A	A
[CAR 03]	A	A	A	A	A	A	A	A	X	A
[DOR 04]	A	A	A	A	A	A	A	A	A	A
[CHA 97]	A	A	A	A	A	A	A	A	A	B
[FLE 05]	A	A	A	A	A	A	A	A	A	A
[NIE 02]	A	A	A	A	A	A	A	A	A	A
[WEI 04]	A	A	A	A	A	A	A	A	X	A
[WEI 05]	A	A	A	A	A	A	A	A	X	A
[WEI 06]	A	A	A	A	A	A	A	A	X	A
[PUH 06]	B	B	B	B	B	B	B	B	X	X
[WAN 06]	B	B	B	B	B	B	B	B	X	X
[YAN 05]	A	A	A	A	A	A	A	A	X	B
[MA 05]	A	A	A	A	A	A	A	A	X	B
[PAR 05]	A	A	A	A	A	A	A	A	X	X
[MIL 06]	A	A	A	A	A	A	A	A	X	A
	A - Impacto alto B - Impacto baixo X - Sem impacto									
	PP_a - *Preparação Léxica* PP_b - *Uniformização da Escrita* PP_c - *Uniformização de Termos* PP_d - *Remoção de stop-words* PP_e - *Remoção de Prefixos e Sufixos* PP_f - *Casamento de Termos Compostos* PP_g - *Reestruturação Hierárquica* PP_h - *Compatibilização de Elementos Complexos e Simples* PP_i - *Compatibilização de Complexos* PP_j - *Remoção de Elementos Coleção*									

balhos relacionados estudados. Os valores na tabela indicam o grau de possível impacto de cada um dos pré-processamentos com relação aos principais trabalhos relacionados estudados. Como pode ser visto na tabela, os *PPs* apresentados tem um alto impacto na grande maioria dos trabalhos, melhorando o resultado dos mesmos.

5.5 Seqüência de execução

Os pré-processamentos apresentados podem ser aplicados de forma independente, ou seja, não existe nenhuma dependência estrita de execução entre eles. Entretanto, para gerarem resultados com qualidade, observar possíveis dependências entre os mesmos e seqüênciá-los corretamente é um fator importante para sua aplicação. Com este objetivo,

estuda-se agora a melhor forma para aplicá-los.

Em relação ao *PP Preparação Léxica*, ele deve ser aplicado antes dos demais, por resolver diferenças na escrita das *tags* que prejudicam a qualidade dos demais *PPs*. Ele uniformiza *tags* compostas por mais de uma palavra, auxiliando a execução de outros *PPs* (por exemplo, o *PP Casamento de Termos Compostos* e o *PP Remoção de stop-words*). Já o *PP Uniformização da Escrita* deve ser posterior a *Preparação Léxica*, porém anterior aos demais, por solucionar problemas que prejudicariam a execução destes algoritmos posteriormente. A combinação do *PP Uniformização da Escrita* com a *Preparação Léxica* também é possível. Pode-se executar a *Preparação Léxica* à medida que as *tags* são analisadas pela *Uniformização da Escrita*.

O *PP Uniformização de Termos* melhora ao ter sua execução posterior à *Preparação Léxica* e pode ser executado em conjunto com o *PP Uniformização da Escrita*, pois beneficia-se dele e pode aproveitar a varredura das estruturas das instâncias executada pelo mesmo. O *PP Remoção de stop-words* deve ser executado após o *PP Preparação Léxica*, pois necessita das transformações feitas pelo mesmo. Sua execução deve, entretanto, ser anterior à *Uniformização da Escrita*, pois pode beneficiá-la ao remover *stop-words* que dificultariam a identificação de termos equivalentes pois, uma vez removidas as *stop-words*, a diferença reside em erros de grafia nos termos que as compõe. O *PP Remoção de Prefixos e Sufixos* deve ser executado em conjunto com a *Preparação Léxica* e anterior aos demais *PPs*.

O *PP Casamento de Termos Compostos* pode ser executado em conjunto com os *PPs* de *Uniformização da Escrita* e de *Uniformização de Termos* e após a execução dos *PPs Preparação Léxica, Remoção de stop-words e Remoção de Prefixos e Sufixos*. Sua execução posterior a estes se deve ao fato de que a preparação léxica é necessária para identificar as palavras que compõem as *tags*. A remoção de *stop-words* ajuda em casos onde o número de palavras compondo termos equivalentes é distinto devido à existência de preposições, por exemplo. Ainda, o *PP Remoção de Prefixos e Sufixos* também auxilia da mesma forma que *PP Remoção de stop-words*, removendo termos que poderiam gerar uma quantidade de palavras diferentes compondo os termos e, assim, posteriormente impedindo a identificação de termos equivalentes pelo *PP Casamento de Termos Com-*

postos.

O *PP Reestruturação Hierárquica* deve ser aplicado após os pré-processamentos relacionados às *tags* dos elementos, de forma que diferenças na escrita das mesmas não prejudiquem a identificação de conflitos hierárquicos. Sua aplicação deve ser anterior a dos demais pré-processamentos relacionados à estrutura das instâncias, pois eles tomam em consideração a hierarquia dos elementos para seu processamento. Diferenças nas hierarquias prejudicam ou impedem sua execução com sucesso.

Os *PPs* de *Compatibilização de Elementos Complexos e Simples*, *Compatibilização de Complexos* e *Remoção de Elementos Coleção* podem ser aplicados em conjunto, mas após o *PP Reestruturação Hierárquica.* O *PP Compatibilização de Complexos* deve ser executado apenas após o *PP Compatibilização de Elementos Complexos e Simples*, pois um elemento complexo transformado em simples não precisa mais ser considerado pelo *PP Compatibilização de Complexos*. Já o *PP Remoção de Elementos Coleção* deve ser aplicado após o *PP Compatibilização de Elementos Complexos e Simples*, pois um elemento complexo transformado em simples não necessita ser analisado por ele.

A Tabela 5.3 apresenta uma visão geral das dependências entre os *PPs* de acordo com as considerações feitas nesta seção. Cabe ressaltar que estas dependências são *desejáveis* para que os algoritmos obtenham bons resultados, porém não são obrigatórias.

Tabela 5.3: Dependências entre os *PPs*

	Algoritmo	**Depende de**
1	*Preparação Léxica*	-
2	*Uniformização de Termos*	-
3	*Uniformização da Escrita*	1
4	*Remoção de Prefixos e Sufixos*	-
5	*Remoção de stop-words*	1
6	*Casamento de Termos Compostos*	1, 5, 4
7	*Reestruturação Hierárquica*	1 a 6
8	*Compatibilização de Elementos Complexos e Simples*	1 a 7
9	*Compatibilização de Complexos*	1 a 7
10	*Remoção de Elementos Coleção*	3, 7

5.6 Estudos de caso

Esta seção apresenta alguns estudos de caso com o objetivo de analisar alguns aspectos específicos dos pré-processamentos. Esta análise visa também demonstrar como os pré-processamentos conseguem resolver casos comuns dos problemas que se propõem a tratar.

5.6.1 Uniformização das *tags*

Os *PPs* relacionados à compatibilização das *tags* das instâncias podem ser aplicados sobre grandes conjuntos de instâncias XML. Desta forma, o custo de aplicação de todos os algoritmos em seqüência reduz-se consideravelmente e aumenta-se a possibilidade de sucesso dos mesmos. Para exemplificar esta situação, considera-se o conjunto de *tags* da Figura 5.5. Ele foi elaborado pela variação do termo *data de nascimento* considerando as diversas variações na grafia que busca-se resolver com os *PPs* apresentados. Todas os termos representam o mesmo conceito: a *data de nascimento* de uma pessoa, animal, etc. Porém, a grafia das mesmas não permite identificar, apenas pela comparação de seus caracteres, que elas representam a mesma informação.

Figura 5.5: Termos equivalentes à *data de nascimento*

DataDeNascimento, datadenascimento, data_de_nascimento, data-de-nascimento, datanascimento, DataNascimento, data_nascimento, data-nascimento, nascimento-data, nascimento_data, nascimentodata, NascimentoData, DataDeNascmiento

Para o conjunto de *tags* do exemplo, se cada possível par de *tags* for processado por apenas um *PP* (escolhido manualmente por um especialista), consegue-se resolver apenas 24 dos pares possíveis (aproximadamente 30% dos casos). Caso seja permitido aplicar mais de um algoritmo, consegue-se resolver as incompatibilidades de 53 dos pares possíveis, o que é aproximadamente 67% dos casos.

Caso permita-se aplicar todos os algoritmos (escolhidos para cada caso, de forma manual, por um especialista), e se considere que qualquer *tag* do conjunto pode servir como *ponte* entre duas *tags*, para que possam ser identificadas como equivalentes,

consegue-se resolver as incompatibilidades de escrita entre as *tags* em 100% dos casos.

Por *ponte* entende-se a possibilidade de uma *tag a* ser identificada como equivalente a outra *tag b*, e esta *tag b* é equivalente a outra *tag c*. Mesmo que as *tags a* e *c* não sejam dadas como equivalentes pelos *PPs*, através da *ponte* feita pela *tag b*, consegue-se identificar que *a* e *c* são equivalentes.

Infelizmente, a aplicação manual não é prática nem possível no contexto em que os *PPs* são propostos. Portanto, precisa-se definir uma forma de aplicar os *PPs* obtendo o melhor resultado possível de forma automática.

Considerando a análise sobre a seqüência de execução dos algoritmos apresentada anteriormente, sugere-se inicialmente a seguinte ordem de execução: *Preparação Léxica*, *Uniformização da Escrita*, *Uniformização de Termos*, *Remoção de stop-words*, *Casamento de Termos Compostos*. Esta seqüência de aplicação do algoritmos de pré-processamento consegue resolver as incompatibilidades de escrita entre 46 dos pares possíveis do exemplo, o que representa 58% dos casos.

A partir desta seqüência base, analisou-se possíveis modificações nas mesmas e nos *PPs* contidos na mesma, buscando melhorar o resultado final. Nesta análise, detectou-se que, pela remoção do *PP Uniformização de Termos* e a inclusão do *PP Uniformização da Escrita* dentro do *PP Casamento de Termos Compostos*, como auxiliar ao processo de identificar termos equivalentes no mesmo, consegue-se um resultado superior ao da ordem básica apresentada anteriormente.

A remoção do *PP Uniformização de Termos* deve-se ao fato que o *PP Casamento de Termos Compostos* é capaz de substituí-lo (porém com custo de execução maior). A inclusão do *PP Uniformização da Escrita* dentro do *PP Casamento de Termos Compostos* conseguiu resolver os casos onde palavras compostas tinham erros de grafia que não eram capturados pela seqüência anterior de execução. Nesta nova composição dos algoritmos conseguiu-se resolver as diferenças na escrita entre 53 dos possíveis pares de *tags*, o que representa 68% dos casos.

Dos casos não resolvidos, destacou-se, em uma análise posterior, aqueles que representavam casos mais raros, como, por exemplo, *datanascimento* e *nascimentodata*. Apesar de possível, a inversão dos termos componentes desta *tag* composta é um caso a

parte e possivelmente existente apenas ao considerar *tags* em linguas distintas (*birthdate* e *datanascimento*). Desconsiderando esta situação, a aplicação dos *PPs* obteve sucesso em 84% dos casos.

5.6.2 *Compatibilização de Elementos Complexos e Simples*

Ao comparar instâncias de dados XML, alguns atributos podem apresentar diferentes formas de representação, como no exemplo da Figura 5.6. Neste caso, trabalhos que utilizam a estrutura dos elementos para fazer o casamento [DOR 04, MIL 06, MA 05] apresentam problemas para o casamento das propriedades, dada a diferença estrutural. Para estes casos, a planificação do termo complexo ajuda a comparação. Esta planificação tende a não apresentar bons resultados se a comparação dos dados dos elementos é feita via distância de edição entre o conteúdo completo deles. Porém, como a maioria dos trabalhos estudados considera a comparação por *tokens* (palavras, etc.) dos dados dos elementos, este problema não é crítico.

Figura 5.6: Exemplo de transformação de elementos complexos em simples

Instância 1	**Instância 2**
<autor> <nome> <primeiroNome>Paulo</primeiroNome> <sobreNome>Ramos</sobreNome> </nome> <livrosPublicados>SimInt, Integração</livrosPublicados> <instituicao>UFSC</instituicao> </autor>	<autor> <nome>Paulo Ramos</nome> <livrosPublicados> <livro>SimInt</livro> <livro>Integração</livro> </livrosPublicados> <instituicao>UFSC</instituicao> </autor>

Portanto, ao aplicar a transformação de elementos complexos em simples, aumenta-se a possibilidade de casar as instâncias. Esta planificação dos dados para a comparação perde a semântica associada a eles. Mesmo assim, dado o fato de não ser possível uma tentativa de casamento baseada na estrutura pelo fato de um dos elementos ser puramente textual, entende-se que é melhor permitir a comparação, mesmo que precária, entre os conteúdos. No melhor caso, consegue-se comparar os mesmos e definir um grau de similaridade. No pior caso, evita-se que a diferença estrutural acabe impedindo a comparação entre dois elementos e por conseqüente "engane" trabalhos que se orientam pela estru-

tura para a comparação, fazendo com que a comparação ocorra apenas entre parte dos atributos e gere resultados incorretos.

5.6.3 *Compatibilização de Complexos*

A compatibilização de elementos complexos propõe duas abordagens distintas, com efeitos diferentes nos trabalhos relacionados. A primeira abordagem sugerida é a remoção dos elementos não-comuns. Esta abordagem pode causar dois impactos entre os trabalhos que consideram a estrutura: trabalhos baseados na edição entre as árvores dos documentos [CHA 97, CHA 96, NIE 02, KAI 04] têm a distância de edição reduzida. Para outros trabalhos que comparam apenas a estrutura [FLE 05, MEL 02b, BUT 04], sua aplicação não faz muito sentido, pois ao aplicar o pré-processamento não haverá mais diferenças estruturais entre as instâncias.

Nos trabalhos que se guiam pela estrutura [CAR 03, DOR 04, MA 05, MIL 06] porém não a modificam, ele não influi. Nos trabalhos que abstraem a estrutura [WEI 04, BRO 98], a remoção dos elementos não-comuns auxilia em casos onde estes têm uma grande massa de dados que acabaria prejudicando a comparação. Por exemplo, um conteúdo textual longo referente à sinopse de um filme, em uma instância de dados XML que representa um filme, conforme mostra a Figura 5.7.

A segunda abordagem identifica os elementos não-comuns e insere-os como subelementos em um novo subelemento lista. Esta abordagem influi de várias formas nos trabalhos que consideram a estrutura das instâncias. Para alguns [CHA 97, CHA 96, NIE 02, KAI 04] a distância de edição é reduzida, pois agora apenas o conteúdo dos elementos terão que ser "editados" para compatibilizar as estruturas. Para outros [FLE 05, MEL 02b, BUT 04], sua aplicação não faz muito sentido pelas mesmas razões da primeira abordagem.

Para aqueles que utilizam a estrutura como guia sem modificá-la [CAR 03, DOR 04, MA 05, MIL 06], os atributos não-comuns tornam-se um novo atributo a ser analisado. Pela Figura 5.8, vê-se que a quantidade de dados em comum entre os termos não equivalentes contribui positivamente ou negativamente para o grau de similaridade entre as instâncias. Para Milano et al. [MIL 06], tal abordagem gera um novo atributo multi-valorado

Figura 5.7: Compatibilização de Elementos Complexos

Instância 1	Instância 2
<filme> <nome>A Rocha</nome> <ano>1996</ano> <sinopse>Um general (Ed Harris), herói na Guerra do Vietnã, e seus comandados se apoderam de poderosas armas químicas e se instalam na prisão de Alcatraz, com 81 reféns. De lá eles ameaçam disparar as armas sobre São Francisco se 100 milhões de dólares não forem pagos, sendo que grande parte desta quantia será para doar às famílias de soldados americanos que morreram em missões secretas e nunca foram reconhecidos pela pátria. Para combatê-los, um grupo de elite é mandado para a ilha e entre eles está um jovem especialista em armas bioquímicas (Nicolas Cage) e o único homem (Sean Connery) que escapou do presídio, só que agora ele tem de entrar e, se possível, completar sua missão e sair vivo.</sinopse> </filme>	<filme> <nome>A Rocha</nome> <ano>1996</ano> <diretor>Sean Connery</diretor> </filme>

que tem seus atributos mapeados da melhor forma possível, conforme a abordagem proposta.

Figura 5.8: Compatibilização de Elementos Complexos

Instância 1	Instância 2
<corrida> <autodromo>Córrego</autodromo> <primeiroLugar>João</primeiroLugar> <segundoLugar>Pedro</segundoLugar> <terceiroLugar>Carlos</terceiroLugar> </corrida>	<corrida> <autodromo>Córrego</autodromo> <Vencedor>João</Vencedor> <Vice>Pedro</Vice> <Terceiro>Carlos</Terceiro> </corrida>
<corrida> <autodromo>Córrego</autodromo> <lista> <item>João>/item> <item>Pedro</item> <item>Carlos</item> </lista> </corrida>	<corrida> <autodromo>Córrego</autodromo> <lista> <item>João</item> <item>Pedro</item> <item>Carlos</item> </lista> </corrida>

5.6.4 *Reestruturação Hierárquica* - casos especiais

No processo de reestruturação hierárquica, dois casos particulares requerem um tratamento especial de forma que o pré-processamento produza o resultado correto e esperado. Para estes dois casos, que são a *reestruturação hierárquica múltipla* e a *rees-*

truturação hierárquica com elemento simples, faz-se uma análise detalhada e se sugere como fazer com que o *PP Reestruturação Hierárquica* resolva corretamente as diferenças estruturais entre as instâncias.

5.6.4.1 *Reestruturação Hierárquica* múltipla

Um ponto importante sobre a reestruturação hierárquica é quando um dos elementos a ser invertido possui múltiplos filhos que devem ser invertidos. A Figura 5.9 exemplifica este caso. Neste caso, se a inversão hierárquica for aplicada sem um tratamento especial, obtém-se o resultado mostrado na Figura 5.10. Este, obviamente, não é o resultado esperado.

Figura 5.9: Exemplo de Conflito Hierárquico

Instância 1	Instância 2

<clube>
  <nome>Clube A</nome>
  <membro>
  <nome>João</membro>
  </membro>
  <membro>
  <nome>Pedro</nome>
  </membro>
</clube>
	<membro>
	  <nome>João</nome>
	  <clube>
	  <nome>Clube A</nome>
	  </clube>
	  <clube>
	  <nome>Clube B</nome>
	  </clube>
	</membro>

Figura 5.10: Execução incorreta da Reestruturação Hierárquica

Instância 1	Instância 2

<membro>
  <nome>João</nome>
  <membro>
  <nome>Pedro</nome>
  <clube>
  <nome>Clube A</nome>
  </clube>
  </membro>
</membro>
	<membro>
	  <nome>João</nome>
	  <clube>
	  <nome>Clube A</nome>
	  </clube>
	  <clube>
	  <nome>Clube B</nome>
	  </clube>
	</membro>

Para evitar tal caso, sugere-se que no processo de inversão hierárquica, ao identificar um elemento a ser invertido, inverta-se ele e todos os *irmãos* que tenham a mesma *tag*. Neste processo, para cada irmão, uma cópia do elemento sendo invertido em paralelo seria gerada. Para o exemplo da Figura 5.9, o resultado seria o da Figura 5.11. Vê-se que, com este artifício adicional no processo de inversão, permite-se a comparação, com melhores

resultados, entre as duas instâncias de dados. O aumento do volume de dados na inversão proposta justifica-se face ao comprometimento com a semântica dos dados envolvidos.

Figura 5.11: Execução correta da Reestruturação Hierárquica

Instância 1	Instância 2

<membro>
  <nome>João</nome>
  <clube>
  <nome>Clube A</nome>
  </clube>
</membro>
<membro>
  <nome>Pedro</nome>
  <clube>
  <nome>Clube A</nome>
  </clube>
</membro>
	<membro>
	  <nome>João</nome>
	  <clube>
	  <nome>Clube A</nome>
	  </clube>
	  <clube>
	  <nome>Clube B</nome>
	  </clube>
	</membro>

5.6.4.2 *Reestruturação Hierárquica* com elementos simples

Outro caso especial a tratar na *inversão hierárquica* é quando um dos elementos é um elemento simples. Um exemplo é apresentado na Figura 5.12. Neste caso, ao realizar a inversão, têm-se duas possibilidades de transformação do elemento simples em complexo, para viabilizar a inversão hierárquica:

Figura 5.12: Conflito Hierárquico com elemento simples

Instância 1	Instância 2

<clube>
  <nome>Clube A</nome>
  <cidade>Florianópolis**</nome>**
</clube>
<clube>
  <nome>Clube B</nome>
  <cidade>Florianópolis**</cidade>**
</clube>
	<cidade>
	  <nome>Florianópolis</nome>
	  <clube>
	  <nome>Clube A</nome>
	  </clube>
	  <clube>
	  <nome>Clube B</nome>
	  </clube>
	</cidade>

1. Gerar um elemento misto;

2. Gerar um subelemento para conter o valor do elemento invertido. Neste caso, a questão é a nomeação do elemento. Uma heurística possível é identificar se o elemento a ser invertido, na outra instância, tem apenas um subelemento simples - se isto for verdade, assume-se a *tag* deste elemento para o novo elemento a ser gerado.

Considerando o exemplo dado, as duas abordagens apresentariam os resultados das Figuras 5.13 e 5.14 respectivamente.

Figura 5.13: Conflito Hierárquico com elemento simples resolvido - primeira abordagem

Instância 1	Instância 2
<cidade>Florianópolis <clube> <nome>Clube A</nome> </clube> <clube> <nome>Clube B</nome> </clube> **</cidade>**	**<cidade>** <nome>Florianópolis</nome> <clube> <nome>Clube A</nome> </clube> <clube> <nome>Clube B</nome> </clube> **</cidade>**

Figura 5.14: Conflito Hierárquico com elemento simples resolvido - segunda abordagem

Instância 1	Instância 2
<cidade> <nome>Florianópolis</nome> <clube> <nome>Clube A</nome> </clube> <clube> <nome>Clube B</nome> </clube> **</cidade>**	**<cidade>** <nome>Florianópolis</nome> <clube> <nome>Clube A</nome> </clube> <clube> <nome>Clube B</nome> </clube> **</cidade>**

5.6.4.3 Conflito hierárquico cíclico

Um conflito hierárquico cíclico ocorre quando um elemento aparece como sucessor e antecessor de um mesmo elemento em uma das instâncias sendo comparadas. Neste caso, opta-se pelo não tratamento do caso, dada a dificuldade em assegurar a semântica das informações ao ser realizada a inversão das estruturas. Tal caso tende a ocorrer em elementos de estrutura complexa, onde processos de matching e integração automáticos tendem a não apresentar bons resultados devido à alta complexidade estrutural dos mesmos.

Um possível exemplo deste caso seria é mostrado na Figura 5.15. Percebe-se que o elemento `cidade` é um elemento anterior e sucessor de `vendedor`.

Figura 5.15: Exemplo de conflito hierárquico cíclico

Instância 1

```
<vendedores>
   <por_cidade>
      <cidade>
         <nome>Florianópolis</nome>
         <vendedor>
            <nome>João</nome>
         </vendedor>
         <vendedor>
            <nome>Pedro</nome>
         </vendedor>
      </cidade>
   </por_cidade>
   <vendedor_por_atuacao>
      <vendedor>
         <nome>Pedro</nome>
         <cidade>Joaçaba</cidade>
         <cidade>Florianópolis</cidade>
      </vendedor>
   <vendedor_por_atuacao>
</vendedores>
```

5.7 Algoritmos combinados

A partir da análise realizada sobre os pré-processamentos apresentados e considerando as dependências e relações entre eles, sugerem-se agora algoritmos que combinam os *PPs*, reduzindo seu custo de aplicação e melhorando o resultado de sua aplicação.

5.7.1 Algoritmos relacionados a *tags*

Conforme já comentado durante a análise dos estudos de caso sobre os algoritmos, o *PP Uniformização de Termos* pode ser descartado quando a aplicação do *PP Casamento de Termos Compostos* é realizada. Com base na ordem de aplicação sugerida nos estudos de caso e considerando a alteração do *PP Casamento de Termos Compostos* proposta, sugere-se o Algoritmo 13 para compatibilizar as *tags* da instâncias sendo comparadas. Este combina os *PPs Remoção de Prefixos e Sufixos, Preparação Léxica, Uniformização da Escrita, Uniformização de Termos, Remoção de stop-words*.

5.7.2 Algoritmos relacionados à estrutura

Em relação aos *PPs* relacionados à estrutura das instâncias, pode-se combinar a aplicação de todos eles à exceção do *PP Reestruturação Hierárquica*, que deve ser aplicado isoladamente, dada as inversões hierárquicas que precisa realizar, assim como a

Algoritmo 13 : Primeira composicao de algoritmos (algoritmosCompostos1 - $O(n^2)$)

Entrada: Conjunto de termos a uniformizar - A
Saída: Mapeamento entre os termos equivalentes - M

```
proc algoritmosCompostos1(A)
    1. Para cada a em A
      2. Para cada b em A
        a1 ← removeSeparadores(a)
        b1 ← removeSeparadores(b)
        3. Se (a1 ≡ b1)
          4. Se(a1 ≡ a)
            adicionaMapeamento(a, b, M)
            remove(a, A)
            AbortaLoop
          Se não
            adicionaMapeamento(b, a, M)
            remove(b, A)
            ContinuaLoop
          Fim Se
        Fim Se
        a1 ← procRemStopWords(ProcLexico(a))
        b1 ← procRemStopWords(ProcLexico(b))
        5. Se compostaEquivalente(a1, b1) Então
          adicionaMapeamento(a, a1, M)
          adicionaMapeamento(b, a, M)
        Fim Se
      Fim Para
    Fim Para
Retorna M
```

dependência dos demais *PPs* com a correta hierarquia dos elementos para sua aplicação. O Algoritmo 14 introduz a versão combinada destes algoritmos.

5.8 Aplicação conjunta *vs* em pares

Em relação aos pré-processamentos apresentados, a aplicação sobre um conjunto de instâncias pode ser feita a cada par possível de instâncias ou sobre todas as instâncias ao mesmo tempo. As duas formas de aplicação, dependendo do pré-processamento considerado, têm vantagens e desvantagens.

Nos algoritmos relacionados as *tags* dos elementos, que incluem *Preparação Léxica*, *Uniformização da Escrita*, *Uniformização de Termos*, *Remoção de stop-words*, *Remoção de Prefixos e Sufixos* e *Casamento de Termos Compostos*, a aplicação pode ser feita sobre todo o conjunto de instâncias, sem desvantagens do ponto de vista de qualidade do resultado. Pelo contrário, conforme demonstrado no estudo de caso sobre estes *PPs*, a aplicação sobre o conjunto inteiro de *tags* pode trazer benefícios ao processo.

Com relação ao *PP Reestruturação Hierárquica*, a aplicação sobre um conjunto de

Algoritmo 14 : Segunda composição de algoritmos (algoritmosCompostos2 - $O(n^2)$)

Entrada: Conjunto de nodos raízes das instâncias a compatibilizar - A
Saída: Conjunto de nodos raízes das instâncias compatibilizadas - A

```
proc algoritmosCompostos2(A)
    1. Para cada a em A
       tags ← TagsElementosFilhos(n)
       2. Se (Tamanho(tags) = 1) ∧ (ElementoColecao(a, tags[0]))
          remove(a,A)
          3. Para cada f em ElementosFilhos(a)
             TrocaPai(f, Pai(a))
             adicionaFilhos(A, filhos(a))
          Fim Para
       Fim Se
    Fim Para
    L ← agrupaPorTags(A)
    4. Para cada l em L
       simples ← simplifica(l)
       5. Se¬simples
          filhos ← ∅
          6. Para cada k em l
             filhos ← FilhosDe(k) − filhos
          Fim Para
          7. Para cada k em l
             removeFilhos(k, filhos)
             adicionaFilhos(B, filhos(k))
          Fim Para
          algoritmosCompostos2(B)
       Fim Se
    Fim Para

Retorna(A)
```

instâncias tende a prejudicar sua execução, pois pode impedir a correção de hierarquias inversas por detectar casos de conflito hierárquico cíclico, onde um elemento é ao mesmo tempo antecessor e sucessor de outro elemento.

Tal fato pode ocorrer por considerar todas as instâncias em conjunto. Entretanto, ao considerá-las em pares, isto pode não ocorrer e a correção hierárquica ser realizada com sucesso. Por exemplo, se considerar-se três instâncias onde duas delas apresentam conflitos hierárquicos, entre as duas a correção do conflito seria possível. Mas caso a terceira instância tenha um conflito cíclico com as mesmas *tags* que as outras duas instâncias, o algoritmo acaba não corrigindo nenhuma delas, quando poderia pelo menos corrigir as duas primeiras.

Considerando que o *PP Reestruturação Hierárquica* foi aplicado sobre o conjunto de instâncias de uma única vez ou que não foi aplicado, pode-se considerar a aplicação dos pré-processamentos restantes relacionados à estrutura das instâncias em conjunto. Tal aplicação, porém, da mesma forma que a reestruturação hierárquica, pode acarretar resultados inferiores em relação à uma aplicação par a par, ou mesmo resultados inválidos.

No caso do *PP Compatibilização de Elementos Complexos e Simples*, a aplicação sobre o conjunto pode resultar na transformação desnecessária de elementos complexos em simples, que poderiam ser mantidos como complexos para serem comparados com algumas das instâncias, respeitando melhor a semântica dos dados.

Já o *PP Compatibilização de Complexos* sofre do mesmo problema do *PP Compatibilização de Elementos Complexos e Simples*. Ao ser aplicado em conjuntos de instâncias, remove propriedades dos elementos não presentes em todos os elementos. Entretanto, se algumas instâncias dentro do conjunto possuíssem tais propriedades, pelo menos na comparação entre elas as propriedades poderiam ser utilizadas, caso o *PP* não as removesse. Desta forma, a comparação possivelmente obteria um melhor resultado (se tais propriedades permitissem, por exemplo, distinguir objetos muito próximos porém distintos).

5.9 Conclusão

Este capítulo realiza uma análise mais detalhada dos algoritmos de *PP* apresentados no Capítulo 4. Esta análise foi composta inicialmente de um estudo sobre o impacto dos algoritmos em trabalhos relacionados. Este estudo demonstrou como os *PPs* apresentam, na grande maioria dos casos, um impacto positivo nos trabalhos relacionados.

Em seguida, analisou-se as dependências desejáveis entre os algoritmos e estabeleceu-se uma ordem sugerida para sua execução. A partir desta ordem eleita, foram estabelecidos algoritmos combinados para a aplicação dos *PPs*, que, em conjunto com os estudos de caso feitos para cada um dos *PPs*, completaram a análise feita, neste capítulo, sobre os *PPs*.

O próximo capítulo apresenta alguns testes preliminares, feitos a partir da implementação dos algoritmos de *PPs*, com o objetivo de confirmar seu impacto sobre alguns dos trabalhos relacionados.

Capítulo 6

Experimentos

6.1 Introdução

Uma vez revisados os trabalhos existentes sobre similaridade entre dados semi-estruturados, em especial instâncias de dados no padrão XML, foram apresentandos pré-processamentos para otimizar a similaridade estrutural entre dados no padrão XML, que têm o objetivo de melhorar a qualidade da comparação dos trabalhos relacionados existentes. No capítulo anterior fez-se um estudo de alguns aspectos específicos dos pré-processamentos introduzidos, assim como o estudo de alguns casos particulares de sua aplicação.

Faz-se necessário, após este estudo, demonstrar, através de experimentos, como os pré-processamentos apresentados são efetivamente capazes de melhorar a qualidade das comparações entre as instâncias de dados efetuadas pelos trabalhos relacionados. Este capítulo apresenta um ambiente de experimentos e avalia os resultados da aplicação dos pré-processamentos.

Inicialmente são definidas as massas de dados, bem como a forma pela qual elas foram preparadas para os experimentos. Define-se também os trabalhos relacionados eleitos para serem utilizados durante os experimentos. Em seguida, definem-se conjuntos de aplicação dos pré-processamentos, visando analisar o impacto deles no processo de comparação dos trabalhos relacionados eleitos. A aplicação conjunta dos algoritmos também é considerada.

Uma vez definido o ambiente de experimentação, detalha-se como os experimentos foram feitos e como os resultados foram obtidos, visando uma análise posterior. Por

fim, faz-se uma análise geral dos resultados obtidos, destacando os benefícios dos pré-processamentos aplicados.

6.2 Definições iniciais

Duas massas de dados de diferentes domínios foram utilizadas nos experimentos. A primeira massa de dados vem do domínio de publicações científicas, mais especificamente as publicações ocorridas durante o evento VLDB 2006[1], obtidas a partir do site DBLP [2]. Para estas publicações, os seguintes dados foram considerados: *titulo, autores, ano de publicação, evento, página, url*. A idéia nesta massa de dados é simular uma consulta sobre algumas fontes de dados sob o seguinte critério: *"publicações feitas no VLDB 2006"*.

A segunda massa de dados utilizada foi obtida a partir de um banco de dados da área médica, de uma clínica que executa exames diversos em Santa Catarina. Os dados contém exames executados por pacientes em um dado período. Dada a natureza sigilosa dos dados, não foram recuperadas informações como CPF ou RG, a fim de garantir a privacidade dos mesmos. Os nomes dos pacientes foram trocados para o nome de outros pacientes, de forma que não fosse possível identificar uma pessoa pela combinação *nome e data de nascimento*. Desta segunda massa de dados, os seguintes dados foram considerados: *nome do paciente, data de nascimento, nome do exame, código do exame, data do exame, hora do exame, médico responsável pelo exame, convênio sob o qual o exame foi realizado, valor dos exames*.

Para fins de aplicação, os algoritmos de pré-processamento foram agrupados em 6 grupos distintos: no grupo S_1 foram incluídos os seguintes algoritmos: {PPs *Preparação Léxica*, *Uniformização da Escrita*, *Uniformização de Termos*, *Casamento de Termos Compostos*, *Remoção de stop-words*}. O segundo grupo, S_2, incluiu o *PP Remoção de Prefixos e Sufixos*. O terceiro grupo, S_3, compõe-se do *PP Compatibilização de Elementos Complexos e Simples*. O quarto grupo, S_4, incluiu o *PP Compatibilização de Complexos*. O penúltimo grupo, S_5, compõe-se do *PP Reestruturação Hierárquica*. O último grupo,

[1] http://aitrc.kaist.ac.kr/ vldb06/
[2] http://www.informatik.uni-trier.de/ ley/db/

S_6, compõe-se de todos os pré-processamentos definidos, aplicados na ordem definida no capítulo anterior.

Estes grupos foram definidos buscando separar os *PPs* conforme as características dos problemas que visam resolver e a forma como os resolvem. O grupo S_1 trata de incoerências na escrita das *tags* oriundas de diferentes nomenclaturas ou erros na escrita. O grupo S_2 trata de incoerências na escrita das *tags* porém oriundas da estruturação das instâncias. O grupo S_3 relaciona-se aos algoritmos que realizam uma abstração da estrutura, no caso apenas o *PP Compatibilização de Elementos Complexos e Simples*. Já o grupo S_4 contém os pré-processamentos que alteram a estrutura da instância pela remoção de elementos com valor semântico - no caso, apenas o *PP Compatibilização de Complexos*. O penúltimo grupo, S_5, contém os algoritmos que modificam a organização da estrutura da instância sem remover elementos, no caso o *PP Reestruturação Hierárquica*. Já o último grupo, S_6, aplica todos os algoritmos definidos, para analisar sua performance quando aplicados em conjunto, considerando a análise do capítulo anterior.

Para analisar o impacto dos algoritmos de pré-processamentos sobre os trabalhos relacionados, escolheu-se algoritmos que representassem as duas categorias essenciais de trabalhos que os pré-processamentos buscam auxiliar: aqueles baseados na estrutura para a comparação [DOR 04, MIL 06, MA 05] e aqueles que utilizam, porém alteram [CHA 97, CHA 96, NIE 02], a estrutura das instâncias comparadas. Os trabalhos que abstraem a estrutura não foram considerados pelo fato dos *PPs* não terem um impacto alto neles, e por eles não serem os mais adequados a processos de comparação com vistas a uma posterior integração.

Considerando os trabalhos relacionados, diversos deles poderiam ser utilizados nos testes, porém os resultados tenderiam a ser muito semelhantes. Isto se deve ao fato de que, apesar de variar a forma como calculam a similaridade, a dependência da estrutura para seus processos de comparação é muito similiar. Desta forma, decidiu-se escolher duas formas de comparação que melhor representassem as duas categorias de trabalhos. O primeiro trabalho eleito foi o de Milano et al. [MIL 06], que representa aqueles que tomam a estrutura como guia para a comparação, porém não realizam qualquer modificação sobre a mesma, comparando apenas os elementos equivalentes. Este trabalho foi eleito

por ser um dos mais recentes e por ter uma abordagem completamente automatizada, que é coerente com o contexto no qual os *PPs* são propostos.

O segundo trabalho escolhido foi a distância de edição entre árvores implementada pelo pacote *SimPack* [3]. Sua distância de edição considera operações de inclusão, exclusão e substituição de nodos. A escolha deste trabalho, em detrimento de outros trabalhos analisados, deve-se ao fato que este trabalho permite considerar dados e estrutura durante o processo de comparação das árvores das instâncias (sem distinguir dados de estrutura, tal como os trabalhos relacionados). Ao mesmo tempo, sua implementação de distância de edição entre árvores tende a apresentar resultados semelhantes aos outros trabalhos [CHA 97, CHA 96, NIE 02], pois os aspectos que estes melhoram não são suficientes para sanar a maioria dos problemas que são tratados pelos *PPs* e que causam o real impacto negativo na qualidade da comparação (nomeação de nodos, inversão hierárquica, etc.).

O pacote *SimPack* também foi escolhido por ter uma implementação já testada e aplicada em diversos trabalhos [ZIE 06b, ZIE 06a, BER 06].

6.3 Execução

Para a implementação dos testes, adotou-se a linguagem Java[4] dada a disponibilidade de recursos (bibliotecas, *frameworks*) úteis aos experimentos. Para a carga e manipulação dos dados no padrão XML, adotou-se a biblioteca Xerces[5] da fundação Apache[6]. Para a geração dos gráficos dos resultados, foi utilizada a biblioteca JFreeChart[7].

O algoritmo de Milano et al. [MIL 06] foi implementado seguindo o algoritmo apresentado no artigo, que define o método de comparação proposto. Para o algoritmo de distância de edição de árvores, como já comentado, foi adotada a implementação elaborada na biblioteca *SimPack*. Os pesos definidos para as operações foram unitários e iguais para simular uma aplicação em uma massa de dados da qual não se tem conhecimento prévio, e portanto não se pode definir pesos particulares às operações.

[3] http://www.ifi.unizh.ch/ddis/simpack.html
[4] http://java.sun.com/
[5] http://xerces.apache.org/
[6] http://www.apache.org/
[7] http://www.jfree.org/jfreechart/

Para a geração das massas de dados, foi utilizado o seguinte critério: das massas de dados apresentadas anteriormente, foram eleitas aleatoriamente 40 instâncias em cada experimento. A partir de cada uma destas instâncias de dados, três versões alternativas foram geradas, com estruturas definidas especificamente para cada teste realizado, com o objetivo de simular os problemas que os pré-processamentos buscam solucionar. Estas versões alternativas foram geradas por um aplicativo onde, a partir de um esquema fornecido e uma instância de dados (oriunda de uma das fontes de dados utilizadas), uma nova instância de dados é gerada a partir dos dados da instância original.

Em cada teste, apenas os dados de uma das fontes foi utilizado. Esta forma de obtenção e geração dos dados para testes busca representar uma situação real de consulta sob várias fontes de dados. A inclusão de erros, como a inversão ou remoção de letras, nos dados das instâncias, não foi considerada, para buscar identificar a melhora específica no processo de comparação devido à correção das diferenças estruturais entre as instâncias promovida pelos *PPs*.

Em cada experimento, um dos conjuntos de pré-processamentos definido anteriormente foi aplicado, utilizando um dos dois algoritmos de comparação escolhidos para comparar as instâncias. Para cada execução, foram colhidos os dados de precisão e revocação sem e com a aplicação do conjunto de pré-processamentos. A precisão e revocação foram calculadas varrendo o resultado do processo de comparação, utilizando como critério de ordenação o valor de similaridade obtido pelo algoritmo. A partir destes dados, um gráfico precisão *vs* revocação foi gerado, buscando demonstrar a melhora pela aplicação do conjunto de pré-processamentos.

Para os algoritmos que necessitam de suporte de repositórios semânticos (*Uniformização de Termos*, *Remoção de stop-words*, *Casamento de Termos Compostos*, *Remoção de Elementos Coleção*) foi utilizado o *Wordnet* em conjunto com um repositório semântico elaborado para os testes, contendo as equivalências entre alguns termos dos domínios trabalhados, assim como as *stop-words* consideradas. Tal repositório fez-se necessário visto não se conhecer, atualmente, repositórios semânticos com as características necessárias para os experimentos realizados. A criação destes repositórios é um tópico de trabalho futuro, discutido no Capítulo 7.

6.4 Resultados

O primeiro experimento realizado, E_1, busca demonstrar a melhora pela aplicação do conjunto S_1 de pré-processamentos. Foram utilizadas as estruturas da Figura 6.1 para gerar as instâncias do teste. A partir destas instâncias, o processo de comparação sem e com os pré-processamentos foi executado e os resultados podem ser vistos nas Figuras 6.2 e 6.3. Vê-se a melhora obtida pelo pré-processamento, com valores excelentes de precisão e revocação uma vez que as diferenças nas estruturas das instâncias foram removidas.

Figura 6.1: Esquemas do Experimento E_1

Dados bibliográficos	Dados médicos
`artigo( nome, autor, ano, evento, pagina, url )`	`atendimento( nome_do_paciente, exame, medico, data, hora, valor, laudo )`
`article(title , author, year, event, pages, site )`	`appointment( patient-name, exam, doctor, day, time, cost, result )`
`artigo( titulo, escritor, ano, congresso, paginas, www )`	`atendimento(nome_cliente, procedimento, clinico, data, horario, custo, resultado)`

Figura 6.2: Resultados do Experimento E_1

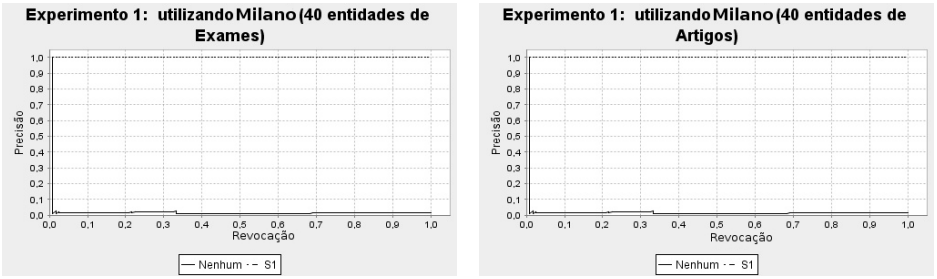
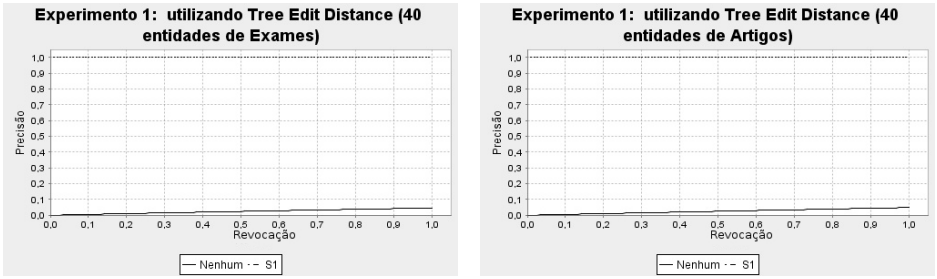


Figura 6.3: Resultados do Experimento E_1



Cabe comentar que o aumento na revocação em determinados pontos dos gráficos ocorre devido à forma de ordenar os resultados das comparações realizadas. Adotou-se uma ordenação dos resultados baseada apenas no valor de similaridade encontrado, para representar uma situação real de análise de similaridade entre dados de domínios desconhecidos. Dado este fato, ocorreram casos onde instâncias similares tiveram valores de similaridade iguais ao de instâncias distintas, e tal fato levou as instâncias distintas em alguns momentos a ficarem acima das instâncias equivalentes nos *rankings* de similaridade gerados, fazendo a revocação iniciar baixa e aumentar em seguida.

No segundo experimento, E_2, analisou-se o impacto do conjunto de pré-processamentos S_2. As instâncias utilizadas da fonte de dados bibliográficos tinham as estruturas da Figura 6.4. Conforme vê-se nas Figuras 6.5 e 6.6, houve melhora nos resultados da comparação pela aplicação dos pré-processamentos. A queda brusca de revocação no gráfico sem a aplicação dos *PPs* deve-se a forma de ordenação dos resultados pela similaridade unicamente, que neste caso gerou um grupo de elementos realmente equivalentes no início dos resultados, e em seguida ocorreu a mistura entre elementos equivalentes e erroneamente identificados como equivalentes na ordenação, o que ocasionou a queda na revocação.

Figura 6.4: Esquemas do Experimento E_2

Dados bibliográficos	**Dados médicos**

`artigo( nome, autor, ano, evento, pagina )`
 	
`artigo( artigo_nome, artigo_autor, ano, evento, pagina, artigo_url )`
 	
`artigo( nome_artigo,autor_artigo, ano, evento, pagina, url_artigo )`
	`atendimento( paciente_atendimento, exame_atendimento, data, hora, valor, laudo )`
	
	`atendimento( paciente,exame,medico,data,hora,valor,laudo )`
	
	`atendimento( atendimento_paciente, atendimento_exame, atendimento_medico, data, hora, valor, laudo )`

Novamente, devido à forma de ordenação dos resultados das comparação, a revocação aumentou no decorrer da análise. Um fator que contribuiu para esta particularidade dos resultados no caso da comparação feita via *Tree Edit Distance* foi que ela não diferencia dados de conteúdo, portanto as diferenças entre os nomes dos elementos foram tratadas da mesma forma que as diferenças entre os conteúdos das instâncias. Assim, instâncias distintas (por terem conteúdos diferentes) muitas vezes obtiveram um valor para

Figura 6.5: Resultados do Experimento E_2

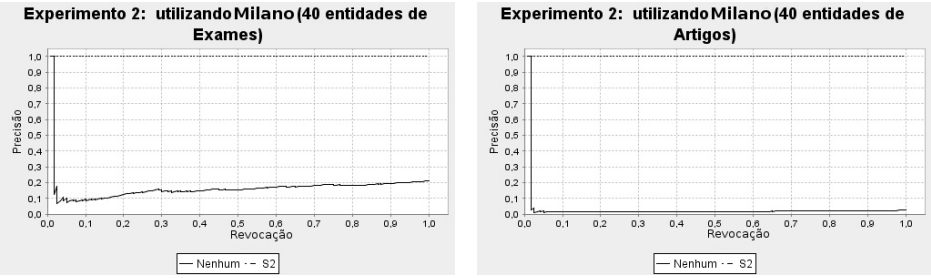
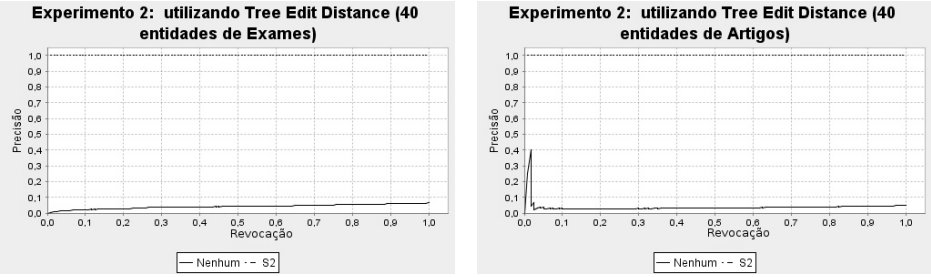


Figura 6.6: Resultados do Experimento E_2



a *distância de edição* entre elas igual ao de instâncias similares, devido ao fato destas últimas, dadas as diferentes estruturas, apresentarem uma distância de edição igual a das instâncias distintas.

O terceiro experimento, E_3, verificou como a conversão de elementos complexos para simples quando o mesmo elemento na outra instância é simples influencia o processo de comparação. Neste experimento, as instâncias utilizadas da fonte bibliográfica possuem estruturas conforme mostra a Figura 6.7. Nas Figuras 6.8 e 6.9 vê-se a melhora que o pré-processamento do conjunto S_3 teve na comparação das instâncias.

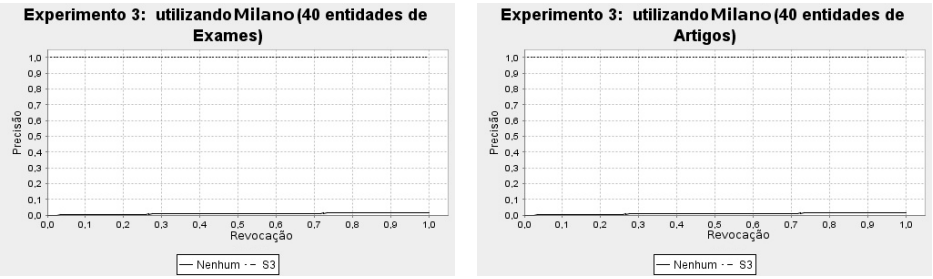
A melhora apresentada vem do fato que, a partir da simplificação dos elementos complexos, foi possível comparar elementos que antes não eram comparáveis por nenhuma das duas abordagens. No caso de Milano et al. [MIL 06], a diferença entre as estruturas fazia a comparação ser feita apenas pelas propriedades que eram comparáveis entre as instâncias, originando muitos casos de falsos positivos. No caso da *Tree Edit Distance*, a grande diferença estrutural entre as instâncias (dada a complexidade mais alta

Figura 6.7: Esquemas do Experimento E_3

Dados bibliográficos	Dados médicos

`artigo(titulo(principal), autores(autor(nome)), ano, evento(nome), paginas(conjunto), url(link))`
 	
`artigo(titulo, autores, ano, evento, paginas, url)`
 	
`artigo( titulo(principal), autores(autor), ano, evento, paginas(conjunto(inicial_final)), url(link(completo)) )`
	`atendimento(paciente(nome), exame(identificacao), medico(dados(nome)), data, hora, valor(completo), laudo)`
	
	`atendimento(paciente, exame, medico, data, hora, valor, laudo)`
	
	`atendimento(paciente(nome(completo)), exame(identificacao(nome)), medico(dados), data, hora, valor, laudo)`

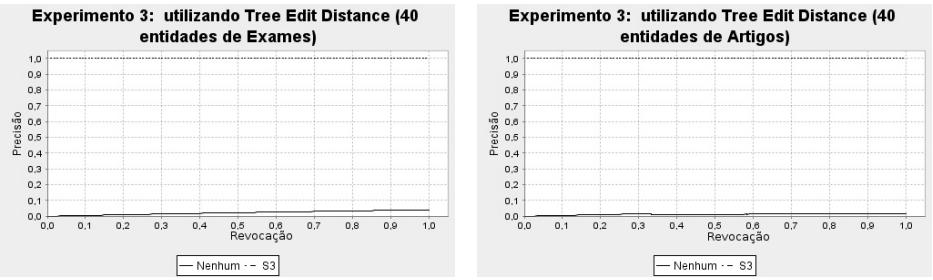
Figura 6.8: Resultados do Experimento E_3



daquelas que estruturavam os dados de forma mais detalhada) gerou uma grande quantidade de falsos negativos, pois instâncias equivalentes foram dadas como distintas pela alta distância de edição, tão alta quanto a de instâncias distintas, haja vista o fato já comentado da *Tree Edit Distance* não distinguir estrutura de dados.

O quarto experimento, E_4, analisou a influência do conjunto de pré-processamentos S_4. Este conjunto compatibiliza os elementos complexos das instâncias. As instâncias utilizadas tinham estruturas definidas conforme a Figura 6.10. Pelas Figuras 6.11 e 6.12,

Figura 6.9: Resultados do Experimento E_3



tal pré-processamento influencia os algoritmos baseados em distância de edição de árvores. Entretanto, no caso de algoritmos baseados na estrutura como guia da comparação, como Milano et al. [MIL 06], a influência do mesmo é reduzida ou nula.

Figura 6.10: Esquemas do Experimento E_4

Dados bibliográficos	Dados médicos
artigo(titulo, autores, evento)	atendimento(paciente, exame, medico, data, hora, valor, laudo)
artigo(titulo, autores, ano, evento, paginas, url)	atendimento(paciente, exame, medico, data)
artigo(titulo, autores, url)	atendimento(paciente, exame, medico, data, hora)

Figura 6.11: Resultados do Experimento E_4

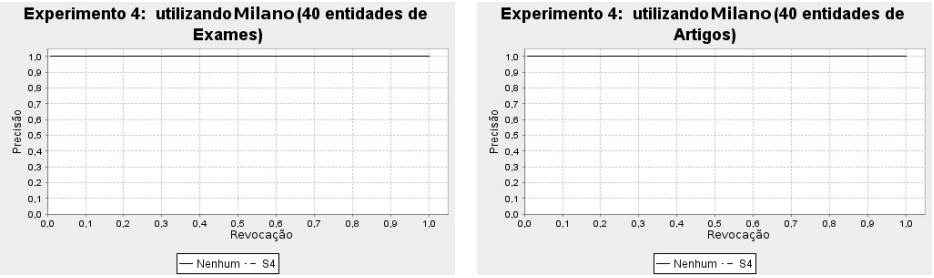
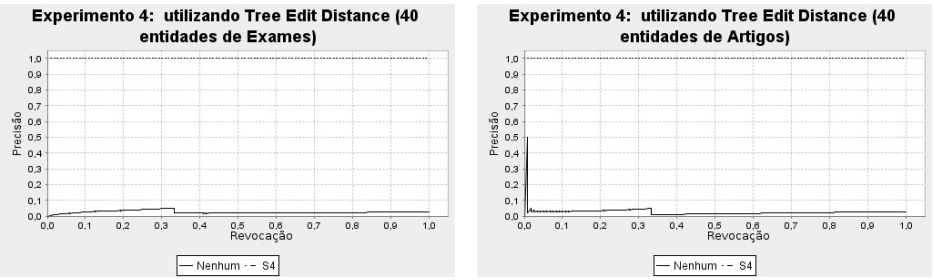


Figura 6.12: Resultados do Experimento E_4



Esta influência nula deve-se ao fato que de, se um elemento não está presente em ambas as instâncias, ele não entrará no cálculo da comparação, portanto sua influência será nula. Como o experimento E_4 focou-se em instâncias com conjuntos distintos de

propriedades, porém sem problemas em relação as propriedades em comum, o impacto do conjunto S_4 de *PPs* torna-se nulo.

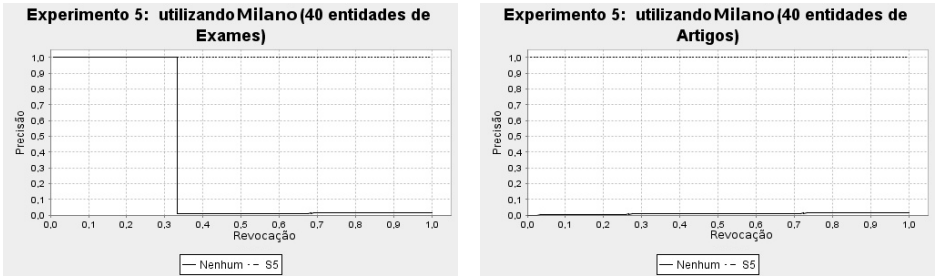
O penúltimo experimento, E_5, testou a influência que diferenças na hierarquia dos elementos têm sobre o processo de comparação das instâncias. Para tal, elaborou-se as estruturas apresentadas na Figura 6.13. Conforme vê-se nas Figuras 6.14 e 6.15, o impacto dos pré-processamentos na qualidade dos resultados da comparação justifica sua aplicação. A queda brusca observada nos resultados sem a aplicação dos *PPs* deve-se às mesmas razões citadas para o experimento

Figura 6.13: Esquemas do Experimento E_5

Dados bibliográficos	**Dados médicos**

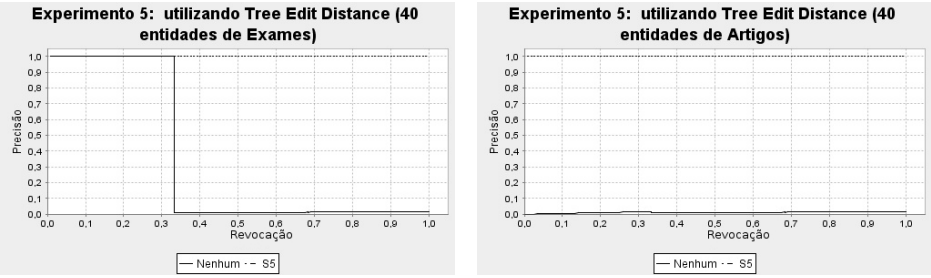
`referencia(artigo(titulo, autor(nome), ano, evento(nome), paginas, url))`
 	
`referencia(autor(nome, artigo(titulo, ano, evento, paginas, url)))`
 	
`referencia(evento(nome, artigo(titulo, autor(nome), ano, paginas, url)))`
	`atendimento(exame(agendamento(data, hora, paciente, medico), nome, valor, laudo))`
	
	`atendimento(agendamento(exame(nome, valor, laudo),paciente, medico, data, hora))`
	
	`atendimento(agendamento(exame(nome, valor, laudo), paciente, medico, data, hora))`

Figura 6.14: Resultados do Experimento E_5



Neste experimento, as diferentes hierarquias entre as instâncias levou a comparação via *Tree Edit Distance* a identificar distâncias de edição muito altas para instâncias similares, devido ao alto número de operações que precisou realizar para tornar as instâncias iguais. Este alto número de operações deve-se à necessidade de remover uma série de nodos das árvores das instâncias de dados e reinserí-las, de forma a tornar suas estruturas iguais. No caso de Milano et al. [MIL 06], as diferenças na hierarquia levaram a um grande número de falsos positivos, pois sem elementos para comparar, o algoritmo com-

Figura 6.15: Resultados do Experimento E_5



parou as instâncias apenas pelas propriedades que encontrou, que não eram suficientes para caracterizar corretamente as instâncias como equivalents.

O último experimento, E_6, analisa a aplicação dos todos os algoritmos em conjunto seguindo a ordem apresentada no capítulo anterior, sobre instâncias com as estruturas da Figura 6.16. Neste experimento, como combina todos os *PPs* apresentados, ele apresenta todos os problemas introduzidos em experimentos anteriores. Através das Figuras 6.17 e 6.18, vê-se que os mesmos influenciam positivamente o resultado da comparação, eliminando as diferenças estruturais que prejudicavam a comparação.

Figura 6.16: Esquemas do Experimento E_6

Dados bibliográficos	**Dados médicos**
`referencia(artigo(titulo(principal), autor(nome), ano(ocorrencia), evento(nome), paginas, url(arquivo)))`	`atendimento(exame(atendamento(data, hora, paciente, medico ), nome, valor)))`
`reference(author(name, article(article_title, year, event(name), pages, url)))`	`appointment(schedule(exam(name, cost), patient, doctor, date, time))`
`reference(author(name, article(article_title, year, event(name), pages, url)))`	`atendimento( agendamento( exame( exame_nome, exame_valor), agendamento_cliente, agendamento_medico, agendamento_data, agendamento_hora))`

6.5 Conclusão

Este capítulo definiu um ambiente de teste e executou um conjunto de experimentos visando analisar o impacto que os pré-processamentos propostos têm sobre a comparação de instâncias de dados no padrão XML. Estas instâncias foram geradas a partir de dados reais, utilizando estruturas elaboradas contendo diferenças estruturais, com o intuito

Figura 6.17: Resultados do Experimento E_6

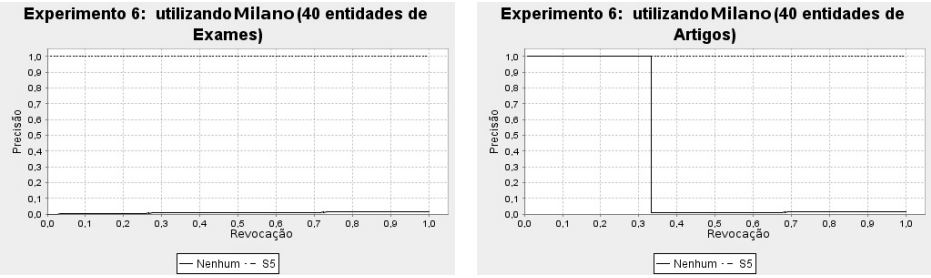
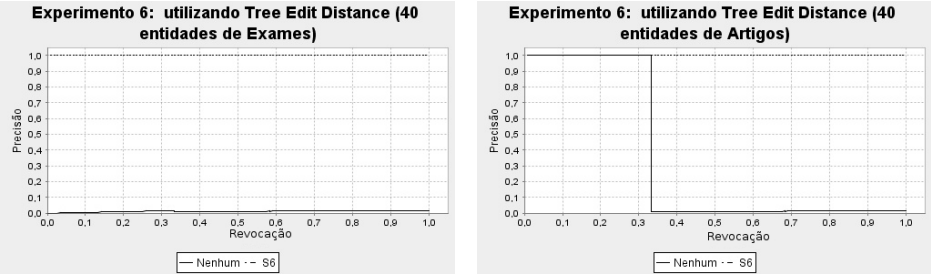


Figura 6.18: Resultados do Experimento E_6



de verificar se os pré-processamentos resolviam estes problemas. Pelos experimentos realizados, verificou-se que os resultados são satisfatórios e justificam a aplicação dos pré-processamentos.

Pelos experimentos, ficou claro também a importância que estruturas compatíveis têm no processo de comparação. Em Milano et al. [MIL 06], o algoritmo de comparação considera que as estruturas sejam compatíveis para que o resultado obtido seja adequado. Já no algoritmo de distância de edição de árvores, tal fato não é uma prerrogativa para a comparação. Contudo, ele é importante para que os resultados sejam adequados, conforme ficou claro nos experimentos realizados.

Face a estes fatos, a aplicação dos pré-processamentos propostos é justificada, dado o impacto positivo que os mesmos tiveram nas abordagens de comparação analisadas.

Capítulo 7

Conclusão

Com o advento da Web, o acesso à informação em escala mundial tornou-se uma realidade. Dada a facilidade de acesso e publicação, ela é um dos principais meios de troca de informações entre as pessoas. Cada vez mais dados de diferentes domínios e de diferentes origens (bibliotecas, universidades, órgãos governamentais, etc.) têm sido publicados na Web. Esta publicação ocorre em diversos formatos, porém, dois deles se destacam: HTML e XML. O primeiro apresenta-se como uma linguagem para formatação e criação de páginas na Web, sem características semânticas. Já no formato XML, parte da semântica dos dados é mantida pela forma como estes são organizados [CAR 03, MEL 02a].

Apesar do aumento no número de dados disponibilizados na Web, em *fontes de dados*, isto não significa necessariamente que a informação diponível tenha aumentado no mesmo ritmo. Para que dados tornem-se informação, faz-se necessário que os mesmos sejam acessados e consultados de forma prática [MEL 02a]. Para que este fato torne-se realidade, um primeiro esforço é determinar os dados equivalentes entre duas ou mais fontes de dados. Para uma ou duas fontes de dados comparadas, um processo manual é possível. Quando aumenta o número de fontes ou elas não são conhecidas previamente, abordagens mais automatizadas fazem-se necessárias.

Para determinar as instâncias de dados equivalentes entre as fontes, diversas abordagens já foram propostas, as quais foram revisadas no Capítulo 3, com relação à forma de trabalho e aspectos que consideram. Desta revisão, ficou claro que a estrutura das instâncias desempenha um papel importante na maioria dos trabalhos revisados, o que justifica

a busca de meios de torná-las o mais homogêneas possíveis. Esta busca foi o foco deste trabalho.

Este trabalho apresentou um conjunto de algoritmos de pré-processamento com o objetivo de reduzir diferenças estruturais entre instâncias de dados no padrão XML. Esta redução visa facilitar a comparação das mesmas pelos trabalhos relacionados à análise de similaridade entre dados no padrão XML. Esta uniformização das estruturas visa tanto permitir comparações com maior qualidade (ao tornar os campos equivalentes mais próximos quanto à nomenclatura e estrutura) quanto também permitir a comparação entre instâncias de esquemas distintos que estejam sob um mesmo domínio e possam vir a representar a mesma entidade.

Esta busca por uma uniformização das estruturas das instâncias se justifica pelo fato que, em instâncias de dados XML, a estrutura das mesmas é o melhor indicativo da semântica dos dados, ou seja, a estrutura das instâncias deve ser entendida como um guia para compará-las. Neste caso, diferenças na forma das estruturas, oriundas não de diferentes dados, mas apenas de diferentes representações dos mesmos, não devem interferir na comparação. Portanto, quanto menores (ou inexistentes) forem as diferenças entre as instâncas (do ponto de vista estrutural), melhor será a comparação entre as mesmas do ponto de vista semântico.

Os algoritmos de *PPs* propostos têm como característica fundamental reduzir as diferenças estruturais entre instâncias de dados XML que contém a mesma informação, através do uso de um conjunto de heurísticas para identificar elementos equivalentes. A fim de torná-los aptos à realidade que a *Web* trouxe, eles não requerem que exista um *schema* dos dados disponível, haja vista que não é um requisito de instâncias de dados XML possuir um esquema que descreva a sua estrutura.

Vale observar, porém, que o trabalho apresentado não é "definitivo" ou capaz de resolver todos os casos e possibilidades, mas sim um primeiro passo em um processo de pesquisa para buscar formas de tentar melhorar o processo de comparação de instâncias XML. Por isso, define-se uma abordagem mais aberta e passível de adaptações e evoluções, de forma que possa ser melhorada ou estendida.

Conforme demonstrado pelos experimentos e estudos de caso, a estrutura desempe-

nha um papel crítico na comparação de dados XML. Isto fica demonstrado, também, pela necessidade de boa parte dos trabalhos relacionados estudados de considerarem que as instâncias de dados comparadas já têm esquemas compatíveis. Tanto trabalhos que não modificam [DOR 04, MIL 06, MA 05] quanto trabalhos que modificam a estrutura [CHA 97, CHA 96, NIE 02] para comparar as instâncias sofrem com as diferenças estruturais das mesmas e se beneficiam consideravelmente com os algoritmos de pré-processamento propostos.

Muitos dos trabalhos atuais de comparação entre documentos XML foca apenas nos elementos dos mesmos, utilizando basicamente duas formas diferentes para definir quais elementos comparar: todos eles, atribuindo pesos ou não aos mesmos; ou comparar apenas alguns elementos, atribuindo ou não pesos aos mesmos. Em ambos os casos, a intervenção de um usuário geralmente é necessária para melhorar os resultados dos processos quando instâncias de fontes distintas são comparadas. Tal abordagem, à medida que aumenta o número de fontes na *Web*, tornar-se-á proibitiva, dada a necessidade de um usuário especialista para definir parâmetros à comparação.

Abordagens que automatizam tanto o processo de comparação quanto o de parametrização da comparação irão possibilitar melhores resultados. Tais abordagens, para que obtenham sucesso, necessitarão de estruturas compatíveis entre as instâncias. Este trabalho apresenta uma contribuição neste contexto, ou seja, propõe um mecanismo automático de compatibilização das estruturas de instâncias XML através de um conjunto de algoritmos de *pré-processamento*.

Neste trabalho, através de um estudo dos trabalhos relacionados à comparação de documentos XML, averigou-se a importância de estruturas compatíveis entre as instâncias para a quase totalidade dos mesmos, de modo que os resultados produzidos fossem satisfatórios. Com isto em mente, estabeleceu-se um conjunto de pré-processamentos, responsáveis por alterar instâncias sendo comparadas de forma que as diferenças estruturais fossem reduzidas ou até eliminadas, produzindo instâncias com estruturas equivalentes.

Para cada pré-processamento apresentado, demonstrou-se os casos onde ele reduz as diferenças estruturais, assim como elaborou-se um algoritmo para exemplificar seu funcionamento, assim como suas complexidades foram estabelecidas. Em todos os algo-

ritmos, a complexidade não ultrapassou $O(n^2)$, o que foi considerado um bom resultado, pois manteve-se a complexidade em escala polinomial.

Através de alguns estudos de caso, demonstrou-se como os *PPs* propostos são capazes de corrigir diferenças estruturais entre as instâncias e se analisou de forma breve o impacto dos mesmos em trabalhos relacionados. Estudou-se também como os *PPs* se relacionam, identificando o impacto que um *PP* tem sobre o resultado dos outros *PPs*, se aplicado antes ou depois. A partir desta análise estabeleceu-se uma ordem adequada à aplicação dos mesmos, buscando otimizar o resultado final de sua aplicação. Além desta análise da ordem de aplicação, definiu-se também possíveis combinações para a sua aplicação, conseguindo-se, com isto, reduzir a complexidade e custo para aplicação dos mesmos.

A fim de testar os pré-processamentos, duas fontes de dados de domínios distintos foram escolhidas e várias possíveis estruturas para representar os seus dados foram geradas. Os algoritmos de pré-processamento foram aplicados sobre estes conjuntos de instâncias, conseguindo resolver com sucesso as diferenças estruturais entre as instâncias e demonstrando como estas diferenças, que antes levavam os algoritmos de comparação escolhidos a péssimos resultados, foram eliminadas e permitiram aos algoritmos produzirem resultados com excelentes níveis de precisão e revocação.

Uma vez apresentados os algoritmos de pré-processamento e demonstradas as melhoras promovidas pelos mesmos, convém comentar sobre suas limitações e possíveis trabalhos futuros visando reduzir estas limitações.

Uma primeira limitação dos algoritmos propostos é a dependência de repositórios semânticos capazes de identificar termos equivalentes entre as estruturas das instâncias. Entre estas estruturas identificam-se ontologias, bases de dados terminológicas, thesauri, etc. Estas estruturas têm sido objeto de estudo constante, tendo sofrido grandes avanços. Entretanto, ainda sofrem com a pouca popularidade e utilização. Esta baixa adoção vêm principalmente da dificuldade em gerá-las.

Em relação a este fato, entretanto, entende-se que, com o aumento do uso da Web como forma de publicação das informações, estes repositórios passem a ganhar mais relevância e passem a ser adotadas por mais grupos e instituições. Assim sendo, eles

tendem a desenvolver-se, crescendo em recursos e dados disponíveis, o que por conseqüência influenciaria positivamente os pré-processamentos propostos. Um projeto nesta área é o *EuroWordNet*[VOS 97], que busca implementar um repositório semântico equivalente ao *Wordnet*, porém, entre várias línguas. Já existem também alguns trabalhos [BEN 00, FAR 98, FEL 98, KWO 98] no sentido de gerar novos repositórios semânticos mais completos a partir de outros existentes, o que auxilia também a popularização e adoção deles.

Outra possível melhora nos algoritmos de pré-processamento é a criação de um sistema de *cache* que permita identificar estruturas de instâncias já analisadas anteriormente e a partir dos dados contidos no *cache* promover as alterações nas instâncias diretamente, sem a necessidade de analisar as mesmas. Desta forma, ganhar-se-ia performance à medida que o número de instâncias tratadas aumentasse.

A pesquisa de outros possíveis algoritmos de pré-processamento também é um objeto de estudo futuro, assim como melhorias nos pré-processamentos existentes. Dentre as possíveis melhoras nos algoritmos existentes, cita-se o tratamento de casos mais complexos pelo algoritmo de reestruturação hierárquica. Outra melhoria seria possibilitar ao algorimo de tratamento de elementos complexos identificar elementos equivalentes baseados em sua estrutura e conteúdo, armazenando tal conhecimento para que, posteriormente, os algoritmos relacionados às *tags* sejam capazes de corrigí-los já durante seu processamento. Por exemplo, identificar que *datanascimento* e *datadenascimento* são equivalentes.

Em relação ao *PP Reestruturação Hierárquica*, uma melhora para o mesmo seria estabelecer heurísticas para definir qual das duas instâncias deveria ser analisada e qual deveria sofrer as inversões de hieraquia, tendo em vista aspectos como performance ou qualidade da transformação.

A transformação do trabalho atual em um *WebService*, que possa ser utilizado por processos de integração, é outro trabalho em desenvolvimento. Em [SAN 07] são descritos os aspectos da implementação de um *framework* de comparação entre instâncias de dados XML. Este *framework* já preve a aplicação dos algoritmos de pré-processamento (tanto em conjunto, utilizando as ordens estabelecidas neste trabalho, quanto individual-

mente). No momento os algoritmos de pré-processamento estão sendo integrados ao *framework* e este terá posteriormente uma interface *Web* para a utilização de suas funções. Tal *framework* também servirá para a execução de testes mais extensos e em diversas situações com os *PPs* apresentados.

Referências Bibliográficas

[ANA 02] ANANTHAKRISHNA, R.; CHAUDHURI, S.; GANTI, V. Eliminating fuzzy duplicates in data warehouses. In: VERY LARGE DATA BASES, 2002. Morgan Kaufmann, 2002. p.586–597.

[BEN 00] BENTIVOGLI, L.; PIANTA, E.; PIANESI, F. **Coping With Lexical Gaps When Building Aligned Multilingual Wordnets**.

[BER 01] BERGAMASCHI, S. et al. Semantic integration of heterogeneous information sources. **Data & Knowledge Engineering**, [S.l.], v.36, n.3, p.215–249, 2001.

[BER 06] BERNSTEIN, A.; KIEFER, C. Imprecise RDQL: towards generic retrieval in ontologies using similarity joins. In: Haddad, H., editor, SYMPOSIUM ON APPLIED COMPUTING, 2006. ACM, 2006. p.1684–1689.

[BRO 98] BRODER, A. On the resemblance and containment of documents. In: SEQS: SEQUENCES '91, 1998. [s.n.], 1998.

[BUT 04] BUTTLER, D. A short survey of document structure similarity algorithms. In: INTERNATIONAL CONFERENCE ON INTERNET COMPUTING, 2004. [s.n.], 2004. p.3–9.

[CAR 03] CARVALHO, J. C. P.; DA SILVA, A. S. Finding similar identities among objects from multiple web sources. In: Chiang, R. H. L.; Laender, A. H. F.; Lim, E.-P., editors, WEB INFORMATION AND DATA MANAGEMENT, 2003. ACM, 2003. p.90–93.

[CHA 96] CHAWATHE, S. S. et al. Change detection in hierarchically structured information. In: PROCEEDINGS OF THE ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1996. **Proceedings...** New York: ACM Press, 1996. v.25, 2 of **ACM SIGMOD Record**, p.493–504.

[CHA 97] CHAWATHE, S. S.; GARCIA-MOLINA, H. Meaningful change detection in structured data. **SIGMOD Rec.**, New York, NY, USA, v.26, n.2, p.26–37, 1997.

[CHE 00] CHEN, J. et al. NiagaraCQ: A scalable continuous query system for Internet databases. **SIGMOD Record (ACM Special Interest Group on Management of Data)**, [S.l.], v.29, n.2, p.379–390, 2000.

[COH 03] COHEN, W. W.; RAVIKUMAR, P.; FIENBERG, S. E. A comparison of string distance metrics for name-matching tasks. In: Kambhampati, S.; Knoblock, C. A., editors, INFORMATION INTEGRATION ON THE WEB, 2003. [s.n.], 2003. p.73–78.

[CON 00] CONSORTIUM, W. W. W. **Extensible Markup Language (XML) 1.0 (Second edition) – W3C Recommendation**. Available at http://www.w3.org/TR/2000/WD-xml-2e-20000814.

[DAT 03] DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. Editora CAMPUS, 2003.

[DAU 02] DAUM, B. **Arquitetura de Sistemas com XML**. Editora CAMPUS, 2002.

[DOR 04] DORNELES, C. F.; HEUSER, C. A.; LIMA, A. E. N. Measuring similarity between collection of values. **Web Information and Data Management**, [S.l.], 2004.

[ELM 99] ELMAGARMID, A.; RUSINKIEWICZ, M.; SHETH, A. **Management of Heterogenous and Autonomous Database Systems**. Morgan Kaufmann, 1999.

[FAR 98] FARRERES, X.; RIGAU, G. **Using WordNet for Building WordNets**.

[FEL 98] FELLBAUM, C. Towards a representation of idioms in WordNet. In: Harabagiu, S., editor, USE OF WORDNET IN NATURAL LANGUAGE PROCESSING SYSTEMS: PROCEEDINGS OF THE CONFERENCE, p.52–57. Association for Computational Linguistics, Somerset, New Jersey, 1998.

[FLE 05] FLESCA, S. et al. Fast detection of XML structural similarity. **IEEE Transactions on Knowledge and Data Engineering**, [S.l.], v.17, n.2, p.160–175, 2005.

[GIO 99] GIONIS, A.; INDYK, P.; MOTWANI, R. Similarity search in high dimensions via hashing. In: VERY LARGE DATA BASES, 1999. [s.n.], 1999. p.518–529.

[GON 06] GONÇALVES, R.; DOS SANTOS MELLO, R. Similaridade entre documentos semi-estruturados. In: II ESCOLA REGIONAL DE BANCO DE DADOS, 2006. [s.n.], 2006.

[GON 07] GONÇALVES, R.; DOS SANTOS MELLO, R. Improving XML instances comparison with preprocessing algorithms. In: Wagner, R.; Revell, N.; Pernul, G., editors, DEXA, 2007. Springer, 2007. v.4653 of **Lecture Notes in Computer Science**, p.13–22.

[KAI 04] KAILING, K. et al. Efficient similarity search for hierarchical data in large databases. In: Bertino, E. et al., editors, INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, 2004. Springer, 2004. v.2992 of **Lecture Notes in Computer Science**, p.676–693.

[KWO 98] KWONG, O. Y. Aligning WordNet with additional lexical resources. In: Harabagiu, S., editor, USE OF WORDNET IN NATURAL LANGUAGE PROCESSING SYSTEMS: PROCEEDINGS OF THE CONFERENCE,

p.73–79. Association for Computational Linguistics, Somerset, New Jersey, 1998.

[LEN 02] LENZERINI, M. Data integration: A theoretical perspective. In: PROCEEDINGS OF THE TWENTY-FIRST ACM SIGMOD-SIGACT-SIGART SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS (PODS-02), 2002. **Proceedings...** New York: ACM Press, 2002. p.233–246.

[LU 79] LU, S.-Y. A tree-to-tree distance and its application to cluster analysis. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [S.l.], v.1, n.2, p.219–224, 1979.

[MA 05] MA, Y.; CHBEIR, R. Content and structure based approach for XML similarity. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY, 2005. IEEE Computer Society, 2005. p.136–140.

[MAR 01] MARIAN, A. et al. Change-centric management of versions in an XML warehouse. In: VERY LARGE DATA BASES, 2001. **Proceedings...** Orlando: Morgan Kaufmann, 2001. p.581–590.

[MEL 02a] MELLO, R. **Uma Abordagem Bottom-Up para a Integração Semântica de Esquemas XML**. Universidade Federal do Rio Grande do Sul, 2002. Tese de Doutorado.

[MEL 02b] MELNIK, S.; GARCIA-MOLINA, H.; RAHM, E. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 2002. **Proceedings...** San Jose, CA: [s.n.], 2002.

[MES 03] MESITI, M.; BERTINO, E.; GUERRINI, G. An abstraction-based approach to measuring the structural similarity between two unordered XML documents. In: INTERNATIONAL SYMPOSIUM ON

INFORMATION AND COMMUNICATION TECHNOLOGIES, 2003. Trinity College Dublin, 2003. p.316–321.

[MIL 95] MILLER, G. A. Wordnet: A lexical database for english. **Communications of the ACM**, [S.l.], v.38, n.11, p.39–41, 1995.

[MIL 06] MILANO, D.; SCANNAPIECO, M.; CATARCI, T. Structure aware XML object identification. In: CLEAN DATABASES, 2006. [s.n.], 2006.

[MON 96] MONGE, A. E.; ELKAN, C. P. The field matching problem: Algorithms and applications. In: Simoudis, E.; Han, J. W.; Fayyad, U., editors, INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 1996. AAAI Press, 1996. p.267.

[NAV 01] NAVARRO, G. A guided tour to approximate string matching. **ACM Computing Surveys**, [S.l.], v.33, n.1, p.31–88, 2001.

[NIE 02] NIERMAN, A.; JAGADISH, H. V. Evaluating structural similarity in XML documents. In: WEB AND DATABASES, 2002. [s.n.], 2002. p.61–66.

[PAR 05] PARK, U.; SEO, Y. An implementation of XML documents search system based on similarity in structure and semantics. In: INTERNATIONAL WORKSHOP ON CHALLENGES IN WEB INFORMATION RETRIEVAL AND INTEGRATION, 2005. IEEE Computer Society, 2005. p.97–103.

[PUH 06] PUHLMANN, S.; WEIS, M.; NAUMANN, F. XML duplicate detection using sorted neighborhoods. In: Ioannidis, Y. E. et al., editors, INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, 2006. Springer, 2006. v.3896 of **Lecture Notes in Computer Science**, p.773–791.

[REG 93] REGNEMALM, I. **The euclidean distance transform**. Linköping, Sweden: Linkpöping University, Dept. of Electrical Engineering, 1993. Tese de Doutorado.

[SAN 06] SANTOS, F. D.; DOS SANTOS MELLO, R. BDTerm: um sistema de gerenciamento de bases de dados terminológicas. In: II ESCOLA REGIONAL DE BANCO DE DADOS, 2006. [s.n.], 2006.

[SAN 07] SANTOS, F. D. et al. Um framework para suporte à preparação e comparação de similaridade de documentos XML. In: III ESCOLA REGIONAL DE BANCO DE DADOS, 2007. [s.n.], 2007.

[SHA 90] SHASHA, D.; ZHANG, K. Fast algorithms for the unit cost editing distance between trees. **Journal of Algorithms**, Duluth, MN, USA, v.11, n.4, p.581–621, 1990.

[SHA 97] SHASHA, D.; ZHANG, K. Approximate tree pattern matching. In: PATTERN MATCHING ALGORITHMS, p.341–371. Oxford University Press, 1997.

[TAI 79] TAI, K.-C. The tree-to-tree correction problem. **Journal of the ACM**, New York, NY, USA, v.26, n.3, p.422–433, 1979.

[VOS 97] VOSSEN, P.; LETTEREN, C. C. **EuroWordNet: a multilingual database for information retrieval**.

[WAN 94] WANG, J. T.-L. et al. A system for approximate tree matching. **IEEE Transactions on Knowledge and Data Engineering**, [S.l.], v.6, n.4, p.559–571, 1994.

[WAN 03] WANG, Y.; DEWITT, D. J.; YI CAI, J. X-diff: An effective change detection algorithm for XML documents. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 2003. [s.n.], 2003. p.519–530.

[WAN 06] WAN, X.; YANG, J. Using proportional transportation similarity with learned element semantics for XML document clustering. In: Carr, L. et al., editors, WORLD WIDE WEB CONFERENCE, 2006. ACM, 2006. p.961–962.

[WEI 04] WEIS, M.; NAUMANN, F. Detecting duplicate objects in XML documents. In: Naumann, F.; Scannapieco, M., editors, INFORMATION QUALITY IN INFORMATIONAL SYSTEMS, 2004. ACM, 2004. p.10–19.

[WEI 05] WEIS, M.; NAUMANN, F. DogmatiX tracks down duplicates in XML. In: Özcan, F., editor, SIGMOD CONFERENCE, 2005. ACM, 2005. p.431–442.

[WEI 06] WEIS, M.; NAUMANN, F. Detecting duplicates in complex XML data. In: Liu, L. et al., editors, INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 2006. IEEE Computer Society, 2006. p.109.

[YAN 05] YANG, J.; CHEUNG, W. K.; CHEN, X. Integrating element and term semantics for similarity-based XML document clustering. In: Skowron, A. et al., editors, WEB INTELLIGENCE, 2005. IEEE Computer Society, 2005. p.222–228.

[ZIE 06a] ZIEGLER, P. et al. Detecting similarities in ontologies with the SOQA-simpack toolkit. In: Ioannidis, Y. E. et al., editors, INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, 2006. Springer, 2006. v.3896 of **Lecture Notes in Computer Science**, p.59–76.

[ZIE 06b] ZIEGLER, P. et al. Generic similarity detection in ontologies with the SOQA-simpack toolkit. In: Chaudhuri, S.; Hristidis, V.; Polyzotis, N., editors, SIGMOD CONFERENCE, 2006. ACM, 2006. p.751–753.