Filipe Guédes Venâncio

Proposta de uma Função de Similaridade para Listas HTML Extraídas da Web

Florianópolis, SC

6 de julho de 2017

Filipe Guédes Venâncio

PROPOSTA DE UMA FUNÇÃO DE SIMILARIDADE PARA LISTAS HTML EXTRAÍDAS DA WEB

Trabalho de conclusão de curso submetido ao Bacharelado em Ciências da Computação para a obtenção do grau de Bacharel em Ciências da Computação.
Orientador: Ronaldo dos Santos Mello

Florianópolis, SC
6 de julho de 2017

Venâncio, Filipe Guédes
Proposta de uma Função de Similaridade para Listas HTML Extraídas da Web : / Filipe Guédes Venâncio; orientador, Ronaldo dos Santos Mello. - Florianópolis, SC 2017.
101 p.

- Universidade Federal de Santa Catarina, Centro Tecnológico - CTC.
.

Inclui Referências

1.Web. 2. Listas da Web. 3. Similaridade. 4. Ciências da Computação. 5. Monografia. I. Mello, Ronaldo dos Santos . II. Universidade Federal de Santa Catarina. . II. Proposta de uma Função de Similaridade para Listas HTML Extraídas da Web.

Filipe Guédes Venâncio

Proposta de uma Função de Similaridade para Listas HTML Extraídas da Web

Trabalho de conclusão de curso submetido ao Bacharelado em Ciências da Computação para a obtenção do grau de Bacharel em Ciências da Computação.

Trabalho aprovado. Florianópolis, 6 de julho de 2017

Prof^o Dr^o Ronaldo dos Santos Mello

Profª Drª Carina Friedrich Dorneles

Prof^o Dr^o Frank Augusto Siqueira

Florianópolis, SC
6 de julho de 2017

AGRADECIMENTOS

É necessário acreditar na pessoa e em seu potencial para poder incentiva-la, estimula-la. Por isso agradeço todos os dias por ter pessoas como minha mãe, minha irmã, minha namorada e demais familiares, que sempre acreditaram no meu potencial e me estimularam a seguir pelo caminho que me trouxe até este trabalho. Da mesma forma agradeço a todos os mestres que acreditaram em mim, tendo a paciência para me ensinar a questionar e até mesmo a como controlar minha curiosidade. Em especial agradeço ao professor Ronaldo, pelas inúmeras conversas e orientações. Não podendo me esquecer dos amigos que acumulei até aqui e que me acompanham nesta jornada.

“Uma longa viagem de mil milhas inicia-se com o movimento de um pé”

(Lao Tzu)

RESUMO

A Web tornou-se uma fonte rica em dados, diversificada pela popularização dos sites, redes de relacionamento e aplicativos, sendo utilizada para a extração seletiva de conteúdo útil para consumo humano. Entretanto, a extração e análise dos dados contidos na Web são um desafio devido ao crescimento das massas de dados e a variabilidade da representação destas informações. Entre estas informações estão as listas HTML, que tendem a ser apenas um agrupamento onde os itens de dados presentes nela possuem um contexto comum, como por exemplo, uma lista de informações sobre cidades ou uma lista de músicas. Alguns trabalhos relacionados buscam a comparação das listas HTML que possuam características semelhantes e que seguem um determinado padrão, pois assumem que as listas são provenientes de respostas produzidas por aplicativos e sistemas. Diferente desses trabalhos, este trabalho de conclusão de curso considera listas HTML extraídas da Web com contextos desconhecidos, que necessitam de uma análise e padronização de sua estrutura, de forma a considerar uma possível variabilidade estrutural, visando determinar se elas dizem respeito a um mesmo assunto. O objetivo principal deste trabalho é propor uma técnica de comparação entre listas HTML que resulte em um escore de similaridade que possa ser utilizado para diversas finalidades, como integração de dados e buscas aproximadas de dados com foco em listas na Web.

Palavras-chave: Dados na Web, Similaridade, Listas na Web.

LISTA DE ABREVIATURAS E SIGLAS

CSS	Cascading Style Sheets
HTML	HiperText Markup Language
Web	World Wide Web
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
UML	Unified Modeling Language

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de documento HTML. 23
Figura 2 – Exemplo de lista na Web e sua especificação em HTML. 24
Figura 3 – Exemplo de estrutura de listas heterogêneas 25
Figura 4 – Exemplo arquivo XML. 27
Figura 5 – Exemplo de formato de dados de entrada para o processo de comparação de listas na Web. 36
Figura 6 – Processo de comparação de listas na Web. 39
Figura 7 – Lista exemplo de uso - listTestA. 43
Figura 8 – Lista exemplo de uso - listTestB. 43
Figura 9 – Exemplo de uso - Matriz de similaridade. 44
Figura 10 – Exemplo de uso - Lista ordenada de coeficientes de similaridade entre linhas das 2 listas Web. 45
Figura 11 – Exemplo de uso - Registro de saída. 45
Figura 12 – *SimListLibrary* - Diagrama de classes. 46
Figura 13 – SimListLibrary - Arquivo de configuração pom.xml. . . . 47
Figura 14 – SimListLibrary - Trecho de código do teste unitário (ScopeTest). 48
Figura 15 – SimListLibrary - Trecho de código SimilarityTestCase. . 49
Figura 16 – SimListLibrary - Diagrama de classes de teste. 50
Figura 17 – Experimento 1: Lista 1 54
Figura 18 – Experimento 1: Lista 2 55
Figura 19 – Experimento 1: Lista 3 56
Figura 20 – Experimento 1: Lista 4 56
Figura 21 – Experimento 1: Lista 5 57
Figura 22 – Experimento 1: Lista 6 57

LISTA DE TABELAS

Tabela 1 – Comparativo de trabalhos correlatos. 33
Tabela 2 – Comparativo deste trabalho com os trabalhos correlatos. . 51
Tabela 3 – Tabela de configurações utilizadas na função *WebListSim*. 53
Tabela 4 – Experimento 1: Tabela de similaridades entre as listas. . 58
Tabela 5 – Resultados do Experimento 1. 59
Tabela 6 – Resultados do Experimento 2. 60
Tabela 7 – Resultados do Experimento 3. 61
Tabela 8 – Resultados do Experimento 4. 61

SUMÁRIO

Lista de ilustrações 13

Lista de tabelas 15

1 INTRODUÇÃO 19
1.1 Objetivo Geral 20
1.2 Objetivos Específicos 20
1.3 Metodologia 20
1.4 Estrutura do Trabalho 21

2 FUNDAMENTAÇÃO TEÓRICA 23
2.1 Dados na Web 23
2.2 Listas na Web 24
2.3 Documentos XML 26
2.4 Similaridade 27

3 TRABALHOS RELACIONADOS 31
3.1 Problemática Top-K 31
3.2 Tratamento de outras estruturas na Web 32
3.3 Tratamento de coleção de itens 32
3.4 Comparativo dos trabalhos correlatos 33

4 FUNÇÃO WEBLISTSIM 35
4.1 Formato dos dados de entrada 35
4.2 Função de similaridade 36
4.3 Processo de Comparação 38
4.4 Exemplo de uso 43
4.5 Desenvolvimento 46
4.6 Comparação com trabalhos correlatos 51

5 EXPERIMENTOS 53
5.1 Método de avaliação 53
5.2 Experimento 1 54

5.3 Experimento 2 59
5.4 Experimento 3 60
5.5 Experimento 4 61

6 CONCLUSÃO 63

REFERÊNCIAS 65

A CÓDIGO FONTE: WEBLISTSIM 67
A.1 Código: SimList.java 67
A.2 Código: LoaderListXML.java 69
A.3 Código: Document.java 70
A.4 Código: ListOf.java 72
A.5 Código: Line.java 73
A.6 Código: Text.java 75
A.7 Código: ListAnalyser.java 76
A.8 Código: DocumentNormalizer.java 83

B ARTIGO SBC 91

1 INTRODUÇÃO

O surgimento da Internet foi um marco na História e seu crescimento vem proporcionando muitas pesquisas relacionadas aos dados que a compõe. Muitos desses estudos são voltados para a descoberta de conhecimento através de meios computacionais, a qual é dificultada uma vez que a maior parte das informações presentes na Internet está disposta para o consumo humano e necessita de interpretação para ser compreendida.

Não diferente dos demais dados da *World Wide Web* (ou simplesmente *Web*), as listas são apresentadas das mais variadas formas, dificultando assim um tratamento computacional adequado para tarefas como extração, comparação e clusterização. Alguns estudos que tratam de listas provenientes da Web as denominam *WebLists*, sendo considerada como uma lista na Web não apenas as devidamente representadas pelas marcações HTML (*HiperText Markup Language*), mas também estruturas e conjuntos de dados que apresentam indícios de ser uma lista. Outro detalhe é que estes estudos abordam listas na Web limitando-as a um conjunto simples de dados ou inferindo um conjunto de características específicas dentro de um contexto.

Um exemplo é a proposta de Fagin, Kumar e Sivakumar (2003), que realiza comparações de listas na Web dentro do contexto da problemática *Top K* (Seção 3.1). Nesta comparação são mensuradas similaridades e dissimilaridades, bem como apresentadas noções de qualidade e equivalência de funções utilizadas na comparação, além de considerar somente listas na Web que apresentam uma importância posicional.

Assim sendo, a área de pesquisa relacionada ao gerenciamento de dados do tipo lista na Web carece ainda de soluções para o tratamento de listas quaisquer, incluindo a descoberta de listas similares na Web. Este trabalho busca então contribuir com a descoberta de listas similares na Web, apresentando uma medida genérica que resulta em um coeficiente o qual poderá ser utilizado para os mais diversos fins.

Dentre os benefícios do uso desta proposta, estão o auxilio em processos de integração de listas HTML e na busca de conteúdo na Web por similaridade. Já no contexto da extração de conhecimento, este trabalho pode ajudar com a identificação de sublistas, na contextualização e classificação de

dados.

1.1 OBJETIVO GERAL

O objetivo deste trabalho é propor uma função de similaridade para listas provenientes da Web já previamente extraídas, função essa genérica, ou seja, não influenciada por aspectos semânticos de determinadas aplicações, mas sim baseada na similaridade textual dos seus elementos e nos dados ao redor das mesmas presentes nas páginas Web.

1.2 OBJETIVOS ESPECÍFICOS

Os seguintes objetivos específicos foram identificados para atender o objetivo geral:

- *Formalização de lista genérica*: Estudar a representação de listas na Web e o entendimento de sua estrutura, visando a formalização de uma estrutura genérica que serve como dado de entrada e como base para a determinação da métrica de similaridade;
- *Construção da função de similaridade*: Definir o que são listas similares e construir um método de comparação, método esse que, através da atribuição de pesos aos componentes de uma lista genérica, fornece um coeficiente que represente quão similares são duas listas;
- *Aplicação e avaliação da medida proposta*: Elaborar um algoritmo que realize a comparação de listas, aplicando a medida de similaridade proposta. Pretende-se, ainda, avaliar este algoritmo através de experimentos sobre um grupo de listas previamente conhecidas (*grupo de controle*).

1.3 METODOLOGIA

A metodologia a ser seguida para a elaboração deste trabalho possui 3 etapas. A primeira etapa envolve o estudo da problemática e o relato de trabalhos relacionados. A segunda etapa visa a formalização de listas genéricas

e definição da função de similaridade a partir da análise de uma amostra de listas reais. Já na terceira etapa, ocorre o desenvolvimento do algoritmo de comparação das listas, assim como a execução de testes sobre uma amostra de listas reais e avaliação dos resultados obtidos.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em 6 capítulos. Este primeiro capítulo apresenta uma breve introdução, contendo a motivação para a realização deste trabalho, os objetivos e a metodologia abordada.

O segundo capítulo mostra a fundamentação teórica sobre a estrutura de listas na Web, assim como a disposição dos dados na Web, em especial, a disposição das listas. Este capítulo aborda ainda a noção de similaridade de dados e algumas métricas existentes. Na sequência, o terceiro capítulo apresenta e discute alguns trabalhos correlatos, apresentando um comparativo entre as abordagens.

O quarto capítulo apresenta a função *WebListSim* como tratamento para a problemática apresentada, mostrando a métrica de similaridade proposta e definições realizadas na construção da função, mostrando, ao final do capítulo, um pequeno exemplo de uso. O quinto capítulo descreve a realização de testes e os resultados obtidos, e as conclusões são descritas no sexto capítulo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 DADOS NA WEB

A proposta da Web apresentada por Lee (1989), formalizou os dados em documentos através da linguagem de marcação HTML, sendo estes documentos chamados de hipertextos. Um exemplo deste tipo de documento pode ser observado na Figura 1. Conforme mostra a Figura, Por ser uma linguagem de marcação, suas instruções são descritas na forma de tags que indicam os limites do documento (<*html*>) e outras formatações para a apresentação de dados nele, como <*head*>, para o cabeçalho do documento.

Figura 1 – Exemplo de documento HTML.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Example<title>
5          <link rel="stylesheet" href="styl
6      </head>
7      <body>
8          <h1>
9              <a href="/">Header</a>
10         </h1>
11         <nav>
12             <a href="one/">One</a>
13             <a href="two/">Two</a>
14             <a href="three/">Three</a>
15         </nav>
```

Fonte: Elaborada pelo autor.

Mesmo existindo o padrão HTML para a disposição de dados na Web, isto não significa que todas as características dos dados estejam devidamente declaradas. Isto ocorre pois, em suma, os dados contidos na Web são destinados para o uso humano, o que explica a influência da organização visual dos documentos através do uso de tecnologias visuais como o *CSS* (*Cascading Style Sheets*), que podem transformar uma sucessão de parágrafos em uma lista.

Abstraindo a problemática das disposições dos dados no hipertexto, Cafarella (2009) classifica os dados dispostos na Web, podendo estes serem estruturados, semiestruturados e não estruturados. Esta classificação consi-

dera a organização dos dados como um todo, definindo cada dado de acordo com sua estrutura e apresentação, e enfatizando que cada tipo de dado necessita de um tratamento adequado para ser compreendido e utilizado.

Nas definições mais usuais desses tipos de dados, um dado é entendido como *estruturado* quando possui uma definição clara de estrutura, como uma lista (Seção 2.2). No caso de dados ditos *semiestruturados*, temos padrões que tentam estruturar dados fracamente estruturados, como os documentos *XML* (*eXtensible Markup Language*) - Seção 2.3 -, que admitem longos conteúdos textuais mesclados a atributos estruturados. Já os dados *não estruturados* são representados por textos puro (ou mesmo outras mídias, como imagens), os quais não possuem uma indicação clara de estrutura, mas podem ter conteúdo útil.

2.2 LISTAS NA WEB

As estruturas de dados em forma de listas são as mais comuns entre as presentes na Web e são classificadas como dados ditos estruturados, visto que a lista tem por conceito ser uma coleção de itens que tendem a possuir um contexto comum. Esse contexto define uma correlação entre os itens e pode estar presente de maneira implícita ou explícita. O fato do contexto, na maior parte das vezes, estar implícito, diferentemente de uma tabela cujo cabeçalho auxilia na determinação da intenção dos dados, torna a comparação computacional de duas listas um desafio.

Figura 2 – Exemplo de lista na Web e sua especificação em HTML.

As 5 cidades mais populosas de SC.

- Joinville, 562.151
- Florianópolis, 469.690
- Blumenau, 338.876
- São José, 232.309
- Criciúma, 206.918

```
1 <!DOCTYPE html>
2 <hmtl> <head> <meta charset="UTF-8"> </head>
3 <body> As 5 cidades mais populosas de SC
4 <ul> <li>Joinville, 562.151</li>
5      <li>Florianópolis, 469.690</li>
6      <li>Blumenau, 338.876</li>
7      <li>São José, 232.309</li>
8      <li>Criciúma, 206.918</li>
9 </ul> </body> </html>
```

Fonte: Elaborada pelo autor.

Como demonstrado no exemplo da Figura 2, o texto disponibilizado em HTML fornece apenas indícios da existência de uma lista ao apresentar as marcações <ul> e <li> nos casos mais simples. Entretanto, uma lista pode

ser representada por outras marcações, como uma subseção de marcações <p> ou <div>, assim como foi mostrado por Agostinho e Mello (2015) ao tratar da extração de listas na Web.

Outras informações adicionais são passíveis da interpretação pelo usuário com base no seu conhecimento sobre o domínio tratado e em identificar o contexto. Neste exemplo, através do título da lista, um leitor humano pode interpretar e afirmar que a lista compreende uma coleção das cidades mais populosas dentro de uma região chamada SC. Após ter a compreensão que cada item representa uma cidade, ainda é possível ao leitor interpretar os valores de cada item, julgando que, entre os dois valores, o primeiro aparenta ser o nome da cidade e o segundo valor, por ser numérico, pode ser interpretado como o tamanho da população.

A interpretação dos dados na Web por parte de um leitor é muitas vezes subjetiva, criando assim um grau de incerteza. Este grau de incerteza, no caso de listas, é derivado da falta de mais indícios que caracterizam os dados, uma vez que, diferentemente de estruturas como tabelas, listas não possuem definição de colunas e rótulos. Este tipo de problema é recorrente ao tratarmos listas genéricas extraídas de uma origem com informações diversificadas.

Além da questão da incerteza, listas podem divergir em outras características que dificultam a sua comparação, podendo ter disposição diferenciada de seu conteúdo e variações da estrutura. Assim sendo, deve ser verificado se a lista é completa e homogênea ou se é heterogênea.

Figura 3 – Exemplo de estrutura de listas heterogêneas

Lista de carros vendidos

- Gol, 37 unidades
- Onix - Chevrolet, 29 un.
- Duster, 13 un. / SUV
- Vectra, 9 un. / SEDAN
- A3 - Audi / SPORT
- Fiat UNO (Atractive), 37 carros | COMPACTO
- UP - Volkswagen, 16 unidades

Fonte: Elaborada pelo autor.

Na Figura 3 é apresentado um exemplo de lista heterogênea, onde

pode ser observada a falta de padronização dos dados. Esta falta de padrão é aparente no uso de abreviaturas, na troca das palavras *unidades* por *carros*, na variação da ordem dos elementos presentes em cada linha e, por fim, na falta de alguns elementos que compõem uma linha, apresentando linhas incompletas.

Ao contrário das listas heterogêneas, as listas homogêneas tratam de dados que são apresentados de maneira uniforme, com a mesma quantidade de elementos e muitas vezes seguindo uma ordem para apresentar os dados de cada item. Um exemplo é a lista apresentada na Figura 2.

Uma outra característica das listas, que pode influenciar no entendimento correto da mesma, está relacionada à ordem de apresentação dos elementos, uma vez que, em muitos casos, na Web, as listas seguem regras impostas pelo autor, podendo seguir a ordem alfabética, de entrada de dados ou outra regra que seja relevante. Este problema, assim como os outros já citados, dificulta tanto na separação dos dados quanto na comparação de duas listas distintas, implicando na possibilidade de listas que tratem do mesmo assunto serem consideradas diferentes caso não apresentem os seus elementos em uma mesma ordem.

2.3 DOCUMENTOS XML

O *XML* é um padrão recomendado pela *W3C* (*World Wide Web Consortium*), que foi criado para unir a flexibilidade das linguagens de marcação com a simplicidade dos documentos *HTML*. Este padrão tem como principal objetivo disponibilizar um meio onde possa ser estruturado e compartilhado qualquer tipo de dado, de forma que qualquer *software* possa ler ou utilizar estes dados.

Um documento *XML* é definido seguindo regras específicas do padrão. Entre as regras está a valorização do conteúdo e a delimitação do mesmo utilizando *tags*. A *tag* é uma etiqueta utilizada para marcação dos dados, representando a estrutura do dado e sendo definida entre os sinais de < e />. A tag também pode ser definida entre os sinais < e >, necessitando assim ser finalizada com a repetição da mesma entre os sinais </ e >, indicando assim o fim da estrutura. Diferente do padrão HTML, onde as *tags* são fixas e com

um propósito específico, em XML as *tags* podem ser definidas de acordo com a intenção do dado que irão representar.

Figura 4 – Exemplo arquivo XML.

```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <MUSICAS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <MUSICA>
      <NOME>A Fórmula Do Amor</NOME>
      <CANTOR>Kid Abelha</CANTOR>
      <LETRA>Eu tenho gestos aptos</LETRA>
    </MUSICA>
  - <MUSICA>
      <NOME>A Viagem</NOME>
      <CANTOR>Roupa Nova</CANTOR>
      <LETRA>Há tanto tempo que eu deixei você</LETRA>
    </MUSICA>
  - <MUSICA>
      <NOME>Águas De Março</NOME>
      <CANTOR>Elis Regina</CANTOR>
      <LETRA>É pau é pedra</LETRA>
    </MUSICA>
  </MUSICAS>
```

Fonte: http://www.tomasvasquez.com.br/blog/

Como pode ser visto na Figura 4, um documento *XML* tem em sua primeira linha a declaração de suas características básicas, como a versão da linguagem *XML* utilizada e a codificação dos caracteres. Após estas definições básicas, são utilizadas as *tags*, que podem representar desde dados simples até estruturas complexas. No exemplo na Figura 4 é criada uma *tag* principal chamada <*MUSICAS*>, qual representa uma lista de dados reconhecidos como <*MUSICA*>. Cada dado <*MUSICA*> contém as *tags* <*NOME*>, <*CANTOR*> e <*LETRA*>.

O padrão XML foi utilizado neste trabalho na caracterização do arquivo de entrada, qual contém os dados das listas HTML e da página Web, de onde as lista foram préviamente extraídas. Isto devido as características de flexibilidade e simplicidade do padrão XML, assim como mostrados no exemplo.

2.4 SIMILARIDADE

A similaridade é um conceito utilizado na comparação de dados que indica quão parecidos são os dados comparados. Esta medida pode ser obtida através do uso de dois tipos de função: a de distância e a de similaridade.

No caso de uma função de distância, ocorre a verificação da dissimi-

laridade entre dois objetos, resultando em valores dentro do intervalo $[0,\infty)$. O valor apresentado indica a diferença entre os objetos, que são ditos similares com valores próximos a 0 e muito diferentes quando o valor é alto. Um exemplo de função de distância é a função de *Levenshtein*, que é utilizada para a comparação de frases. Ela conta as transformações necessárias, seja a transformação uma inserção, deleção ou troca de posição de letras, para que a frase 1 torne-se idêntica à frase 2.

Já as funções de similaridade, como mencionado por Chen, Ma e Zhang (2009), são inversas às funções de distância. Esta inversão se deve ao fato das funções de similaridade serem uma normalização das funções de distância para o intervalo $[0,1]$. Neste intervalo são identificadas as similaridades dos objetos comparados, sendo os objetos considerados mais similares quando obtidos valores próximos de 1, enquanto valores próximos de 0 significa baixa similaridade.

Neste trabalho é apresentada uma métrica baseada em funções de similaridade que compreenda as adequações necessárias para a comparação de listas provenientes da Web. Esta métrica considera como similar listas que apresentem um mesmo conteúdo, contendo um número adequado de elementos similares. Em outras palavras, duas listas são consideradas similares quando uma delas seja um subconjunto da outra, ou ambas se complementem. É importante ressaltar que a ordenação dos elementos da lista, neste caso, não é considerada interessante, visto que os dados extraídos da Web são ordenados de acordo com a intenção de cada autor e, caso fosse considerada, poderia gerar uma dissimilaridade indesejada.

Este trabalho toma como base um trabalho anterior do nosso grupo de pesquisa (Silva e Mello (2015)). Para a construir a base da nova métrica, seguindo as análises apresentadas em Silva e Mello (2015), foram escolhidas as funções *Cosseno*, *Euclides* e *Jaccard* para serem utilizadas inicialmente neste trabalho, pois estas foram as funções que apresentaram melhor desempenho na comparação de dados estruturados na Web em experimentos preliminares. A Equação 2.1, também apresentada em Silva e Mello (2015), também é aplicada neste trabalho para transformar as distâncias de Euclides e Cosseno em

coeficientes de similaridade.

$$Similaridade = \frac{1}{1 + Distancia} \tag{2.1}$$

Apesar do trabalho relatado por Silva e Mello (2015) tratar da similaridade entre tabelas na Web, as análises realizadas neste trabalho foram consideradas partindo do princípio que a lista pode ser vista como uma tabela de coluna única, podendo esta ser multivalorada e sem rótulos. Ressalta-se ainda que as análises servem como ponto de partida e que muitas definições não servem para as estruturas de lista, uma vez que as listas possuem uma estrutura diferenciada das tabelas, que necessitam de adequações específicas.

Para utilizar a função *Cosseno*, faz-se necessário calcular a frequência de cada termo (TF). A frequência de termos é obtida através da divisão da quantidade de ocorrências de uma palavra pelo total de palavras presentes no documento, sendo importante para definir os termos mais relevantes do documento. Este artifício pode ser ludibriado, uma vez que elementos textuais, como artigos e preposições de uma língua, podem inferir maior relevância devido ao seu número de ocorrências, sendo necessária a especificação de uma heurística como a frequência invertida de documentos (IDF). A IDF busca relacionar a quantidade de documentos em que uma determinada palavra aparece, de forma que palavras mais comuns se tornem irrelevantes.

A relação multiplicativa entre TF e IDF é definida pelo algoritmo *TF-IDF*, apresentado na Equação 2.2. O TF-IDF, em resumo, apresenta a frequência dos termos mais relevantes. Uma vez obtidos os valores de TF-IDF, é então calculada a dissimilaridade dos documentos através da Equação 2.3 que apresenta a função *Cosseno*.

$$fTotalTermo_{t,d} = fTermoDoc_{t,d} \cdot \ln\left(\frac{quantidadeTotalDoc}{quantidadeDocComTermo}\right) \tag{2.2}$$

$$Cosseno(W_i, W_j) = \frac{\sum_{l=1}^{t} fTotalTermo_{l,i} \cdot fTotalTermo_{l,j}}{\sqrt{\sum_{l=1}^{t} fTotalTermo_{l,i}^2} \cdot \sqrt{\sum_{l=1}^{t} fTotalTermo_{l,j}^2}} \tag{2.3}$$

Já a função *Euclidiana*, apesar de utilizar o conceito de frequências da mesma forma como a função *Cosseno*, acaba enfatizando as dissimilaridades através da diferença dos quadrados. O cálculo da função Euclidiana pode ser visto na Equação 2.4. Por fim, há a função de similaridade *Jaccard*, que mede a similaridade entre dois conjuntos de palavras utilizando os conceitos de interseção e união, sendo a interseção o número de elementos idênticos, a união o total de elementos e a similaridade definida pela intersecção dividida pela união. Esta função é apresentada na Equação 2.5.

$$Euclidiana(W_i, W_j) = \sum_{l=1}^{t} \sqrt{fTotalTermo_{l,i}^2 - fTotalTermo_{l,j}^2} \tag{2.4}$$

$$Jaccard = \frac{numeroPalavrasIguais}{numeroTotalDePalavras} \tag{2.5}$$

As funções de similaridade escolhidas para este trabalho (e recém apresentadas) são utilizadas nas comparações dos elementos estruturais das listas na Web, como no caso dos elementos e título da lista, sendo os resultados destas funções utilizados, por sua vez, em uma variação da métrica *SubSetSim* apresentada por Silva e Mello (2015), que provém da métrica *SetSim* definida em Dorneles et al. (2004) para a comparação de conjuntos de dados. A equação *SetSim* (Equação 2.6), foi construída para comparar cada elemento da lista A com todos os termos da lista B, sendo considerado apenas a maior pontuação alcançada para cada elemento. Esta equação divide o somatório das maiores pontuações de cada elemento pelo tamanho da maior lista.

$$\begin{aligned} SetSim(E_p, E_d) &= \frac{\sum_{E_p^i.n = E_d^j.n}(max(sim(E_p^i, [E_d^1, ..., E_d^m])))}{max(m,n)}, \\ onde(1 \leq i \leq n)e(1 \leq j \leq m) \end{aligned} \tag{2.6}$$

3 TRABALHOS RELACIONADOS

Dentre os trabalhos que lidam com listas na Web, são poucos os que abordam a determinação de similaridade entre listas e eles estão geralmente relacionados à problemática conhecida como *Top-K*. Desta maneira, buscou-se também trabalhos que tratam de conceitos complementares para se obter mais subsídios para o problema tradato neste trabalho. Como conceitos complementares a listas na Web, foram consideradas também abordagens que lidam com outros tipos de estruturas dispostas na Web, bem como o tratamento de coleções de itens. Esses trabalhos são descritos a seguir.

3.1 PROBLEMÁTICA TOP-K

A problemática Top-K foi originalmente concebida para resolver a desqualificação causada nos buscadores de conteúdo na Web. Essa desqualificação ocorre devido ao crescimento dos dados que, por sua vez, dificulta os processos de classificação, agregação e indexação de conteúdo, refletindo na qualidade dos resultados obtidos. Para mensurar a qualidade dos resultados obtidos nos buscadores, foi então necessária a comparação destes, os quais são apresentados como uma lista de conteúdos disponíveis na Web. Neste contexto, as listas possuem relevância posicional uma vez que os melhores resultados devem sempre aparecer nas primeiras posições da lista.

Diversas são abordagens relacionadas a essa problemática (Fagin, Kumar e Sivakumar (2003), Wong et al. (2011) e Pal e Michel (2016)), em geral, elas propõem a comparação das listas de resultado considerando os primeiros "k" elementos já ordenados pela importância de cada item. Elas definem um procedimento para a permutação da importância dos itens onde as listas são divididas em sublistas *Top-i*, onde $i \leq k$, de forma que, para cada dupla de lista *Top-i*, é feita uma comparação e a ponderação das comparações de listas *Top-i* acaba evidenciando as dissimilaridades.

Trabalhos neste contexto lidam com listas na Web, porém muito singulares. O procedimento aplicado é limitado, atendendo apenas situações onde há a necessidade de se comparar listas ordenadas.

3.2 TRATAMENTO DE OUTRAS ESTRUTURAS NA WEB

No contexto de tratamento de outras estruturas de dados presentes na Web, um trabalho relevante foi o desenvolvido anteriormente no nosso grupo de pesquisa (GBD/UFSC) por Silva e Mello (2015), o qual trata de dados na Web dispostos em forma de tabelas. O trabalho apresenta um diferencial ao introduzir o tratamento de diferentes estruturas de tabelas presentes na Web para fins de determinação de similaridade. Ao considerarmos que uma lista na Web pode ser encarada como uma tabela Web simplificada, com apenas uma coluna sem rótulo, na forma de uma tabela multivalorada composta, pode-se utilizar algumas ideias abordadas no trabalho como ponto de partida.

Dentre as ideias utilizadas, estão a utilização de elementos "*externos*"a lista presentes na página Web na comparação e as funções de similaridades sugeridas. No entanto, se tratando de listas, como não existe uma definição exata de coluna, faz-se necessário um tratamento diferenciado para dados multivalorados, o qual não é tratado no trabalho desenvolvido, uma vez que as tabelas possuem maior organização estrutural dos dados em diversas colunas.

3.3 TRATAMENTO DE COLEÇÃO DE ITENS

O trabalho de Dorneles et al. (2004) aborda a determinação de similaridade para diversas estruturas complexas, algumas representando coleções de valores contidas em documentos *XML*, além de apresentar métricas de similaridade para cada tipo de estrutura.

Dentre as métricas descritas, duas indicadas para coleções de dados foram investigadas. A primeira é a métrica *listSim*, a qual foi descartada por tratar de itens que apresentam uma mesma ordem, impondo restrições que este trabalho busca evitar. A outra métrica, a *setSim*, mostrou-se promissora ao comparar itens entre conjuntos, apesar de não tratar dados multivalorados.

Mesmo aparentemente promissora, a métrica *setSim* induz a comportamentos que podem depreciar o coeficiente de similaridade buscado. Dentre os comportamentos identificados, tem-se a caracterização do relacionamento dos itens através de uma função sobrejetora, ou seja, na comparação entre conjuntos A e B, mais de um elemento do conjunto A pode estar relacio-

nando com um mesmo elemento do conjunto B. Este comportamento não foi considerado relevante, pois admite-se a inexistência de elementos repetidos dentro de uma lista, tornando este tipo de associação (n para 1) um depreciador que pode mascarar uma dissimilaridade, ao buscar sempre pela relação com maior escore.

Um outro comportamento indesejável é a ponderação utilizada, que divide a soma das melhores comparações de itens pelo tamanho da lista com maior número de elementos, prejudicando na identificação de sublistas. Este problema pode ser resolvido pelo uso da equação *subSetSim* apresentada por Silva e Mello (2015), a qual considera como divisor o tamanho da lista com menor quantidade de elementos.

3.4 COMPARATIVO DOS TRABALHOS CORRELATOS

A Tabela 1 apresenta um comparativo dos trabalhos relacionados, indicando o tipo de estrutura que o trabalho trata para fins de comparação, o contexto onde os dados estão inseridos (dados na Web de modo geral ou dados no formato XML) e suas limitações em relação aos propósitos deste trabalho. Conforme comentado anteriormente, nenhum dos trabalhos relacionados atende plenamente os objetivos deste trabalho, porém todos eles serviram de inspiração para o seu desenvolvimento.

Tabela 1 – Comparativo de trabalhos correlatos.

Trabalho	Tipo de Estrutura	Contexto	Limitações
Pal e Michel (2016)	Lista	Web	Trata apenas lista Top-k
Wong et al. (2011)	Lista	Web	Trata apenas lista Top-k
Fagin, Kumar e Sivakumar (2003)	Lista	Web	Trata apenas lista Top-K
Silva e Mello (2015)	Tabela	Web	Trata apenas tabelas Não trata adequadamente dados multivalorados
Dorneles et al. (2004)	Lista	XML	Trata lista com ordem Não trata dados multivalorados e sublistas Comparação estilo função sobrejetora

4 FUNÇÃO WEBLISTSIM

Este capítulo apresenta a função de similaridade proposta neste trabalho, denominada função *WebListSim*, a qual leva em conta as particularidades de listas genéricas presentes da Web. Além da função propriamente dita, este capítulo detalha ainda o processo de comparação no qual ela é utilizada, bem como suas entradas, saídas, planejamento e desenvolvimento, mostrando ao final do capítulo um exemplo de uso.

4.1 FORMATO DOS DADOS DE ENTRADA

Considerando a problemática da comparação de listas presentes na Web em termos da diversificação das suas representações, foi definido um formato genérico de dados de entrada para o processo de comparação. Este formato deriva do arquivo XML resultante do extrator desenvolvido por Agostinho e Mello (2015), o qual foi modificado para considerar não apenas os componentes da estrutura de uma lista e o endereço eletrônico da página que contém a lista, mas também outros dados da página, que foram julgados relevantes por auxiliar na determinação do contexto da lista.

Na Figura 5, é apresentado um exemplo de arquivo de entrada. Ele embute o conjunto das informações consideradas relevantes na comparação dentro da *tag* raiz <*document*>. Dos elementos da página, chamados de dados externos, foram considerados o título da página disposto na *tag* <*titlePage*>, o endereço eletrônico da página em <*link*> e o parágrafo antecessor à lista em <*text*>.

Já a lista extraída é representada na *tag* <*list*>, qual contém internamente a definição da *tag* <*title*> para um possível título da lista, seguido da *tag* <*lines*> constituída por sua vez por uma ou mais *tags* <*line*>. Ddentro de cada *tag* <*line* há a *tag* <*elements*>, e dentro da *tag* <*elements*> encontra-se uma ou mais *tags* <*element*>, onde é disponibilizado o conteúdo de cada elemento da lista.

Figura 5 – Exemplo de formato de dados de entrada para o processo de comparação de listas na Web.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <document>
3    <link>http://exame.abril.com.br/seu-dinheiro/noticias/os-carros-mais-roubados-em-2012#1</link>
4    <list>
5      <title></title>
6      <lines>
7        <line>
8          <elements>
9            <element>1º lugar: Hyundai HR</element>
10           <element>Quantidade de roubados/furtados em 2012: 804</element>
11           <element>Frota em 2012: 62.179</element>
12           <element>Frequência de roubos/furtos: 1,293%</element>
13         </elements>
14       </line>
15       <line>
16         <elements>
17           <element>2º lugar: Fiat Stilo</element>
18           <element>Quantidade de roubados/furtados em 2012: 1.126</element>
19           <element>Frota em 2012: 90.896</element>
20           <element>Frequência de roubos/furtos: 1,239%</element>
21         </elements>
22       </line>
23     </lines>
24   </list>
25   <titlePage>Os carros mais roubados em 2012 | EXAME.com</titlePage>
26   <text>
27     <paragraph>Clique nas fotos e veja quais foram os 10 carros com maior índice de roubos.</paragraph>
28   </text>
29 </document>
```

Fonte: Elaborada pelo autor.

4.2 FUNÇÃO DE SIMILARIDADE

Para realizar a comparação de duas listas na Web foi elaborada neste trabalho a função de similaridade *WebListSim*. Esta nova função considera os dados de 2 listas a serem comparadas mais algumas informações externas a elas e presentes nas páginas Web onde elas se encontram, conforme descrito na seção anterior. A função retorna um coeficiente de similaridade entre os valores 0 e 1, sendo que, quanto mais perto do valor 1, mais similares são as 2 listas comparadas, podendo o coeficiente ser igual a 1 quando elas são idênticas ou quando uma das listas é um subconjunto dos elementos da outra lista.

A função *WebListSim* foi planejada separando os itens a serem comparados em dois conjuntos. O primeiro conjunto é formado por informações da página externas à lista, que são o título da página Web, o endereço eletrônico e o parágrafo antecessor à lista. A subfunção *EP* (*Element's Page*) determina a similaridade deste conjunto de informações externas. O segundo conjunto

considera os elementos propriamente ditos da lista e o título da lista, sendo calculada a similaridade deste segundo conjunto de informações calculada pela subfunção *SL* (*Similarity List*).

No cálculo da função *WebListSim*, são atribuidos pesos associados as funções *EP* e *SL*, possibilitando definir configurações que indicam a importância de cada conjunto de dados na comparação, de acordo com o objetivo desejado. Caso os pesos não forem configurados, foi assumido por *default* o peso de 30% (*trinta por cento*), para o conjunto *EP*, de forma que os outros 70% (*setenta por cento*), sejam atribuídos a *SL*.

A Equação 4.1 apresenta a função *WebListSim*. Ela recebe como parâmetros os 2 documentos XML (d_1 e d_2), no formato de entrada descrito na seção anterior, contendo às informações das 2 listas a serem comparadas, e invoca as 2 funções de similaridade secundárias *EP* e *SL*, considerando as variáveis *pesoEP* e *pesoSL*, para as funções *EP* e *SL* respectivamente.

$$WebListSim(d_1, d_2) = pesoEP \cdot EP(d_1, d_2) + pesoSL \cdot SL(d_1, d_2) \quad (4.1)$$

No cálculo da função *EP*, os componentes título da página Web (*TP*), endereço eletrônico da página Web (*EE*) e o parágrafo imediatamente anterior à definição da lista na página Web (*P*) são comparados separadamente utilizando uma função de similaridade *Sim*. Esta função pode ser escolhida entre as 3 funções de similaridade de dados textuais disponibilizadas neste trabalho: *Cosseno*, *Euclidiana* ou *Jaccard*. O resultado da função *EP* é a média aritmética simples dos coeficientes retornados por cada uma das 3 comparações, como mostrado na Equação 4.2.

$$EP(d_1, d_2) = \frac{Sim(TP_1, TP_2) + Sim(EE_1, EE_2) + Sim(P_1, P_2)}{3} \quad (4.2)$$

Já para o cálculo da função secundária *SL*, foi definida uma adaptação da métrica *setSim*, uma vez que, como comentado na Seção 3.3, ela possui alguns comportamentos indesejados para os objetivos da função proposta neste

trabalho. Entre as adaptações realizadas, estão as alterações para o reconhecimento de sublistas, a inserção do título da lista e a melhoria na comparação das linhas das listas, uma vez que a métrica *setSim* permite o relacionamento n para 1 - comportamento de *função sobrejetora* -, ou seja, ao comparar duas listas **A** e **B**, a métrica permite que uma ou mais linhas da lista B sejam relacionadas, por similaridade, com uma mesma linha da lista A, podendo degradar o método, uma vez que pode gerar uma similaridade maior que a esperada nas listas que possuem muitos elementos parecidos. O objetivo é refinar a similaridade permitindo um mapeamento 1 para 1 entre as linhas de duas listas, assumindo não existir linhas redundantes em uma lista.

A variação *subSetSim*, definida em Silva e Mello (2015), corrige o primeiro comportamento de forma a permitir o conceito de subconjuntos, o qual também é útil para tratar o problema de sublistas, trocando o denominador da métrica para considerar o menor grupo de elementos ao invés do maior. No caso do comportamento de função sobrejetora, foi alterado o somatório dos valores de similaridade dos itens para que sejam admitidas apenas relações unárias e ainda adicionado, de forma ponderada, a similaridade entre os títulos das listas (TL_1 e TL_2) através da equação *SimT*. Todas essas alterações propostas podem ser observadas na Equação 4.3, onde a função *Sim* é a função de similaridade a ser aplicada (*Cosseno*, *Euclidiana* ou *Jaccard*).

$$SL(d_1,d_2) = \frac{SimT(TL_1,TL_2) + Max(Sim(d_1^1,[d_2^1,...,d_2^m])) + ... + Max(Sim(d_1^n,[d_2^1,...,d_2^{m-(n-1)}]))}{(n+1)},$$
$$sendo\ n = \ tamanho\ da\ lista\ d_1,\ m = \ tamanho\ da\ lista\ d_2\ e\ n < m. \quad (4.3)$$

Um exemplo do uso das funções *WebListSim*, *EP* e *SL* pode ser observado na seção 4.4.

4.3 PROCESSO DE COMPARAÇÃO

O processo básico de comparação considera 2 arquivos XML de entrada provenientes de uma extração de 2 listas ocorrida na Web. Estas entradas são comparadas no processo para a determinação do coeficiente de similaridade entre elas, passando por 2 etapas. A Figura 6 apresenta este processo. Ele recebe essas 2 listas de entrada, que passam pela primeira etapa

de *leitura e recuperação dos dados*. Esta etapa realiza a leitura dos arquivos de entrada no formato *XML* com os dados de cada lista, conforme descrito na seção 4.1. Após os dados serem devidamente carregados, eles são passados para a segunda etapa de *comparação* para gerar, por fim, o coeficiente de similaridade.

Figura 6 – Processo de comparação de listas na Web.

Fonte: Elaborada pelo autor.

O Algoritmo 1 foi desenvolvido para a etapa de Comparação. Ele recebe como entrada os arquivos chamados de *doc1* e *doc2* (2 arquivos XML de entrada), em conjunto com o objeto *FuncaoSim*, que representa uma das funções de similaridade pré-definidas: *Cosseno*, *Euclidiana* ou *Jaccard*. O corpo do algoritmo realiza a chamada para a função *calculaSimPag*, que calcula a similaridade entre os dados externos às listas considerados pela métrica de similaridade proposta (TP, EE e P), armazenando o valor na variável *EP*. Na sequência, é realizada a chamada à função *compareList*, qual calcula a similaridade entre os elementos das listas contidas em *doc1* e *doc2*, armazenando o valor na variável *SL*. Por fim, é calculado o coeficiente final de similaridade utilizando a Equação 4.3.

o Algoritmo 2 apresenta um pseudocódigo que representa a função *calculaSimPag* invocada no Algoritmo 1. Ele recebe como entrada os dados das páginas *pag1* e *pag2*, assim como o objeto *FuncaoSim* que indica a função de similaridade a ser utilizada. Para cada página pode-se acessar o atributo *title* representando o título da página, o *link* qual contém o endereço eletrônico da página, e o atributo *text* que mantém o parágrafo antecessor à lista.

Algoritmo 1: - Algoritmo WebListSim

```
Entrada: doc1, doc2, FuncaoSim, pesoEP (Op), pesoSL (Op)
Saída: WebListSim
início
    se pesoEP não informado então
        pesoEP = 0,3;
    fim
    se pesoSL não informado então
        pesoSL = 0,7;
    fim
    EP = calculaSimPag(doc1.pag, doc2.pag, FuncaoSim);
    SL = compareList(doc1.list, doc2.list, FuncaoSim);
    WebListSim = pesoEP · EP + pesoSL · SL;
fim
```

Cada atributo é comparado de forma individual, chamando a função *compare* do objeto *FuncaoSim*, que recebe o conteúdo dos atributos e os compara. O resultado de *compare* é o valor de similaridade, que passa a ser armazenado em uma variável específica. No caso do atributo *title*, o valor de similaridade retornado é guardado na variável *TP*. Já no caso dos atributos *link* e *text*, os valores são atribuídos às variáveis *EE* e *P*, respectivamente. Uma vez definidos os escores de similaridade para cada atributo da página, é então utilizada a Equação 4.2, para calcular a similaridade total entre os dados da página, sendo esta retornada através da variável *EP*.

Um detalhe a ser observado no Algoritmo 2, é que no caso de uma das páginas não conter algum dos atributos, seja *title*, *link* ou *text*, o atributo faltante não é considerado no cálculo da variável *EP*. Foi atribuido este comportamento, visto não fazer sentido realizar a comparação com um objeto não existente.

O Algoritmo 3, apresenta a função *compareList* invocada no Algoritmo 1. Este algoritmo compara os dados internos da lista de forma que os itens da menor lista são relacionados aos elementos da maior lista que

Algoritmo 2: - Função calculaSimPag

```
Entrada: pag1, pag2, FuncaoSim
Saída: EP
início
    nVar = 0;
    TP = 0;
    EE = 0;
    P = 0;
    se pag1.title.existe() e pag2.title.existe() então
        nVar++;
        TP = FuncaoSim.compare(
                        pag1.title, pag2.title);
    fim
    se pag1.link.existe() e pag2.link.existe() então
        nVar++;
        EE = FuncaoSim.compare(
                        pag1.link, pag2.link);
    fim
    se pag1.text.existe() e pag2.text.existe() então
        nVar++;
        P = FuncaoSim.compare(
                        pag1.text, pag2.text);
    fim
    EP = (TP + EE + P)/nVar;
fim
```

apresentarem melhor similaridade, desconsiderando-se os demais itens. Os melhores valores de similaridade são somados. A soma das similaridades é dividida pelo tamanho da menor lista, seguindo as definições proposta na Equação 4.3. Por fim, o algoritmo retorna o resultado para o algoritmo *WebSimList*, dando continuidade ao calculo do coeficiente de similaridade proposto.

Algoritmo 3: - Função compareList

```
Entrada: list1, list2, fSim
Saída: SL
início
    menor = list1; maior = list2;
    se list1.tamanho > list2.tamanho então
        maior = list1; menor = list2;
    fim
    List<Relacao> listRel = List.emptyList;
    para i = 1 ; i< menor.tamanho; i++ faça
        para j = 1; j < maior.tamanho; j++ faça
            ponto =
              fSim.compare(menor.pegaL(i), maior.pegaL(j));
            rel = Relacao.nova;
            rel.a = "A"+ (i + 1);
            rel.b = "B"+ (j + 1);
            rel.p = ponto;
            listRel.adiciona(rel);
        fim
    fim
    listRel.ordena(Modo.decrescente, Relacao.ponto);
    List<String> jaSelec = List.emptyList;
    SOR = 0; TL = 0; hTL = 0;
    para i=q ; i < listRel.tamanho; i++ faça
        rel = listRel.pega(i);
        se jaSelec.nContem(rel.a) e jaSelec.nContem(rel.b)
        então
            jaSelec.adiciona(rel.a);
            jaSelec.adiciona(rel.b); SOR += rel.p;
        fim
    fim
    se list1.pegaTitulo().existe() e list2.pegaTitulo().existe()
    então
        hTL++;
        TL = fSim.compare(
                list1.pegaTitulo(), list2.pegaTitulo());
    fim
    TL = fSim.compare(list1.pegaTitulo(), list2.pegaTitulo());
    SL = (TL + SOR) / (menor.tamanho + 1);
fim
```

4.4 EXEMPLO DE USO

Duas listas foram escolhidas como exemplo para demonstrar o uso da função *WebListSim*: *listTestA* e *listTestB*. Elas podem ser vistas nas Figuras 7 e 8, respectivamente. *lisTestA* é uma lista com cinco municípios catarinenses mais populosos, enquanto *listTestB* é uma lista com os 10 municípios catarinenses com maior PIB.

Figura 7 – Lista exemplo de uso - listTestA.

```
UseExample.java   *listTestA.xml   listTestB.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<document>
   <link>https://pt.wikipedia.org/wiki/Lista_de_municípios_de_Santa_Catarina_por_população</link>
   <list>
      <title>Ordem crescente dos municípios mais populosos</title>
      <lines>
        <line> <elements> <element>Joinville, 569.645</element> </elements> </line>
        <line> <elements><element>Florianópolis, 477.798</element> </elements> </line>
        <line> <elements> <element>Blumenau, 343.745</element> </elements> </line>
        <line> <elements> <element>São José, 236.029</element> </elements> </line>
        <line> <elements> <element>Chapecó, 209.553</element> </elements> </line>
     </lines>
   </list>
   <titlePage>Lista de municípios de Santa Catarina por população - Wikipédia, a enciclopédia livre</titlePage>
   <text>
      <paragraph>Esta é uma lista de municípios de Santa Catarina por população segundo a estimativa do IBGE para 2016[1].</paragraph>
   </text>
</document>
```

Fonte: Elaborada pelo autor.

Figura 8 – Lista exemplo de uso - listTestB.

```
UseExample.java   listTestA.xml   listTestB.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<document>
   <link>https://pt.wikipedia.org/wiki/Lista_de_municípios_de_Santa_Catarina_por_PIB</link>
   <list>
      <title></title>
      <lines>
         <line> <elements> <element>Joinville, 18.473.990</element> </elements> </line>
         <line> <elements> <element>Itajaí, 15.235.108</element> </elements> </line>
         <line> <elements> <element>Florianópolis, 9.806.534</element> </elements> </line>
         <line> <elements> <element>Blumenau, 8.950.141</element> </elements> </line>
         <line> <elements> <element>Jaraguá do Sul, 5.259.384</element> </elements> </line>
         <line> <elements> <element>São José, 4.784.758</element> </elements> </line>
         <line> <elements> <element>Chapecó, 4.149.295</element> </elements> </line>
         <line> <elements> <element>São Francisco do Sul, 3.984.152</element> </elements> </line>
         <line> <elements> <element>Criciúma, 3.566.411</element> </elements> </line>
         <line> <elements> <element>Brusque, 2.944.597</element> </elements> </line>
      </lines>
   </list>
   <titlePage>Lista de municípios de Santa Catarina por PIB - Wikipédia, a enciclopédia livre</titlePage>
   <text>
      <paragraph>Esta lista tem como objetivo listar os municípios do estado brasileiro de Santa Catarina por Produto Interno Bruto (PIB
   </text>
</document>
```

Fonte: Elaborada pelo autor.

Estas listas exemplo servem como entrada para a função *WebListSim* no Algoritmo 1, junto com uma função de similaridade escolhida entre as opções já apresentadas. Para este exemplo foi selecionada a função de distância *Euclidiana* normalizada para similaridade através da Equação 2.1, com pesos 0.3 para elementos da página *Web* e 0.7 para dados da lista.

Ao receber a entrada, o Algoritmo 1 calcula a similaridade dos elementos da página *Web*, armazenando o resultado na variável EP, através do Algoritmo 2. No calculo de EP é considerada a comparação do título da página de *listTestA* (*Ẅista de municípios de Santa Catarina por população - Wikipédia, a enciclopédia livre*¨) com o título da página de *listTestB* (Ẅista de municípios de Santa Catarina por PIB - Wikipédia, a enciclopédia livre¨), obtendo um coeficiente igual a 0.9166 armazenado em *TP*. Da mesma forma, são ainda comparados o endereço eletrônico, gerando um coeficiente igual a 0.9230 armazenado em *EE* e o parágrafo próximo a lista obtendo coeficiente igual a 0.8789 armazenado em *P*. Com a definição dos coeficientes *TP*, *EE* e *P*, é então calculado *EP* através da Equação 4.2, gerando um *EP* igual a 0.9062.

Dando sequência ao processo, o Algoritmo 1 inicia a comparação dos elementos da lista através da invocação do Algoritmo 3. Ele verifica que a *listTestA* é a menor, nomeando suas linhas de *A1* até *A5*, ao mesmo tempo que nomeia as linhas da *lisTestB* de *B1* até *B10*. As linhas são comparadas e são calculados os coeficientes de similaridade entre elas, construindo, assim, a matriz de similaridade da Figura 9.

Figura 9 – Exemplo de uso - Matriz de similaridade.

x	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10
A1	0,5528	0,4708	0,4708	0,4708	0,5528	0,5149	0,4708	0,5848	0,4708	0,4708
A2	0,4708	0,4708	0,5528	0,4708	0,5528	0,5149	0,4708	0,5848	0,4708	0,4708
A3	0,4708	0,4708	0,4708	0,5528	0,5528	0,5149	0,4708	0,5848	0,4708	0,4708
A4	0,5000	0,5000	0,5000	0,5000	0,5615	0,6508	0,5000	0,6279	0,5000	0,5000
A5	0,4708	0,4708	0,4708	0,4708	0,5528	0,5149	0,5528	0,5848	0,4708	0,4708

Fonte: Elaborada pelo autor.

As células da matriz de similaridade são ordenadas de forma decrescente através da pontuação de similaridade e mantidas em uma lista de relações entre as linhas das 2 listas. O primeiro elemento da lista é a relação entre as linhas *A4* e *B6*. As primeiras posições da lista ordenada são mostradas na Figura 10.

Com o mapeamento das linhas concluído, as pontuações das relações são somadas e mantidas na variável *SOR* e é calculada a similaridade entre os

Figura 10 – Exemplo de uso - Lista ordenada de coeficientes de similaridade entre linhas das 2 listas Web.

```
A4 - B6 = 0.65078485
A1 - B8 = 0.5847726
A2 - B3 = 0.5527864
A3 - B4 = 0.5527864
A5 - B5 = 0.5527864
```

Fonte: Elaborada pelo autor.

títulos das listas na variável *TL* (Algoritmo 3). Neste ponto, a variável *SOR* é somada com a variável *TL* e dividida pelo número de linhas da menor lista acrescido de 1, obtendo-se, assim, o coeficiente de similaridade *SL* igual a 0.5809.

Após a obtenção dos coeficientes *EP* e *SL*, é dado continuidade ao Algoritmo 1 para o cálculo do coeficiente da função *WebListSim*. Seguindo a definição de pesos escolhidos neste exemplo, sendo 0.3 para *EP* e 0.7 para *SL*, obtém-se o coeficiente final da *WebListSim* é igual a 0.6785. O registro de saída gerado pelo algoritmo pode ser observado na Figura 11.

Figura 11 – Exemplo de uso - Registro de saída.

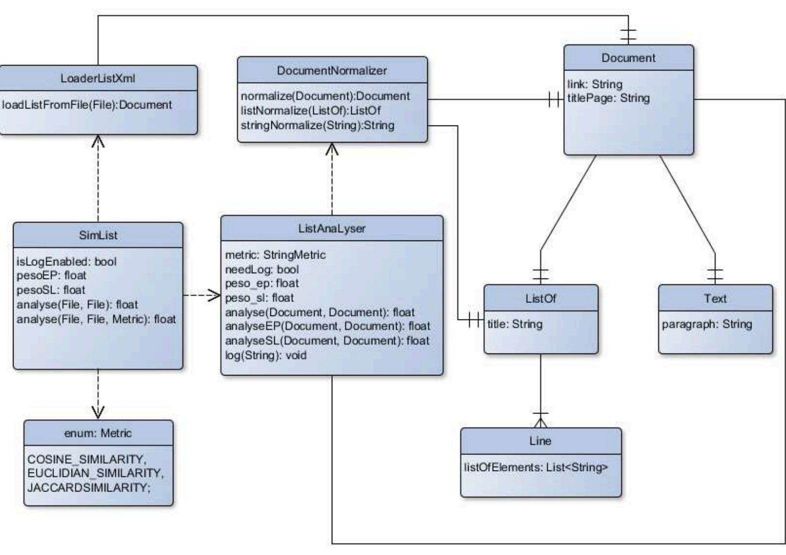
```
EuclideanDistance [Whitespace]

TP = 0.9166667
EE = 0.9230769
P = 0.87891614
EP =  (TP+EE+P)/3 = 0.90621996

x    B1     B2     B3     B4     B5     B6     B7     B8     B9     B10
A1 0,5528 0,4708 0,4708 0,4708 0,5528 0,5149 0,4708 0,5848 0,4708 0,4708
A2 0,4708 0,4708 0,5528 0,4708 0,5528 0,5149 0,4708 0,5848 0,4708 0,4708
A3 0,4708 0,4708 0,4708 0,5528 0,5528 0,5149 0,4708 0,5848 0,4708 0,4708
A4 0,5000 0,5000 0,5000 0,5000 0,5615 0,6508 0,5000 0,6279 0,5000 0,5000
A5 0,4708 0,4708 0,4708 0,4708 0,5528 0,5149 0,5528 0,5848 0,4708 0,4708

A4 - B6 = 0.65078485
A1 - B8 = 0.5847726
A2 - B3 = 0.5527864
A3 - B4 = 0.5527864
A5 - B5 = 0.5527864

SOR = 2.8939166
TL = 0.5917517
SL = (TL + SOR) / (menor + 1) = 0.5809447

WebListSim = EP * 0.3 + SL * 0.7 = 0.6785273
```

Fonte: Elaborada pelo autor.

4.5 DESENVOLVIMENTO

Para o desenvolvimento da função *WebListSim* foi planejada e construída uma biblioteca orientada a objetos chamada *SimListLibrary*. No diagrama de classes presente na Figura 12 pode ser observada a estrutura criada para a implementação da função, a qual separa as tarefas de carga dos dados na classe *LoaderListXml*. Esta classe conhece a estrutura de um *Document* e é chamada pela classe *SimList*. A classe *SimList* detém a configuração padrão da função, como os pesos aplicados a *EP*, *SL* e a métrica escolhida através do enumerador *Metric*.

Figura 12 – *SimListLibrary* - Diagrama de classes.



Fonte: Elaborada pelo autor.

A classe *ListAnalyser*, por sua vez, é encarregada da análise dos dados, recebendo os dados do tipo *Document*. *Document* é a representação da lista em tempo de execução da análise, sendo constituída por um *Text* e um *ListOf*. O *Text* representa o parágrafo e o objeto *ListOf* é a lista constituída pelo atributo *Title* e as linhas representadas por *Line*. É importante ressaltar que

os elementos da página *Web* são representados nos atributos *titlePage* e *link* da classe *Document* juntamente com o parágrafo no atributo *paragraph* da classe *Text*.

A linguagem *Java* versão *8* foi a escolhida para a implementação, sendo considerado o ambiente operacional *Windows*. A *IDE Eclipse* [1] versão *Neon 4.6.3* foi utilizada para auxiliar no desenvolvimento, em conjunto com o controle de versionamento *Git* [2] e a ferramenta de gerenciamento de bibliotecas *Maven* [3]. Na Figura 13 pode-se observar o arquivo *pom.xml* de configuração da ferramenta *Maven*. Neste arquivo são listadas todas as dependências de bibliotecas externas das quais a *SimListLibrary* necessita.

Figura 13 – SimListLibrary - Arquivo de configuração pom.xml.

```
SimListLibrary/pom.xml    SimList.java
7      <groupId>br.ufsc.inf.tcc</groupId>
8      <artifactId>SimListLibrary</artifactId>
9      <version>0.0.1-SNAPSHOT</version>
10     <packaging>jar</packaging>
11
12     <name>SimListLibrary</name>
13     <url>http://maven.apache.org</url>
14
15     <properties>
16         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17         <lombok.version>1.16.6</lombok.version>
18         <javax.version>2.2.12</javax.version>
19      </properties>
20
21     <dependencies>
22         <dependency>
23             <groupId>com.github.mpkorstanje</groupId>
24             <artifactId>simmetrics-core</artifactId>
25             <version>4.1.1</version>
26         </dependency>
27         <dependency>
28             <groupId>org.projectlombok</groupId>
29             <artifactId>lombok</artifactId>
30             <version>${lombok.version}</version>
31             <scope>provided</scope>
32         </dependency>
33         <dependency>
34             <groupId>javax.xml.bind</groupId>
35             <artifactId>jaxb-api</artifactId>
36             <version>${javax.version}</version>
37         </dependency>
38         <dependency>
39             <groupId>junit</groupId>
40             <artifactId>junit</artifactId>
41             <version>4.11</version>
42         </dependency>
43     </dependencies>
```

Fonte: Elaborada pelo autor.

[1] *IDE Eclipse* é ambiente de desenvolvimento personalizável através de plugins e um dos mais utilizados para programação em Java.

[2] *Git* é ferramenta desenvolvida para gerenciamento de código fonte e controle de versionamento, com ênfase em velocidade.

[3] *Apache Maven* é uma ferramenta que automatiza a compilação através de um arquivo de configuração que permite descrever bibliotecas externas, ordem de compilação, entre outros detalhes.

A biblioteca externa *Simmetrics-core* contém implementações consistentes para calcular a distância e similaridade de *strings*, enquanto a *Jaxb-api* possui todo o tratamento adequado à carga de arquivos *XML* para objetos da linguagem *Java*. Ambas correspondem ao processo de análise e carga, respectivamente.

A dependência *lombok* é um auxílio para programação, usada para gerar códigos básicos, como construtores e métodos de acesso a dados, através de notações em código. O *JUnit* é utilizado para a realização de testes desenvolvidos para servirem como base de averiguação da integridade das funções de comparação e experimentos. Alguns exemplos são os testes de inversão da entrada sobre a estrutura *XML* e o tratamento a dados não preenchidos, que podem ser vistos no trecho de código de teste unitário mostrado na Figura 14.

Figura 14 – SimListLibrary - Trecho de código do teste unitário (ScopeTest).

```
SimList.java   ListAnalyser.java   ScopeTest.java   ElementListTest.java
1  package br.ufsc.inf.tcc.simlist.functional;
2
3  import java.util.Arrays;
13
14 public class ScopeTest extends SimilarityTestCase {
15
16    Document doc1 = new Document("ww.euclides.com/olamundo/lista1", "title",
17          new Text("One paragraph is necessary for test! So i wrote some words and whatever."), null);
18    Document doc2 = new Document("www.euclides.com/olamundo/lista1", "title",
19          new Text("One paragraph is necessary for test! So i wrote some words and whatever."), null);
20    Document doc3 = new Document(null, null, null,
21          new ListOf("A good title's list",
22                Arrays.asList(
23                      new Line(Arrays.asList("Number: 1", "City: Florianópolis", "Citizens: 400.000")),
24                      new Line(Arrays.asList("Number: 2", "City: São Paulo", "Citizens: 11.400.000")))));
25
26    @Test
27    public void testOrder1() {
28       this.testSimilarity(this.doc1, this.doc2, Metric.EUCLIDIAN_SIMILARITY);
29    }
30
31    @Test
32    public void testOrder2() {
33       this.testSimilarity(this.doc2, this.doc1, Metric.EUCLIDIAN_SIMILARITY);
34    }
35
36    @Test
37    public void testFault1() {
38       this.testDissimilarity(this.doc3, this.doc1, Metric.EUCLIDIAN_SIMILARITY);
39    }
40
41 }
```

Fonte: Elaborada pelo autor.

A Figura 14 mostra também que a classe *ScopeTest* é uma especialização de *SimilarityTestCase* (Figura 15), a qual contém uma instância *SimList* usada para a comparação e que realiza a confirmação do resultado esperado através de chamadas estáticas a funções da classe *SimilarityAssertion*. Para

a confirmação do resultado, é determinado na classe *SimilarityAssertion* um limiar igual a $0,7$ de forma que, na chamada *asseritIsClose* é verificado se a pontuação é maior ou igual ao limiar aceito para indicar similaridade, enquanto a chamada *assertIsNotClose* verifica se a pontuação é menor, confirmando a dissimilaridade. De forma mais detalhada, pode-se observar a estrutura de teste criada no diagrama apresentado na Figura 16.

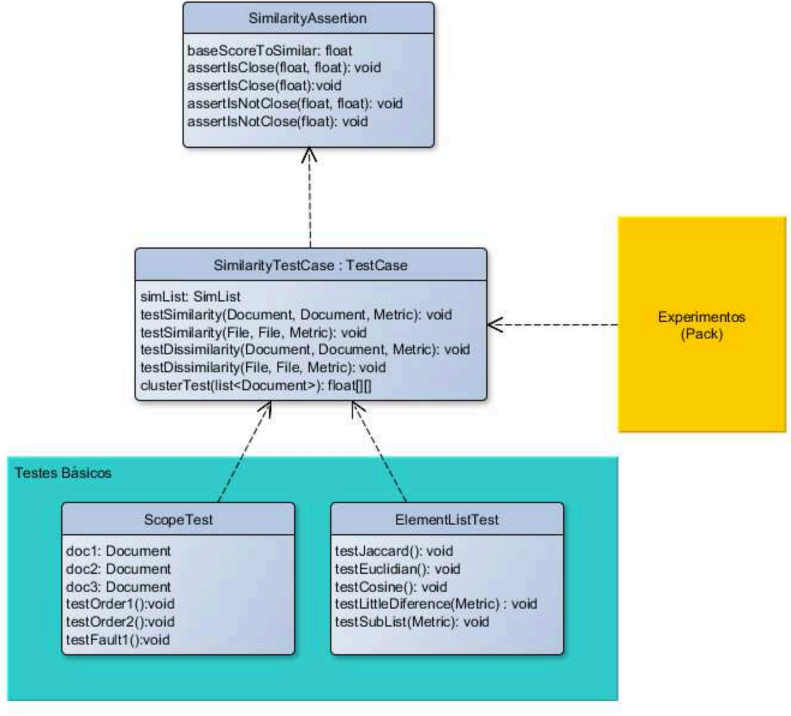
Figura 15 – SimListLibrary - Trecho de código SimilarityTestCase.

SimList.java ListAnalyser.java SimilarityTestCase.java ScopeTest.java

```
1  package br.ufsc.inf.tcc.simlist.common;
2
3⊕ import java.io.File;
10
11 public abstract class SimilarityTestCase extends TestCase {
12
13    SimList simList = new SimList(0.3f, 0.7f, true);
14
15⊖   public void testSimilarity(Document d1, Document d2, Metric m) {
16       Float score = this.simList.analyse(d1, d2, m);
17       SimilarityAssertion.assertIsClose(score);
18    }
19
20⊖   public void testDissimilarity(Document d1, Document d2, Metric m) {
21       Float score = this.simList.analyse(d1, d2, m);
22       SimilarityAssertion.assertIsNotClose(score);
23    }
24
25⊖   public void testSimilarity(File f1, File f2, Metric m) {
26       Float score = 0.0f;
27       try {
28          score = this.simList.analyse(f1, f2, m);
29          SimilarityAssertion.assertIsClose(score);
30       } catch (Exception e) {
31          e.printStackTrace();
32       }
33    }
34
35⊖   public void testDissimilarity(File f1, File f2, Metric m) {
36       Float score = 0.0f;
37       try {
38          score = this.simList.analyse(f1, f2, m);
39          SimilarityAssertion.assertIsNotClose(score);
40       } catch (Exception e) {
41          e.printStackTrace();
42       }
43    }
```

Fonte: Elaborada pelo autor.

Figura 16 – SimListLibrary - Diagrama de classes de teste.



Fonte: Elaborada pelo autor.

Durante os testes foram percebidas algumas particularidades que necessitaram de tratamento. Uma das necessidades é a abordada pela classe *DocumentNormalizer* (diagrama figura 12), que trata possíveis inconsistências no arquivo *XML*, derivado da possível falta de elementos. Esta falta pode ser exemplificada com 2 arquivos, onde o primeiro está completo e o segundo está faltando o título da página *Web*. Neste caso, não faz sentido compararmos um elemento com outro inexistente, sendo assim é assumido valor 0 de similaridade para o dado em questão. No entanto, caso o dado não estiver disponível nos dois arquivos de entrada é considerado similaridade igual a 1 para este dado, de modo que o cálculo do coeficiente de similaridade não seja prejudicado.

Um outro ponto tratado diz respeito à normalização das frases pelo uso da expressão regular *"\\p{Punct}"*, que troca todos os tipos de pontuação, acentos e demais caracteres especiais por espaço. Este tratamento é relevante visto que os dados podem ter erros de grafia, além de melhorar o desempenho de algumas métricas, como a *Cosseno*. A melhoria na função *Cosseno* é visível, por exemplo, na comparação dos endereços eletrônicos *"www.google.com"* e *"https://www.google.com.br"*. Sem este tratamento, a similaridade seria igual à 0, visto que a função reconheceria cada endereço como uma palavra inteira. Ao passar pela normalização, os endereços passariam a ser *"www google com"* e *"https www google com br"*, formando conjuntos de palavras que, aplicados à função, resultam em um coeficiente próximo à 1.

4.6 COMPARAÇÃO COM TRABALHOS CORRELATOS

A Tabela 2 compara este trabalho e os já presentes na literatura, sendo uma extensão da tabela 1. Ela apresenta novamente as características abordadas em cada trabalho, assim como as suas limitações. Em suma, este trabalho propõe uma função de similaridade para a comparação de listas HTML na Web que não apresenta as limitações dos demais trabalhos, como a relevância posicional dos itens e um tratamento para dados multivalorados, possibilitando assim a utilização da função proposta para os mais diversos fins.

Tabela 2 – Comparativo deste trabalho com os trabalhos correlatos.

Trabalho	Tipo de Estrutura	Contexto	Limitações
Pal e Michel (2016)	Lista	Web	Trata apenas listas Top-k
Wong et al. (2011)	Lista	Web	Trata apenas listas Top-k
Fagin, Kumar e Sivakumar (2003)	Lista	Web	Trata apenas listas Top-K
Silva e Mello (2015)	Tabela	Web	Trata apenas tabelas; Não trata adequadamente dados multivalorados
Dorneles et al. (2004)	Lista	XML	Trata lista com ordem; Não trata dados multivalorados e sublistas; Realiza comparação estilo função sobrejetora
Venâncio e Mello(2017)	Lista	Web	Não apresenta as limitações acima, pois não leva em conta a ordem e trata dados multivalorados.

5 EXPERIMENTOS

Este capítulo apresenta experimentos realizados para avaliar a função proposta *WebListSim*. Inicialmente, é apresentada a forma de avaliação utilizada, assim como os parâmetros utilizados na configuração da função. Na sequência, são apresentados dois experimentos: o primeiro contém 6 listas e detalha o experimento passo a passo, retratando uma situação onde as listas pertencem ao mesmo domínio. Já no segundo experimento, a função é utilizada sobre um grupo maior de 36 listas pertencentes a outros domínios. No final do capítulo são mostradas outras análises experimentais.

5.1 MÉTODO DE AVALIAÇÃO

Para avaliar a função *WebListSim* proposta, nove configurações distintas foram utilizadas em cada experimento. Estas configurações (Tabela 3) foram planejadas para analisar as 3 funções de similaridade consideradas neste trabalho e a influência dos pesos atribuídos a cada parte da função. Assumiu-se sempre um peso maior para a similaridade dos dados da lista em si (Peso SL), em relação ao peso dos dados da página Web (Peso EP), uma vez que considera-se que os dados da página Web servem apenas como um provável complemento para a compreensão da semântica da lista.

Tabela 3 – Tabela de configurações utilizadas na função *WebListSim*.

Configuração	Função Similaridade	Peso EP	Peso SL
Config_1	Cosseno	$0,2$	$0,8$
Config_2	Cosseno	$0,3$	$0,7$
Config_3	Cosseno	$0,4$	$0,6$
Config_4	Euclidiana	$0,2$	$0,8$
Config_5	Euclidiana	$0,3$	$0,7$
Config_6	Euclidiana	$0,4$	$0,6$
Config_7	Jaccard	$0,2$	$0,8$
Config_8	Jaccard	$0,3$	$0,7$
Config_9	Jaccard	$0,4$	$0,6$

Fonte: Elaborada pelo autor.

Para avaliar adequadamente a métrica proposta neste trabalho, a massa

de dados experimental foi analisada manualmente, tendo sido marcadas as listas similares. Uma vez conhecidas as listas similares, a massa de dados foi imposta à função *WebListSim*, tendo sido comparadas todas as listas entre si e extraídos os escores de similaridade para cada configuração definida.

Para cada configuração executada foi utilizado um limiar (*threshold*) de $0,7$ para considerar um par de listas como similar. Esta determinação automática de similaridade foi então comparada com as listas manualmente definidas como similares. Mediante esta análise foi possível calcular as tradicionais medidas *Precision*, *Recall* e *F-Measure*, sendo a *F-Measure* a média harmônica das duas primeiras medidas, calculada pela equação 5.1. O limiar $0,7$ foi escolhido por ser um valor não muito rigoroso nem muito aberto, considerando a natureza de dados na Web, que são bastante heterogêneos. Esse limiar foi também considerado o mais adequado em experimentos de similaridade entre tabelas na Web realizados em trabalhos anteriores do GBD/UFSC.

$$F - Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.1}$$

5.2 EXPERIMENTO 1

Neste experimento 1 foram selecionadas as 6 listas apresentadas nas Figuras 17, 18, 19, 20, 21 e 22. Estas listas pertencem a um domínio comum, ou seja, estão inseridas no contexto de *cidades violentas*.

Figura 17 – Experimento 1: Lista 1

Fonte: http://g1.globo.com/

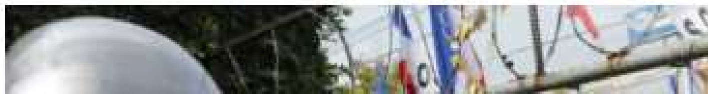
Figura 18 – Experimento 1: Lista 2

3. 2. San Pedro Sula

(Wikimedia Commons/ Gervaldoz)

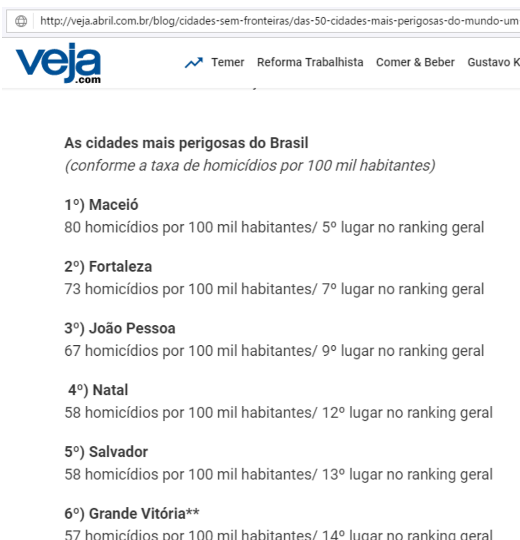
País	Honduras
Homicídios em 2015	886
Habitantes	797.065
Taxa de homicídios por 100 mil hab	111.03

4. 3. San Salvador



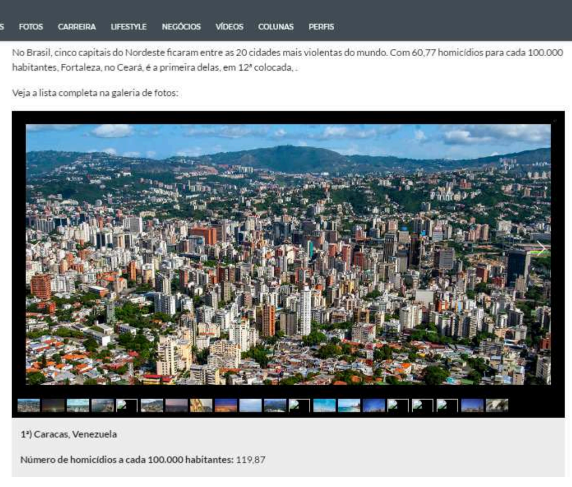
Fonte: http://exame.abril.com.br/

Figura 19 – Experimento 1: Lista 3



Fonte: http://veja.abril.com.br/

Figura 20 – Experimento 1: Lista 4



Fonte: http://www.forbes.com.br/

Figura 21 – Experimento 1: Lista 5

www.pragmatismopolitico.com.br/2015/02/25-cidades-mais-violentas-mundo.html

Pública e Justiça Penal, do México, é liderada pela hondurenha San Pedro Sula. A cidade, que tem uma taxa de homicídios de 171,20 para cada 100.000 habitantes, encabeça o ranking pelo quarto ano consecutivo.

No entanto, o que mais preocupa é o aumento das cidades brasileiras entre as mais violentas do mundo. São 11 brasileiras entre as 25 mais violentas.

Veja o ranking abaixo:

1. San Pedro Sula – Honduras
Habitantes: 769.025 | Mortes em 2014: 1.317 | Taxa de homicídios: 171,2

2. Caracas – Venezuela
Habitantes: 3.273.863 | Mortes em 2014: 3.797 | Taxa de homicídios: 115,98

3. Acapulco – México
Habitantes: 847.735 | Mortes em 2014: 883 | Taxa de homicídios: 104,16

4. João Pessoa – Brasil
Habitantes: 780.738 | Mortes em 2014: 620 | Taxa de homicídios: 79,41

5. Distrito Central – Honduras
Habitantes: 1.195.456 | Mortes em 2014: 928 | Taxa de homicídios: 77,65

Fonte: http://www.pragmatismopolitico.com.br/

Figura 22 – Experimento 1: Lista 6

do Sul.

O levantamento leva em conta a taxa de homicídios por grupo de 100 mil habitantes no ano passado. De acordo com a ONG, foram levantados dados disponibilizados pelos governos em suas páginas na internet e consideradas só cidades com mais de 300 mil. Essa foi a quarta edição do ranking. Dos 16 municípios do Brasil no ranking das cidades mais violentas do mundo, seis vão receber jogos da Copa do Mundo: Fortaleza, Natal, Salvador, Manaus, Recife e Belo Horizonte.

As brasileiras da lista mexicana

Maceió (5ª colocada) - 79,76 homicídios por 100 mil habitantes; Fortaleza (7ª) - 72,81; João Pessoa (9ª) - 66,92; Natal (12ª) - 57,62; Salvador (13ª) - 57,51; Vitória (14ª) - 57,39; São Luís (15ª) - 57,04; Belém (16ª) - 48,23; Campina Grande (25ª) - 46; Goiânia (28ª) - 44,56; Cuiabá (29ª) - 43,95; Manaus (31ª) - 42,53; Recife (39ª) - 36,82; Macapá (40ª) - 36,59; Belo Horizonte (44ª) - 34,73 e Aracaju (46ª) - 33,36.

Fonte: https://oglobo.globo.com/

Durante a analise das listas foram encontrados dois subcontextos, um relacionado as "*cidades mais violentas do mundo*"e o outro as "*cidades mais violentas do Brasil*". Esses 2 subcontextos indicam a presença de um grau de dissimilaridade evidenciado nas listas menores pertencentes ao subcontexto das "*cidades mais violentas do mundo*" (Listas 4 e 5) que, por possuirem cidades do âmbito mundial e em menor quantidade, acabam relatando poucas cidades brasileiras e assim menos elementos similares quando comparadas a listas do subcontexto das "*cidades mais violentas do Brasil*" (Listas 3 e 6). Seguindo este raciocínio, as listas foram manualmente analisadas e marcadas na tabela de similaridade (Tabela 4). Uma marcação igual a 1 indica listas similares e 0 indica listas não similares.

Tabela 4 – Experimento 1: Tabela de similaridades entre as listas.

-	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5	Lista 6
Lista 1	1	1	1	1	1	1
Lista 2	1	1	1	1	1	1
Lista 3	1	1	1	0	0	1
Lista 4	1	1	0	1	1	0
Lista 5	1	1	0	1	1	1
Lista 6	1	1	1	0	1	1

Fonte: Elaborada pelo autor.

Na sequência, as listas foram comparadas entre si utilizando a função *WebListSim* nas configurações propostas. Ao se confrontar a Tabela 4 com os escores de similaridade obtidos (considerando-se o limiar $0,7$ para indicar se 2 listas são similares ou não), obteve-se as medidas de *Precision*, *Recall* e *F-Measure* apresentadas na Tabela 5.

Observando os valores de *Precision*, pode-se verificar que, ao se comparar listas que contém um domínio comum, a função *WebListSim* reconhece bem duas listas similares. Porém, ao analisarmos os valores de *Recall*, percebe-se uma degradação de desempenho considerável para as funções *Cosseno* e *Jaccard*, uma vez que estas deixaram de considerar muitas similaridades.

Esta analise se reflete na medida *F-Measure*, mostrando que a função *Euclidiana* é a mais apropriada para este caso. Um outro detalhe a observar é

Tabela 5 – Resultados do Experimento 1.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,352941	0,521739
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,352941	0,521739
Config_3(Cosseno, 0.4, 0.6)	**1,000000**	0,352941	0,521739
Config_4(Euclidiana, 0.2, 0.8)	0,875000	0,823529	0,848485
Config_5(Euclidiana, 0.3, 0.7)	0,875000	0,823529	0,848485
Config_6(Euclidiana, 0.4, 0.6)	0,882353	**0,882353**	**0,882353**
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,352941	0,521739
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,352941	0,521739
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,352941	0,521739

Fonte: Elaborada pelo autor.

que a alteração dos pesos da função *WebListSim* só influenciou nos resultados obtidos através do uso da função euclidiana, qual obteve melhor resultado na *Configuração 6*, com os pesos 0.4 e 0.6, para *EP* e *SL* respectivamente.

Compreendendo que as listas não possuem dados duplicadas e entendendo a função euclidiana, equação 2.4, temos então que o coeficiente gerado pela função irá representar se uma dada palavra existe na outra linha. Desta maneira é então quantificado os termos exclusivos entre as listas, que quando normalizado para a medida de similaridade, acaba por mostrar a similaridade, diferente das funções de Jaccard e Cosseno, quais calculam a os termos em comum.

5.3 EXPERIMENTO 2

Neste experimento foram consideradas mais 30 listas pertencentes a outros domínios. Estas 30 listas, junto com as 6 listas do Experimento 1, foram impostas à função *WebListSim*. A Tabela 6 mostra os resultados obtidos considerando as mesmas configurações do Experimento 1.

Na Tabela 6 pode-se observar que as funções *Cosseno* e *Jaccard* continuaram com um bom índice de *Precision* e melhoraram o *Recall*, porém a melhora foi pouco expressiva quando comparado com os resultados do Experimento 1. Pode-se observar também que o melhor desempenho em termos de *F-measure* foi obtido na *Configuração 3*, a qual considerou peso $0,4$ para os

Tabela 6 – Resultados do Experimento 2.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,487179	0,655172
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,487179	0,655172
Config_3(Cosseno, 0.4**,** 0.6)	**1,000000**	0,500000	**0,666667**
Config_4(Euclidiana, 0.2, 0.8)	0,410256	**0,820513**	0,547009
Config_5(Euclidiana, 0.3, 0.7)	0,429487	**0,858974**	0,572650
Config_6(Euclidiana, 0.4, 0.6)	0,381215	**0,884615**	0,532819
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,461538	0,631579
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,461538	0,631579
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,461538	0,631579

Fonte: Elaborada pelo autor.

elementos da página, peso $0,6$ para a similaridade da lista e utilizou a função *Cosseno*.

Neste experimento a função *Euclidiana* perdeu a sua expressividade devido à diminuição do índice de *Precision*, gerando mais similaridades que o esperado, embora tenha trazido uma boa parte dos dados desejados, como mostra os valores de *Recall*. Após uma análise mais minuciosa, percebeu-se que parte desta perda é atribuída a dois fatores. O primeiro fator é a realização da comparação de números, a qual gerou uma maior similaridade em alguns casos e diminuiu a *Precision*. Já o segundo fator está relacionado à diminuição do *Recall*, devido ao não tratamento de abreviações e sinônimos.

5.4 EXPERIMENTO 3

A comparação de números foi considerada como um dos fatores de degradação do desempenho nos experimentos anteriores. A fim de melhorar este desempenho, foi elaborado um terceiro experimento que, através de algumas modificações, buscou descartar comparações numéricas, ou seja, foram comparados apenas dados do tipo *string*. Na Tabela 7, pode-se observar os resultados obtidos, onde a *Configuração 4* obteve o melhor *F-Measure*.

Um detalhe interessante que pode ser observado, ao compararmos a melhor configuração do Experimento 2 com a do Experimento 3, é a alteração dos pesos indicando a necessidade de adequação dos pesos ao tratar

Tabela 7 – Resultados do Experimento 3.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,474359	0,643478
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,474359	0,643478
Config_3(Cosseno, 0.4, 0.6)	**1,000000**	0,474359	0,643478
Config_4(Euclidiana, 0.2**,** 0.8)	0,818182	0,692308	**0,750000**
Config_5(Euclidiana, 0.3, 0.7)	0,683544	0,692308	0,687898
Config_6(Euclidiana, 0.4, 0.6)	0,549020	**0,717949**	0,622222
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,461538	0,631579
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,461538	0,631579
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,461538	0,631579

Fonte: Elaborada pelo autor.

determinados domínios. Além disso, houve uma melhora significativa do *F-Measure*, indicando a função *Euclidiana* como a mais adequada.

5.5 EXPERIMENTO 4

Outro experimento realizado sobre a mesma massa de dados do *Experimento 2*, incluindo as alterações de *Experimento 3* para a comparação apenas de dados textuais (*strings*), foi a variação das funções de similaridade visando verificar como os resultados iriam se comportar ao se utilizar funções diferentes em cada parte da equação da função *WebListSim*. Este quarto experimento considerou peso $0,3$ para os elementos da página e $0,7$ para a similaridade da lista.

De forma a reduzir a quantidade de combinações possíveis a testar, foi fixada a função *Euclidiana* como métrica de similaridade no cálculo de *SL* na função *WebListSim*. Assim sendo, variou-se apenas a métrica associada ao calculo de *EP*, obtendo-se como resultado a Tabela 8.

Tabela 8 – Resultados do Experimento 4.

Métrica EP	Métrica SL	Precision	Recall	F-Measure
Euclidiana	Euclidiana	0,683544	**0,692308**	**0,687898**
Cosseno	Euclidiana	**1,000000**	0,512821	0,677966
Jaccard	Euclidiana	**1,000000**	0,461538	0,631579

Fonte: Elaborada pelo autor.

De acordo com a Tabela 8, pode-se notar que houve uma degradação de desempenho ao misturarmos funções de similaridade em diferentes partes da função *WebListSim*, quando testado sob a perspectiva da função Euclidiana. No entanto, o resultado é um indicativo para a necessidade de se explorar mais as possibilidades de composição, motivando a realização de mais experimentos visando a investigação de uma combinação mais adequada para listas a serem comparadas em diferentes domínios. Uma tendência que parece estar ocorrendo é o melhor desempenho da função *Euclidiana*, assim como da função *Cosseno*, em múltiplos domínios.

6 CONCLUSÃO

A *Web* é uma das maiores fontes de informação da atualidade, sendo um ótimo objeto de estudo para a descoberta de conhecimento. Diante da problemática da determinação eficiente de similaridade entre duas listas provenientes da *Web*, bem como a carência de propostas efetivas na literatura, este trabalho propõe e avalia uma função denominada *WebListSim*, visando contribuir para a sua solução. Em um primeiro momento, a função obteve êxito ao reconhecer listas que participam de um mesmo macrodomínio, porém não identificou diferenças mais sutis que pudessem estar dividindo este macrodomínio em subcontextos, como mostrado no experimento 1. Já no experimento 2, realizado com listas de vários domínios e com uma massa de dados maior, houve uma degradação do desempenho causado pela consideração de números isolados na comparação e pelo não tratamento de sinônimos e abreviações. Mesmo assim, consideram-se promissores os resultados desta pesquisa, dada a grande variedade (e também volume) de dados similares presentes na *Web* e a alta complexidade para se determinar com exatidão a similaridade entre eles.

Diversos trabalhos futuros podem ser imaginados para esta pesquisa. Um exemplo seria um estudo mais detalhado de outras formas de comparação para dados de listas na *Web*, tentando responder à seguinte pergunta: será que os dados considerados na equação definida para a *WebListSim* são os dados necessários e suficientes para uma boa determinação de similaridade entre listas na *Web*?

Outras possibilidades de trabalhos futuros são a consideração e avaliação de outras funções de similaridade embutidas na função *WebListSim*, como a *Levenshtein* ou a *Jaro*, bem como explorar mais a aplicação de funções de similaridade diferentes em diferentes partes da função *WebListSim*, da forma como foi iniciado na Seção 5.5. Experimentos com volumes maiores de listas também são interessantes para verificar se é possível indicar as melhores combinações de funções.

REFERÊNCIAS

AGHAEI, S.; NEMATBAKHSH, M.; FARSANI, H. *Evolution of the Wolrd Wide Web: From Web 1.0 to Web 4.0*. [S.l.], 2012.

AGOSTINHO, F. L. A.; MELLO, R. S. *Um algoritmo para extração de listas estruturadas da Web baseado em heurísticas*. Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil, 2015.

CAFARELLA, M. J. *Extracting and Managing Structured Web Data*. University of Washington, 2009.

CHAPMAN, S. *SimMetrics, a similarity metric library. (2006)*. Disponível em: <https://sourceforge.net/projects/simmetrics/>.

CHEN, S.; MA, B.; ZHANG, K. *On the similarity metric and the distance metric*. Department of Computer Science, The University of Western Ontario, London, Ontario, Canada, 2009.

DORNELES, C. F. et al. *Measuring similarity between collection of values*. UFRGS, Porto Alegre, Brazil and UFMA, Manaus, Brazil, 2004.

FAGIN, R.; KUMAR, R.; SIVAKUMAR, D. *Comparing top k lists*. San Jose, CA 95120: IBM Almaden Research Center, 650 Harry Road, 2003. Disponível em: <http://researcher.watson.ibm.com/researcher/files/us-fagin/topk.pdf>. Acesso em: 03 jul. 2016.

LEE, T. B. *Information Management: A Proposal*. CERN, 1989. Disponível em: <https://www.w3.org/History/1989/proposal.html>. Acesso em: 04 jun. 2016.

MAKHOUL, J. et al. Performance measures for information extraction. In: *In Proceedings of DARPA Broadcast News Workshop*. [s.n.], 1999. p. 249–252. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.4637>.

PAL, K.; MICHEL, S. *Efficient Similarity Search across Top-k Lists under the Kendall's Tau Distance*. 2016. 6:1–6:12 p. Disponível em: <http://dl.acm.org/citation.cfm?doid=2949689.2949709>.

PIETERSE, V.; BLACK, P. E. *Levenshtein distance, in Dictionary of Algorithms and Data Structures [online]*. Disponível em: <https://xlinux.nist.gov/dads/HTML/Levenshtein.html>.

POWERS, D. M. W. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness And Correlation*. [S.l.], 2007. Disponível

em: <http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf>.

SILVA, F. R.; MELLO, R. S. *A frameworkfor establishing similarity between Web tables*. University Federal of Santa Catarina, Florianópolis, SC, Brazil, 2015.

W3C. *Cascading Style Sheets, level 2 - CSS2 Specification*. Disponível em: <https://www.w3.org/TR/REC-CSS2/cover.html>.

W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Disponível em: <https://www.w3.org/TR/xml/>.

W3C. *World Wide Web Consortium(W3C)*. Disponível em: <www.w3c.br/Home/WebHome>.

WONG, H. S. et al. Incremental top-k list comparison approach to robust multi-structure model fitting. *CoRR*, abs/1105.6277, 2011. Disponível em: <http://arxiv.org/abs/1105.6277>.

A CÓDIGO FONTE: WEBLISTSIM

A.1 CÓDIGO: SIMLIST.JAVA

```
1  package br.ufsc.inf.tcc.simlist;
2
3  import java.io.File;
4
5  import
       br.ufsc.inf.tcc.simlist.analyser.ListAnalyser;
6  import br.ufsc.inf.tcc.simlist.data.Document;
7  import
       br.ufsc.inf.tcc.simlist.loader.LoaderListXml;
8  import lombok.Getter;
9  import lombok.Setter;
10
11 @Getter
12 @Setter
13 public class SimList {
14
15   public enum Metric {
16     //@formatter:off
17     COSINE_SIMILARITY,
18     EUCLIDIAN_SIMILARITY,
19     JACCARD_SIMILARITY,
20     LEVENSHTEIN_SIMILARITY,
21     JARO_SIMILARITY;
22     //@formatter:on
23   }
24
25   private boolean isLogEnabled = false;
26   private Float pesoEP = 0.3f;
27   private Float pesoSL = 0.7f;
28
```

```
29    public SimList() {
30    }
31
32    public SimList(Float pesoEp, Float pesoSl,
          boolean enableLog) {
33      this.isLogEnabled = enableLog;
34      this.pesoEP = pesoEp;
35      this.pesoSL = pesoSl;
36    }
37
38    public Float analyse(File file1, File file2,
          Metric metric) throws Exception {
39
40      return this.analyse(file1, file2, metric,
            metric);
41    }
42
43    public Float analyse(File file1, File file2,
          Metric metricEp, Metric metricSL) throws
          Exception {
44
45      this.validateFiles(file1, file2);
46
47      Document doc1 =
            LoaderListXml.loadListFromFile(file1);
48      Document doc2 =
            LoaderListXml.loadListFromFile(file2);
49
50      return this.analyse(doc1, doc2, metricEp,
            metricSL);
51    }
52
53    public Float analyse(Document doc1, Document
```

```
        doc2, Metric metricEp, Metric metricSL) {
54
55      ListAnalyser analyser = new
          ListAnalyser(metricEp, metricSL,
          this.pesoEP, this.pesoSL);
56      analyser.setLog(this.isLogEnabled);
57      return analyser.analyse(doc1, doc2);
58    }
59
60    private void validateFiles(File... files)
        throws Exception {
61      for (File file : files) {
62        if (!file.exists()) {
63          throw new Exception("Arquivo " +
              file.getName() + " n\~ao
              encontrado!");
64        } else if (!file.isFile()) {
65          throw new Exception("Arquivo " +
              file.getName() + " n\~ao \'e um
              arquivo v\'alido!");
66        }
67      }
68    }
69
70  }
```

A.2 CÓDIGO: LOADERLISTXML.JAVA

```
1  package br.ufsc.inf.tcc.simlist.loader;
2
3  import java.io.File;
4
5  import javax.xml.bind.JAXBContext;
6  import javax.xml.bind.Unmarshaller;
```

```
7
8  import br.ufsc.inf.tcc.simlist.data.Document;
9
10 public class LoaderListXml {
11
12   public static Document loadListFromFile(File
         file) throws Exception {
13     JAXBContext context =
           JAXBContext.newInstance(Document.class);
14     Unmarshaller um =
           context.createUnmarshaller();
15
16     return (Document) um.unmarshal(file);
17   }
18
19 }
```

A.3 CÓDIGO: DOCUMENT.JAVA

```
1  package br.ufsc.inf.tcc.simlist.data;
2
3  import javax.xml.bind.annotation.XmlRootElement;
4
5  @XmlRootElement
6  public class Document {
7    private String link;
8    private String titlePage;
9
10   private Text text;
11   private ListOf list;
12
13   public Document() {
14
15   }
```

```
16
17      public Document(String link, String
            titlePage, Text text, ListOf list) {
18        this.link = link;
19        this.titlePage = titlePage;
20        this.text = text;
21        this.list = list;
22      }
23
24      public String getLink() {
25        return this.link;
26      }
27
28      public void setLink(String link) {
29        this.link = link;
30      }
31
32      public String getTitlePage() {
33        return this.titlePage;
34      }
35
36      public void setTitlePage(String titlePage) {
37        this.titlePage = titlePage;
38      }
39
40      public Text getText() {
41        return this.text;
42      }
43
44      public void setText(Text text) {
45        this.text = text;
46      }
47
```

```
48    public ListOf getList() {
49      return this.list;
50    }
51
52    public void setList(ListOf list) {
53      this.list = list;
54    }
55
56  }
```

A.4 CÓDIGO: LISTOF.JAVA

```
1  package br.ufsc.inf.tcc.simlist.data;
2
3  import java.util.List;
4
5  import javax.xml.bind.annotation.XmlAccessType;
6  import
       javax.xml.bind.annotation.XmlAccessorType;
7  import javax.xml.bind.annotation.XmlElement;
8  import
       javax.xml.bind.annotation.XmlElementWrapper;
9
10 @XmlAccessorType(XmlAccessType.FIELD)
11 public class ListOf {
12   private String title;
13
14   @XmlElementWrapper(name = "lines")
15   @XmlElement(name = "line")
16   private List<Line> ListOfLines;
17
18   public ListOf() {
19   }
20
```

```
21    public ListOf(String title, List<Line> lines)
          {
22      this.title = title;
23      this.ListOfLines = lines;
24    }
25
26    public String getTitle() {
27      return this.title;
28    }
29
30    public void setTitle(String title) {
31      this.title = title;
32    }
33
34    public List<Line> getLines() {
35      return this.ListOfLines;
36    }
37
38    public void setLines(List<Line> lines) {
39      this.ListOfLines = lines;
40    }
41
42  }
```

A.5 CÓDIGO: LINE.JAVA

```
1  package br.ufsc.inf.tcc.simlist.data;
2
3  import java.util.List;
4
5  import javax.xml.bind.annotation.XmlAccessType;
6  import
       javax.xml.bind.annotation.XmlAccessorType;
7  import javax.xml.bind.annotation.XmlElement;
```

```
8  import
       javax.xml.bind.annotation.XmlElementWrapper;
9
10 @XmlAccessorType(XmlAccessType.FIELD)
11 public class Line {
12
13   @XmlElementWrapper(name = "elements")
14   @XmlElement(name = "element")
15   private List<String> ListOfElements;
16
17   public Line() {
18   }
19
20   public Line(List<String> elements) {
21     this.ListOfElements = elements;
22   }
23
24   public List<String> getElement() {
25
26     return this.ListOfElements;
27   }
28
29   public void setElement(List<String> elements)
        {
30
31     this.ListOfElements = elements;
32   }
33
34   public String getAllElemets() {
35
36     String line = "";
37
38     for (String element : ListOfElements) {
```

```
39          line += (" " + element);
40        }
41
42        return line;
43      }
44
45  }
```

A.6 CÓDIGO: TEXT.JAVA

```
1   package br.ufsc.inf.tcc.simlist.data;
2
3   import javax.xml.bind.annotation.XmlType;
4
5   @XmlType
6   public class Text {
7     private String paragraph;
8
9     public Text() {
10    }
11
12    public Text(String paragraph) {
13      this.paragraph = paragraph;
14    }
15
16    public String getParagraph() {
17      return this.paragraph;
18    }
19
20    public void setParagraph(String paragraph) {
21      this.paragraph = paragraph;
22    }
23  }
```

A.7 CÓDIGO: LISTANALYSER.JAVA

```java
package br.ufsc.inf.tcc.simlist.analyser;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.simmetrics.StringMetric;
import org.simmetrics.metrics.StringMetrics;

import br.ufsc.inf.tcc.simlist.SimList.Metric;
import br.ufsc.inf.tcc.simlist.data.Document;
import br.ufsc.inf.tcc.simlist.data.Line;
import lombok.AllArgsConstructor;

public class ListAnalyser {

  private StringMetric metricEp;
  private boolean needLog = false;
  private Float peso_ep;
  private Float peso_sl;
  private StringMetric metricSL;

  public ListAnalyser(Metric metricEp, Metric
      metricSL, Float pesoEP, Float pesoSL) {
    this.peso_ep = pesoEP;
    this.peso_sl = pesoSL;

    this.metricEp = this.getInstance(metricEp);
    this.metricSL = this.getInstance(metricSL);

  }

```

```
32     private StringMetric getInstance(Metric
           metricEnum) {
33       switch (metricEnum) {
34         case COSINE_SIMILARITY:
35           return StringMetrics.cosineSimilarity();
36         case EUCLIDIAN_SIMILARITY:
37           return
               StringMetrics.euclideanDistance();
38         case JACCARD_SIMILARITY:
39           return StringMetrics.jaccard();
40         case JARO_SIMILARITY:
41           return StringMetrics.jaro();
42         case LEVENSHTEIN_SIMILARITY:
43           return StringMetrics.levenshtein();
44       }
45       return null;
46     }
47
48     public Float analyse(Document doc1, Document
           doc2) {
49       Document nDoc1 =
             DocumentNormalizer.normalize(doc1);
50       Document nDoc2 =
             DocumentNormalizer.normalize(doc2);
51
52       Float webListSim = this.analyseEP(nDoc1,
             nDoc2) * this.peso_ep +
             this.analyseSL(nDoc1, nDoc2) *
             this.peso_sl;
53       this.log("\nWebListSim = EP * " +
             this.peso_ep + " + SL * " + this.peso_sl
             + " = " + webListSim);
54       return webListSim;
```

```
55    }
56
57    private Float analyseEP(Document doc1,
          Document doc2) {
58
59      Float TP = 0f;
60      Float EE = 0f;
61      Float P = 0f;
62
63      int div = 0;
64
65      if (this.canCompare(doc1.getTitlePage(),
            doc2.getTitlePage())) {
66        TP =
              this.metricEp.compare(doc1.getTitlePage(),
              doc2.getTitlePage());
67        div++;
68      }
69
70      if (this.canCompare(doc1.getLink(),
            doc2.getLink())) {
71        EE =
              this.metricEp.compare(doc1.getLink(),
              doc2.getLink());
72        div++;
73      }
74
75      if
            (this.canCompare(doc1.getText().getParagraph(),
            doc2.getText().getParagraph())) {
76        P =
              this.metricEp.compare(doc1.getText().getParagraph()
              doc2.getText().getParagraph());
```

```
77          div++;
78        }
79
80        Float EP = (TP + EE + P) / div;
81
82        this.log("\n" + this.metricEp.toString() +
              "\n\nTP = " + TP);
83        this.log("EE = " + EE);
84        this.log("P = " + P);
85        this.log("EP =  (TP+EE+P)/3 = " + EP);
86
87        return EP;
88      }
89
90      private Float analyseSL(Document doc1,
           Document doc2) {
91
92        List<Line> linesList1 =
              doc1.getList().getLines();
93        List<Line> linesList2 =
              doc2.getList().getLines();
94
95        List<Line> minorList = new ArrayList<>();
96        List<Line> majorList = new ArrayList<>();
97        List<Line> temp = new ArrayList<>();
98
99        if (linesList1.size() < linesList2.size()) {
100         minorList.addAll(linesList1);
101         majorList.addAll(linesList2);
102         temp.addAll(linesList2);
103       } else {
104         minorList.addAll(linesList2);
105         majorList.addAll(linesList1);
```

```
106         temp.addAll(linesList1);
107       }
108
109       List<Relation> listRel = new
             ArrayList<Relation>();
110
111       if (!minorList.isEmpty() &&
             !majorList.isEmpty()) {
112         Float[][] mSim = new
               Float[minorList.size()][majorList.size()];
113
114         for (int i = 0; i < minorList.size();
               i++) {
115           Line line = minorList.get(i);
116           for (int j = 0; j < majorList.size();
                 j++) {
117             Line line2 = majorList.get(j);
118             Float score =
                   this.metricSL.compare(line.getAllElemets(),
                   line2.getAllElemets());
119             listRel.add(new Relation("A" + (i +
                   1), "B" + (j + 1), score));
120             mSim[i][j] = score;
121           }
122         }
123         this.log(mSim);
124         Collections.sort(listRel);
125       }
126
127       List<String> selItens = new
             ArrayList<String>();
128       Float SOR = 0f;
129
```

```
130        for (Relation relation : listRel) {
131          if (selItens.contains(relation.r1) ||
               selItens.contains(relation.r2)) {
132            continue;
133          }
134          selItens.add(relation.r1);
135          selItens.add(relation.r2);
136          SOR += relation.s;
137          this.log(relation.r1 + " - " +
               relation.r2 + " = " + relation.s);
138        }
139
140        this.log("\nSOR = " + SOR);
141
142        Float TL = 0f;
143
144        if
             (this.canCompare(doc1.getList().getTitle(),
             doc2.getList().getTitle())) {
145          TL =
               this.metricSL.compare(doc1.getList().getTitle(
               doc2.getList().getTitle());
146        }
147
148        this.log("TL = " + TL);
149
150        Float SL = (TL + SOR) / (minorList.size() +
             1);
151
152        this.log("SL = (TL + SOR) / (menor + 1) = "
             + SL);
153
154        return SL;
```

```
155     }
156
157     private void log(Float[][] mSim) {
158       if (this.needLog) {
159
160         System.out.println();
161         int lenght = mSim[0].length;
162         for (int i = 0; i <= mSim.length; i++) {
163           for (int j = 0; j <= lenght; j++) {
164             if (i == 0 && j == 0) {
165               System.out.printf("%15s", "x");
166             } else if (i == 0) {
167               System.out.printf("%15s", "B" + j);
168             } else if (j == 0) {
169               System.out.printf("%15s", "A" + i);
170             } else {
171               System.out.printf("%15s", "" +
                      mSim[i - 1][j - 1]);
172             }
173           }
174           System.out.println();
175         }
176         System.out.println();
177
178       }
179     }
180
181     private boolean canCompare(String s1, String
          s2) {
182       return !s1.isEmpty() && !s2.isEmpty();
183     }
184
185     public void setLog(boolean isLogEnabled) {
```

```
186       this.needLog = isLogEnabled;
187     }
188
189     private void log(String message) {
190       if (this.needLog) {
191         System.out.println(message);
192       }
193     }
194
195     @AllArgsConstructor
196     private class Relation implements
          Comparable<Relation> {
197       public String r1;
198       public String r2;
199       public Float s;
200
201       @Override
202       public int compareTo(Relation o) {
203         if (this.s > o.s) {
204           return -1;
205         } else if (this.s < o.s) {
206           return 1;
207         }
208         return 0;
209       }
210
211     }
212
213   }
```

A.8 CÓDIGO: DOCUMENTNORMALIZER.JAVA

```
1   package br.ufsc.inf.tcc.simlist.analyser;
2
```

```
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.List;
6
7  import br.ufsc.inf.tcc.simlist.data.Document;
8  import br.ufsc.inf.tcc.simlist.data.Line;
9  import br.ufsc.inf.tcc.simlist.data.ListOf;
10 import br.ufsc.inf.tcc.simlist.data.Text;
11
12 public class DocumentNormalizer {
13
14   private static List<String> stopWords =
         StopWords.getAllPortugueseStopWords();
15
16   public static Document normalize(Document
         doc) {
17
18     String link =
          stringNormalize(doc.getLink());
19     String titlePage =
          stringNormalize(doc.getTitlePage());
20     Text text = new Text(doc.getText() != null
          ?
          stringNormalize(doc.getText().getParagraph())
          : "");
21
22     ListOf list = listNormalize(doc.getList());
23
24     return new Document(link, titlePage, text,
          list);
25   }
26
27   private static ListOf listNormalize(ListOf
```

```
       list) {
28
29      if (list == null) {
30        list = new ListOf("", new
            ArrayList<Line>());
31      }
32
33      String title =
          stringNormalize(list.getTitle());
34
35      List<Line> lines = new ArrayList<>();
36      for (Line line : list.getLines()) {
37        List<String> elements = new ArrayList<>();
38        for (String element : line.getElement()) {
39          elements.add(stringNormalize(element));
40        }
41        lines.add(new Line(elements));
42      }
43
44      return new ListOf(title, lines);
45    }
46
47    private static String stringNormalize(String
        string) {
48      if (string != null) {
49        String withOutP =
            string.replaceAll("\\p{Punct}", " ");
50        String withOutPSW =
            removeStopWords(withOutP);
51        String withOutPSWN =
            removeNumbers(withOutPSW);
52        return withOutPSWN;
53      }
```

```
54        return "";
55      }
56
57      private static String removeStopWords(String
          withOutP) {
58        String r = "";
59        for (String s : withOutP.split(" ")) {
60          if (!stopWords.contains(s)) {
61            r = r + s + " ";
62          }
63        }
64        return r;
65      }
66
67      private static String removeNumbers(String
          withOutPSW) {
68        String r = "";
69        for (String s : withOutPSW.split(" ")) {
70          if (!s.matches("^[0-9]*$")) {
71            r = r + s + " ";
72          }
73        }
74        return r;
75      }
76
77      static class StopWords {
78
79        public static List<String>
            getAllPortugueseStopWords() {
80          return Arrays.asList(
81                "de", "a", "o", "que", "e", "do",
                    "da", "em", "um", "para",
82                "\'e", "com", "n\~ao", "uma", "os",
```

```
        "no", "se", "na", "por", "mais",
83      "as", "dos", "como", "mas", "foi",
        "ao", "ele", "das", "tem",
84      "\'a", "seu", "sua", "ou", "ser",
        "quando", "muito", "h\'a", "nos",
85      "j\'a", "est\'a", "eu", "tamb\'em",
        "s\'o", "pelo", "pela", "at\'e",
        "isso",
86      "ela", "entre", "era", "depois",
        "sem", "mesmo", "aos", "ter",
87      "seus", "quem", "nas", "me",
        "esse", "eles", "est\~ao",
        "voc\^e",
88      "tinha", "foram", "essa", "num",
        "nem", "suas", "meu", "\'as",
89      "minha", "t\^em", "numa", "pelos",
        "elas", "havia", "seja", "qual",
90      "ser\'", "n\'os", "tenho", "lhe",
        "deles", "essas", "esses",
        "pelas",
91      "este", "fosse", "dele", "tu",
        "te", "voc\^es", "vos", "lhes",
        "meus",
92      "minhas", "teu", "tua", "teus",
        "tuas", "nosso", "nossa",
        "nossos",
93      "nossas", "dela", "delas", "esta",
        "estes", "estas", "aquele",
94      "aquela", "aqueles", "aquelas",
        "isto", "aquilo", "estou",
        "est\'a",
95      "estamos", "est\~ao", "estive",
        "esteve", "estivemos",
```

```
            "estiveram",
96      "estava", "est\'avamos", "estavam",
            "estivera", "estiv\'eramos",
97      "esteja", "estejamos", "estejam",
            "estivesse", "estiv\'essemos",
98      "estivessem", "estiver",
            "estivermos", "estiverem",
            "hei", "h\'a",
99      "havemos", "h\~ao", "houve",
            "houvemos", "houveram",
            "houvera",
100     "houv\'eramos", "haja", "hajamos",
            "hajam", "houvesse",
            "houv\'essemos",
101     "houvessem", "houver", "houvermos",
            "houverem", "houverei",
            "houver\'a",
102     "houveremos", "houver\~ao",
            "houveria", "houver\'iamos",
            "houveriam",
103     "sou", "somos", "s\~ao", "era",
            "\'eramos", "eram", "fui",
            "foi", "fomos",
104     "foram", "fora", "f\^oramos",
            "seja", "sejamos", "sejam",
            "fosse",
105     "f\^ossemos", "fossem", "for",
            "formos", "forem", "serei",
            "ser\'a",
106     "seremos", "ser\~ao", "seria",
            "ser\'iamos", "seriam", "tenho",
            "tem",
107     "temos", "t\^em", "tinha",
```

```
            "t\'inhamos", "tinham", "tive",
            "teve",
108         "tivemos", "tiveram", "tivera",
            "tiv\'eramos", "tenha",
            "tenhamos",
109         "tenham", "tivesse",
            "tiv\'essemos", "tivessem",
            "tiver", "tivermos",
110         "tiverem", "terei", "ter\'a",
            "teremos", "ter\~ao", "teria",
111             "ter\'iamos", "teriam");
112     }
113
114   }
115
116 }
```

B ARTIGO SBC

Proposta de uma Função de Similaridade para Listas HTML Extraídas da Web

Filipe G. Venâncio[1], Ronaldo dos Santos Mello[1]

[1]Departamento de Informática e Estatística - INE
Universidade Federal de Santa Catarina (UFSC)
Campus Trindade – Caixa Postal 476 – 88.010-970 – Florianópolis – SC – Brasil

f.venancio@grad.ufsc.br, r.mello@ufsc.br

Abstract. *This meta-paper describes a propose to comparing HTML list extracted of Web with unknown context, which needed some formalization and analyzes to consider structural variability and determining if two lists are in the same context. The main objective of this work is to propose a comparison technique between HTML lists that results in a similarity score that can be used for several purposes, such as data integration and approximate searches of data with a focus on lists in the Web*

Resumo. *Este artigo apresenta uma proposta de comparação de listas HTML extraídas da Web com contextos desconhecidos, que necessitam de uma análise e padronização de sua estrutura, de forma a considerar uma possível variabilidade estrutural, visando determinar se elas dizem respeito a um mesmo assunto. O objetivo principal deste trabalho é propor uma técnica de comparação entre listas HTML que resulte em um escore de similaridade que possa ser utilizado para diversas finalidades, como integração de dados e buscas aproximadas de dados com foco em listas na Web.*

1. Introdução

Não diferente dos demais dados da *World Wide Web* (ou simplesmente *Web*), as listas são apresentadas das mais variadas formas, dificultando assim um tratamento computacional adequado para tarefas como extração, comparação e clusterização. Alguns estudos que tratam de listas provenientes da Web as denominam *WebLists*, sendo considerada como uma lista na Web não apenas as devidamente representadas pelas marcações HTML (*HiperText Markup Language*), mas também estruturas e conjuntos de dados que apresentam indícios de ser uma lista. Outro detalhe é que estes estudos abordam listas na Web limitando-as a um conjunto simples de dados ou inferindo um conjunto de características específicas dentro de um contexto.

Um exemplo é a proposta de [Fagin et al. 2003], que realiza comparações de listas na Web dentro do contexto da problemática *Top K*. Nesta comparação são mensuradas similaridades e dissimilaridades, bem como apresentadas noções de qualidade e equivalência de funções utilizadas na comparação, além de considerar somente listas na Web que apresentam uma importância posicional.

Assim sendo, a área de pesquisa relacionada ao gerenciamento de dados do tipo lista na Web carece ainda de soluções para o tratamento de listas quaisquer, incluindo a descoberta de listas similares na Web. Este trabalho busca então contribuir com a descoberta

de listas similares na Web, propondo uma função de similaridade genérica, ou seja, não influenciada por aspectos semânticos de determinadas aplicações, mas sim baseada na similaridade textual dos seus elementos e nos dados ao redor das mesmas presentes nas páginas Web, resultando em um coeficiente qual poderá ser utilizado para os mais diversos fins.

2. Função WebListSim

A função de similaridade proposta neste trabalho, denominada função *WebListSim*, considera as particularidades presentes nas listas através de uma representação genérica das listas presentes da Web.

2.1. Formato dos dados de entrada

Considerando a problemática da comparação de listas presentes na Web em termos da diversificação das suas representações, foi definido um formato genérico de dados de entrada para o processo de comparação. Este formato deriva do arquivo XML resultante do extrator desenvolvido por [Agostinho and Mello 2015], o qual foi modificado para considerar não apenas os componentes da estrutura de uma lista e o endereço eletrônico da página que contém a lista, mas também outros dados da página, que foram julgados relevantes por auxiliar na determinação do contexto da lista.

Na Figura 1, é apresentado um exemplo de arquivo de entrada. Ele embute o conjunto das informações consideradas relevantes na comparação dentro da *tag* raiz ¡*document*¿. Dos elementos da página, chamados de dados externos, foram considerados o título da página disposto na *tag* ¡*titlePage*¿, o endereço eletrônico da página em ¡*link*¿ e o parágrafo antecessor à lista em ¡*text*¿.

```xml
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <document>
3    <link>http://exame.abril.com.br/seu-dinheiro/noticias/os-carros-mais-roubados-em-2012#1</link>
4    <list>
5      <title></title>
6      <lines>
7        <line>
8          <elements>
9            <element>1º lugar: Hyundai HR</element>
10           <element>Quantidade de roubados/furtados em 2012: 804</element>
11           <element>Frota em 2012: 62.179</element>
12           <element>Frequência de roubos/furtos: 1,293%</element>
13         </elements>
14       </line>
15       <line>
16         <elements>
17           <element>2º lugar: Fiat Stilo</element>
18           <element>Quantidade de roubados/furtados em 2012: 1.126</element>
19           <element>Frota em 2012: 90.896</element>
20           <element>Frequência de roubos/furtos: 1,239%</element>
21         </elements>
22       </line>
23     </lines>
24   </list>
25   <titlePage>Os carros mais roubados em 2012 | EXAME.com</titlePage>
26   <text>
27     <paragraph>Clique nas fotos e veja quais foram os 10 carros com maior índice de roubos.</paragraph>
28   </text>
29 </document>
```

Figure 1. Exemplo do formato de dados de entrada.

Já a lista extraída é representada na *tag* ¡*list*¿, qual contém internamente a definição da *tag* ¡*title*¿ para um possível título da lista, seguido da *tag* ¡*lines*¿ constituída por sua vez por uma ou mais *tags* ¡*line*¿. Ddentro de cada *tag* ¡*line* há a *tag* ¡*elements*¿, e dentro da *tag* ¡*elements*¿ encontra-se uma ou mais *tags* ¡*element*¿, onde é disponibilizado o conteúdo de cada elemento da lista.

2.2. Função de similaridade

Para realizar a comparação de duas listas na Web foi elaborada neste trabalho a função de similaridade *WebListSim*. Esta nova função considera os dados de 2 listas a serem comparadas mais algumas informações externas a elas e presentes nas páginas Web onde elas se encontram, conforme descrito na seção anterior. A função retorna um coeficiente de similaridade entre os valores 0 e 1, sendo que, quanto mais perto do valor 1, mais similares são as 2 listas comparadas, podendo o coeficiente ser igual a 1 quando elas são idênticas ou quando uma das listas é um subconjunto dos elementos da outra lista.

A função *WebListSim* foi planejada separando os itens a serem comparados em dois conjuntos. O primeiro conjunto é formado por informações da página externas à lista, que são o título da página Web, o endereço eletrônico e o parágrafo antecessor à lista. A subfunção *EP* (*Element's Page*) determina a similaridade deste conjunto de informações externas. O segundo conjunto considera os elementos propriamente ditos da lista e o título da lista, sendo calculada a similaridade deste segundo conjunto de informações calculada pela subfunção *SL* (*Similarity List*).

No cálculo da função *WebListSim*, são atribuidos pesos associados as funções *EP* e *SL*, possibilitando definir configurações que indicam a importância de cada conjunto de dados na comparação, de acordo com o objetivo desejado. Caso os pesos não forem configurados, foi assumido por *default* o peso de 30% (*trinta por cento*), para o conjunto *EP*, de forma que os outros 70% (*setenta por cento*), sejam atribuídos a *SL*.

A Equação 1 apresenta a função *WebListSim*. Ela recebe como parâmetros os 2 documentos XML (d_1 e d_2), no formato de entrada descrito na seção anterior, contendo às informações das 2 listas a serem comparadas, e invoca as 2 funções de similaridade secundárias *EP* e *SL*, considerando as variáveis *pesoEP* e *pesoSL*, para as funções *EP* e *SL* respectivamente.

$$WebListSim(d_1, d_2) = pesoEP \cdot EP(d_1, d_2) + pesoSL \cdot SL(d_1, d_2) \quad (1)$$

No cálculo da função *EP*, os componentes título da página Web (*TP*), endereço eletrônico da página Web (*EE*) e o parágrafo imediatamente anterior à definição da lista na página Web (*P*) são comparados separadamente utilizando uma função de similaridade *Sim*. Esta função pode ser escolhida entre as 3 funções de similaridade de dados textuais disponibilizadas neste trabalho: *Cosseno*, *Euclidiana* ou *Jaccard*. O resultado da função *EP* é a média aritmética simples dos coeficientes retornados por cada uma das 3 comparações, como mostrado na Equação 2.

$$EP(d_1, d_2) = \frac{Sim(TP_1, TP_2) + Sim(EE_1, EE_2) + Sim(P_1, P_2)}{3} \quad (2)$$

Já para o cálculo da função secundária *SL*, foi definida uma adaptação da métrica *setSim* [Dorneles et al. 2004], uma vez que ela possui alguns comportamentos indeseja-

dos para os objetivos da função proposta neste trabalho. Entre as adaptações realizadas, estão as alterações para o reconhecimento de sublistas, a inserção do título da lista e a melhoria na comparação das linhas das listas, uma vez que a métrica *setSim* permite o relacionamento n para 1 - comportamento de *função sobrejetora* -, ou seja, ao comparar duas listas **A** e **B**, a métrica permite que uma ou mais linhas da lista *B* sejam relacionadas, por similaridade, com uma mesma linha da lista *A*, podendo degradar o método, uma vez que pode gerar uma similaridade maior que a esperada nas listas que possuem muitos elementos parecidos. O objetivo é refinar a similaridade permitindo um mapeamento 1 para 1 entre as linhas de duas listas, assumindo não existir linhas redundantes em uma lista.

A variação *subSetSim*, definida em [Silva and Mello 2015], corrige o primeiro comportamento de forma a permitir o conceito de subconjuntos, o qual também é útil para tratar o problema de sublistas, trocando o denominador da métrica para considerar o menor grupo de elementos ao invés do maior. No caso do comportamento de função sobrejetora, foi alterado o somatório dos valores de similaridade dos itens para que sejam admitidas apenas relações unárias e ainda adicionado, de forma ponderada, a similaridade entre os títulos das listas (TL_1 e TL_2) através da equação *SimT*. Todas essas alterações propostas podem ser observadas na Equação 3, onde a função *Sim* é a função de similaridade a ser aplicada (*Cosseno*, *Euclidiana* ou *Jaccard*).

$$SL(d_1, d_2) = \frac{SimT(TL_1, TL_2) + Max(Sim(d_1^1, [d_2^1, ..., d_2^m])) + ... + Max(Sim(d_1^n, [d_2^1, ..., d_2^{m-(n-1)}]))}{(n+1)},$$
$$sendo\ n = \ tamanho\ da\ lista\ d_1,\ m = \ tamanho\ da\ lista\ d_2\ e\ n < m. \quad (3)$$

2.3. Processo de Comparação

O processo básico de comparação considera 2 arquivos XML de entrada provenientes de uma extração de 2 listas ocorrida na Web. Estas entradas são comparadas no processo para a determinação do coeficiente de similaridade entre elas, passando por 2 etapas. A Figura 2 apresenta este processo. Ele recebe essas 2 listas de entrada, que passam pela primeira etapa de *leitura e recuperação dos dados*. Esta etapa realiza a leitura dos arquivos de entrada no formato *XML* com os dados de cada lista. Após os dados serem devidamente carregados, eles são passados para a segunda etapa de *comparação* para gerar, por fim, o coeficiente de similaridade.

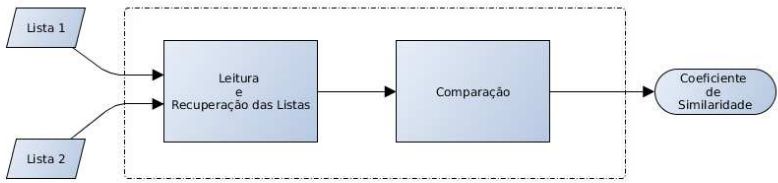


Figure 2. Processo de comparação de listas na Web.

2.4. Comparação trabalhos correlatos

A Tabela 1 compara este trabalho e os já presentes na literatura. Ela apresenta as características abordadas em cada trabalho, assim como as suas limitações. Em suma, este

trabalho propõe uma função de similaridade para a comparação de listas HTML na Web que não apresenta as limitações dos demais trabalhos, como a relevância posicional dos itens e um tratamento para dados multivalorados, possibilitando assim a utilização da função proposta para os mais diversos fins.

Table 1. Comparativo deste trabalho com os trabalhos correlatos.

Trabalho	Tipo de Estrutura	Contexto	Limitações
[Pal and Michel 2016]	Lista	Web	Trata apenas listas Top-k
[Wong et al. 2011]	Lista	Web	Trata apenas listas Top-k
[Fagin et al. 2003]	Lista	Web	Trata apenas listas Top-K
[Silva and Mello 2015]	Tabela	Web	Trata apenas tabelas; Não trata adequadamente dados multivalorados
[Dorneles et al. 2004]	Lista	XML	Trata lista com ordem; Não trata dados multi-valorados e sublistas; Realiza comparação estilo função sobrejetora
Venâncio e Mello(2017)	Lista	Web	Não apresenta as limitações acima, pois não leva em conta a ordem e trata dados multivalorados.

3. Experimentos

Inicialmente, é apresentada a forma de avaliação utilizada, assim como os parâmetros utilizados na configuração da função. Na sequência, são apresentados dois experimentos: o primeiro contém 6 listas e detalha o experimento passo a passo, retratando uma situação onde as listas pertencem ao mesmo domínio. Já no segundo experimento, a função é utilizada sobre um grupo maior de 36 listas pertencentes a outros domínios. No final do capítulo são mostradas outras análises experimentais.

3.1. Método de avaliação

Para avaliar a função *WebListSim* proposta, nove configurações distintas foram utilizadas em cada experimento. Estas configurações (Tabela 2) foram planejadas para analisar as 3 funções de similaridade consideradas neste trabalho e a influência dos pesos atribuídos a cada parte da função. Assumiu-se sempre um peso maior para a similaridade dos dados da lista em si (Peso SL), em relação ao peso dos dados da página Web (Peso EP), uma vez que considera-se que os dados da página Web servem apenas como um provável complemento para a compreensão da semântica da lista.

Para avaliar adequadamente a métrica proposta neste trabalho, a massa de dados experimental foi analisada manualmente, tendo sido marcadas as listas similares. Uma vez conhecidas as listas similares, a massa de dados foi imposta à função *WebListSim*, tendo sido comparadas todas as listas entre si e extraídos os escores de similaridade para cada configuração definida.

Para cada configuração executada foi utilizado um limiar (*threshold*) de $0,7$ para considerar um par de listas como similar. Esta determinação automática de similaridade

Table 2. Tabela de configurações utilizadas na função *WebListSim*.

Configuração	Função Similaridade	Peso EP	Peso SL
Config_1	Cosseno	$0,2$	$0,8$
Config_2	Cosseno	$0,3$	$0,7$
Config_3	Cosseno	$0,4$	$0,6$
Config_4	Euclidiana	$0,2$	$0,8$
Config_5	Euclidiana	$0,3$	$0,7$
Config_6	Euclidiana	$0,4$	$0,6$
Config_7	Jaccard	$0,2$	$0,8$
Config_8	Jaccard	$0,3$	$0,7$
Config_9	Jaccard	$0,4$	$0,6$

foi então comparada com as listas manualmente definidas como similares. Mediante esta análise foi possível calcular as tradicionais medidas *Precision*, *Recall* e *F-Measure*, sendo a *F-Measure* a média harmônica das duas primeiras medidas, calculada pela equação 4. O limiar $0,7$ foi escolhido por ser um valor não muito rigoroso nem muito aberto, considerando a natureza de dados na Web, que são bastante heterogêneos. Esse limiar foi também considerado o mais adequado em experimentos de similaridade entre tabelas na Web realizados em trabalhos anteriores do GBD/UFSC.

$$F - Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

3.2. Experimento 1

Neste experimento 1 foram selecionadas 6 listas pertencentes a um domínio comum, ou seja, lista que estão inseridas no contexto de *cidades violentas*. Durante a analise das listas foram encontrados dois subcontextos, um relacionado as "*cidades mais violentas do mundo*" e o outro as "*cidades mais violentas do Brasil*". Esses 2 subcontextos indicam a presença de um grau de dissimilaridade evidenciado nas listas menores pertencentes ao subcontexto das "*cidades mais violentas do mundo*" (Listas 4 e 5) que, por possuirem cidades do âmbito mundial e em menor quantidade, acabam relatando poucas cidades brasileiras e assim menos elementos similares quando comparadas a listas do subcontexto das "*cidades mais violentas do Brasil*" (Listas 3 e 6). Seguindo este raciocínio, as listas foram manualmente analisadas e marcadas na tabela de similaridade (Tabela 3). Uma marcação igual a 1 indica listas similares e 0 indica listas não similares.

Na sequência, as listas foram comparadas entre si utilizando a função *WebListSim* nas configurações propostas. Ao se confrontar a Tabela 3 com os escores de similaridade obtidos (considerando-se o limiar $0,7$ para indicar se 2 listas são similares ou não), obteve-se as medidas de *Precision*, *Recall* e *F-Measure* apresentadas na Tabela 4.

Observando os valores de *Precision*, pode-se verificar que, ao se comparar listas que contém um domínio comum, a função *WebListSim* reconhece bem duas listas similares. Porém, ao analisarmos os valores de *Recall*, percebe-se uma degradação de desempenho considerável para as funções *Cosseno* e *Jaccard*, uma vez que estas deixaram de considerar muitas similaridades.

Table 3. Experimento 1: Tabela de similaridades entre as listas.

-	Lista 1	Lista 2	Lista 3	Lista 4	Lista 5	Lista 6
Lista 1	1	1	1	1	1	1
Lista 2	1	1	1	1	1	1
Lista 3	1	1	1	0	0	1
Lista 4	1	1	0	1	1	0
Lista 5	1	1	0	1	1	1
Lista 6	1	1	1	0	1	1

Fonte: Elaborada pelo autor.

Table 4. Resultados do Experimento 1.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,352941	0,521739
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,352941	0,521739
Config_3(Cosseno, 0.4, 0.6)	**1,000000**	0,352941	0,521739
Config_4(Euclidiana, 0.2, 0.8)	0,875000	0,823529	0,848485
Config_5(Euclidiana, 0.3, 0.7)	0,875000	0,823529	0,848485
Config_6(Euclidiana, 0.4, 0.6)	0,882353	**0,882353**	**0,882353**
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,352941	0,521739
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,352941	0,521739
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,352941	0,521739

Fonte: Elaborada pelo autor.

Esta analise se reflete na medida *F-Measure*, mostrando que a função *Euclidiana* é a mais apropriada para este caso. Um outro detalhe a observar é que a alteração dos pesos da função *WebListSim* só influenciou nos resultados obtidos através do uso da função euclidiana, qual obteve melhor resultado na *Configuração 6*, com os pesos 0.4 e 0.6, para *EP* e *SL* respectivamente.

Compreendendo que as listas não possuem dados duplicadas e entendendo a função euclidiana de similaridade, temos então que o coeficiente gerado pela função irá representar se uma dada palavra existe na outra linha. Desta maneira é então quantificado os termos exclusivos entre as listas, que quando normalizado para a medida de similaridade, acaba por mostrar a similaridade, diferente das funções de Jaccard e Cosseno, quais calculam a os termos em comum.

3.3. Experimento 2

Neste experimento foram consideradas mais 30 listas pertencentes a outros domínios. Estas 30 listas, junto com as 6 listas do Experimento 1, foram impostas à função *WebListSim*. A Tabela 5 mostra os resultados obtidos considerando as mesmas configurações do Experimento 1.

Na Tabela 5 pode-se observar que as funções *Cosseno* e *Jaccard* continuaram com um bom índice de *Precision* e melhoraram o *Recall*, porém a melhora foi pouco expressiva quando comparado com os resultados do Experimento 1. Pode-se observar também que o melhor desempenho em termos de *F-measure* foi obtido na *Configuração 3*, a qual

Table 5. Resultados do Experimento 2.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,487179	0,655172
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,487179	0,655172
Config_3(Cosseno, 0.4, 0.6)	**1,000000**	0,500000	**0,666667**
Config_4(Euclidiana, 0.2, 0.8)	0,410256	**0,820513**	0,547009
Config_5(Euclidiana, 0.3, 0.7)	0,429487	**0,858974**	0,572650
Config_6(Euclidiana, 0.4, 0.6)	0,381215	**0,884615**	0,532819
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,461538	0,631579
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,461538	0,631579
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,461538	0,631579

considerou peso $0,4$ para os elementos da página, peso $0,6$ para a similaridade da lista e utilizou a função *Cosseno*.

Neste experimento a função *Euclidiana* perdeu a sua expressividade devido à diminuição do índice de *Precision*, gerando mais similaridades que o esperado, embora tenha trazido uma boa parte dos dados desejados, como mostra os valores de *Recall*. Após uma análise mais minuciosa, percebeu-se que parte desta perda é atribuída a dois fatores. O primeiro fator é a realização da comparação de números, a qual gerou uma maior similaridade em alguns casos e diminuiu a *Precision*. Já o segundo fator está relacionado à diminuição do *Recall*, devido ao não tratamento de abreviações e sinônimos.

3.4. Experimento 3

A comparação de números foi considerada como um dos fatores de degradação do desempenho nos experimentos anteriores. A fim de melhorar este desempenho, foi elaborado um terceiro experimento que, através de algumas modificações, buscou descartar comparações numéricas, ou seja, foram comparados apenas dados do tipo *string*. Na Tabela 6, pode-se observar os resultados obtidos, onde a *Configuração 4* obteve o melhor *F-Measure*.

Table 6. Resultados do Experimento 3.

Configuração	Precision	Recall	F-Measure
Config_1(Cosseno, 0.2, 0.8)	**1,000000**	0,474359	0,643478
Config_2(Cosseno, 0.3, 0.7)	**1,000000**	0,474359	0,643478
Config_3(Cosseno, 0.4, 0.6)	**1,000000**	0,474359	0,643478
Config_4(Euclidiana, 0.2, 0.8)	0,818182	0,692308	**0,750000**
Config_5(Euclidiana, 0.3, 0.7)	0,683544	0,692308	0,687898
Config_6(Euclidiana, 0.4, 0.6)	0,549020	**0,717949**	0,622222
Config_7(Jaccard, 0.2, 0.8)	**1,000000**	0,461538	0,631579
Config_8(Jaccard, 0.3, 0.7)	**1,000000**	0,461538	0,631579
Config_9(Jaccard, 0.4, 0.6)	**1,000000**	0,461538	0,631579

Um detalhe interessante que pode ser observado, ao compararmos a melhor configuração do Experimento 2 com a do Experimento 3, é a alteração dos pesos indicando a necessidade de adequação dos pesos ao tratar determinados domínios. Além

disso, houve uma melhora significativa do *F-Measure*, indicando a função *Euclidiana* como a mais adequada.

3.5. Experimento 4

Outro experimento realizado sobre a mesma massa de dados do *Experimento 2*, incluindo as alterações de *Experimento 3* para a comparação apenas de dados textuais (*strings*), foi a variação das funções de similaridade visando verificar como os resultados iriam se comportar ao se utilizar funções diferentes em cada parte da equação da função *WebListSim*. Este quarto experimento considerou peso $0,3$ para os elementos da página e $0,7$ para a similaridade da lista.

De forma a reduzir a quantidade de combinações possíveis a testar, foi fixada a função *Euclidiana* como métrica de similaridade no cálculo de *SL* na função *WebListSim*. Assim sendo, variou-se apenas a métrica associada ao calculo de *EP*, obtendo-se como resultado a Tabela 7.

Table 7. Resultados do Experimento 4.

Métrica EP	Métrica SL	Precision	Recall	F-Measure
Euclidiana	Euclidiana	0,683544	**0,692308**	**0,687898**
Cosseno	Euclidiana	**1,000000**	0,512821	0,677966
Jaccard	Euclidiana	**1,000000**	0,461538	0,631579

Fonte: Elaborada pelo autor.

De acordo com a Tabela 7, pode-se notar que houve uma degradação de desempenho ao misturarmos funções de similaridade em diferentes partes da função *WebListSim*, quando testado sob a perspectiva da função Euclidiana. No entanto, o resultado é um indicativo para a necessidade de se explorar mais as possibilidades de composição, motivando a realização de mais experimentos visando a investigação de uma combinação mais adequada para listas a serem comparadas em diferentes domínios. Uma tendência que parece estar ocorrendo é o melhor desempenho da função *Euclidiana*, assim como da função *Cosseno*, em múltiplos domínios.

4. Conclusão

A *Web* é uma das maiores fontes de informação da atualidade, sendo um ótimo objeto de estudo para a descoberta de conhecimento. Diante da problemática da determinação eficiente de similaridade entre duas listas provenientes da *Web*, bem como a carência de propostas efetivas na literatura, este trabalho propõe e avalia uma função denominada *WebListSim*, visando contribuir para a sua solução. Em um primeiro momento, a função obteve êxito ao reconhecer listas que participam de um mesmo macrodomínio, porém não identificou diferenças mais sutis que pudessem estar dividindo este macrodomínio em subcontextos, como mostrado no experimento 1. Já no experimento 2, realizado com listas de vários domínios e com uma massa de dados maior, houve uma degradação do desempenho causado pela consideração de números isolados na comparação e pelo não tratamento de sinônimos e abreviações. Mesmo assim, consideram-se promissores os resultados desta pesquisa, dada a grande variedade (e também volume) de dados similares presentes na *Web* e a alta complexidade para se determinar com exatidão a similaridade entre eles.

Diversos trabalhos futuros podem ser imaginados para esta pesquisa. Um exemplo seria um estudo mais detalhado de outras formas de comparação para dados de listas na *Web*, tentando responder à seguinte pergunta: será que os dados considerados na equação definida para a *WebListSim* são os dados necessários e suficientes para uma boa determinação de similaridade entre listas na *Web*?

Outras possibilidades de trabalhos futuros são a consideração e avaliação de outras funções de similaridade embutidas na função *WebListSim*, como a *Levenshtein* ou a *Jaro*, bem como explorar mais a aplicação de funções de similaridade diferentes em diferentes partes da função *WebListSim*, da forma como foi iniciado na Seção 3.4. Experimentos com volumes maiores de listas também são interessantes para verificar se é possível indicar as melhores combinações de funções.

5. References

[Knuth 1984], [Boulic and Renault 1991], [Smith and Jones 1999].

References

Agostinho, F. L. A. and Mello, R. S. (2015). *Um algoritmo para extração de listas estruturadas da Web baseado em heurísticas*. Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil.

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Dorneles, C. F., Heuser, C. A., Lima, A. E. N., Da Silva, A., and Moura, E. (2004). *Measuring similarity between collection of values*. UFRGS, Porto Alegre, Brazil and UFMA, Manaus, Brazil.

Fagin, R., Kumar, R., and Sivakumar, D. (2003). *Comparing top k lists*. San Jose, CA 95120: IBM Almaden Research Center, 650 Harry Road.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Pal, K. and Michel, S. (2016). Efficient similarity search across top-k lists under the kendall's tau distance.

Silva, F. R. and Mello, R. S. (2015). *A frameworkfor establishing similarity between Web tables*. University Federal of Santa Catarina, Florianópolis, SC, Brazil.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.

Wong, H. S., Chin, T., Yu, J., and Suter, D. (2011). Incremental top-k list comparison approach to robust multi-structure model fitting. *CoRR*, abs/1105.6277.