UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

INATAN LOPES HERTZOG

ICNDE - Um novo método para Identificação e Correção de ruído para extração de dados

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientadora: Profa. Dra. Renata Galante
Co-orientador: MsC. Edimar Manica

Porto Alegre
2016

“Os sonhos das pessoas não têm fim.“

— MARSHALL D. TEACH, ONE PIECE

AGRADECIMENTOS

Agradeço primeiramente a minha Família, minha mãe (Nazaré Hertzog) que sempre me deu suporte. Meu irmão (Inácio Hertzog) que sempre me apoiou. Principalmente ao meu falecido pai (Amaury Hertzog) que sempre foi aquele que me fez escolher esse curso e sempre foi à inspiração para eu me levantar.

Agradeço a minha namorada Desiree Elias que mudou completamente a minha vida e teve que perder muitas noites de sono com esse trabalho. Agradeço a Maria Soni Cardoso por sempre acreditar em mim e procurar qualquer forma de me ajudar.

Agradeço aos meus amigos que se mantiveram juntos um ajudando o outro. Em especial ao Mateus Riad que em todos os momentos difíceis estava disposto a me ajudar. Ao Cristiano Ruschel que também foi um grande apoio psicológico nessa nova etapa da minha vida. E ao Leonardo Carvalho que além de dar apoio fez questão de ler o meu trabalho e apontar tudo o que eu poderia melhorar.

Agradeço também a minha Orientadora Renata Galante e ao meu Coorientador Edimar Manica que sempre estiveram disponíveis a me ajudar independente do horário, além de serem atenciosos e queridos.

Agradeço aos professores que me ajudaram a chegar nessa nova etapa, eles foram essênciais para eu crescer.

RESUMO

Através da web são encontradas muitas entidades que são descritas em páginas. A fim de obter os valores de tais entidades é necessária a utilização de um extrator de dados da web. Extratores de dados são utilizados por empresas dirigidas a dados para fazer consulta direta em motores de busca. Cada extrator de dados emprega sua própria abordagem. No entanto, os métodos de extração existentes são propensos a falhas na extração de valores. Este trabalho tem como objetivo primário propor um método que identifica e corrige ruído para extração de dados e como objetivos secundários, avaliar os casos de falha de extratores de dados de abordagens diferentes, esta avalição foi utilizada como subsídio para propor o método. Para essa avaliação dos casos de falha, foi utilizada uma base de dados real de diferentes domínios de aplicação que possuem um conjunto de diferentes sites que acumulam um total de 125k páginas. O avaliador também foi desenvolvido neste trabalho. A avaliação permitiu a identificação dos tipos mais comuns de ruídos que ocorrem nos valores extraídos pelos métodos. Em resposta a isso, o método ICNDE (*Identification and Correction of Noise in Data Extraction* - Identificação e Correção do Ruído para Extração de Dados) foi proposto como uma ferramenta que não apenas identifica o ruído nos dados extraídos, mas que também o corrige. O ICNDE usa um extrator de dados para obter regras que extraem valores de atributos em páginas web e realiza um pós-processamento para eliminar prefixos, sufixos e outros tipos de valores considerados ruídos. O pós-processamento utiliza procedimentos de anotação e tokenização para identificar os ruídos presentes nos valores extraídos, gerando uma saída composta dos valores extraídos sem ruídos. A eficácia, o percentual de erros e o desempenho do pós-processamento do ICNDE também foram avaliados. Para realizar esse experimento, foram utilizados métodos como $baselines$ com duas abordagens diferentes, um baseado em XPath e outro baseado em árvore. O experimento mostrou que a etapa de pós-processamento aumentou a eficácia tanto no método baseado em XPath (ganho de F1 de 13%) quanto no método baseado em árvore (ganho de F1 de 11%), além disso, o percentual de erro diminui nos dois métodos.

Palavras-chave: Extração de dados. Páginas-entidade. Tokenização. Anotação. Ruído.

ICNDE - A new method to Identification and Correction of Noise to Data Extraction

ABSTRACT

Through the web are found many entities that are described by pages. In order to obtain the values of such entities, the use of a web data extrator is a necessity. Data extractors are used by data-driven companies to do direct search queries on search engines. Each data extractor employs its own approach. However, the existing extraction methods are prone to failures in extracting values. The main objective of this work is to propose a method that identifies and corrects noise for data extraction and as secondary objectives to evaluate the cases of data extraction failures of different approaches, this evaluation was used as a subsidy to propose the method. For this assessment of failure cases, we used a real database of different application domains that have a set of different sites that accumulate a total of 125k pages. The evaluator was also developed in this work. The evaluation allowed the identification of the most common types of noise that occur in the values extracted by the methods. In response to this, the ICNDE (Data Identification and Correction of Noise to Data Extraction) method was proposed as a tool that not only identifies the noise in the extracted data, but also the Correct ICNDE uses a data extractor to obtain rules that extract attribute values from web pages and performs post-processing to eliminate prefixes, suffixes, and other types of values considered as noise. Post-processing uses annotation and tokenization procedures to identify the noises present in the extracted values, generating a composite output of the noise-free values. The efficacy, percentage of errors and post-processing performance of the ICNDE were also evaluated. To perform this experiment we used methods from two different approaches, one based on XPath and another based on tree. The experiment showed that the postprocessing step increased the efficacy of both the XPath-based method (13% F1 gain) and the tree-based method (F1 gain of 11%), and the error percentage decreased Two methods.

Keywords: Data extraction. Entity page. Tokenization. Annotation. Noise.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AERD	Attribute Extraction based on Redundancy Detection
CSV	Comma Separated Values File
DOM	Document Object Model
HTML	HyperText Markup Language
ICNDE	Identification and Correction of Noise to Data Extracion
Weir	Web-Extraction and Integration of Redundant data
Wadar	Wrapper And Data Repair in Web Data Extraction
XML	extensible Markup Language
XPath	XML Path Language

SUMÁRIO

1 INTRODUÇÃO

A Internet é um dos meios de comunicação que possui maior volume de informações. Sua estrutura está dividida em diferentes sites, especializados em áreas de conhecimento específicas. Por exemplo, sites de listas telefônicas que têm como objetivo fornecer dados de empresas contendo nome, endereço e telefone. Esta informação está disponível aos usuários. Supondo que uma empresa faça uma busca por domínios para extrair nome, telefone e endereço de possíveis clientes para $marketing$, essa deve realizar algum método de captura. O método pode ser realizado de maneira manual ou de maneira automática, através de aplicações de extração de dados da web. O método manual em geral, é ineficiente porque exige que uma pessoa seja deslocada de uma função para realizar esta tarefa. Já o método automático, além de não necessitar de pessoas para realizar essa função tem maior desempenho.

Existe um volume amplo de sites que utilizam $templates$ que compartilham a mesma regra de formação de $tags$ para a geração de páginas dentro do site. Segundo (GIBSON; PUNERA; TOMKINS, 2005), uma fração significante de páginas da web, 40-50%, são dinamicamente geradas populando páginas baseadas em $tempalte$, ou seja, páginas com uma mesma estrutura fixa. Em outros casos, os sites possuem um padrão de formatação interna em suas páginas, o qual permite que uma ferramenta possa realizar buscas analisando a sua estruturação para a extração de dados. As ferramentas, em geral, fazem leitura do HTML das páginas, cuja redundância em suas informações facilita a identificação de dados.

Página-entidade é uma página que publica dados que representam uma entidade de um determinado domínio, este conceito foi definido por (BLANCO et al., 2008). A extração de dados retira valores de atributos de páginas-entidade permitindo que esses dados estejam disponíveis às aplicações ou aos usuários. Por exemplo, deseja-se extrair informações de uma página que descreve um livro. O extrator de dados extrai os valores do livro, como título, autor, ISBN e data de publicação. A extração de dados apresenta casos de falha, como dados extraídos com ruído. Por exemplo, a data de publicação extraída tem como valor *Pub date:10/02/2005*, *Pub date:* é o ruído no valor da data de publicação.

O principal objetivo desse trabalho é propor um método extração de dados que realiza tanto a extração de dados quanto a eliminação do ruído nos dados extraídos. Os objetivos secundários desse trabalho são à análise dos problemas entre métodos de ex-

tração de dados de duas abordagens, que posteriormente foram usadas como $baselines$. Experimentos foram realizados com e sem o uso do método proposto.

Escolhemos duas abordagens distintas de extração de dados (uma baseada em XPath e outra baseada em árvore) para analisar os casos de falha. Assim, foram definidos os tipos de erros que ocorrem com maior frequência na saída obtida pelos métodos utilizados. Alguns casos de erros apresentam ruído junto a um valor extraído, muitos destes casos podem ser prefixos, sufixos ou outros tipos de ruído que se encontram presentes no valor extraído. Os tipos de erros que apresentam ruído também são classificados pelos diferentes motivos que os causam.

Através dos tipos de erros com ruído identificados é proposto o método que identifica e corrige o ruído presente nos dados extraídos. Desta forma, é possível identificar prefixos, sufixos ou outros tipos de ruído presentes em um valor de atributo. Após a identificação, esse ruído é eliminado do valor de atributo e é salvo na base de dados. Baseado no conhecimento obtido por essas análises, foi criado o método ICNDE que extrai dados e elimina o ruído existente nos mesmos. O ICNDE utiliza um extrator de dados para a etapa de extração e realiza a eliminação de ruído através de uma etapa de pós-processamento.

Foram realizados experimentos utilizando 125k páginas com ICNDE. Os extratores utilizados tem duas abordagens diferentes, um baseado em XPath e outro baseado em árvore. O experimento comparou o uso e o não uso da etapa de pós-processamento do método ICNDE. Através desta comparação, a abordagem baseada em XPath obteve 13% de ganho em F1 e a abordagem baseada em árvore obteve 11% de ganho em F1. Os resultados são estatisticamente relevantes. O número total de erros diminuiu com o uso do ICNDE.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta conceitos e trabalhos relacionados necessários para a compreensão. O Capítulo 3 apresenta os experimentos com o objetivo de verificar os tipos de ruído que ocorrem nos dados extraídos pelos métodos de extração existentes. O Capítulo 4 apresenta o método ICNDE e suas etapas. O Capítulo 5 apresenta os experimentos realizados para avaliar o método ICNDE. O Capítulo 6 apresenta a conclusão deste trabalho.

2 LEVANTAMENTO BIBLIOGRÁFICO

Esse capítulo tem como objetivo apresentar o levantamento Bibliográfico relacionado ao desenvolvimento deste trabalho. A Seção 2.1 apresenta os conceitos necessários para a compreensão deste trabalho. A Seção 2.2 discute os trabalhos relacionados. Por fim, a Seção 2.3 apresenta uma descrição mais detalhada dos $baselines$ utilizados neste trabalho.

2.1 Definições Preliminares

Os três principais conceitos utilizados neste trabalho são: **domínio**, **entidade** e **atributo**. O termo **domínio de aplicação** (ou **domínio**) é usado neste trabalho com o mesmo significado que o termo conceito é usado por (DALVI et al., 2009): objeto do interesse do usuário da web que está buscando informação ou tentando realizar alguma tarefa. Um exemplo de domínio é jogador de futebol. Uma entidade é uma instância de um domínio. Por exemplo, *O senhor do anéis*. Um atributo é uma propriedade de uma entidade. Por exemplo, título, autor e data de publicação. Os demais conceitos são de **página-entidade**, **valores de atributos**, **regras**, **redundância**, **páginas com e sem o uso de *templates***, **DOM** e **XPath**.

Página-entidade é uma página que publica dados que representam uma entidade de um determinado domínio, este conceito foi definido por (BLANCO et al., 2008). **Regras** são expressões que extraem valores de atributos de páginas de um determinado site. **Instâncias** são os valores extraídos através das regras. **Ruído** é uma parte da instância que não compõe um valor de um atributo. Por exemplo, na instância *Pub date: 09/12/2015*, o valor *Pub date:* é ruído.

Diferentes sites que publicam páginas-entidade do mesmo domínio possuem **redundância de conteúdo**. A redundância de conteúdo pode ocorrer em dois níveis (ZHU et al., 2011): entidade e atributo. A **redundância em nível de entidade** - também denominada **redundância em nível de instância** e **redundância em nível de objeto** - ocorre quando dois ou mais sites publicam entidades em comum. Por exemplo, dois sites que publicam dados sobre o mesmo livro. A **redundância em nível de atributo** (*attribute level*) - também denominada **redundância em nível de esquema** (*schema level*) e **redundância em nível de item de dados** (*data-item level*) - ocorre quando dois ou mais sites publicam atributos em comum. Por exemplo, dois sites que publicam o título e o autor.

Páginas baseadas em $template$ são páginas que seguem um padrão fixo pré-definido através do código HTML (*HyperText Markup Language* - Linguagem de Marcação de Hipertexto) para a representação de dados. Páginas sem $template$ são páginas sem padrão algum, criadas manualmente. **DOM** (*Document Object Model* - Modelo de Objeto de Documento) é um interface de programação de aplicação (API) usada para definir a estrutura lógica de documentos, como HTML ou XML (Extensible Markup Language - Linguagem de Marcação Extensível), e como são acessados e manipulados. A árvore DOM é a representação de um página HTML em uma estrutura de árvore em que cada nodo é uma tag HTML (por exemplo, p, br,li, entre outras). A raiz da árvore é uma $tag\ HTML$ e as folhas são os nodos textuais. A linguagem **XPath** é baseada em uma representação de árvore do documento XML.

2.2 Trabalhos Relacionados

Nesta seção são apresentados os trabalhos relacionados à extração de dados. A Seção 2.2.1 apresenta a descrição dos trabalhos relacionados e a Seção 2.2.2 apresenta um quadro comparativo entre os trabalhos relacionados.

2.2.1 Descrição dos trabalhos relacionados

O método Weir (-*Extraction and Integration of Redundant Data* - Extração e Integração de Dados Redundantes na); proposto por (BRONZI et al., 2013), tem o objetivo de extrair valores de atributos publicados em páginas baseadas em $template$. O Weir gera, para cada site, regras de extração baseadas em expressões XPath através da análise da árvore DOM das páginas do site. Essas regras são filtradas comparando as regras geradas para diferentes sites do mesmo domínio. A premissa é que regras de diferentes sites que extraem o valor do mesmo atributo são similares. A similaridade é medida através de uma função de similaridade que compara apenas valores extraídos em páginas que descrevem a mesma entidade. O método explora a redundância em nível de atributo e em nível de entidade.

O método Trinity proposto por (SLEIMAN; CORCHUELO, 2014), tem o objetivo de extrair valores de atributos publicados em páginas baseadas em $template$. O Trinity transforma a estrutura textual das páginas do site em uma árvore ternária. O método em

uma visão geral, gera uma árvore de prefixos, separadores e sufixos a partir dos padrões que se repetem nas páginas-entidade do site. A partir da árvore são retiradas expressões regulares para extrair os valores dos atributos publicados nas páginas.

O Wadar (*Wrapper And Data Repair in Data Extraction* - $Wrapper$ e Reparação de dados na extração de dados), proposto por (ORTONA et al., 2015), tem como objetivo extrair os dados publicados nas páginas-entidade e utilizar um pós-processamento para eliminar ruídos. Para a extração de dados, o método Wadar é baseado em XPath extraído da Árvore DOM. Após isso, os valores extraídos pelas expressões XPath são segmentadas a fim de eliminar possíveis ruídos. Essa segmentação é resultado obtido através de expressões regularas que são aprendidas utilizando reconhecedores de entidades e cadeias de $Markov$.

O método FindAttrPos (*Find Attribute Position* - Encontre a posição do atributo), proposto por (GULHANE et al., 2010), tem o objetivo de extrair valores de atributos publicados em páginas baseadas em $template$. O FindAttrPos gera regras de extração baseadas em XPath. Após, ele seleciona as regras através de uma comparação entre os valores que a regra extrai e uma base de conhecimento. A base de conhecimento apresenta algumas entidades que pertencem às páginas.

O método AERD (*Attribute Extraction based on Redundancy Detection* - Extração de atributos baseada em detecção de redundância) proposto por (LI; ZHU; YIN, 2012), tem o objetivo de extrair valores de atributos publicados em páginas baseadas em $template$. O AERD gera regras de extração baseadas em XPath. Após, ele seleciona as regras através de uma comparação entre os valores que a regra extrai e uma base de conhecimento. A base de conhecimento são exemplos de valores de atributos que não são necessariamente da mesma entidade. O AERD precisa que o usuário anote os rótulos dos atributos para a comparação.

2.2.2 Quadro comparativo dos métodos

Os métodos AERD e FindAttPos necessitam de uma base de conhecimento necessária para a sua execução. O Trinity deixa $tags$ nos valores extraídos em sua saída. O Weir extrai valores de atributos que possuem prefixos e sufixos considerados como ruído. O Wadar precisa de reconhecedores de entidade para aplicar as cadeias de $Markov$. A Tabela 2.1 apresenta a comparação entre as principais características dos métodos.

O Trinity e o Weir formam os $baselines$ escolhidos para esse trabalho, uma vez

Tabela 2.1: Comparação entre os métodos de extração de dados

Método	Anotação do usuário	Árvore	Base de conhecimento	Eliminação de ruído	Expressões regulares	Reconhecedor de entidade	XPath
Weir							**X**
Trinity		**X**			**X**		
Wadar				**X**	**X**	**X**	**X**
FindAttrPos			**X**				**X**
AERD	**X**		**X**				**X**

que ambos não necessitam de uma base de conhecimento prévia. O Weir possui uma abordagem baseada em XPath e o Trinity uma abordagem baseada em árvore. Ambos os métodos são automáticos, independentes de padrões HTML específicos e são métodos de extração de dados de páginas-entidade baseadas em $template$.

2.3 Descrição dos *Baselines*

Está seção apresenta de maneira detalhada os métodos Trinity e Weir. A Seção 2.3.1 descreve o Weir e a Seção 2.3.2 descreve o Trinity.

2.3.1 Weir

O método Weir proposto por (BRONZI et al., 2013), possui como entrada um conjunto de páginas de diferentes sites de um mesmo domínio. Para cada site, são geradas as regras para o processo de extração que passam por uma filtragem utilizando heurísticas pré-definidas. Logo após, as páginas são correlacionadas aos seus respectivos identificadores, que são únicos dentro do seu respectivo site. Concluídos esses passos, uma filtragem é realizada com as regras de extração, com o objetivo de eliminar as regras consideradas fracas, ou as que extraem os valores de um atributo apenas para uma parte das páginas de um site ou extraem valores para diferentes atributos das páginas de um site.

As regras de extração de sites diferentes que tiveram origem em um mesmo atributo são unidas em grupos que recebem o nome de mapeamentos, enquanto que as regras sem mapeamento são descartadas. A saída desse processo é composta pelos mapeamentos gerados, ou seja, o conjunto com os dados extraídos a partir dos sites de entrada. O Weir pode ser classificado em 6 etapas:

1. Encontrar as regras de extração;
2. Filtrar as regras utilizando heurísticas;
3. Relacionar os identificadores às páginas;
4. Remover regras fracas;
5. Criar mapeamento;
6. Extrair os valores dos sites relacionados.

A associação de regras de extração, etapa 1, obtém um coleção de regras de extração que aparecem em páginas de um respectivo site. Esta coleção é gerada a partir da análise da árvore DOM das páginas e considera os nodos textuais que aparecem exatamente uma vez em ao menos 40% das páginas do site e que compartilham de um mesmo valor e de uma mesma regra. Esses nodos são classificados como nodos $templates$ e suas regras consistem no valor da regra, da raiz até o nó textual, eliminando-se o valor do índice, como por exemplo: html/body/h1/text(). Na maioria dos casos, os nodos $templates$ são os próprios rótulos de atributo.

Também são geradas regras de extração para os nodos candidatos, que são os nodos que não foram classificados como nodos de $template$. Estes possuem três tipos de regras de extração:

- Regras com índice do nodo raiz até o nodo candidato.
  Exemplo: /html[1]/body[1]/ul[1]/li[3]/text();
- Regras de índices a partir dos nodos com um atributo id até o nodo candidato.
  Exemplo: //p[@id='author']/text ();
- Regras com índices de nodos templates até nodos candidatos.
  Exemplo: //td[$contains$(text(),'Volume')]/../td[2]/text(), supondo que
  /td[$contains$(text(),'Volume')] é um nodo template.

Como forma de estabelecer um limite à geração de regras a partir de nodos $template$, a implementação do método Weir deve estipular uma distância máxima em número de nodos entre o nodo $template$ e um nodo candidato.

Na etapa 2 é realizada a filtragem de regras utilizando heurísticas com o objetivo de excluir as regras que extraem valores irrelevantes. São descartadas as regras que não extraem valores em pelo menos 20% das páginas do site. Caso mais de uma regra extraia os mesmos valores em uma mesma página, a regra de extração que possui maior preferência é escolhida e as demais regras são descartadas. Para o caso das regras apresentarem tipos diferentes, terá maior preferência a regra que começar em um nodo $template$, en-

quanto que aquele que iniciar na raiz terá menor preferência. Nos casos em que as regras são do mesmo tipo, a preferência é pela regra que apresenta a menor distância.

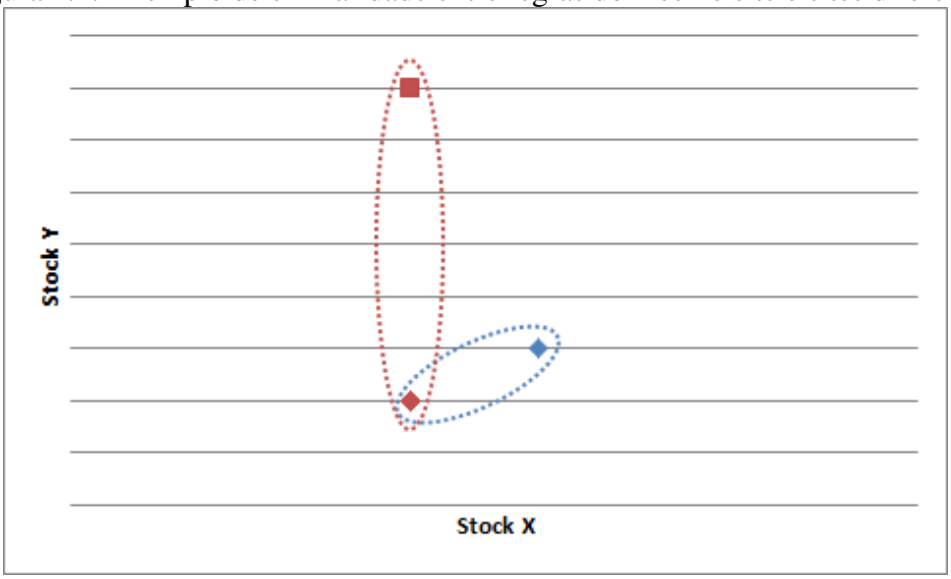
A etapa 3 consiste na localização e associação dos identificadores de cada valor da página. Dado que as próximas etapas exploram redundância em nível de instância, faz-se necessário conhecer qual a entidade descrita na página. Esta etapa recebe como entrada, além das regras já processadas nas etapas anteriores, um conjunto de identificadores com origem em uma parcela dos sites de extração. Por exemplo, no caso de sites de livros, um conjunto de ISBNs. Para estes sites, são encontrados os conjuntos de regras com o maior número de intersecções entre os valores extraídos e os identificadores. O conjunto com maior volume de intersecções é selecionado e então é feita a associação dos valores extraídos por essas regras a seus respectivos identificadores. Esses valores encontrados são utilizados para gerar um novo conjunto de identificadores e o processo é reiniciado para encontrar identificadores de outros sites.

Na etapa 4 é realizada a remoção de regras de extração fracas, através do processamento dos pares de regras de todos sites que não possuem valores com intersecção entre os valores já ordenados de maneira não decrescente em virtude de suas distâncias. A distância é expressa a partir de uma função que explora à redundância em nível de instância, o que exige que os sites compartilhem um número mínimo de valores de atributos. Os pares de regras que apresentam valores relevantes são processados primeiro. Caso uma regra apresente uma intersecção com outra considerada relevante, o par da primeira regra é marcada como fraca e descartada. Ao final, restarão apenas os pares de regras marcados como relevantes.

Na etapa 5 é criado o mapeamento que identifica as regras de diferentes sites que extraem valores de um mesmo atributo. Cada regra forma um mapa único, no qual todos os pares de regras são processados e ordenados pela distância entre seus valores em ordem não decrescente. Caso uma regra de um par pertencente a um determinado site seja marcado como completa, então o mapeamento das regras dos pares é marcado como completo. Caso contrário, os mapeamentos que contém cada um dessas regras são unidas. Essa estratégia assume que os pares de regras de site diferentes que extraem valores de um mesmo atributo estão próximos, caso contrário extraem valores de atributos diferentes. No exemplo apresentado pela Figura 2.1, a cor representa um site e a forma geométrica representa regras de atributos que extraem os mesmos valores. Nesse exemplo, pode-se notar que é possível visualizar que valores extraídos de mesmos atributos de sites diferentes (círculo azul) são mais próximos do que valores de atributos diferentes de sites iguais

(círculo vermelho). As regras de sites diferentes que se correspondem são mantidas, caso contrário, são descartadas.

Figura 2.1: Exemplo de similaridade entre regras do mesmo site e sites diferentes.



Na etapa 6, a extração de dados utiliza os mapeamentos realizados na etapa anterior para a extração dos valores dos sites de entrada. O Weir organiza os valores extraídos junto com suas respectivas regras na saída.

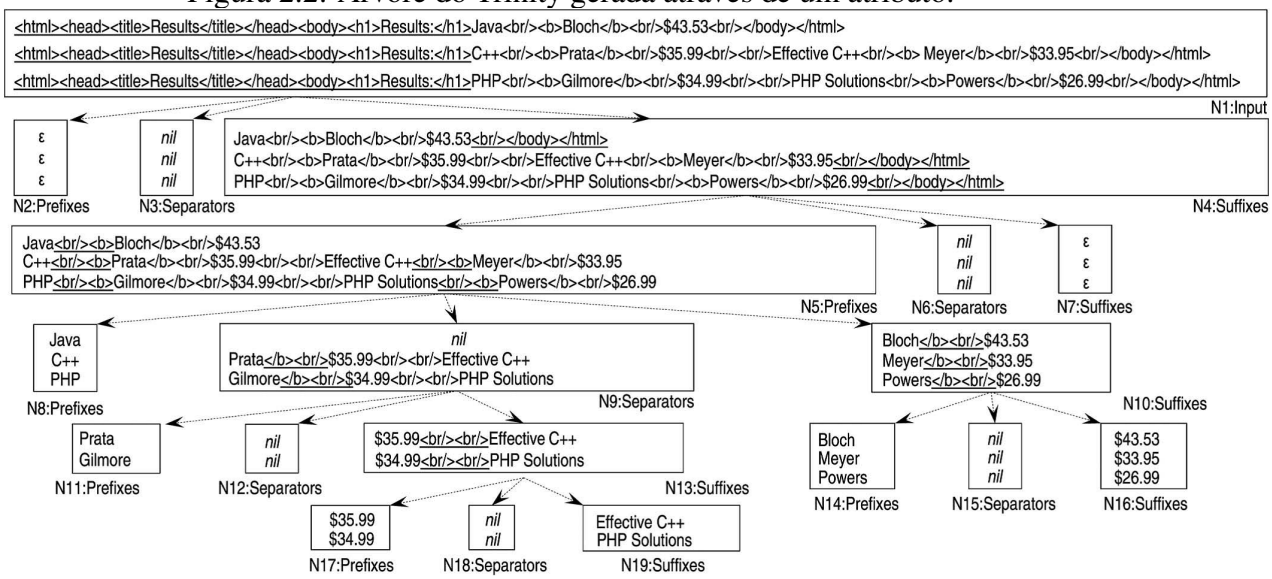
2.3.2 Trinity

O método Trinity proposto por (SLEIMAN; CORCHUELO, 2014),utiliza uma árvore ternária cujo os nodos da árvores são definidos como uma tupla composta por (t, a, p, e, s), na qual t é uma coleção de texto; a é um tipo de texto que compartilha o padrão de t; p é o nodo de prefixo; e é o nodo separador; e s é nodo sufixo.

O Trinity recebe como entrada um conjunto de páginas de um site. Para cada página de um site existem regras de diferentes tamanhos e que correspondem ao caminho textual completo das $tags$ de um determinado atributo dentro de uma página . A captura das regras necessita de delimitadores, para um tamanho definido por um valor mínimo e um valor máximo determinado pelo usuário, com o objetivo de evitar um grande volume de registros desnecessários, ou ainda, de evitar o descarte de regras procuradas. Esses delimitadores são definidos pelo usuário e funcionam como uma expressão regular para a captura de regras textuais.

A criação da árvore ternária toma como raiz a regra retirada da página e procura um padrão na estrutura do caminho textual das $tags$ para todas as regras retiradas. Logo

Figura 2.2: Árvore do Trinity gerada através de um atributo.



Fonte: (SLEIMAN; CORCHUELO, 2014)

após, é verificado se é possível expandir o nodo raiz, procurando por possíveis subsequências e dividindo-as em nodos filhos. Após a definição das subsequências, os nodos são classificados como prefixos, separadores e sufixos. Caso um dos nodos não tenha subsequências das regras presentes, então este nodo não é inserido como filho. A construção da árvore é executada recursivamente até a árvore ternária ser concluída.

A geração bem sucedida de uma árvore de busca faz com que o Trinity aprenda o $template$ da página descrita, armazenando-a e utilizando-a para a construção de uma expressão regular. Para gerar estas expressões, verifica-se o atributo é opcional caso o valor possa ser nulo ou multivalorado caso o atributo possua mais de um valor. Concluída a descoberta dos tipos, o $template$ é gerado como uma expressão, na qual os prefixos e sufixos esperados correspondem aos nodos folhas e os separadores aos nodos vazios caso não existam atributos multivalorados. Após a expressão regular ser criada, a extração é realizada e salva.

3 TRINITY E WEIR: AVALIAÇÃO EXPERIMENTAL DOS CASOS DE FALHA

Este capítulo apresenta os resultados da avaliação experimental dos casos de falha dos métodos Weir e Trinity. O objetivo dessas análises foi levantar subsídios para a proposição de um método que identifica e corrige os casos que apresentam ruído para extração de dados, que está detalhado no Capítulo 4. A Seção 3.1 introduz a configuração dos experimentos. A Seção 3.2 descreve os experimentos e apresenta os resultados obtidos. A Seção 3.3 apresenta as considerações finais do experimento de avaliação.

3.1 Configuração dos Experimentos

Esta seção apresenta a base de dados na Seção 3.1.1, as métricas utilizadas para avaliar o experimento na Seção 3.1.2 e a metódologia na Seção 3.1.3

3.1.1 Base de dados

Os dados de entrada para a análise foram adquiridos a através do site SWDE Codeplex[1], que tem como objetivo fornecer dados para pesquisas relacionadas à extração de dados de páginas web. O gabarito foi retirado de (ORTONA et al., 2015)[2] o qual apresenta uma solução esperada de um usuário que extrai manualmente as informações da base de dados.

O site SWDE organiza a base de dados adquirida em oito domínios que representam áreas de interesse específicas, que são exibidos na Tabela 3.1. Cada domínio é composto por coleções de 10 sites reais contendo no máximo 2000 páginas em codificação UTF-8, salvas como um arquivo de extensão htm e nomeadas com um identificador único em relação ao seu site.

3.1.2 Métricas

As métricas utilizadas para avaliar esse experimento são:

- Precisão: A razão entre o total de valores extraídos corretamente e o total de valores

[1] https://swde.codeplex.com/
[2] http://diadem.cs.ox.ac.uk/wadar.

Tabela 3.1: Organização da base de dados SWDE

Domínio	Nº Sites	Nº Páginas	Nº Atributos	Atributos
Carros	10	17,923	4	modelo, preço, motor e economia de gasolina
Livros	10	20,000	5	título, autor, ISBN 13, editora e data de publicação
Câmeras	10	5,258	3	modelo, preço e fabricante
Empregos	10	20,000	4	título, companhia, localização e data de postagem
Filmes	10	20,000	4	título, diretor, gênero e classificação
Jogadores de NBA	10	4,405	4	nome, time, tamanho e altura
Restaurantes	10	20,000	4	nome, endereço, telefone e especialidade
Universidades	10	16,705	4	nome, telefone, site e tipo

extraídos. A precisão é calculada como segue:

$$P = \frac{verdadeiros\,positivos}{falsos\,positivos + verdadeiros\,positivos} \tag{3.1}$$

- Revocação: A razão entre o total de valores extraídos corretamente e o total de valores corretos. A revocação é calculada como segue:

$$R = \frac{verdadeiros\,positivos}{falsos\,negativos + verdadeiros\,positivos} \tag{3.2}$$

- F1: medida que combina a precisão e a revocação. Corresponde à média harmônica entre os dois coeficientes, calculado como:

$$F1 = 2 * \frac{P * R}{P + R} \tag{3.3}$$

3.1.3 Metodologia

A implementação de extração de dados é dividida em quatro pacotes:

1. Configuração e definição dos caminhos para diretórios da base de dados.
2. Definição de métricas da extração de dados.

3. Estruturação do $dataset$.
4. Implementação do método de extração de dados (Trinity e Weir).

As etapas um a três são pacotes nativos para funcionamento do projeto de extrator de dados retirado de (MANICA; DORNELES; GALANTE, submetido para publicação 2017). A etapa quatro trata do método de extração de dados implementado para o mesmo e é o foco de discussão deste capítulo. Nesta etapa, foram executados três algoritmos. O primeiro é o método de extração de dados (Trinity ou Weir), o segundo é o mapeamento da saída deste e o terceiro é a tradução do conteúdo minerado para regras de atributos, separadas em um arquivo csv (*Comma Separated Values File* - Arquivo de Valores Separados por Ponto e Vírgula) que apresenta uma chave relacionando o identificador da página e o valor extraído como registro.

Todas as etapas foram executadas em um Notebook Samsung com Windows 10, processador intel i7 de 4ª geração com 4 núcleos e 8Gb de memória RAM.

Na quarta etapa do projeto há um avaliador nativo que verifica se os valores extraídos estão corretos. Os cálculos são realizados para a medição da porcentagem de acertos dos métodos, salvos em um arquivo csv dentro do diretório do respectivo site, no qual também há um arquivo log de erros gerados pelo método de extração. O avaliador utilizado apresenta alguns problemas que inviabilizaram seu uso nesse trabalho. Em especial dois casos: não considerar atributos multivalorados e permitir ruído no valor de atributo.

Dentro desses casos, o gabarito apresenta multivalores para alguns atributos. O gabarito é estruturado em diretórios para cada domínio, pastas que apresentam para cada site um arquivo csv em que são armazenados os registros das páginas deste site, identificadas por seu identificador e valores de atributo.

Visto que o avaliador apresentado no projeto fornecido não era preciso para esse caso, houve a necessidade de desenvolver um novo avaliador. O novo avaliador tem como objetivo considerar atributos multivalorados e considera valores com ruído como valores incorretos. O novo avaliador classifica os tipos de erros que ocorrem para cada caso na extração de dados.

O avaliador desenvolvido foi escrito na linguagem $Python$, por ser de fácil manipulação de cadeias de caracteres, estruturas de dados e expressões regulares. O $script$ realiza iterações para todos os sites. Para cada site, é realizada uma leitura no gabarito e nas regras pertencentes ao respectivo site. Os valores das respostas são formatados como um dicionário de uma coleção de valores, cujas chaves representam o nome dos atributos e o índice da coleção é o identificador da página. As regras não apresentam o nome

Figura 3.1: Fluxograma das comparações dentro das iterações.

do atributo a que pertencem, sendo lidas como uma matriz em que a primeira dimensão representa o número de regras e a segunda representa a quantidade de páginas.

A avaliação realizada pelo autor consiste em uma série de comparações, que itera sobre as chaves, páginas e índice de regras. A primeira comparação verifica o quão próximo do valor desejável se encontra o valor extraído, processo que busca se o valor indicado no gabarito está contido em cada regra, em caso afirmativo, é verificada a igualdade entre os valores. Caso a comparação seja verdadeira o acerto é contabilizado para a página. Do contrário, é computado um erro de $contains$ (acerto parcial). A Figura 3.1 apresenta o fluxograma das comparações do avaliador. Após, são realizados os cálculos dos coeficientes de precisão, revocação e F1.

O cálculo das métricas é computado por página e por atributo, o que resulta para o site em uma média aritmética por atributo. Esses valores são salvos em um arquivo csv em que as linhas correspondem aos atributos, enquanto que as colunas são o conjunto do $dataset$, site, número de valores totais, precisão, revocação e F1. O arquivo é salvo dentro de um diretório de respostas, contido na pasta referente ao domínio extraído.

Nesse diretório também é salvo um arquivo de texto como log de erros, contabilizando os erros totais e de $contains$ (por atributo e total). Seu conteúdo apresenta também o identificador da página, o nome do atributo e o valor esperado em casos de ausência de informação. Já em casos de $contains$, este conteúdo é apresentado junto com o valor extraído para a comparação. O último arquivo salvo é um csv que lista os valores de atributo, identificador da página, valor correto e extraído, para melhor visualização dos erros de $contains$. No diretório raiz são salvos três arquivos que listam os erros de $contains$,

de falta e totais; separados por sites e métodos de extração de dados.

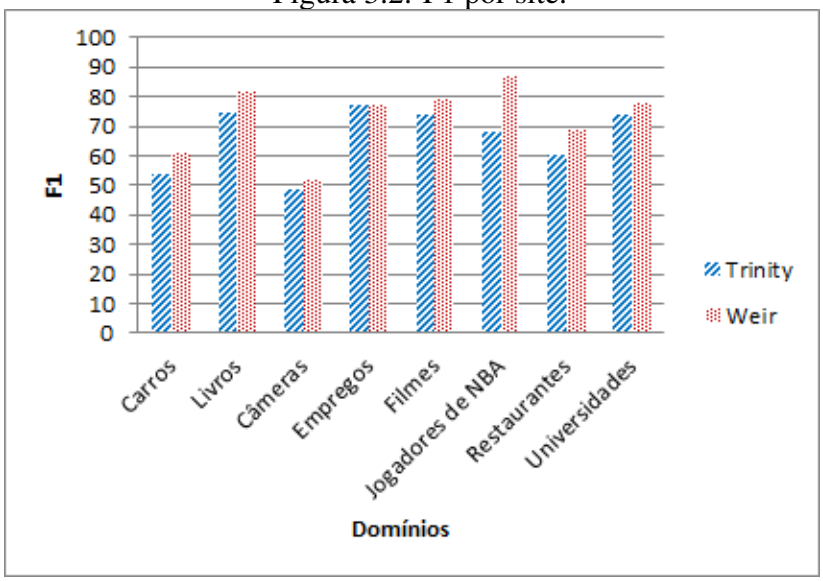
3.2 Descrição dos Resultados do Experimento

Esta seção apresenta a descrição dos resultados do experimento. A Seção 3.2.1 descreve a análise da eficácia dos métodos, a Seção 3.2.2 introduz os tipos de erros e a Seção 3.2.3 apresenta os tipos de erros de $contains$.

3.2.1 Eficácia dos métodos Trinity e Weir

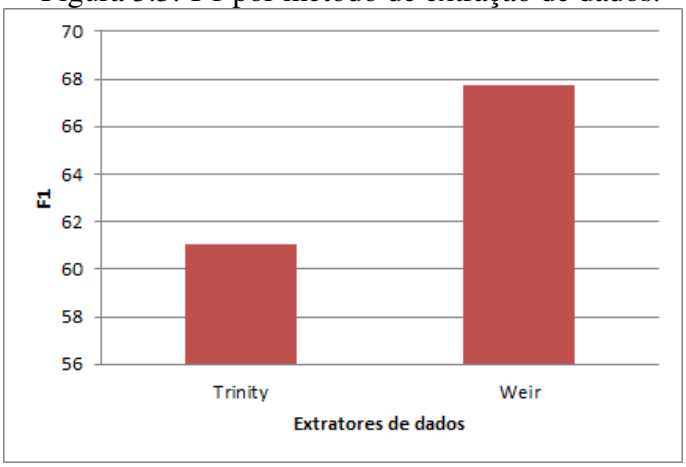
Essa seção apresenta a eficácia dos métodos Trinity e Weir. Após a execução do $script$ de avaliação desenvolvido, todos os resultados das métricas foram armazenados em planilhas para calcular a média de F1 por site e por método de extração de dados. Desta maneira, fica visível quais são os sites mais problemáticos e quais dos métodos de extração são mais eficazes.

Figura 3.2: F1 por site.



Na Figura 3.2, os domínios que apresentam menor F1 são *Câmeras*, $Carros$ e $Restaurantes$, uma vez que possuem atributos que extraem valores como endereços e modelos que possuem muitas informações em apenas um dado. Apenas em $Empregos$ o método Trinity obteve a mesma eficácia do que no método Weir. No restante dos domínios de aplicação, o método Weir apresentou maiores resultados, devido a sua abordagem, em *Jogadores de NBA* o Weir apresentou o maior F1.

Figura 3.3: F1 por método de extração de dados.



O método Weir é mais eficaz comparado ao Trinity observando a Figura 3.3, uma vez que o Weir possuí uma abordagem XPath que trabalha com um nodo textual para um valor de atributo extraído. O Trinity pela sua abordagem em árvore apresenta mais de um nodo textual para valores de atributos, assim, obtém um número menor de valores corretos com o gabarito utilizado.

3.2.2 Tipos de erros Trinity e Weir

Esta seção apresenta os tipos de erros dos métodos Trinity e Weir. Os erros foram contabilizados por site junto ao método escolhido. Nas Figuras 3.4, 3.5 e 3.6 são apresentados o número de ocorrência destes erros. Os erros foram divididos em:

- Totais: somatório de erros
- *Contains*: valor parcialmente corretos, contêm ruído.
- Falta: valor incorreto

Pode-se se notar nas Figuras 3.4 e 3.5, que na maioria dos casos os erros de $contains$, mesmo obtendo um índice baixo com relação aos de falta, ainda apresentam uma parcela relevante dentro do geral de casos analisados.

Analisando a Figura 3.6, o método Weir apresentou menor quantidade de erros e a proporção de erros de $contains$ é muito menor que a verificada no método Trinity.

Os casos que são considerados parcialmente corretos apresentam maior possibilidade de validação. Na experimentação, a análise dos erros de $contains$ dimensiona a proporção com base em suas aparições em cada site, o qual é dividido por seu total de valores. O resultado desta operação é apresentado na Figura 3.7.

Na Figura 3.7, a proporção de *contains* afeta no máximo 25% da extração de dados. Alguns domínios como *Empregos* e *Jogadores de NBA* apresentam taxas de *contains* muito abaixo das dos demais. Porém, não foram nulas em nenhum dos casos.

3.2.3 Tipos de erros de *contains*

Esta seção introduz os tipos de erros de *contains*. Após a análise dos erros apresentados na seção anterior, pesquisaram-se os motivos que geram os acertos parciais. A partir da análise dos *logs* de cada um dos sites, foi possível descobrir alguns dos casos de falha mais comuns deste tipo, os quais foram classificados da seguinte forma:

Figura 3.4: Erros no método Trinity.

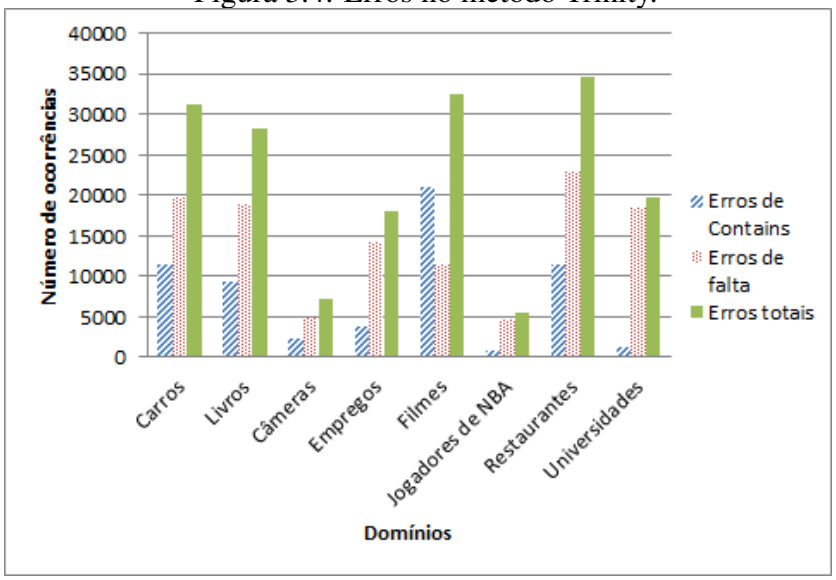


Figura 3.5: Erros no método Weir.

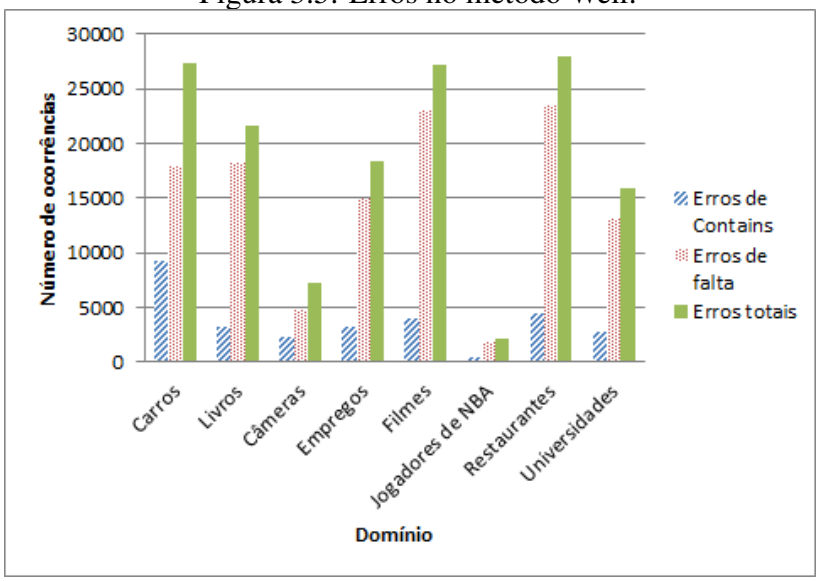


Figura 3.6: Erros por método de extração de dados.

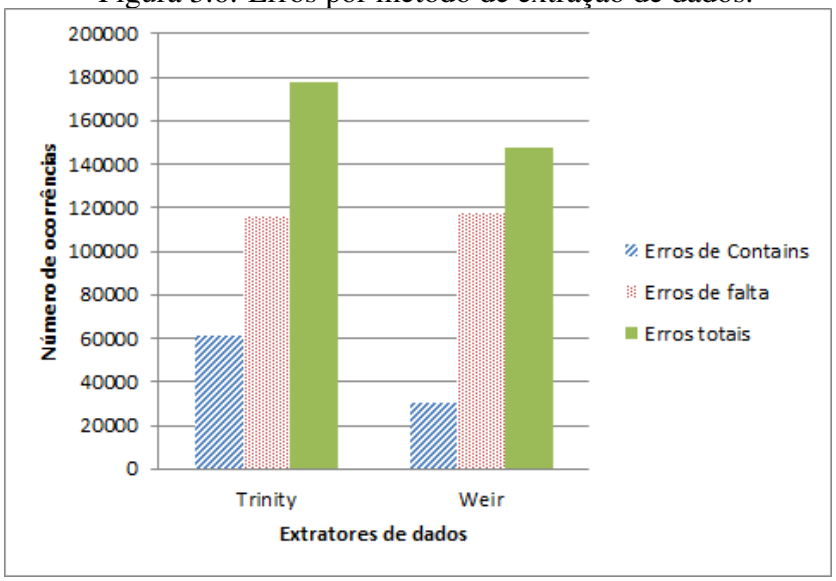
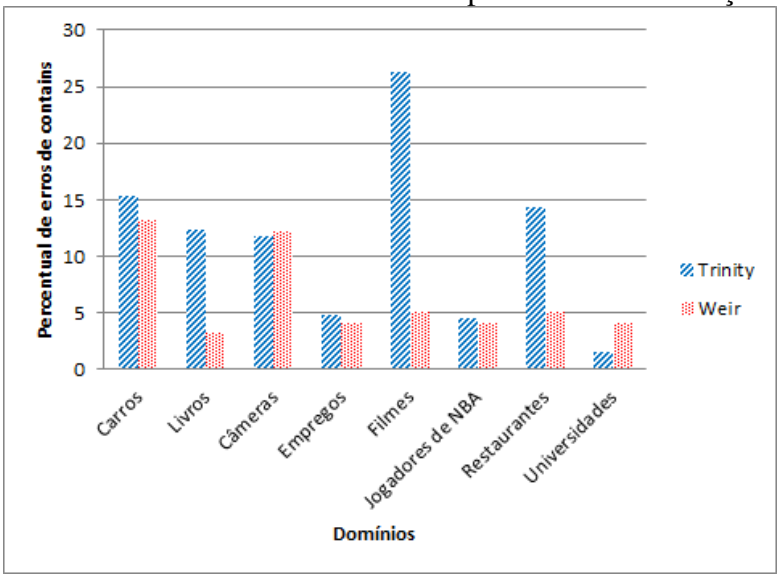


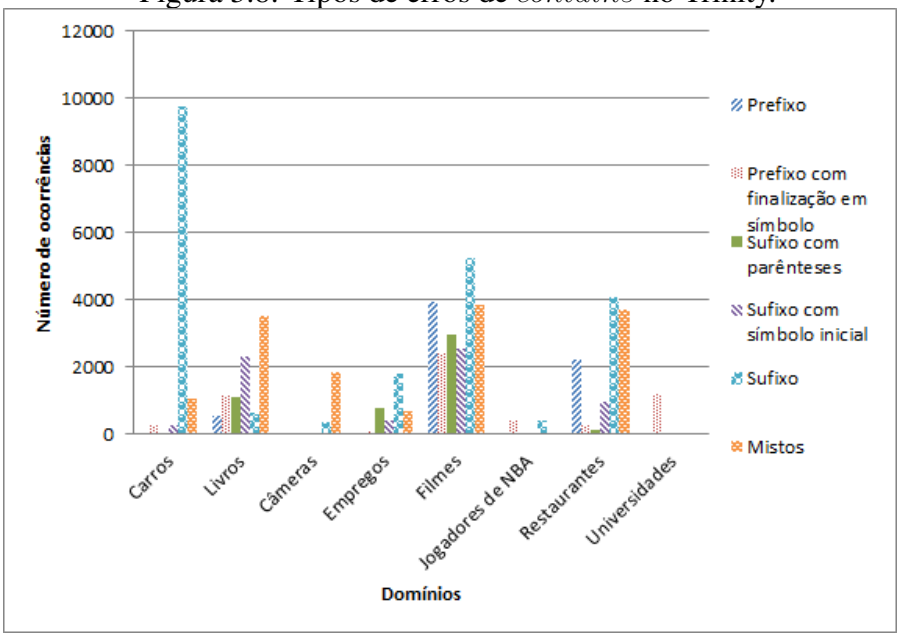
Figura 3.7: Percentual de erros de *contains* por método de extração de dados.



- Prefixos terminados por símbolo: Valor extraído com um rótulo o qual seu ultimo caracter é um símbolo, como por exemplo $phone$:
- Prefixos: Valor extraído com um rótulo sem símbolo, exemplo: $number$
- Sufixos com parênteses: Valor extraído apresentando uma observação junto do valor, por exemplo: título do filme *Batman vs Superman(2016)*
- Sufixos inicializados por símbolo: Valor extraído como um detalhe iniciado com símbolo, como por exemplo: $, limited$
- Sufixos: Valor extraído apresentando detalhe sem nenhuma marcação com símbolo ou parênteses, por exemplo: *Modelo limited*
- Mistos (sufixo e prefixo): Valor extraído possui prefixo e sufixo ao mesmo tempo, por exemplo: *Assista: Batman vs Superman (2016)*

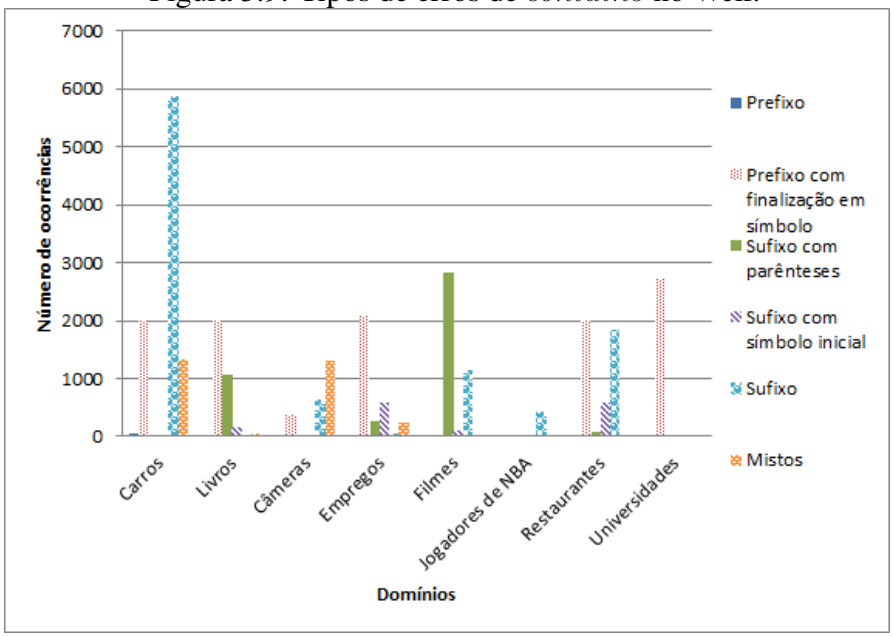
Todos os casos acima estão incluídos apenas nos erros de $contains$ sem que nenhum deles influencie os demais. O avaliador desenvolvido pelo autor, ao computar os erros de $contains$, verifica a qual desses tipos o erro pertence. Cada tipo é, então, contabilizado para arquivos diferentes, que são separados por $site$ e $dataset$. Assim como nos casos de erros anteriores, foram gerados gráficos para identificar a quantidade de cada tipo de erro, os quais são exibidos nas Figuras 3.8, 3.9 e 3.10.

Figura 3.8: Tipos de erros de $contains$ no Trinity.



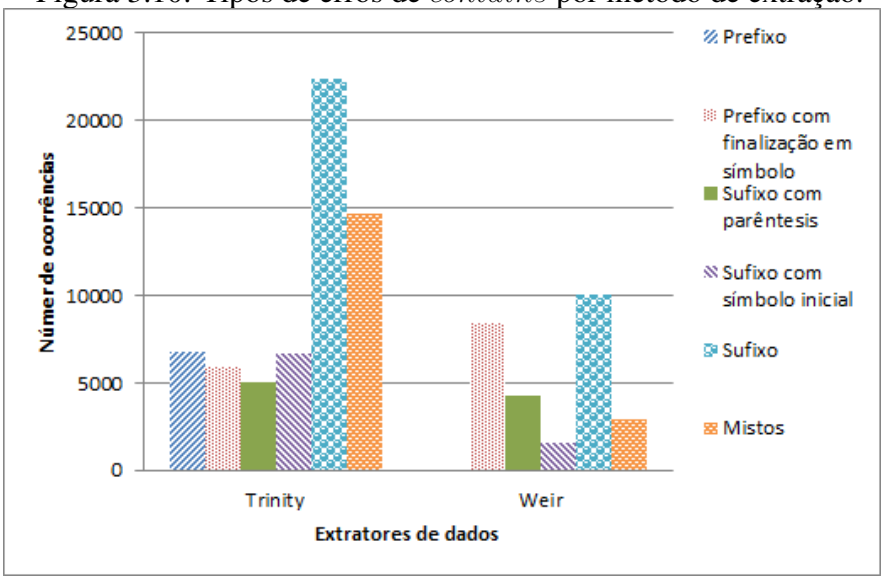
Na Figura 3.8 os casos de sufixo são mais comuns, porém o domínio $Filmes$ apresenta todos tipos de erros de $contains$ de maneira relativamente distribuída. Enquanto que na Figura 3.9 há um maior volume erros de prefixo terminados em símbolos, assim como

Figura 3.9: Tipos de erros de *contains* no Weir.



grande quantidade de erros de sufixo.

Figura 3.10: Tipos de erros de *contains* por método de extração.



Na Figura 3.10 os resultados do método Trinity se mostram bem distribuídos, em contra partida, com maior quantidade de erros. Já o Weir apresenta muitos casos de erros de sufixos, prefixos finalizados com símbolos e sufixo com parênteses.

Após a verificação do acúmulo de erros de *contains* foi averiguada a proporção de cada tipo de erro de acordo com o quadro geral da base de dados e com a mesma lógica utilizada na Figura 3.7. A partir disso, foram gerados os gráficos das Figuras 3.11 e 3.12

Figura 3.11: Percentual de tipos de erros de *contains* no Trinity.

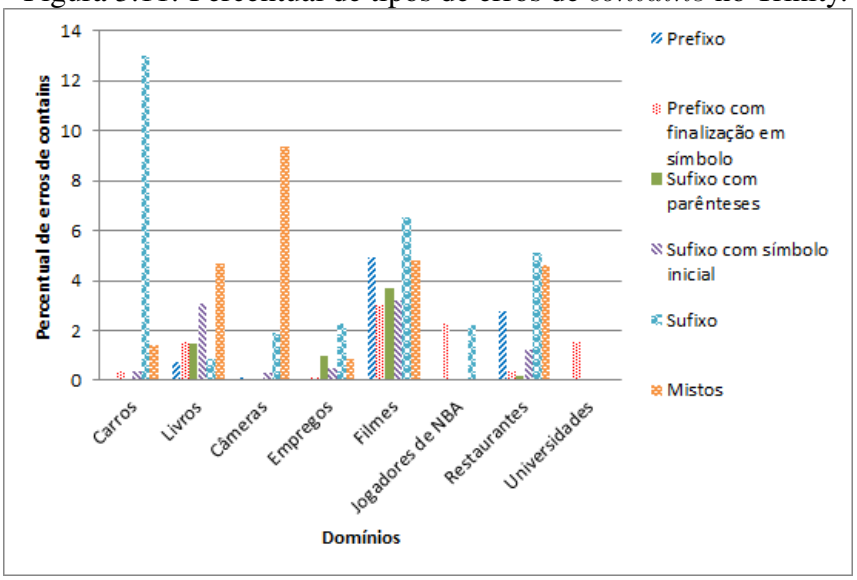
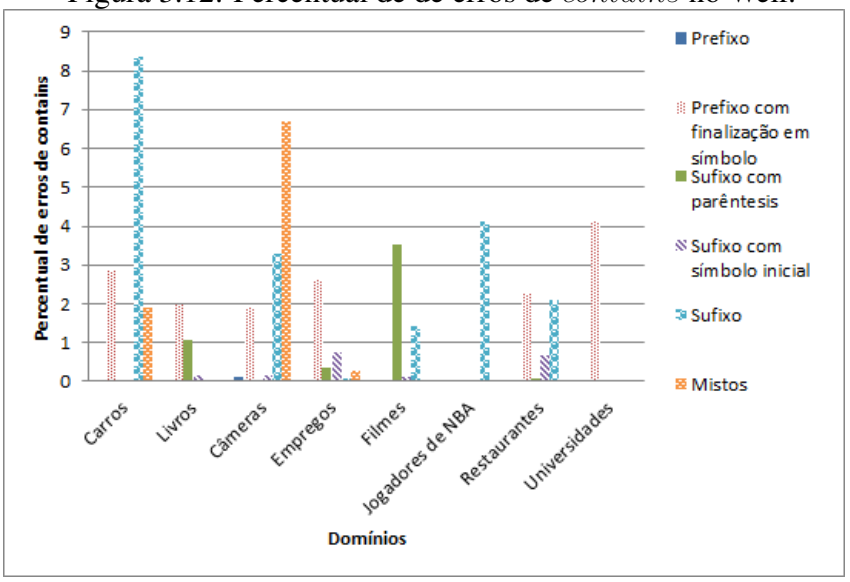


Figura 3.12: Percentual de de erros de *contains* no Weir.



Na Figura 3.11 é possível notar que os casos de sufixo podem alcançar próximo de 14% de ocorrência, como observado no domínio *Carros*. Já na Figura 3.12 este tipo de erro alcança até 8%, enquanto que o erro de prefixos finalizados em símbolo e o de prefixo junto a sufixo não possuem ocorrências maiores que 6%.

3.3 Considerações finais

A avaliação obteve os casos de falha dos métodos Trinity e Weir. O Weir foi o método que apresentou melhor eficácia em F1. O Trinity apresenta um maior número de ocorrência de erros, embora o Weir apresente um número de ocorrência de erros relevante.

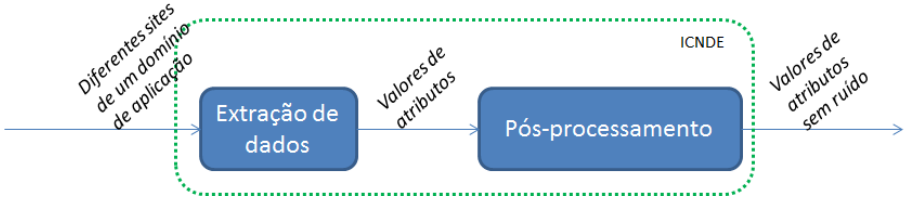
O percentual de erros de $contains$ influencia até 25% de um domínio. Esse percentual é relevante em comparação a um domínio.

É possível criar um método que corrige os casos de erros de $contains$, tratando de cada um de seus tipos classificados e tratando cada caso de ruído. Esse tratamento permitirá a identificação e a correção do ruído. Através da correção, são obtidos resultados mais próximos do gabarito. Assim, são obtidos melhores índices de precisão, revocação e F1 para os métodos utilizando a mesma base de dados.

4 ICNDE - UM NOVO MÉTODO PARA IDENTIFICAÇÃO E CORREÇÃO DE RUÍDO PARA EXTRAÇÃO DE DADOS

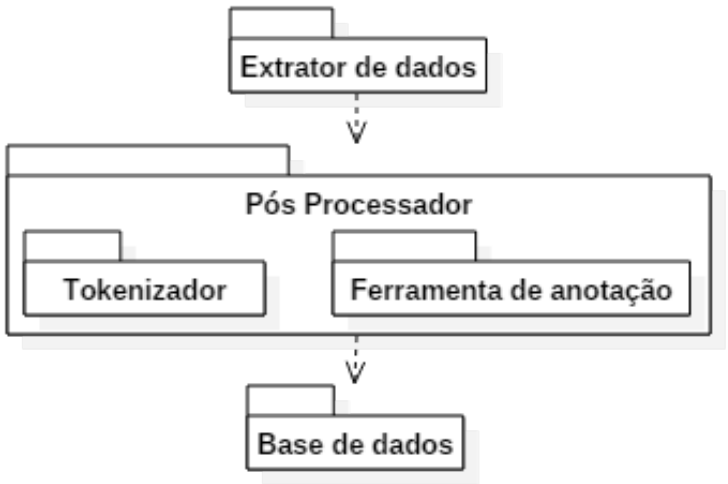
Este capítulo apresenta o ICNDE (*Identification and Correction of Noise to Data Extraction* - Identificação e Correção de Ruído para Extração de Dados) para a extração de dados de páginas-entidades baseadas em $template$. A Figura 4.1 apresenta as principais etapas do ICNDE. O método tem como entrada um conjunto de páginas-entidade de diferentes sites de um domínio de aplicação. A saída é um conjunto de regras que extraem valores de atributos de diferentes sites de um domínio. O método inclui a etapa de extração de dados que realiza a extração dos dados em cada página-entidade (Seção 4.1) e a etapa de pós-processamento, que elimina os ruídos nos dados extraídos (Seção 4.2).

Figura 4.1: Etapas do método.



A arquitetura do ICNDE está ilustrada na Figura 4.2 e é genérica o suficiente para poder ser relacionada à saída de qualquer método de extração de dados. A arquitetura apresenta 5 componentes principais. O extrator de dados extrai os dados de um domínio. O Pós-processamento utiliza um tokenizador e a ferramenta de anotação. O tokenizador

Figura 4.2: Arquitetura do sistema.



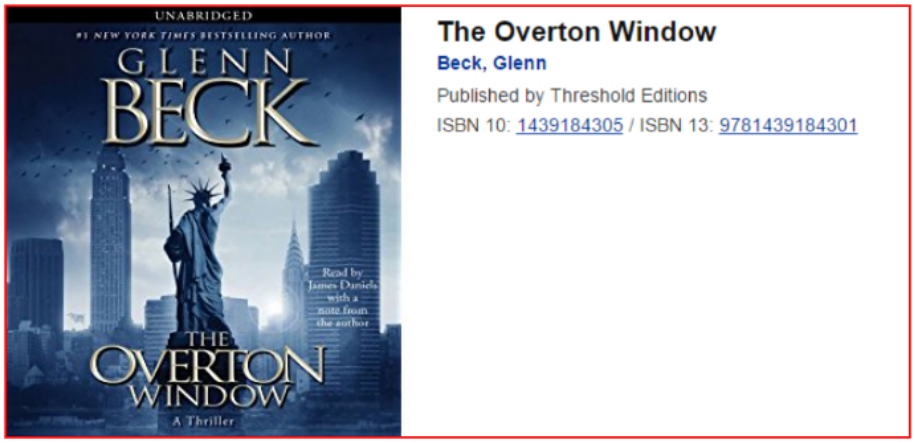
divide as instâncias em *substrings* denominadas como *tokens*, que são avaliados para verificar se são ruído ou não. A ferramenta de anotação identifica tipos de dados que seguem um padrão (por exemplo, data, endereço e telefone). Por fim, a base de dados armazena o ruído encontrado.

A Figura 4.3(a) apresenta um exemplo real de uma página-entidade referente a um livro do site adebooks[1]. Nesse exemplo, a entidade descrita é um livro que é descrito através dos atributos título, autor, editora, entre outros. A Figura 4.3(b) ilustra uma saída esperada pelo ICNDE, uma identificação dos atributos com as suas respectivas instâncias.

Figura 4.3: Exemplo de entrada e saída do método

Entrada

a)



Saída

b)

Livro	
Título	The Overton Window
Autor	Beck, Glenn
Editora	Threshold Editions
ISBN 10	1439184305
ISBN 13	9781439184301

4.1 Extração de dados

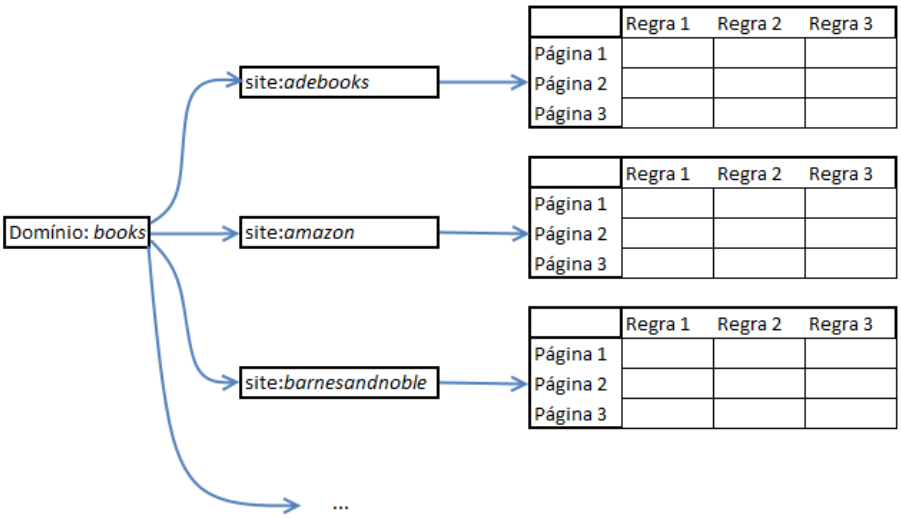
Esta seção apresenta a etapa de Extração de dados do ICNDE, que tem como objetivo extrair dados de atributos de páginas-entidade. A Seção 4.1.1 apresenta uma visão geral dessa etapa e a Seção 4.1.2 apresenta os desafios após a realização desta etapa.

[1]https://www.abebooks.com/Overton-Window-Glenn-Beck-Threshold-Editions/11264262612/bd

4.1.1 Visão geral

A etapa tem uma entrada e uma saída. Obtém-se a saída através de um extrator (por exemplo, Trinity e Weir). O extrator de dados tem como entrada um domínio de aplicação específico com um conjunto de diferentes sites, cada site possui páginas-entidade. O extrator gera regras que extraem valores de atributos analisando as páginas-entidade, essas regras são dividas por sites. A Figura 4.4 apresenta a estrutura de dados de saída dessa etapa. Um domínio possui diversos sites, cada site possui uma matriz bidimensional na qual as páginas são uma dimensão e as regras outra. Cada regra extrai valores de um atributo. As células contêm o valor extraído pela regra da página.

Figura 4.4: Organização da estrutura de dados que compõe a saída da etapa de extração.



4.1.2 Desafios

Após a execução do extrator, são encontrados valores ruidosos. A Tabela 4.1 apresenta 6 exemplos de valores de atributos extraídos com ruído:

1. Valores com $tags$, apresentam $tags$ dentro da instância;
2. Terminadores nos extremos, que são apenas símbolos do template que ficam presentes junto ao valor do atributo;
3. Prefixos terminados com símbolo, os quais geralmente funcionam como um rótulo do atributo;
4. Prefixos, caso mais difícil identificar, devido a falta de alguma marcação;

5. Sufixos iniciados com símbolo, que são comuns de ocorrer, uma vez que no mesmo atributo podem aparecer informações adicionais;
6. Sufixos, que ocorrem pelo mesmo motivo, porém não tem um símbolo como marcação;
7. Sufixos com parênteses, que são informações adicionais nos atributos que estão entre parêntesis.

Tabela 4.1: Casos problemáticos

Linha	Caso	Exemplo de valor
1	Valores com Tags	$Douglas < b > Addams$
2	Terminadores aos extremos	$> O\ Segredo <$
3	Prefixos terminados com símbolo	$Pub\ Date : 20/05/2015$
4	Prefixos	$Pub\ Date\ 20/05/2015$
5	Sufixos iniciado com símbolo	*O Senhor dos Anéis, disponível*
6	Sufixos	$O\ Senhor\ dos\ Aneis\ disponvel$
7	Sufixos com parêntesis	$O\ Hobbit\ (a\ venda)$

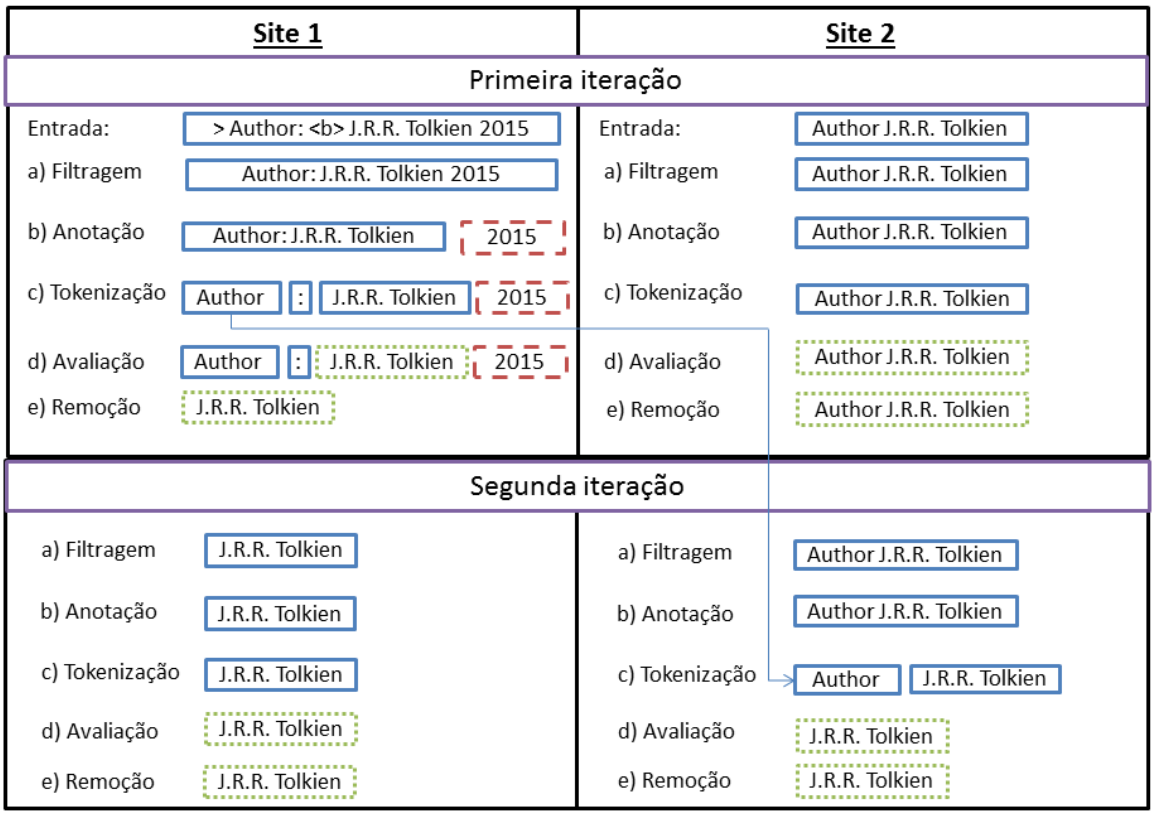
Esses desafios levam a necessidade de uma etapa de pós-processamento, que tem como objetivo eliminar os ruídos presentes nas instâncias.

4.2 Pós-processamento

Esta seção descreve a etapa de pós-processamento, que tem como objetivo eliminar prefixos, sufixos e outros tipos de ruídos extraídos com valores de atributos.

A Figura 4.5 apresenta execução de duas regras de uma mesma entidade de sites diferentes. O exemplo apresenta 5 etapas. Primeiramente, é dada a entrada que é um valor extraído de um atributo. A etapa a realiza a filtragem de $tags$ e terminadores. A etapa b define as anotações, no exemplo a anotação é o valor tracejado representado por uma data. A etapa c realiza a tokenização em que, através de um conjunto de símbolos, valores aprendidos como ruído e anotações. A etapa d apresenta quais $tokens$ são considerados ruído, o valor pontilhado representa o não considerado ruído. A etapa e apresenta a remoção do ruído de uma iteração do pós-processamento. A seta apresentada de um site para outro mostra que um $token$ pode influênciar em outra iteração em qualquer site, uma vez que o valor $author$ foi aprendido como ruído então ele é definido como um separador para a tokenização.

Figura 4.5: Exemplo de pós-processamento.



A Seção 4.2.1 apresenta a estrutura de dados utilizada para o pós-processamento. A Seção 4.2.2 apresenta em detalhes o algoritmo de pós-processamento.

4.2.1 Estrutura de dados

Esta seção introduz a estrutura de dados utilizada. A estrutura de dados é organizada da seguinte maneira: o domínio determina uma lista de diferentes sites; cada site apresenta uma matriz bidimensional em que uma dimensão representa às regras para o domínio do site e a outra as páginas do mesmo. Essa organização foi apresentada anteriormente na Seção 4.4.

O ICNDE neste trabalho utiliza um índice de $tokens$ para contagem de ocorrências em nível global (de domínio) e em nível local (de regra). A Tabela 4.2 apresenta um exemplo de $Livros$ para uma regra que extrai valores de datas de publicação. A ocorrência local é uma lista do número de ocorrências do $token$ em cada regra. Uma lista de $tokens$ global é utilizada para identificar novos $tokens$. Por fim, é utilizada uma lista de $tokens$

confirmados como ruído no pós-processamento do ICNDE.

Tabela 4.2: Exemplo de índices de $tokens$

Chaves	Ocorrências Locais	Ocorrências Globais
’"Pub Date"	(r1,2000),(r2,1000),(r3,2000)	5000
’":"	(r1,2000),(r2,0),(r3,2000)	4000
"25/04/2015"	(r1,1),(r2,7),(r3,2)	10
"16/07/2010"	(r1,0),(r2,2),(r3,0)	2
"27/01/2013"	(r1,1),(r2,0),(r3,2)	3
"08/12/2012"	(r1,0),(r2,0),(r3,0)	1

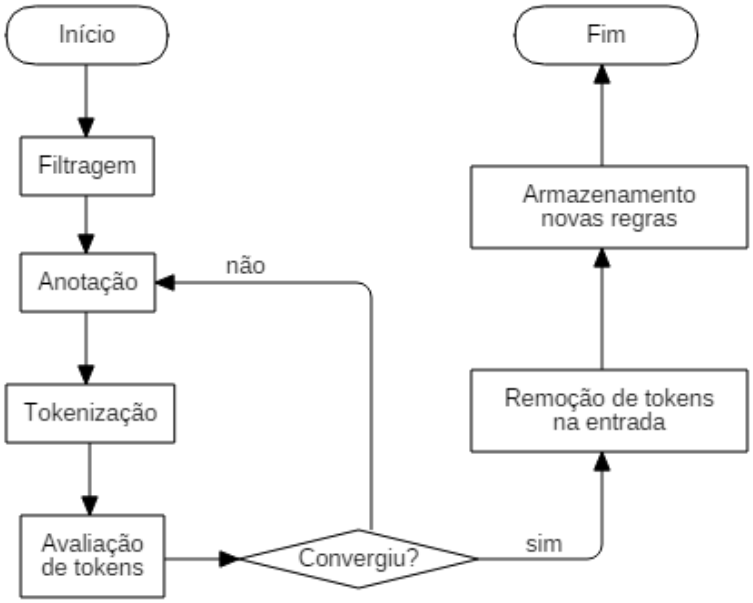
4.2.2 Algoritmo

Esta Seção apresenta o algoritmo de pós-processamento. O algoritmo tem como entrada o domínio com diferentes sites, cada um com um conjunto de regras. A entrada é a estrutura definida na Figura 4.6. Sua saída é o mesmo conjunto, porém com os valores das instâncias corrigidos. É feita a identificação de dados que seguem um padrão como anotações. Em seguida, é feita tokenização que divide a instância a partir de valores pré-definidos. Os $tokens$ gerados são avaliados através de suas ocorrências para classificá-los como ruído. Os $tokens$ classificados como ruído são adicionados em uma lista de $tokens$ confirmados como ruído. Caso essa lista não sofra modificações na iteração significa que ela convergiu. Do contrário, retorna-se ao passo de anotação. Caso a lista de tokens confirmados como ruído convirja, os $tokens$ confirmados são removidos de suas respectivas regras e, por fim, os valores após a remoção são salvos nas instâncias das regras no domínio em seus respectivos sites.

Esse método segue o fluxo apresentado na Figura 4.6. O fluxograma da etapa de pós-processamento possui seis subetapas e um laço. A subetapa de filtragem retira $tags$ e símbolos nos extremos da instância. A subetapa de anotação marca dados que seguem um padrão dentro da instância. A subetapa de tokenização separa a instância em $tokens$ através de valores pré-definidos. A subetapa de avaliação de $tokens$ classifica se o $token$ é ruído ou não. Caso um novo $token$ seja classificado como ruído, significa que a avalição não convergiu - a lista de $tokens$ confirmados como ruídos foi modificada na iteração atual - o fluxo retorna a subetapa de anotação. Caso a avaliação convirja, é feita a subetapa de eliminação dos $tokens$ na entrada, em que todo o ruído é eliminado na

entrada. A subetapa de armazenamento de novas regras salva os arquivos com as regras sem ruído da mesma forma que os arquivos de entrada do pós-processamento.

Figura 4.6: Fluxograma apresentando as etapas do pós-processamento.



A subetapa de filtragem tem como objetivo retirar de cada instância qualquer tipo de símbolo terminador que é apresentado em seus extremos e eliminar $tags$ que podem aparecer na instância. Ambos os filtros são realizados através de expressões regulares. Após essa subetapa é feita uma cópia da instância para as próximas etapas.

A subetapa de anotação marca os tipos de dados que seguem um padrão (por exemplo, datas, $links$, endereços, telefones, entre outros). Esses dados são definidos como atômicos, ou seja, eles não são tokenizados. Esses dados podem ser identificas através de expressões regulares ou de ferramentas como Gate[2](CUNNINGHAM, 2002) ou brat[3](STENETORP et al., 2012). O uso de anotações transforma um valor como um $token$ atômico, ou seja, dados como endereços, datas, números de telefones ou outros tipos de dados que seguem um padrão não segmentado. Um exemplo de anotações de datas é $05/04/2015$ este valor é considerado um valor indivisível, logo não é separado por tokenização. A cópia de instância ao encontrar um valor marcado como anotação o salva como um $token$ e o substitui por um símbolo para a subetapa seguinte.

A subetapa de tokenização divide a instância em $tokens$. A divisão é avaliada através de um conjunto de símbolos especiais (por exemplo, $(|.;,*-\backslash)$) em forma de

[2]https://gate.ac.uk/sale/tao/split.html
[3]http://brat.nlplab.org/

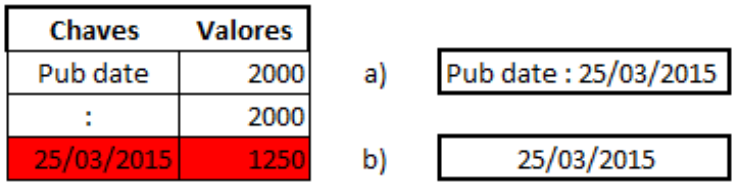
expressão regular. Outra expressão utilizada nessa subetapa é a interpretação de valores entre parênteses (por exemplo, $(available - this\ week)$). Por fim, outro caso para a tokenização é para $tokens$ aprendidos como ruído. Esses casos definem como a instância é dividida. Durante essa subetapa, cada $token$ tem sua ocorrência contada de maneira global (em relação ao domínio) e de maneira local (em relação a regra corrente) em um índice de termos. Após a tokenização, a cópia é completamente dividida e destruída, uma vez que cada $token$ é retirado dela.

A subetapa de avaliação de $tokens$ avalia o índice de $tokens$ de cada regra que contém a contagem de ocorrências de cada $token$, caso a contagem tenha uma proporção maior ou igual Θ na regra, então, o $token$ é classificado como candidato. Caso esse $token$ candidato tenha uma proporção menor ou igual a Φ em relação a todo o domínio, então esse $token$ é classificado como ruído e é adicionado a uma lista local para a regra e a uma lista global para o domínio. Esse $token$ classificado como ruído é aprendido para subetapa de tokenização. O Θ é um limiar local utilizado para definir uma proporção que apresente uma melhor precisão para a definição dos $tokens$ candidatos. Esse limiar é utilizado para verificar se um $token$ é repetido muitas vezes no mesmo site, o que pode significar que esse seja ruído. Por exemplo, $Pub\ date$: se repete em todas as instâncias de uma regra, esse pode ser um prefixo a ser considerado um ruído. Φ é um limiar global em relação ao domínio que avalia a proporção em função da quantidade de páginas presentes de um domínio. Caso um $token$ se repita em muitas regras e em diferentes sites, significa que esse $token$ candidato pode ser um $token$ válido que apenas se repete muitas vezes. Por exemplo, supondo que o valor $public$ se repita muitas vezes em um determinado site, esse também se repete muitas vezes em outros sites, logo ele provavelmente não é um ruído, uma vez que cada site tem sua maneira de organização.

O teste que determina se o algoritmo convergiu funciona da seguinte maneira. Caso a lista de $tokens$ classificados como ruído não seja atualizada, isso significa que o algoritmo convergiu e pode seguir para subetapa de remoção de $tokens$ na entrada. Caso contrário, o fluxo retorna para a subetapa de anotação. É retirada uma nova cópia da instância atual, pois a cópia anterior foi destruída no fim da subetapa de tokenização. Esse laço existe porque na tokenização é identificado o ruído que foi aprendido em iterações anteriores na tokenização em outras instâncias de outras regras. Assim, é possível dividir valores que não possuem marcações por símbolos especiais.

A subetapa de remoção de $tokens$ na entrada pode ser dividida em 3 casos diferentes. O primeiro caso ocorre quando existem $tokens$ válidos na instância, todos os

Figura 4.7: Exemplo de remoção de $tokens$.



$tokens$ classificados como ruído são removidos da instância. O $token$ classificado como ruído não é removido apenas se é um caractere especial e não existe outro $token$ classificado como ruído na instância. O segundo ocorre quando existe apenas um $token$ e esse é classificado como ruído, esse $token$ não é removido, visto que esse $token$ possa ser um valor que se repete muito em uma regra. Por exemplo, 2015 é um valor que localmente se repete muito, removê-lo seria acrescentar erro ao invés de eliminar um erro. O terceiro caso ocorre quando existe apenas $tokens$ confirmados na instância. Nesse caso os $tokens$ são ordenados de maneira decrescente pela quantidade de ocorrências na regra, o que tiver menor quantidade de ocorrências locais não é removido da instância e o restante dos $tokens$ são removidos. A Figura 4.7 apresenta um exemplo deste caso. A Figura 4.7(a) apresenta a instância antes da remoção e a Figura 4.7(b) apresenta a instância após a remoção dos $tokens$ confirmados. Após a remoção, a estrutura do domínio é corrigida e está pronta para a última subetapa.

A subetapa armazenamento das novas regras salva as regras sem ruído e as organiza da mesma maneira que a entrada dessa etapa é fornecida.

O Algoritmo 1 apresenta uma visão de alto nível da solução apresentada neste capítulo. Em relação ao domínio, D é o domínio de entrada após a leitura dos arquivos; $caminhoD$ trata-se do caminho do domínio na base de dados; s é o site; r é a regra; p é a página; D_s é o site do domínio; $D_{s,r}$ é a regra de um site; $D_{s,r,p}$ é uma página de um site.

Em relação ao indice de $tokens$, t é um $token$; l refere-se ao índice de occorências locais; g refere-se ao número de ocorrências globais do índice; $CpInst$ é uma cópia local da instância; $indT_{s,r,t}$ é um $token$ de uma regra; $indT_{s,r,t,l}$ é a quantidade de ocorrências locais do $token$ c na regra r; $indT_{s,r,t,g}$ é o número de ocorrências globais do $token$ t;

Em relação ao restante da estrutura, $lstT$ é a lista de $tokens$ gerados através das anotações e tokenização; $Simb$ é um conjunto de simbolos, expressões e valores que são utilizados para realizar a tokenização e dividir a instância em $tokens$; $indT$ é o índice de

$tokens$ que tem como chave os $tokens$ e é dividio por l para ocorrências locais e por g para ocorrências globais; $lstR$ é uma lista de $tokens$ classificados como ruído.

Algorithm 4.1: Pós-processamento ICNDE

```
   Data: Dados sem ruído
   Result: Correção dos Dados Extraídos
 1 D ← leDadosExtraidos(caminhoD);
 2 FiltraInst(D);
 3 convergiu ← Falso;
 4 while ¬convergiu do
 5     convergiu ← Verdadeiro;
 6     for s ∈ D do
 7         for r ∈ D_s do
 8             for p ∈ D_{s,r} do
 9                 CpInst ← D_{s,r,p};
10                 lstT ← {lstT} ∪ anota(CpInst);
11                 lstT ← {lstT} ∪ tokeniza(CpInst, Simb);
12                 indT_{s,r} ← contaTokens(lstT);
13     for s ∈ indT do
14         for r ∈ indT_s do
15             for c ∈ indT_{s,r} do
16                 if indT_{s,r,t,l} ≥ Θ * D_{s,r} ∧ indT_{s,r,t,l} ≤ φ * indT_{s,r,t,g} ∧ c¬ ∈
                    lstConf_{s,r} then
17                     lstConf_{s,r} ← {lstConf_{s,r}} ∪ c;
18                     convergiu← Falso
19 for s ∈ D do
20     for r ∈ D_s do
21         for p ∈ D_{s,r} do
22             removeRuido(D_{s,r,p}, lstR_{s,r});
23 salvaD(D, caminhoSaida);
```

O Algoritmo 1 realiza a correção de ruído presente nos dados extraídos pelo extrator. Na Linha 1 é dado o caminho da base onde está o domínio. O caminho é colocado em memória na estrutura apresentada na Figura 4.4. Na linha 2 é feita a filtragem das instâncias em todo o domínio, eliminando terminadores nos extremos das instâncias e $tags$. Na linha 3, o valor de convergiu, sendo considerado como falso, uma vez que não foi realizada nenhuma etapa do pós-processamento.

Os laços existentes na linha 4 servem para garantir que a lista de $tokens$ classificados como ruído não foi mais atualizada. Na linha 5, o valor de convergiu é dado com verdadeiro para o caso de nenhuma alteração ser realizada na lista de valores confirma-

dos. As linhas 6, 7 e 8 servem para que a anotação e tokenização sejam realizadas em nível de instância, ou seja, percorre-se cada página de cada regra de cada site do domínio. Na linha 9 é feita uma cópia da instância corrente, pois nas etapas seguintes os $tokens$ encontrados serão retirados dessa cópia. Na Linha 10, a lista de $tokens$ é atualizada com os valores anotados encontrados dentro da instância. A cópia remove essas anotações e salva na lista de $tokens$. Na linha 11, é realizada a tokenização que divide a cópia por um conjunto de símbolos, expressões ou valores aprendidos. Todos esses valores divididos são passados para a lista de $tokens$. Na linha 12, os $tokens$ são contabilizados em um índice de $tokens$ para o domínio e para a regra.

As linhas 13, 14 e 15 navegam no índice de $tokens$ para poder consultar as ocorrências locais de um determinado $token$ para realizar a subetapa de avaliação de $tokens$. Na linha 16 são verificados os limiares e se o $token$ já foi confirmado anteriormente. Na linha 17, o $token$ é adicionado na lista de confirmados. Na linha 18, convergiu é dado como falso, pois a lista de valores confirmados sofreu alguma alteração nessa iteração.

As linhas 19, 20 e 21 são para navegar por cada instância do domínio de entrada para a subetapa de remoção dos $tokens$. Na linha 22, os $tokens$ confirmados são removidos de suas respectivas regras no domínio. Na linha 23, a nova saída é salva no mesmo formato da entrada no caminho definido pelo usuário.

5 AVALIAÇÃO EXPERIMENTAL

Este capítulo apresenta a avaliação experimental do método ICNDE. A Seção 5.1 apresenta a configuração dos experimentos e a Seção 5.2 descreve os experimentos, que incluem a avalição de limiares, a avaliação da eficácia do método ICNDE, a análise do número de ocorrências de erro e a análise da execução do experimento.

5.1 Configuração do experimentos

Está seção está organizada da seguinte forma. A subseção 5.1.1 descreve as características das bases de dados. A subseção 5.1.2 define as métricas de avaliação. A subseção 5.1.3 apresenta as $baselines$. Finalmente, a subseção 5.1.4 discute a metodologia adotada.

5.1.1 Bases de Dados

Os dados de entrada para os experimentos são os mesmos utilizados no Capítulo 3, adquiros através do site SWDE Codeplex, apresentando as mesmas estruturas de domínio, site, regra e página. A base de dados de entrada possui oito domínios, cada um com 10 sites. No total, são 124291 páginas utilizadas na base de dados. A base para esse experimento é saída da execução dos métodos Trinity e Weir sobre essa base de dados.

5.1.2 Métricas

As seguintes métricas são utilizadas:

- Precisão:

$$P = \frac{verdadeiros\,positivos}{falsos\,positivos + verdadeiros\,positivos} \tag{5.1}$$

- Revocação:

$$R = \frac{verdadeiros\,positivos}{falsos\,negativos + verdadeiros\,positivos} \tag{5.2}$$

- F1:

$$F1 = 2 * \frac{P * R}{P + R} \tag{5.3}$$

- Teste t: Avalia se as médias de duas distribuições normais de valores que são estatisticamente diferentes, apresenta bons resultados mesmo quando as distribuições não são perfeitamente normais. Foi utilizado o limiar de α=0,05. Quando o valor da probabilidade p bicaudal calculado é menor que α, existe uma diferença significativa entre os desempenhos dos métodos analisados.
- Tempo de execução: indica o tempo na execução do pós-processamento do ICNDE

5.1.3 *Baselines*

O experimento necessita de um extrator de dados, neste caso foram escolhidas duas abordagens. Então foi escolhido testar uma abordagem baseada em XPath e outra baseada em árvore.

A abordagem baseada em XPath escolhida para comparação foi a Weir (Seção 2.3.1). Essa abordagem foi escolhida porque ela: (i) é um estado da arte na extração de dados de páginas-entidade baseadas em $template$; (ii) é automática; (iii) é independente de padrões HTML específicos (por exemplo, tabelas e listas HTML); (iv) é independente de uma base de conhecimento; e (v) é independente de anotação do usuário.

A abordagem baseada em árvore escolhida para comparação foi a Trinity (Seção 2.3.2). Essa abordagem foi escolhida porque ela: (i) é um estado da arte na extração de dados de páginas-entidade baseadas em $template$; (ii) possui tratamento para atributos com formato variante; (iii) é automática; (iv) é independente de padrões HTML específicos (por exemplo, tabelas e listas HTML); e (v) é independente de uma base de conhecimento.

5.1.4 Metodologia

Todos os experimentos foram executados em um Notebook Samsung com Windows 10, processador intel i7 de 4ª geração com 4 núcleos e 8Gb de memória RAM.

A etapa de extração de dados do método ICNDE necessita de um extrator de dados. Essa etapa foi avaliada com um extrator baseado em XPath (Weir) e com um extrator baseado em árvore (Trinity). A saída foi retirada de (MANICA; DORNELES; GALANTE,

submetido para publicação 2017). O Weir utiliza um estágio de alinhamento que introduz erros, nesse caso foi utilizado o alinhamento manual. As regras que extraem maiores F1 são as regras selecionadas para a extração de cada atributo em cada site. As regras de diferentes sites que extraem os valores de mesmo atributo são alinhadas. Esse mesmo alinhamento foi utilizado no Trinity.

O Weir utiliza dois parâmetros: o $\delta(r)$ - o número máximo de passos do XPath a partir do primeiro $../$ que uma regra de extração deve ter. E min_{pages} - o número mínimo de páginas que define um nodo $template$. Nesse experimento foram utilizado os parâmetros $\delta(r) = 6$ e $min_{pages} = 40$. Já o Trinity tem como entrada os valores mínimo e máximo dos tamanhos de padrões compartilhados para o algoritmo de pesquisa. Foi utilizado min=1 e max=0,05 * m, ondem é o tamanho em $tokens$ de uma página pequena.

Após a execução do método, a saída é salva com um novo nome no diretório de origem. Os valores dos limiares influenciam no ganho da correção, devido a isso foi necessário avaliar os valores dos limiares para buscar o caso que gere maior ganho em F1 na correção. Os casos de testes que avaliam os limiares, executam 10 casos para cada limiar, Θ é executado valendo de 10% a 100% e Φ também com esse intervalo de execução, tendo como incremento 10%. Cada saída desse teste possui no nome do diretório os valores de Θ e Φ. Toda a identificação de anotações foi realizada através de expressões regulares, ao invés de uma ferramenta de anotação.

O experimento utilizou a linguagem Python para a etapa de pós-processamento, uma vez que é uma linguagem em que se tem facilidade para manipular cadeias de caracteres e expressões regulares, sendo assim mais fáceis para desenvolvê-lo o método.

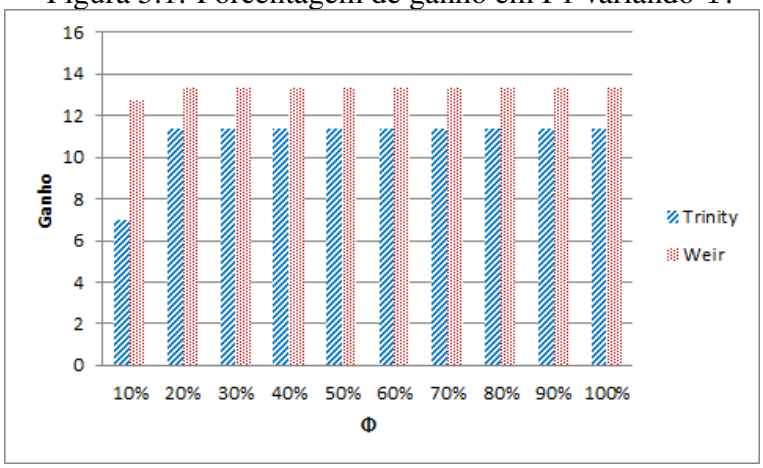
5.2 Descrição dos experimentos

Nesta seção, é apresentada a descrição dos experimentos e seus resultados. A Seção 5.2.1 apresenta o experimento de definição dos limiares, a Seção 5.2.2 descreve a eficácia do ICNDE. A Seção 5.2.3 apresenta a análise dos tipos de erros. A Seção 5.2.4 apresenta o tempo de execução. A Seção 5.2.5 apresenta os casos de falha do ICNDE.

5.2.1 Definição de limeares

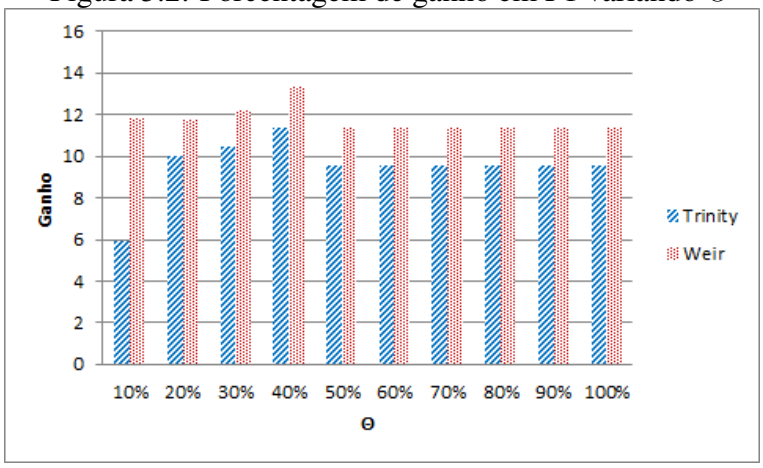
O objetivo deste experimento é encontrar as melhores configurações dos parâmetros para a etapa de pós-processamento, que envolve sobrando definir os limiares Θ e Φ. Para avaliar os limiares foi escolhido o F1 como medida. A avaliação é realizada pelo método para verificar se os valores dos limiares influenciam de maneira diferente em cada método de extração. Para cada teste, um limiar é mantido constante e o outro é variado, o critério de escolha do melhor limiar trata-se aquele gera maior F1. Quando Φ avaliado $\Theta = 40\%$ e quando Θ é avaliado $\Phi = 20\%$.

Figura 5.1: Porcentagem de ganho em F1 variando Φ.



Na Figura 5.1 se mantem constante a porcentagem de ganho em F1 a partir 20% em Φ, além de ser constante também é o melhor caso. A escolha para Φ foi 20%.

Figura 5.2: Porcentagem de ganho em F1 variando Θ



A Figura 5.2 apresenta que a partir de 50% F1 é constante para ambos extratores. O melhor caso de ganho ocorre quando Θ está em 40%, em que o Weir chega a ter um ganho de 13% e o Trinity de 11%. Por ser o maior ganho em F1 nos dois extratores de

dados, o Θ escolhido foi 40%.

Os limiares $\Phi = 20\%$ e $\Theta = 40\%$ foram os parâmetros escolhidos, por terem a melhor eficácia em F1. Os demais experimentos utilizam esses valores de limiares como parâmetros. É importante ressaltar que esses valores de limiares obtêm os melhores casos nos dois métodos de extração de dados.

5.2.2 Eficácia do método

Esse experimento compara a eficácia do ICNDE com os $baselines$. Para medir a eficácia foi escolhido o F1. A Tabela 5.1 apresenta uma comparação dos métodos de extração com e sem o pós-processamento do ICNDE. A diferença percentual de F1 do Weir é de 9% e a diferença percentual de F1 no Trinity é de 7%.

Tabela 5.1: F1 com e sem etapa de pós-processamento do ICNDE

Extrator	Sem pós-processamento	Com pós-processamento	Ganho
Trinity	61,04769997	68,0083327	11,40195738
Weir	67,72112384	76,7551955	13,34010885

Além desses resultados foi avaliado o F1 de cada site, para verificar se a diferença de F1, com ou sem o pós-processamento do ICNDE utilizando o Trinity e utilizando o Weir como extratores, é significativamente relevante. Ambos os métodos de extração através do teste t tiveram valor de p menor que α, que vale 0,05 (no Trinity $p = 0,02$ e no Weir $p = 0,012$), assim, o pós-processamento teve valores estatisticamente relevantes. Logo, o pós-processamento do ICNDE teve uma diferença significante em seus resultados, visto que o pós-processamento de prefixos, sufixos ou outros valores considerados ruídos são eliminados. A partir da eliminação do ruído os valores são mais próximos do gabarito.

A Figura 5.3 mostra que em quatro domínios houve um ganho de F1 principalmente para o domínio de $Livros$ que teve um ponto percentual de aproximadamente 10%. Em domínios como $Carros$ e $Filmes$ não apresentam nenhuma diferença. Já nos domínios *Jogadores de NBA* e *Câmeras* existe uma perda que não supera a 2%.

A Figura 5.4 mostra que em seis domínios houve um ganho de F1, principalmente para o domínio de $Livros$, o qual, novamente obteve o maior ponto percentual de valor aproximadamente 10%. Já nos domínios *Câmeras* e $Filmes$ existe uma perda de F1 que não supera 1%.

Figura 5.3: F1 antes e depois do ICNDE no Trinity.

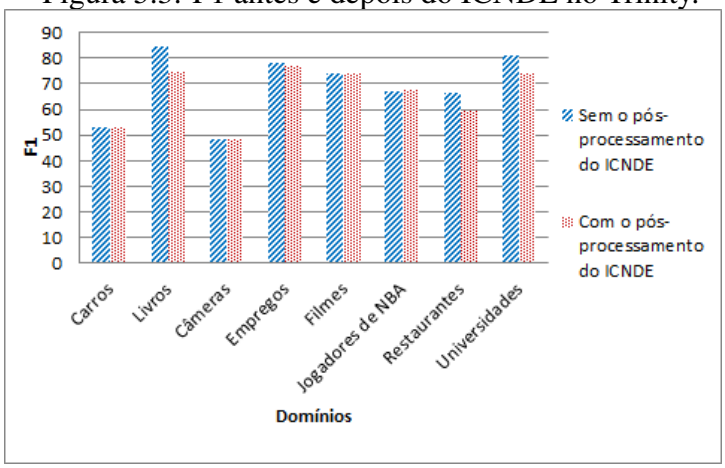
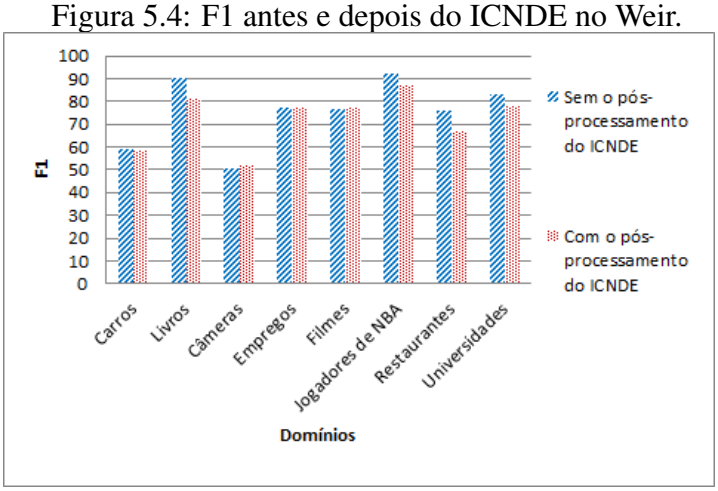


Figura 5.4: F1 antes e depois do ICNDE no Weir.



Além desses resultados, foi avaliado o F1 de cada site para verificar se a diferença de F1, antes e depois do Trinity e do Weir é significativamente relevante. Ambos os método tiveram valor de diferença estatisticamente relevante menor que $0,05$ (no Trinity $0,02$ e no Weir $0,012$). Logo, ICNDE teve uma diferença significante em seus resultados.

5.2.3 Análise de tipos de erros

Este experimento apresenta o efeito do ICNDE sobre os tipos de erros apresentados nas Seções 3.2.2 e 3.2.3. Foi medido o total de erros para o Trinity e o Weir com ou sem o uso do pós-processamento do ICNDE. No restante dos experimentos relacionados a erros, tem-se como base percentual dos erros de $contains$ sobre o total de dados do domínio.

A Figura 5.5 apresenta o total de erros dos métodos Weir e Trinity com ou sem o uso do pós-processamento do ICNDE. No Weir, pode-se observar que existe uma perda próxima a 30000 erros. Já no Trinity, existe uma perda de 20000 erros no total. Isso demonstra que o pós-processamento do ICNDE cumpre o objetivo de diminuir erros. A proporção de erros de $contains$ é analisada, visto que o principal objetivo do ICNDE é eliminar os erros de $contains$ ocorridos na extração de dados.

Na Figura 5.6, os domínios de $Livros$, *Jogadores de NBA* e $Restaurantes$ tiveram queda no percentual de erros de $contains$ utilizando o pós-processamento do ICNDE. O maior percentual no queda de total de erros de $contains$ utilizando Trinity como extrator está no domínio de $Livros$. Os domínios de $Carros$, *Câmeras* e $Filmes$ não tiveram alteração em seu percentual. Porém, em $Empregos$ e $Universidades$ existe um ganho no percentual de erros de $contains$.

Figura 5.5: Total de erros com e sem o pós-processamento do ICNDE.

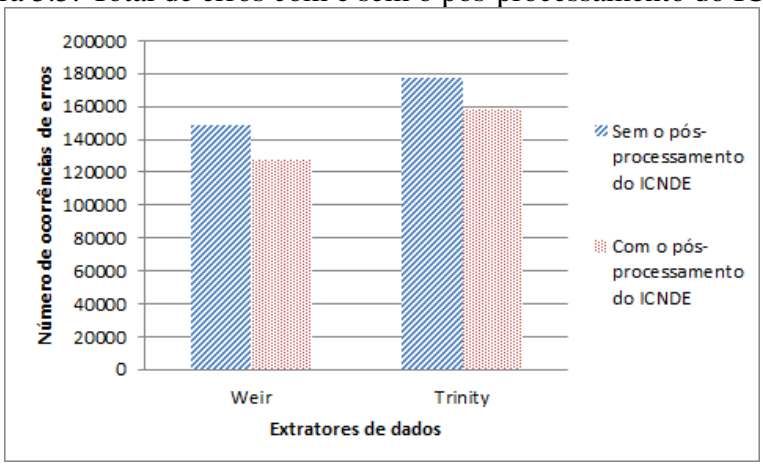


Figura 5.6: Percentual de erros de *contains* do ICNDE utilizando o Trinity.

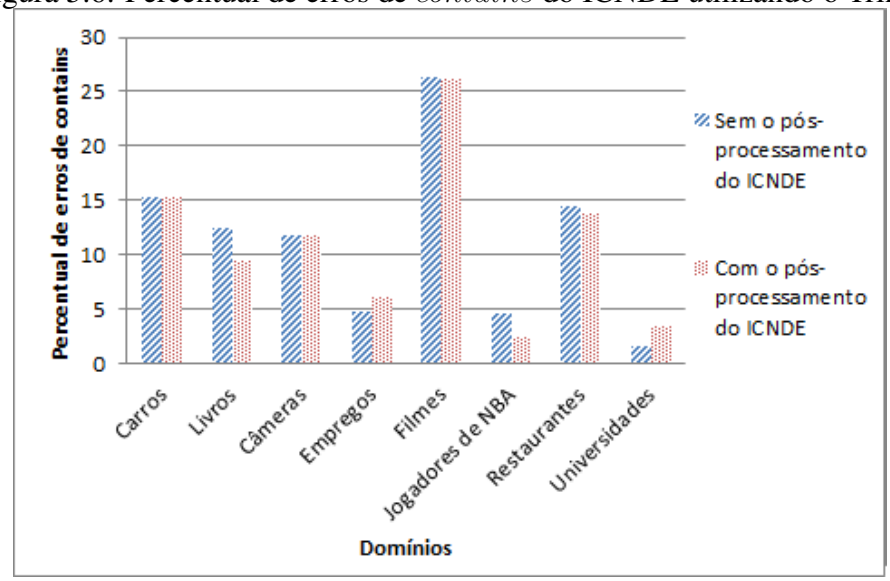
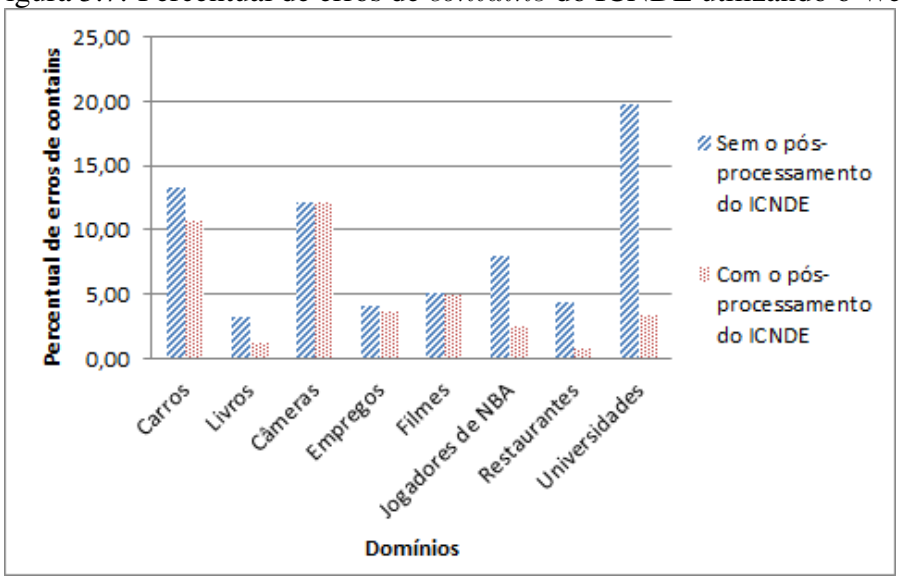


Figura 5.7: Percentual de erros de *contains* do ICNDE utilizando o Weir.

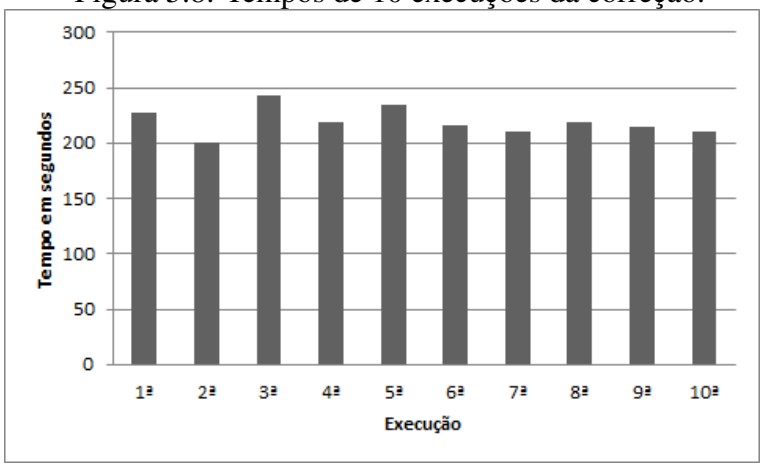


A Figura 5.7 apresenta que em seis domínios houve uma queda no percentual de erros de $contains$ utilizando o pós-processamento do ICNDE. A proporção de erros de $contains$ é massiva nos domínios que apresentaram maior proporção antes de executar o ICNDE. O Weir apresentou maior queda de proporção em $Universidades$. Os domínios *Câmeras* e $Filmes$ não tiveram alteração em seu percentual.

5.2.4 Tempo de execução

Outro fator importante para o experimento é a definição do desempenho do método. Neste caso, a etapa de pós-processamento do método proposto apresenta um medidor em segundos para cada vez que o pós-processamento ICNDE é executado. Para medir o tempo foram realizadas 10 execuções e os seus respectivos tempos estão exibidos na Figura 5.8. Em média, a etapa de pós-processamento do ICNDE leva 3,64 minutos. No máximo 5 iterações foram necessárias para o algoritmo convergir

Figura 5.8: Tempos de 10 execuções da correção.



5.2.5 Casos de Falha

O experimento apresentou alguns casos de falha em que houve perda ao invés de ganhos em F1. Esses casos acontecem quando um $token$ não classificado como ruído é classificado como tal. Desta forma, é possível incluir erros ao invés de elimina-los.

Valores que não deveriam ser anotados. Este caso ocorreu quando um conjunto de atributos tem um padrão similar a outro. Por exemplo, em *Câmeras* existe o atributo $modelo$ que segue um padrão de definição. Em alguns sites o padrão do $modelo$ tem

um trecho que é similar a um padrão de endereço. Por exemplo, $SD, Canon$ possui um padrão similar a expressão regular que identifica endereços. Nesse caso, o valor do atributo $modelo$ foi perdido em função do valor ser marcado de maneira errada. A solução para esse problema é restringir as marcações para cada domínio, ou seja, *endereço* é utilizado em $Restaurantes$, mas não em *Câmeras*. Isso significa que o padrão definido para as anotações de um domínio é influenciado por outro. Se cada domínio tem o seu conjunto próprio de anotações definido esse erro não ocorreria.

Valores não anotados no experimento. Este caso ocorre quando um valor de um atributo possui caracteres especiais que não são considerados como ruído e o valor não é considerado como uma anotação. No experimento, não foram considerados os valores de $altura$ para serem marcados. Este caso ocorreu no domínio de *Jogadores de NBA* onde a $altura$ apresentava valores como caracteres especiais em seus valores, esse valor acaba sendo segmentado na tokenização. Por exemplo, $6'7"$ é um valor considerado correto, porém por não ser marcado é divido por 6 e 7. Para corrigir esse caso de falha, é necessário que o padrão de alturas seja adicionado para ser marcado como anotações.

Uma única regra extrair valores de diferentes atributos de forma concatenada. Este caso ocorre quando é extraído mais de um nodo textual para a mesma regra. Esse caso ocorreu no Trinity - porque ele extrai diversos nodos textuais em um atributo - no domínio *Jogadores de NBA* no atributo $peso$. Nesse caso, dois nodos foram extraídos juntos e após a filtragem de $tags$ os valores de diferentes nodos textuais estão unidos como um único valor. Por exemplo, $220\ Lakers$, o peso do jogador e o time que ele pertence foram interpretados como peso apenas. Cada site de um domínio tem um padrão diferente então nem sempre são os mesmos tipos de valores podem ocorrer juntos no mesmo atributo. Então, uma maneira de eliminar esse caso de falha seria comparar os tipos de regras que extrai os mesmo valores de outros sites e avaliar qual é o tipo mais comum para essa regra e eliminar o ruído daquele atributo, se baseando nos sites que extraem os mesmos valores de atributos.

Como o caso do experimento utilizou anotações apenas para links, endereços, telefones e datas para todos os domínios, isso acarretou alguns casos de falha. Esses casos podem ser corrigidos se cada domínio executado tenha o seu conjunto de anotações já definidas. Assim, é possível avaliar de maneira mais precisa os $tokens$ para retirar qualquer ruído. Outros casos que ocorrem podem ser corrigidos através de uma segmentação mais precisa do $token$.

6 CONCLUSÃO

Esse trabalho apresentou os métodos de extração de dados e analisou casos de falha de duas diferentes abordagens. Tais análises concluíram na elaboração do método ICNDE, para a filtragem de ruídos extraídos junto com valores de atributos pelos métodos de extração de dados. O ICNDE foi analisado utilizando um extrator de dados baseado em árvore (Trinity) e outro baseado em XPath (Weir).

O método ICNDE tem como objetivo utilizar um extrator de dados e eliminar os ruídos presentes em sua saída, o que é realizada em uma etapa adicional de pós-processame- nto. O método utiliza os limiares Θ e Φ como parâmetros para a classificação de ruídos. O Θ é um limiar para o percentual do número de ocorrências de um $token$ em um site. O Φ é um limiar para o percentual do número de ocorrências de um $token$ no domínio. O pós-processamento do método ICNDE realiza uma filtragem de $tags$ e de terminadores que são encontrados nos extremos dos valores. A partir das instâncias filtradas, são anotados os dados que seguem um padrão como dados atômicos. Havendo na sequência a tokenização, que divide a instância corrente a partir de símbolos, expressões regulares e valores aprendidos como ruído. Após a tokenização, os $tokens$ são avaliados através do seu número de ocorrências e, a partir delas, pode-se definir se o $token$ é ruído ou não. Todo ruído encontrado é eliminado da entrada e, por fim, o método retorna como saída as regras geradas pelo extrator, sem o ruído.

Após a definição do método, foi realizado um experimento utilizando novamente o Trinity e o Weir como $baselines$ para avaliar a eficácia do método ICNDE. Os $baselines$ foram testados com e sem o a etapa de pós-processamento do ICNDE. Através do experimento, foram definidos os melhores limiares Θ e Φ para o pós-processamento. Logo após, foi analisado o ganho do pós-processamento em F1 para ambos os métodos de extração de dados. O método apresentou um ganho em F1 de 13% no Weir e ganho em F1 de 11% no Trinity, esta diferença é estatisticamente significativa. Após a avaliação de eficácia, foram analisados os erros totais com e sem o uso do pós-processamento do ICNDE, onde constatou-se uma queda no número de ocorrência de erros com o uso do pós-processamento. Os erros de $contains$, os quais eram a motivação do método ICNDE, foram avaliados e, em ambos os casos, houve uma queda no percentual desses erros quando se emprega o pós-processamento do método. Além dos tipos de erros, o tempo de execução do experimento foi analisado, o qual teve em média 3 minutos de execução.

O ICNDE mostrou um ganho relevante de eficácia durante o seu experimento e conta atualmente com a capacidade de identificar e corrigir ruídos. A eliminação de ruído torna os valores extraídos mais adequados para a utilização em qualquer contexto de aplicação. O que torna possível, por exemplo, criar um banco de dados que armazena entidades de um determinado domínio e, através desse banco, realizar diferentes tipos de consultas referentes a essa entidade.

Os trabalhos futuros com relação ao ICNDE são: (i) melhorar a forma de segmentação; (ii) utilizar uma comparação entre sites e definir através das regras geradas um padrão para todas as regras que extraem os mesmos tipos de valores nos mesmo sites; (iii) testar o método com ferramentas de anotação; (iv) comparar o uso de ferramentas de anotação com uso de apenas expressões regulares.

REFERÊNCIAS

BLANCO, L. et al. Supporting the automatic construction of entity aware search engines. In: **Proceedings of the 10th ACM Workshop on Web Information and Data Management**. New York, NY, USA: ACM, 2008. (WIDM '08), p. 149–156. ISBN 978-1-60558-260-3. Available from Internet: <http://doi.acm.org/10.1145/1458502.1458526>.

BRONZI, M. et al. Extraction and integration of partially overlapping web sources. **Proc. VLDB Endow.**, VLDB Endowment, v. 6, n. 10, p. 805–816, aug. 2013. ISSN 2150-8097. Available from Internet: <http://dx.doi.org/10.14778/2536206.2536209>.

CUNNINGHAM, H. Gate, a general architecture for text engineering. **Computers and the Humanities**, Springer, v. 36, n. 2, p. 223–254, 2002. ISSN 00104817. Available from Internet: <http://www.jstor.org/stable/30204529>.

DALVI, N. et al. A web of concepts. In: **Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems**. New York, NY, USA: ACM, 2009. (PODS '09), p. 1–12. ISBN 978-1-60558-553-6. Available from Internet: <http://doi.acm.org/10.1145/1559795.1559797>.

GIBSON, D.; PUNERA, K.; TOMKINS, A. The volume and evolution of web page templates. In: **Special Interest Tracks and Posters of the 14th International Conference on World Wide Web**. New York, NY, USA: ACM, 2005. (WWW '05), p. 830–839. ISBN 1-59593-051-5. Available from Internet: <http://doi.acm.org/10.1145/1062745.1062763>.

GULHANE, P. et al. Exploiting content redundancy for web information extraction. In: **VLDB Endowment**. [S.l.]: Journal Proceedings of the VLDB Endowment, 2010. v. 3, n. 6, p. 578–587.

LI, X.; ZHU, Y.; YIN, G. Exploiting attribute redundancy in extracting open source forge websites. In: **Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)**. [S.l.]: IEEE, 2012.

MANICA, E.; DORNELES, C. F.; GALANTE, R. R-extractor: an extended method for data extraction from template-based entity-pages. In: **DASFAA**. [S.l.: s.n.], submetido para publicação 2017.

ORTONA, S. et al. Wadar: Joint wrapper and data repair. **Proc. VLDB Endow.**, VLDB Endowment, v. 8, n. 12, p. 1996–1999, aug. 2015. ISSN 2150-8097. Available from Internet: <http://dx.doi.org/10.14778/2824032.2824120>.

SLEIMAN, H. A.; CORCHUELO, R. Trinity: On using trinary trees for unsupervised web data extraction. **IEEE Transactions on Knowledge and Data Engineering**, v. 26, n. 6, p. 1544–1556, June 2014. ISSN 1041-4347.

STENETORP, P. et al. Brat: A web-based tool for nlp-assisted text annotation. In: **Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics**. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012. (EACL '12), p. 102–107. Available from Internet: <http://dl.acm.org/citation.cfm?id=2380921.2380942>.

ZHU, Y. et al. Exploiting attribute redundancy for web entity data extraction. In: ____. **Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation: 13th International Conference on Asia-Pacific Digital Libraries, ICADL 2011, Beijing, China, October 24-27, 2011. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 98–107. ISBN 978-3-642-24826-9. Available from Internet: <http://dx.doi.org/10.1007/978-3-642-24826-9_15>.