Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Extraindo Estrutura de Entidades na Web com Pouca Supervisão

Lucas Almeida Pereira de Lima

Trabalho de Graduação

Recife
16 de dezembro de 2016

Universidade Federal de Pernambuco
Centro de Informática

Lucas Almeida Pereira de Lima

Extraindo Estrutura de Entidades na Web com Pouca Supervisão

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Luciano de Andrade Barbosa*

Recife
16 de dezembro de 2016

Agradecimentos

Gostaria de agradecer em primeiro lugar a Deus, por me fortalecer e me guiar em cada momento da vida. A Ele toda honra e toda glória.

Ao professor Luciano Barbosa, pela oportunidade e pela orientação que tornou esse trabalho possível.

Aos demais professores com quem tive a oportunidade de aprender durante a graduação. Especialmente a Pedro Manhães, Kátia Guimarães e Paulo Gustavo Fonseca, que me proporcionaram um grande crescimento acadêmico e pessoal.

Aos amigos que fiz na faculdade, pela companhia constante e por todas memórias compartilhadas nessa caminhada ao longo desses 5 anos. Em especial a Bertha, Duhan, Guilherme, Mateus e Vinícius.

Aos integrantes do time da Maratona de Programação, projeto que me abriu tantas oportunidades, por me motivarem e me darem tanta alegria e satisfação. Em especial às treinadoras Liliane Salgado e Kátia Guimarães e aos meu companheiros de time Gustavo Stor, Duhan e Mário.

A todos aqueles que fazem parte do Centro de Informática da Universidade Federal de Pernambuco, e o fazem ser um centro de excelência no Brasil.

Aos meus pais, Alexandre e Gedália, por todo apoio, suporte, amor e carinho que sempre me deram, e por terem me proporcionado uma educação de qualidade.

Resumo

A World Wide Web contém uma quantidade vasta de dados de variados domínios, como imóveis, carros e produtos. Uma parte desses dados representa entidades estruturadas, com seus atributos e respectivos valores. A obtenção desses dados pode ser útil, por exemplo, para melhoria dos resultados em engenhos de busca e tomada de decisão apoiada em dados. Uma etapa importante na obtenção desses dados é sua extração a partir de páginas HTML. Infelizmente, a maior parte desses dados não está pronta para consumo automático, tornando a difícil a sua extração. Nesse trabalho, apresentamos REX (Real Estate eXtractor), um extrator que com pouca supervisão é capaz de extrair dados estruturados de imóveis em páginas web. REX inicia o processo de extração a partir da detecção de nós contendo atributos base utilizando conhecimento de domínio. A partir desses nós, detecta regiões contendo informações relevantes. Após a identificação dos atributos presentes na página, REX segmenta os textos das regiões encontradas e extrai os pares atributo-valor. Em avaliações experimentais realizadas utilizando mais de 60 sites de imóveis, REX obteve um bom desempenho, alcançando valores altos de precisão e revocação para a maioria dos sites.

Palavras-chave: recuperação de informação, extração de dados, extração de entidades, dados estruturados, coleta de dados, web

Abstract

The World Wide Web contains a vast amount of data from a diverse set of domains, e.g., real estate, cars and products. A significant amount of this data exhibits a structure representing entities and its attributes and respective values. Data of structured entities can be used, for instance, to improve search engine results, and for data-driven analytics. An important step in acquiring this data is its extraction from HTML pages. Unfortunately, most of this data is not ready for automatic consumption, making its extraction challenging. In this work, we present REX (Real Estate eXtractor), a semi-supervised extractor capable of extracting structured data from entities in the real estate domain. REX first detects nodes that contain important attributes using domain knowledge. Using these nodes, it detects regions that contain relevant information. After the identification of the attributes that appear in the page, REX segments the text found in relevant regions and extract the attribute-value pairs. Experimental evaluations conducted using over 60 sites in the real estate domain show that REX is effective, achieving high values of precision and recall for most of the sites.

Keywords: information retrieval, data extraction, entity extraction, structure data, data collection, web

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A World Wide Web é uma grande fonte de dados estruturados [1]. Grande parte desses dados encontram-se em páginas contendo pares atributo-valor acerca de alguma entidade [2]. Exemplos de dados estruturados são as caixas de informação da Wikipédia (Figura 1.1a) e páginas contendo especificações de produtos (Figura 1.1b). Há interesse considerável em extrair esses dados e construir uma "base de dados estruturados", que possam ser utilizados para diversas aplicações, como na melhoria de engenhos de busca, integração de dados e respostas a perguntas [3]. Por exemplo, o Google utiliza o Mapa do Conhecimento[1] para possibilitar buscas como "Qual a altura da Torre Eiffel?" (Figura 1.2a) e possui ferramentas como o Google Voos, que integra informações a respeito de voos de companhias aéreas ao redor do mundo (Figura 1.2b). Além de recuperação de informação, dados estruturados também podem ser úteis em outras áreas, como visualização de dados e aprendizagem de máquina.

Para poder se fazer uso desses dados, algumas etapas são necessárias, como coleta e extração dos dados. A dificuldade da tarefa de extração de dados depende da forma como os dados se apresentam. Quando há uma estrutura bem definida, como em uma base de dados estruturados, a extração pode ser feita de forma simples e precisa. Por outro lado, em casos como na Figura 1.3, onde a estrutura encontra-se implícita no texto, a extração é muito difícil. Infelizmente, grande parte da informação da Web está apresentada de forma acessível apenas para o usuário humano [4], utilizando HTML para definir a apresentação de objetos como listas e tabelas exibidos na página. Para realizar a extração de dados semi-estruturados, portanto, é preciso utilizar a hierarquia da árvore HTML da página para identificar regiões representando

[1] https://www.google.com/intl/pt-BR_ALL/insidesearch/features/search/knowledge.html

UFPE	
Lema	Virtus impavida *(virtude impávida)*[1]
Fundação	11 de agosto de 1827 (189 anos)
Tipo de instituição	Pública
Mantenedora	Ministério da Educação
Localização	Recife, Pernambuco
Docentes	2 834 [2]
Reitor(a)	Prof. Anísio Brasileiro de Freitas Dourado [3]

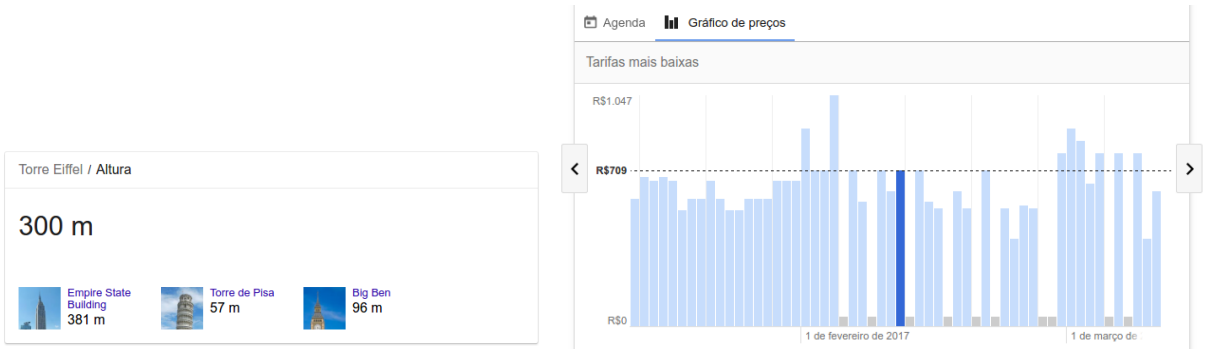
(a) Informações sobre a UFPE na Wikipédia

informações técnicas

Código	120853315
Marca	CD Projekt Red
Título	The Witcher 3: Wild Hunt
Plataforma	PS4
Gênero	RPG / Ação

(b) Informações sobre jogo Witcher 3 no site americanas.com

Figura 1.1: Exemplo de fontes de dados estruturados na web.



(a) Mapa de Conhecimento (b) Google Voos

Figura 1.2: Exemplo de utilização de dados estruturados no Google.

Casa padrão, situado no Jardim Petrolar.
A casa possui 2/4, 1 sala de estar, 1 cozinha, 1 banheiro, quintal.
Não pode ser Financiada.
6,70 metros de frente por 27 comprimento
total de 180,09m² área construída 45m² casa ainda inacabada

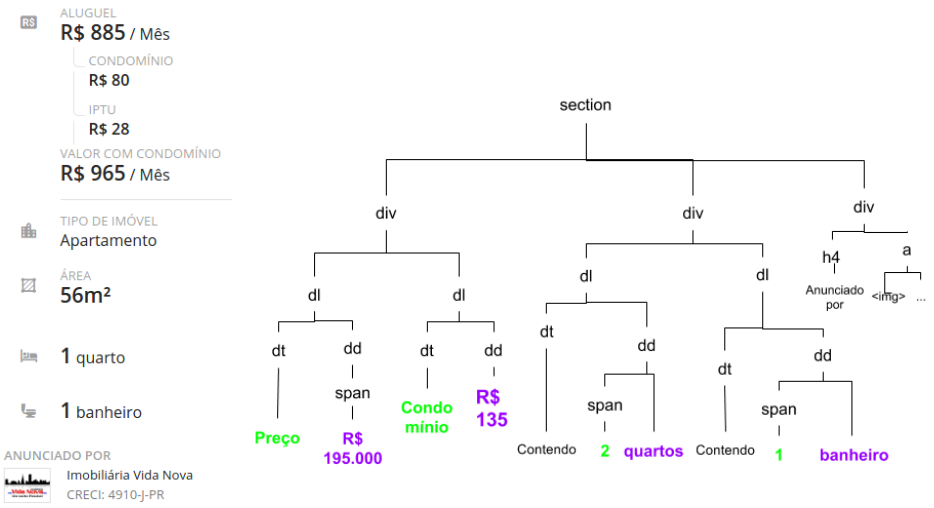
Figura 1.3: Exemplo de página contendo informação latente.

esses dados.

Muitas vezes, a informação exibida em sites é gerada automaticamente através de programas que transformam a informação disponível em uma base de dados em uma página HTML. Com isso, para um mesmo site, é comum que haja um *template*, i.e, uma certa estrutura regular que define a forma como a informação é exibida na página [5]. Na Figura 1.4, mostramos um exemplo de informação exibida no site de imóveis Viva Real, sua representação em forma de árvore HTML e os pares que desejam ser extraídos.

Uma abordagem simples para extração de dados semi-estruturados consiste em analisar páginas de um dado site para identificar características específicas do *template* utilizado por esse site, como caminho na árvore DOM e atributos do HTML, e utilizá-las para extração. Essa estratégia é capaz de alcançar uma boa acurácia, porém precisa de um grande esforço manual para repetir o processo em muitos sites, além do custo de manter os extratores atualizados quando ocorrem mudanças na estrutura. Algumas propostas de extração de dados semi-estruturados [6, 7] atuam de forma supervisionada, onde para a indução automática de um extrator o usuário precisa marcar atributos a serem extraídos em páginas de exemplo. Já outras soluções [8, 9], realizam a indução de extratores através da busca por repetições e regularidades na estrutura HTML utilizando diversas páginas de exemplos ou em páginas com a presença de várias entidades a serem extraídas. Algumas soluções optam por focar em estruturas específicas como tabelas e listas [10, 4] e outras utilizam conhecimento de domínio para guiar a detecção de regiões contendo informação relevante [11, 12, 13]. Uma solução independente de domínio e totalmente automática é o grande objetivo da extração de informação na web [4].

Nesse trabalho, buscamos criar para o domínio de imóveis um extrator único para extrair entidades estruturadas de vários sites, que atue com pouca supervisão humana. O grande desa-



(a) Informação visível para usuário do site

(b) Parte da árvore HTML

Figura 1.4: Exemplo de informação representada de forma semi-estruturada. A estrutura HTML define como a informação deve ser exibida para o usuário. Em verde, os atributos que devem ser extraídos, casados com os valores (em roxo). Os nós com *tag* `dl` e texto "Contendo" estão escondidos.

fio de desempenhar essa tarefa decorre da grande variedade na forma de exibição de conteúdo encontrada entre sites. Embora o foco do trabalho seja na área de imóveis, procuramos utilizar estratégias que possam ser adaptadas de forma simples para outros domínios.

Mais especificamente, nosso objetivo é extrair pares atributo-valor de uma página contendo um imóvel. É comum que uma página de entidade possua diversos atributos, porém estamos interessados nos atributos mais comuns, que aparecem em regiões de destaque na página, como nos exemplos exibidos na Figura 1.5. Essas regiões se assemelham a listas ou tabelas, porém não assumimos a utilização *tags* específicas no HTML.

Para o domínio de imóveis, as regiões de destaque normalmente contêm atributos como número de quartos, área e número de banheiros. Além desses atributos mais comuns do domínio,

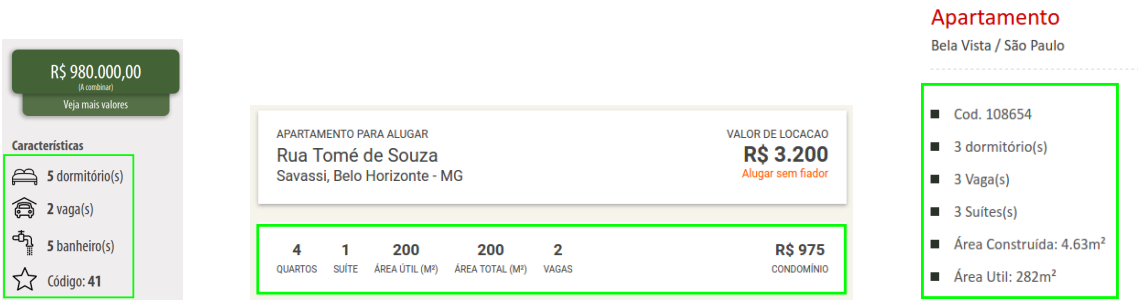


Figura 1.5: Exemplos de regiões de destaque (em verde) em páginas de imóveis.

Figura 1.6: Regiões com atributos relevantes e não relevantes. As regiões contendo atributos relevantes estão em verde, as regiões contendo atributos que são considerados não relevantes em vermelho.

também são considerados relevantes outros atributos que estejam presentes na mesma região. É importante notar que optamos por não focar na extração dos atributos preço e localização. Embora esses atributos sejam importantes, eles normalmente são exibidos de forma diferenciada. Por isso, os consideramos relevantes apenas quando aparecem na mesma região que destaca os atributos mais comuns. A Figura 1.6 mostra um exemplo da diferenciação entre atributos relevantes e não relevantes.

Para guiar o processo de extração, optamos por utilizar conhecimento de domínio, porém buscamos reduzir o trabalho manual em relação a outros trabalhos como [11] e [13]. Também escolhemos trabalhar com páginas de entidade, contendo informação para um imóvel principal. Ao contrário de outras soluções como Roadrunner [8], não utilizamos diversos exemplos de página com o mesmo *template*.

O restante deste trabalho está organizado da seguinte maneira. O Capítulo 2 mostra conceitos fundamentais e trabalhos relacionados na área de extração de informação. No Capítulo 3 detalhamos REX (Real Estate eXtractor), a abordagem desenvolvida. O Capítulo 4 mostra os resultados e a análise da avaliação experimental realizada. Por fim, no Capítulo 5 nós concluímos o trabalho.

CAPÍTULO 2

Fundamentos

Neste capítulo abordamos conceitos básicos relacionados à área do projeto, trabalhos relacionados e definidos o foco do trabalho.

2.1 Conceitos Básicos

2.1.1 HTML

O *layout* de uma página web é descrito utilizando HTML (HyperText Markup Language), que é uma linguagem de marcação que pode ser interpretada por navegadores. Um documento HTML é composto por elementos, onde cada elemento é composto por uma *tag* (comando de formatação na linguagem), atributos, valores e filhos (que podem ser outros elementos ou texto). Por exemplo, `<a href=”www.google.com.br”> Google </a>` representa um documento com *tag* a, atributo href, valor "google.com.br"para esse atributo, e texto Google. A relação entre os elementos é hierárquica, de forma que é possível representar uma página web através da árvore DOM (Document Object Model), que contém um nó raiz representando o documento, além dos elementos, atributos e textos.

Seletores: Em muitas aplicações, é interessante selecionar todos os nós em uma região do documento HTML que obedecem um determinado padrão. Utilizando CSS (Cascading Style Sheets) para definir a aparência de itens HTML, é possível definir um seletor (chamado de *CSS Selector*) e escolher o valor de uma propriedade (cor, tamanho, margem etc) para todos os nós que respeitam o padrão definido por aquele seletor. É possível criar seletores baseados em sequência de *tags*, atributos, posição em relação ao pai e várias outros critérios. Por exemplo, um seletor `div > ul.atributos > li:nth-child(1)`, seleciona os elementos com *tag* `li`, que são o primeiro filho do seu pai, que por sua vez possui *tag* `ul` e atributo *class* contendo “atributos” entre seus valores e são filhos de elementos com *tag* `div`. Outro tipo de seletor comumente utilizado é o XPath (XML Path Language), criado para seleção de nós em documentos XML mas que também pode ser utilizado para selecionar nós em árvores DOM.

2.1.2 Entidades Estruturadas na Web

Uma entidade estruturada é um objeto que contém atributos e valores associados a esses atributos. Por exemplo, um jogador de beisebol possui atributos como clube atual, altura e data de nascimento, e valores associados a esses atributos. Os atributos de entidades em um determinado domínio representam o esquema de dados. Esses esquemas descrevem a estrutura de modo formal, definindo tabelas, campos, relacionamentos etc. Com isso, é possível consultar

diversos tipos de informação de forma precisa utilizando linguagens como SQL (*Structured Query Language*).

Na web, não é possível ter acesso ao esquema de dados utilizado pelo provedor da informação. O usuário tem acesso à informação representada da forma escolhida pelo desenvolvedor do site, de forma que a dificuldade de extrair os dados depende da representação utilizada.

Representações estruturadas: Em casos onde o site deseja prover a informação, normalmente utiliza-se uma linguagem de descrição de conteúdo como XML (eXtensible Markup Language) ou JSON (JavaScript Object Notation). Por exemplo, a Distance Matrix API do Google Maps[1] é um serviço que fornece informações como distância e tempo de viagem para diversos pares origem-destino. O usuário pode consultar a informação realizando requisições HTTP e recebendo um JSON ou um XML contendo as informações desejadas. Na documentação do serviço, são definidos os campos e como eles devem ser acessados.

Representações não-estruturadas: Os casos mais complicados para extração de entidades estruturadas ocorrem quando a informação está disponível apenas em texto, normalmente escrito por um humano. Nesse caso, a estrutura está latente, representada de forma implícita, e é preciso lidar com uma enorme variabilidade na forma de como as informações são colocadas no texto, além de ser capaz de diferenciar os textos que representam as entidades dos textos comuns.

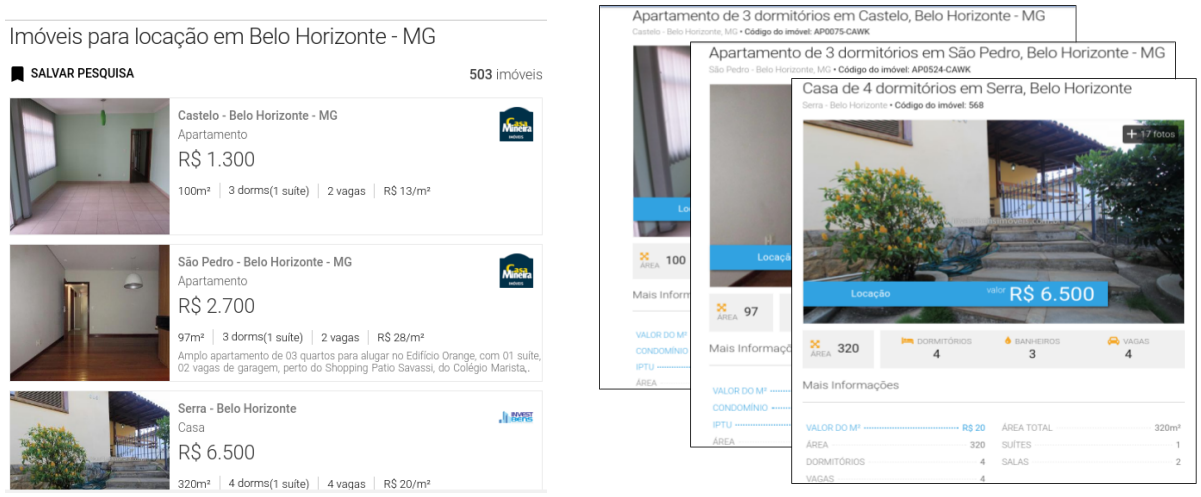
Representações semi-estruturadas: A maior parte da informação disponível na web está representada um meio-termo entre as duas outras representações comentadas. O W3C (World Wide Web Consortium), organização responsável pelos padrões da web, diz que o DOM não define qual informação na página é relevante nem como o documento está estruturado[2]. Porém, dois fatores que ocorrem com frequência contribuem para que a extração seja mais simples que em texto livre: (i) a informação é exibida para o usuário com alguma estrutura, por exemplo, de forma que se assemelha a uma tabela ou uma lista contendo os pares atributo-valor; (ii) a página HTML é gerada a partir de um *template*, que define os elementos da página e sua aparência, e é preenchido utilizando dados provenientes de uma base de dados. Sendo assim, é comum que haja uma regularidade entre atributos em uma mesma região da página e entre páginas de um mesmo site, que podem ser exploradas para identificação de entidades e extração de pares atributo-valor.

2.2 Extração Automática de Entidades Estruturadas

Nessa seção abordamos soluções desenvolvidas por trabalhos anteriores para detecção e extração automática de entidades estruturadas representadas em páginas HTML. Primeiramente apresentamos os tipos de página em que entidades estruturadas são exibidas. Depois mostramos uma forma para classificação de detectores, de acordo com seu grau de supervisão. Por último, apresentamos alguns trabalhos anteriores realizados na área.

[1] https://developers.google.com/maps/documentation/distance-matrix/intro?hl=pt-br

[2] https://www.w3.org/TR/DOM-Level-2-Core/introduction.html



(a) Página de lista (b) Página de entidade

Figura 2.1: Exemplos de página de lista e página de entidade.

2.2.1 Tipos de Páginas com Entidades Estruturadas

A informação estruturada de entidades pode estar presente em dois tipos de página: páginas de lista e páginas de entidade.

Páginas de lista, também chamadas de páginas de índice, são páginas contendo diversas entidades estruturadas, organizadas de forma semelhante a uma lista ou a um grid, como na Figura 2.1a. Essas páginas normalmente contêm os resultados de buscas feitas no site, ou são acessíveis através de um link de entrada. Alguns trabalhos [9, 11] focam nessas páginas, em busca de aproveitar a regularidade encontrada entre as diferentes entidades encontradas em uma mesma página. Parte deles foca na apenas na identificação da região contendo cada entidade, enquanto outros também realizam a extração da informação estruturada para cada entidade.

Páginas de entidades, também chamadas de páginas de detalhe, são páginas que contêm conteúdo para apenas uma entidade, como na Figura 2.1b. A vantagem de extrair informações dessa página é a presença de uma quantidade maior de informações, já que nas páginas de lista o espaço é reduzido e apenas as informações principais de cada entidade são exibidas. Os trabalhos [8, 14] que utilizam esse tipo de página normalmente utilizam diversos exemplos de páginas similares para aprender a estrutura utilizada no site.

2.2.2 Grau de Supervisão dos Extratores

Podemos dividir os extratores em categorias de acordo com o grau de supervisão e generalidade [15].

Extratores específicos: Extratores específicos são extratores focados em um tipo específico de *template*, normalmente para um único site. Eles são construídos observando manualmente páginas de exemplo para aquele site, e observando características que possam ser para extração utilizando seletor de elementos HTML ou expressões regulares. Uma possibilidade é selecionar

diretamente os atributos, e a partir deles navegar ao valor correspondente. Outra possibilidade é encontrar algum nó que seja ancestral de ambos atributo e valor, e a partir dele navegar até os nós desejados.

Navegadores web como Google Chrome, Mozilla Firefox e Opera possuem ferramentas para inspecionar o DOM da página, e identificar o seletor para um nó da árvore. É sempre possível selecionar um único nó já que o caminho entre a raiz e o nó (utilizando a posição em relação ao pai) é único. No caso de entidades estruturadas, isso não é suficiente já que são vários pares atributos-valor que precisam ser encontrados. Porém, normalmente é possível utilizar uma quantidade pequena de seletores para selecionar os elementos desejados, devido à homogeneidade encontrada. Por exemplo, todos elementos contendo atributos podem ter o mesmo valor para o atributo HTML *class*. Outra possibilidade comum é navegar até a raiz da região contendo os pares, e de lá navegar até atributos e valores. Extratores específicos são normalmente simples de serem criados, porém não são escaláveis, já que necessitam de trabalho manual para cada site cuja informação deve ser extraída e é preciso atualizar os extratores quando a estrutura do site muda.

Extratores supervisionados: Extratores supervisionados, como LiXto [6] e WiEN [7], recebem como entrada exemplos de páginas rotuladas com a informação a ser extraída, e têm como objetivo induzir automaticamente uma regra de extração que possa ser utilizada para extrair a informação desejada. O sistema pode sugerir (possivelmente utilizando uma GUI) páginas adicionais a serem rotuladas pelo usuário. A vantagem dessa abordagem é que usuários leigos (sem conhecimento de programação) podem realizar a extração se treinados para utilizar a GUI. Porém, esse tipo de extração apresenta problemas de escalabilidade, já que ainda é preciso intervenção humana.

Outro tipo de extração supervisionada, porém com menos esforço humano, consiste em utilizar técnicas de aprendizagem de máquina para classificar páginas e regiões de uma página, a fim de detectar regiões relevantes, como no DEXTER [10]. Isso é uma abordagem factível quando a maioria das informações está exibida de forma parecida para diversos sites, como em tabelas e listas HTML. Embora seja necessário o trabalho manual de rotulação para classificação, uma vez criado o classificador é possível operar em larga escala.

Extratores semi-supervisionados: Extratores semi-supervisionados são extratores que ainda utilizam algum tipo de entrada humana, porém de forma mais simples que em extratores supervisionados. Por exemplo, IEPAD [16] utiliza a regularidade encontrada em páginas de lista para extração em páginas não-rotuladas, mas precisa que um humano especifique qual a informação relevante para os padrões encontrados. Outras soluções utilizam algum conhecimento de domínio que é configurado pelo usuário [11, 12, 13], como ontologias ou padrões textuais relevantes que podem ser utilizados para facilitar a detecção e extração das entidades.

Extratores não-supervisionados: Extratores não-supervisionados atuam diretamente em páginas não rotuladas e buscam detectar regiões similares utilizando casamentos de padrão [8] ou similaridade entre árvores [9]. Essas técnicas atuam em diversas páginas geradas pelo mesmo *template*, ou em uma página de lista contendo múltiplas entidades, e normalmente são custosas computacionalmente e normalmente não possuem um desempenho muito eficaz.

2.2.3 Trabalhos Relacionados

Nessa seção apresentamos alguns trabalhos relacionados, com um breve resumo da abordagem adotada e o foco do trabalho.

MDR [9] é uma solução não-supervisionada como foco na detecção de entidades em páginas de lista, utilizando semelhança na árvore DOM. Uma extensão, NET [17], lida com o problema de entidades aninhadas, através de um algoritmo *bottom-up* usando distância de edição em árvores. Em ambos os casos, a solução não é capaz de realizar o casamento atributo-valor.

Lerman et al. [4] apresenta uma solução semi-supervisionada combinando a informação presente em páginas de lista e páginas de entidade. Assumindo que páginas de lista e páginas de entidade são visões diferentes para as mesmas entidades, as informação contidas em páginas de entidade são utilizadas como guia para segmentação de páginas de lista. Na avaliação do trabalho, as páginas de entidades relevantes foram baixadas manualmente, e apenas a segmentação das páginas de lista foi avaliada.

Derouiche et al. [13] apresenta uma das primeiras soluções que utiliza conhecimento de domínio para diminuir o esforço manual com relação a extratores supervisionados e aumentar a precisão em relação a extratores não-supervisionados. A indução de um extrator é realizada a partir de páginas de exemplo. Ele atua tanto em páginas de lista quanto em páginas de entidade, porém utiliza atributos de tipos bem definidos, como autor, data, localização, preço etc e extrai informações de páginas com estruturas não muito complexas.

REPEX [12] é uma solução para extração de conteúdo de páginas de fóruns. Ela atua em páginas de *threads*, que se assemelham a páginas de lista, onde as entidades são postagens feitas por usuários. Detectores são utilizados para tipos comuns em páginas desse domínio, como dia, hora e autor de uma postagem. Para encontrar regiões relevantes, é utilizada uma medida de teoria de informação, assumindo que essas regiões são balanceadas de acordo com esses tipos detectados. Após a identificação de regiões relevantes, é capaz de segmentar as regiões e extrair o conteúdo de cada postagem.

DEXTER [10] é uma solução para encontrar sites de produto na web, detectar e extrair especificações de produtos desses sites. Para esse domínio, a maior parte dos sites utiliza listas e tabelas HTML. Com isso, uma estratégia eficaz foi a de treinar um classificador para decidir se uma lista ou página HTML representa uma especificação ou não. Com o foco nesses dois casos, a extração dos pares atributo-valor foi mais simples.

O projeto DIADEM [11] é a solução estado da arte em coleta e extração de informação automática. Para um site contendo informações para um certo domínio, a ferramenta é capaz de explorar o site, identificar dados relevantes e induzir um extrator para coletar as informações. Para isso, utiliza conhecimento de domínio, na forma de ontologia, para definir diversos aspectos sobre os dados, como reconhecedores para os tipos dos atributos e valores e restrições a respeito da forma de apresentação nos sites. Dessa forma, embora a solução atue de forma automática, é preciso um trabalho manual não-trivial para configurá-la para um novo domínio. Outro problema é que a solução é focada apenas em páginas de lista, não tendo suporte para páginas de entidade.

CAPÍTULO 3

Abordagem

Nessa seção apresentamos REX (Real Estate eXtractor), um extrator capaz de extrair automaticamente pares atributo-valor de entidades de imóveis em páginas HTML. A Figura 3.1 ilustra o passo a passo do processo de extração. Dada uma página contendo uma entidade estruturada, REX primeiramente detecta na árvore DOM da página os nós contendo atributos base para o domínio, como quartos, área e banheiros. A partir desses atributos, REX detecta regiões contendo informações estruturadas acerca dos atributos da entidade. Para isso, utiliza os ancestrais comuns aos nós identificados. No próximo passo, identifica os atributos presentes na página, a partir do texto em nós que possuem caminho parecido com os nós contendo os atributos base. Por último, REX segmenta os textos contidos nas subárvores filhas das raízes das regiões detectadas, e identifica os pares atributo-valor a partir dos textos segmentados.

3.1 Detecção de Nós com Atributos Base

Como ponto de partida para o processo de extração, assumimos que para cada domínio há um conjunto de atributos base que aparecerá frequentemente em páginas desse domínio. Para imóveis, por exemplo, existem os seguintes atributos: quartos, área, vagas, condomínio e banheiros. Para esses atributos, implementamos detectores básicos.

Embora a necessidade de utilizar esses atributos seja um fator que diminui a autonomia do extrator, notamos que é preciso apenas conhecimento básico do domínio para escolher atributos adequados, ao contrário de soluções que utilizam configurações mais complexas, como ontologias [11].

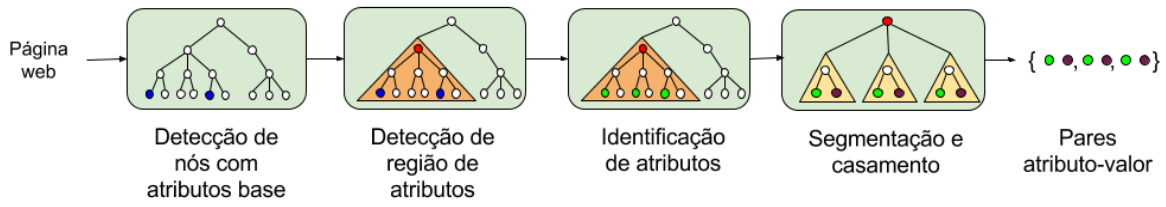


Figura 3.1: Sequência de passos executados para extração. Os nós em azul indicam nós detectados para atributos base, em vermelho a raiz da região de atributos, em verde contendo os atributos e em roxo contendo os valores. A subárvore laranja indica a região de atributos e as subárvores em amarelo indicam as regiões com texto a ser utilizado na segmentação e casamento.

Atributo	Tamanho Limite	Padrão textual
Área	30	`"área"`
Banheiros	30	`"banheiro"`
Condomínio	50	`"condomínio"`
Quartos	30	`"quarto|dorm"`
Vagas	30	`"vaga|garagem"`

Tabela 3.1: Parâmetros de regras para detecção de atributos base.

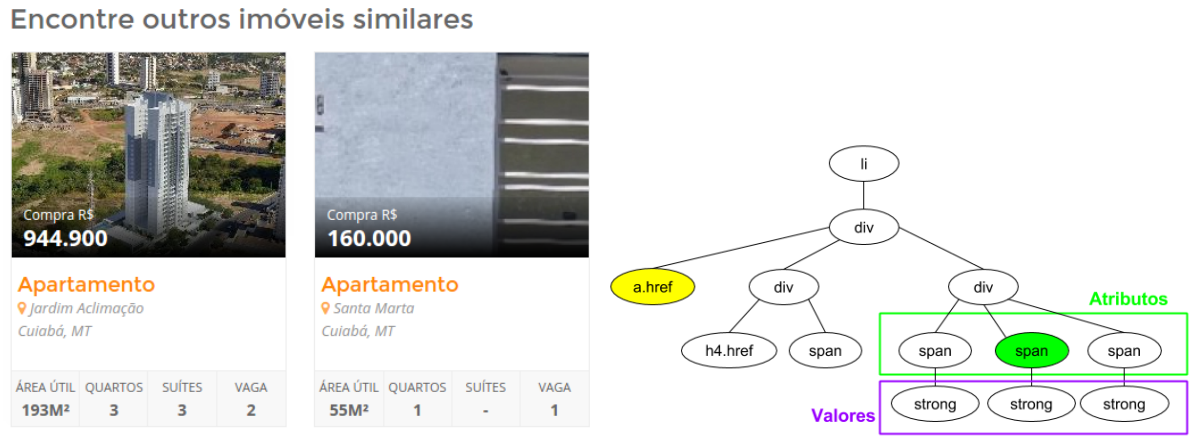
Para cada atributo, uma regra é criada para detectar se um nó da árvore DOM casa com o atributo. Essa regra é normalmente baseada em casamento de padrões. Em busca de alcançar uma boa precisão na detecção, utilizamos heurísticas, que consistem em verificar se o tamanho do texto no nó está dentro do limite definido e evitar nós que possuem ancestrais com *links* ou que são *forms*. Filtramos textos longos porque regiões com informações estruturadas normalmente possuem textos curtos, ao contrário de regiões indesejadas como comentários ou descrições livres. Já *forms* estão normalmente relacionados a campos para consulta e a exibição de informações. Também percebemos que regiões dentro de *links* estavam normalmente relacionadas a informações extras como entidades similares, e não a entidade principal. Veja os parâmetros utilizados para cada extrator na Tabela 3.1.

A última etapa da detecção é a remoção de nós referentes a imóveis similares. Em uma página de uma entidade de imóvel (chamada de entidade principal) é comum que haja sugestões de imóveis similares, para que um cliente interessado na entidade principal possa encontrar outras opções. Como estamos interessados apenas em colher os dados da entidade principal, desejamos ser capazes de detectar e evitar esses casos. A primeira tentativa de atacar esse problema é evitar nós que ocorram dentro de *links*, como explicamos anteriormente. Porém apenas esse filtro não é suficiente, como pode ser visto na Figura 3.2, retirada do site Imóvel Top, onde os atributos dos imóveis similares não são descendentes do nó com o *link*.

Para a entidade principal, é improvável que existam diversos nós contendo um mesmo atributo que possuam o mesmo caminho (sequência de *tags* da raiz até o nó desconsiderando a posição em relação ao pai). Sendo assim, procuramos descobrir caminhos improdutivos e evitar usar nós nesses caminhos em etapas seguintes da extração. Para detectar os caminhos improdutivos, contamos o número de vezes que cada caminho aparece nos nós detectados para um dado atributo, e selecionamos aqueles acima de um limite como improdutivos.

Experimentamos duas formas para a construção do caminho, utilizando apenas as *tags* e utilizando o valor dos atributos HTML *class* além das *tags*. Para o último nó do caminho, sempre utilizamos apenas a *tag*, para evitar problemas onde o atributo *class* especializa o atributo, por exemplo `"<span class='baths'>"`. Quando o nó possui múltiplos valores para o atributo *class*, adotamos a simplificação de utilizar apenas a que aparece primeiro. Em relação aos limites mínimos para considerar um caminho como improdutivo, consideramos um limite mínimo de 3 ocorrências no caso onde só usamos as *tags* e 2 ocorrências quando usamos *class* e *tags*.

Essa estratégia não é perfeita, já que, por exemplo, uma página pode ter apenas uma en-



(a) Informação exibida no site

(b) Parte da árvore HTML para uma das entidades

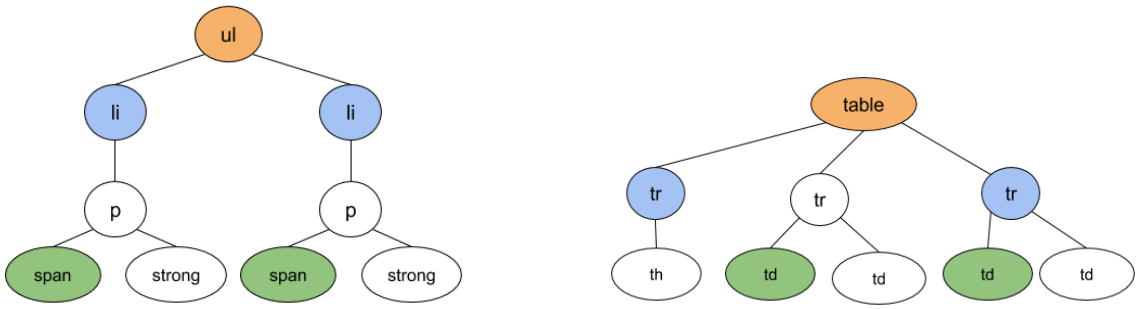
Figura 3.2: Exemplo de página com imóveis similares. A heurística de filtrar nós dentro de *links* não é suficiente porque o nó do atributo (em verde) não é descendente do nó com o *link* para o imóvel similar (em amarelo). Porém, para todos imóveis similares presentes na página, o caminho para o nó em verde será o mesmo, indicando que nós com esse caminho devem ser filtrados.

tidade similar. Utilizar alguma técnica de segmentação que ajude a identificar o conteúdo principal da página seria um bom complemento para lidar com problemas desse tipo. Um detalhe importante para um bom desempenho é o impacto da variabilidade do atributo. Devem ser utilizados atributos com pouca variação, como quartos, ao invés de um atributo como área, que possui variações como "Área útil" e "Área do terreno", que podem ser ocorrências legítimas da presença do mesmo caminho na entidade principal.

3.2 Detecção da Região dos Atributos

Utilizando os detectores de atributos encontramos os nós que casam com cada atributo base. Apenas essa detecção não é suficiente para extração dos pares atributo-valor, visto que é comum que o valor do atributo não esteja no mesmo nó do atributo, e também desejamos ser capazes de extrair pares mesmo para atributos que não possuem detector. Sendo assim, é preciso encontrar regiões da página (nós raízes e suas subárvores) que contém os pares atributo-valor. A Figura 3.3 contém exemplos concretos do objetivo da detecção de região. Na Figura 3.3a os atributos foram detectados nos nós com *tag* `span`. O detector de região precisa então identificar que o nó com *tag* `ul` é a raiz da subárvore que contém os pares atributo-valor. Já na Figura 3.3b, os atributos foram identificados em nós com *tag* `td` e a raiz da região é o nó com *tag* `table`.

Através da análise de páginas do conjunto de desenvolvimento (descrito na Seção 4.1), percebemos que é comum que a raiz da região que contém os atributos e valores seja o menor ancestral comum entre dois nós detectados. Sendo assim, para cada par de nós detectados de atributos diferentes, encontramos seu menor ancestral comum na árvore DOM. Para melhorar



(a) Exemplo para lista HTML

(b) Exemplo para tabela HTML

Figura 3.3: Exemplos de menor ancestral comum, utilizados como raízes de regiões relevantes no REX. Os nós laranjas são os ancestrais comuns dos pares de nós das outras cores (azul e verde).

a precisão, consideramos apenas o caso em que os dois nós estão na mesma profundidade na árvore e também um limite na distância entre a raiz e os nós. Apresentamos essa estratégia no Algoritmo 1.

Algoritmo 1 Detector de raízes

```
1:  function DETECTARRAIZES(detectores, distancia_maxima)
2:      raizes ← {}
3:      n ← detectores.size()
4:      for i ← 1, n do
5:          for j ← i + 1, n do
6:              for each x in detectores[i].nos_detectados do
7:                  for each y in detectores[j].nos_detectados do
8:                      ancestral = MenorAncestralComum(x, y, distancia_maxima)
9:                      if valido(ancestral) then
10:                         raizes.append(ancestral)
11:     return raizes
```

Percebemos que utilizando apenas a abordagem descrita estávamos detectando regiões improdutivas, ou seja, regiões contendo informações indesejadas como descrições textuais de características do imóvel. No exemplo da Figura 3.4, o limite no tamanho do texto utilizado na detecção do nó não é suficiente, já que os nós contendo “Quartos” e “Banheiros” possuem texto com tamanho pequeno. Sendo assim, precisamos utilizar alguma heurística para eliminar raízes de subárvores indesejadas. Optamos por utilizar uma estratégia simples, eliminando as raízes onde a média dos tamanhos dos textos de suas subárvores filhas fosse maior que um limite.

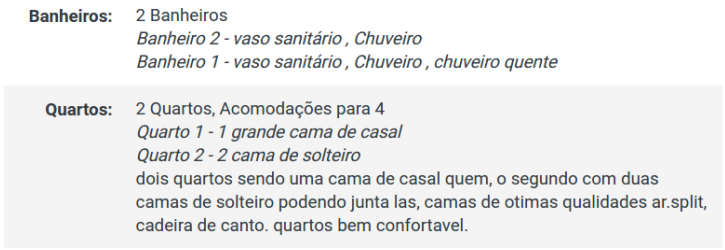


Figura 3.4: Exemplo de região improdutiva que deve ser removida do processo de extração.

3.3 Identificação de Atributos

Além de identificar as regiões contendo os pares atributo-valor, é preciso identificar os atributos que estão presentes nessas regiões. O primeiro passo para alcançar esse objetivo é identificar os nós contendo esses atributos. Depois disso, é preciso identificar, no texto desses nós, esses atributos. Embora os detectores da Seção 3.1 nos ajudem a encontrar nós contendo alguns desses atributos, eles cobrem apenas um subconjunto de todos os atributos do domínio, de forma que existem outros atributos que precisam ser encontrados automaticamente, além de variações no texto dos atributos base.

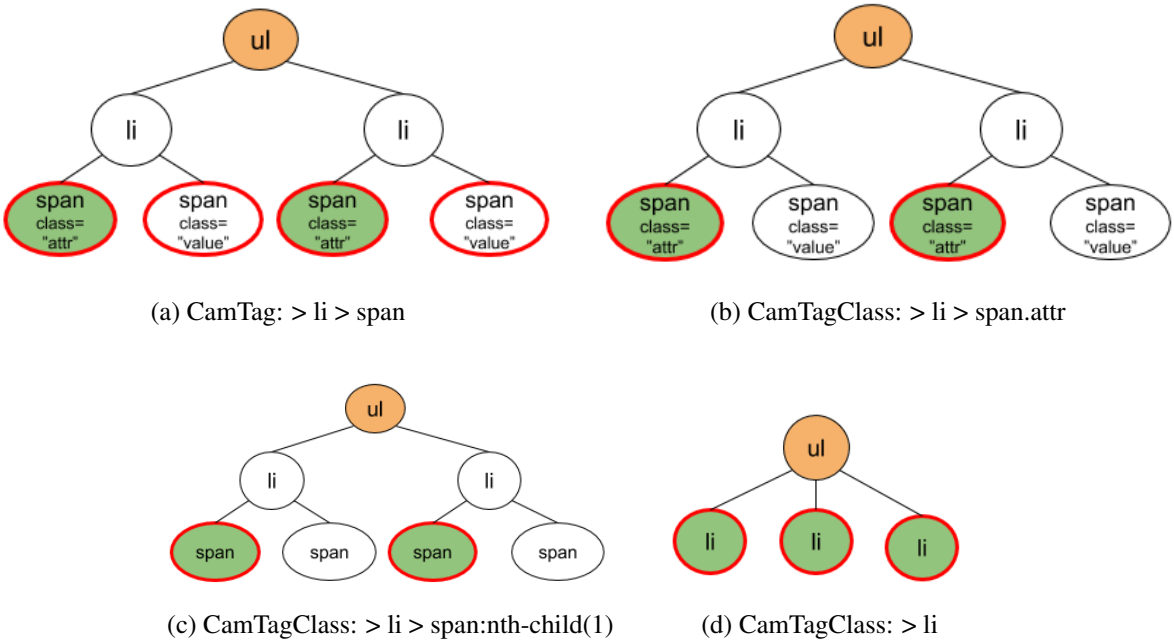
Para a árvore na Figura 1.4b, caso os atributos base fossem condomínio e quartos, a etapa de detecção de nós com atributos base iria detectar o nó com *tag* `dt`, contendo condomínio, e com a *tag* `dd`, contendo quartos. Na etapa de detecção de região de atributos, seria detectado o nó com tag `section` como raiz da região. Já a etapa de identificação de atributos identificaria os nós contendo os demais atributos (`dt` contendo preço e `dd` banheiro) e a representação textual desses atributos: “preço”, “condomínio”, “quartos” e “banheiro”. Esse exemplo também ilustra algumas dificuldades da tarefa, já que cada subárvore filha do nó com *tag* `section` contém múltiplos pares, e cada uma delas possui uma estrutura distinta. Além disso, uma das subárvores não contém pares atributo-valor.

A estratégia que utilizamos para encontrar esses nós contendo atributos é baseada no caminho de cada nó. Percebemos que é comum que conjuntos de atributos possuam caminho parecido, o que é esperado visto que eles são gerados pelo mesmo *template*. Por exemplo, na Figura 3.2b os atributos possuem o mesmo caminho, e na Figura 3.3a os nós com *tag* `span` (que poderiam ser nós contendo atributos) também possuem o mesmo caminho.

Para cada nó detectado na Seção 3.1, identificamos um nó raiz, construímos o caminho até esse nó a partir dessa raiz e utilizamos esse caminho para encontrar outros nós que compartilham do mesmo caminho. Experimentamos formas diferentes na construção do caminho e nós diferentes para utilizar como raiz.

Mais especificamente, dado um nó detectado u, é preciso primeiramente escolher um nó r como nó inicial do caminho, para depois construir o caminho de r a u. Utilizamos duas estratégias diferentes para a escolha do nó inicial:

- RaizDoc: Raiz da árvore HTML do documento.
- RaizSub: Raiz da menor subárvore contendo o nó detectado, dentre as raízes detectadas na Seção 3.2.



(a) CamTag: > li > span

(b) CamTagClass: > li > span.attr

(c) CamTagClass: > li > span:nth-child(1)

(d) CamTagClass: > li

Figura 3.5: Exemplos de estratégias para construção do seletor. Os nós em verde contêm os atributos. Os nós em laranja representam a raiz. Os nós com contorno vermelho representam os nós selecionados pelo seletor. Em cada legenda podem ser encontrados a estratégia e o seletor construído. Na figura (a), os nós contendo valores são selecionados por engano. Na figura (b), o problema é corrigido utilizando o valor do atributo *class*. A figura (c) mostra um caso onde a posição é utilizada já que não há atributo *class*. Por ultimo, a figura (d) mostra o caso onde atributos e valores estão no mesmo nó e a posição não é utilizada porque a raiz é o pai do atributo.

O caminho é construído utilizando um padrão que possa ser utilizado para encontrar outros nós, para isso utilizamos *CSS selector*. Depois de construído o caminho, selecionamos outros nós na subárvore enraizada em *r* utilizando o seletor. As estratégias utilizadas para construção do seletor são:

- CamTag: Sequência de *tags* (Figura 3.5a).
- CamTagClass: Sequência de *tags*, e para o último nó da sequência utilizamos o valor do atributo HTML *class* (Figura 3.5b), ou posição para o pai (Figura 3.5c) se o atributo *class* for inexistente e o pai não for o nó inicial (Figura 3.5d).

Utilizamos três formas diferentes de combinar as estratégias, apresentadas na Tabela 3.2.

A ideia por trás da utilização do valor do atributo *class* é que em alguns casos o seletor da estratégia CamTag para o atributo e para o valor é o mesmo (Figura 3.5a), e alguns sites adicionam informações no *class* como “params_field_label” ou “params_field_value”, que ajudam a diferenciar os dois casos. Ressaltamos que essas informações são adicionadas apenas ao último

Versão	Nó Inicial	Construção do seletor
Identificação DT	RaizDoc	CamTag
Identificação ST	RaizSub	CamTag
Identificação STC	RaizSub	CamTagClass

Tabela 3.2: Escolha de ponto inicial e construção do seletor para cada versão.

nó do caminho. No caso no qual o nó possua múltiplos valores para o atributo *class*, utilizamos apenas a que aparece primeiro. Uma abordagem mais robusta seria procurar identificar automaticamente qual conjunto de valores pro *class* é invariante aos atributos.

Após identificar os nós contendo os atributos, é preciso identificar os atributos dentro do texto desses nós. Em casos mais simples, o texto completo do nó indicará o atributo, porém é possível que o texto também contenha o valor além do atributo. Em casos como "Vagas de Garagem = 1" e "Área: 50m²", consideramos apenas o texto que aparece antes do separador (utilizamos = e : como separadores). Já em casos mais parecidos com texto livre, onde o texto contém dígitos, como "2 a 3 dormitórios", quebramos o texto de acordo com espaço em branco ou − (hífen) e extraímos o maior o sufixo de palavras até a aparição de um número.

Para aumentar a precisão, evitamos atributos muito longos (acima de 30 caracteres) e atributos que sejam valores numéricos. Essa última filtragem é uma forma simples de evitar identificar um texto que representa um valor como atributo, em casos onde o caminho para o atributo e o valor é o mesmo na estrutura do site. Veja o Algoritmo 2 para uma visão geral da expansão de atributos extraídos.

Algoritmo 2 Aprendizagem de atributos

```
1:  function APRENDERATRIBUTOS(detectores)
2:      atributos ← ∅
3:      for each detector in detectores do
4:          for each x in detector.nos_detectados do
5:              raiz ← EscolherRaiz(x)
6:              seletor ← ConstruirSeletor(x, raiz)
7:              nos_similares ← raiz.select(seletor)
8:              for each y in nos_similares do
9:                  texto ← y.texto()
10:                 if texto contém separador then
11:                     texto ← texto[:posicao_separador]      ▷ prefixo até posicao_separador
12:                 if valido(texto) then
13:                     if texto contém dígito then
14:                         atributos ← atributos ∪ ExtrairAtributoDeTexto(texto)
15:                     else
16:                         atributos ← atributos ∪ texto
17:     return atributos
```

3.4 Segmentação e Casamento de Atributos e Valores

Após a detecção de raízes da Seção 3.2, consideramos que o texto de cada subárvore filha é uma fonte de pares atributo-valor. Para ajudar a diferenciar atributos de seus valores, na Seção 3.3 mostramos como identificamos os atributos presentes na página. Utilizando essas informações, realizamos uma segmentação em cada texto, a fim de buscar diferenciar as partes dos textos referentes a atributos e a valores, e por último realizamos o casamento entre atributos e valores para extração dos pares.

O primeiro passo para segmentação é dividir o texto de acordo com espaço ou separadores `:` ou =. Em seguida utilizamos um algoritmo guloso, com o objetivo de agrupar os intervalos de palavras referentes a possíveis atributos. O Algoritmo 3 mostra o funcionamento da segmentação. A cada passo do laço `while` começando na linha 6, procuramos o maior intervalo de palavras começando na posição atual que possa ser um atributo, guardamos os índices do intervalo encontrado (linhas 13 e 14) e continuamos da posição posterior ao fim do intervalo. Caso não haja intervalo, continuamos da próxima posição. Utilizando os índices guardados, segmentamos o texto de acordo com os intervalo entre os índices ordenados (linha 19 em diante). O casamento entre valor e atributo será realizado em seguida.

A segmentação utilizada ajuda o casamento de pares atributo-valor em textos contendo múltiplos pares. Por exemplo, para uma subárvore onde o texto é "Valor do Aluguel R$ 885 / Mês Condomínio R$ 80" onde valor do aluguel e condomínio foram identificados como atributos, o algoritmo de segmentação vai primeiramente encontrar de forma gulosa os intervalos com os atributos Valor do Aluguel e Condomínio, e depois considerar o texto entre esses intervalos também como intervalos (representando possíveis valores), de forma que a segmentação fica [Valor do Aluguel, R$ 885 / Mês, Condomínio, R$ 80].

A última fase é o casamento de atributos e valores para extração dos pares da instância. Após a segmentação dos textos, sabemos quais intervalos são candidatos a atributos e quais são candidatos a valores. Para realizar o casamento, nos concentramos apenas no caso mais simples, onde não há muita variação dentro de um mesmo texto. Por exemplo, assumimos que dentro de um mesmo texto é consistente que o atributo aparece à direita ou à esquerda do valor. Sendo assim, para cada intervalo referente a um possível atributo, tentamos casá-lo com o intervalo ao seu lado (se esse intervalo não for também um atributo). Para aumentar a precisão do REX, consideramos um limite máximo no tamanho do atributo e do valor.

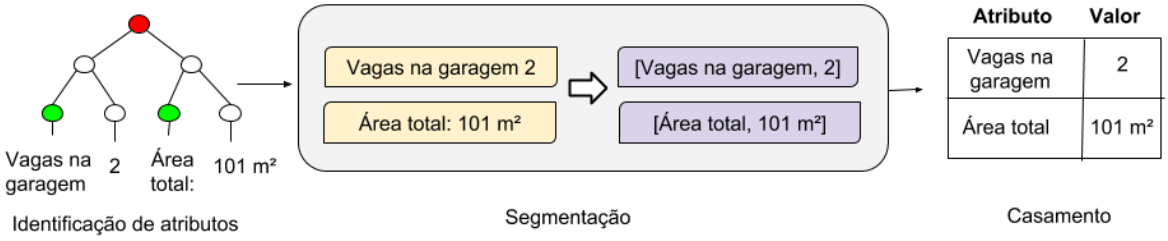


Figura 3.6: Passos finais da extração. Utilizando as regiões e os atributos identificados em passos anteriores, segmenta o texto para realizar o casamento entre atributos e valores.

Algoritmo 3 Segmentação de textos extraídos de subárvores

```
1:  function SEGMENTARTEXTO(texto, atributos)
2:      palavras ← texto.split("[\\h:=]+")
3:      n ← palavras.size()
4:      indices_divisao ← {1}
5:      i ← 1
6:      while i ≤ n do
7:          fim_atributo ← -1
8:          for j ← i, n do
9:              possivel_atributo ← palavras[i : j + 1]
10:             if possivel_atributo in atributos then
11:                 fim_atributo ← j
12:         if fim_atributo ≠ -1 then
13:             indices_divisao.append(i)
14:             indices_divisao.append(fim_atributo + 1)
15:             i ← fim_atributo
16:         i ← i + 1
17:     indices_divisao ← indices_divisao + n
18:     m ← indices_divisao.size()
19:     texto_segmentado ← { }
20:     for i ← 2, m do
21:         comeco ← indices_divisao[i - 1]
22:         fim ← indices_divisao[i]
23:         if comeco ≠ fim then
24:             texto_segmentado.append(palavras[comeco:fim])
25:     return texto_segmentado
```

Algoritmo 4 Casamento de pares atributo-valor

```
1:  procedure CASAMENTOPARES(texto_segmentado, atributos, resultado_extracao)
2:      n ← texto_segmentado.size()
3:      comeca_com_atributo ← texto_segmentado[1] in atributos
4:      for i ← 1, n do
5:          atributo ← texto_segmentado[i]
6:          if atributo not in atributos then
7:              continue
8:          posicao_valor ← i + 1 if comeca_com_atributo else i - 1
9:          if posiscao_valor ≤ n then
10:             valor ← texto_segmentado[posicao_valor]
11:             if par_valido(atributo, valor) then
12:                 valor_atual ← resultado_extracao.get(atributo)
13:                 if SobrescreverValor(valor_atual, valor) then
14:                     resultado_extracao.put(atributo, valor)
```

Embora em uma entidade não haja múltiplos valores atrelados a um único atributo, é possível que no processo de extração mencionado ocorram múltiplos casamentos para um mesmo atributo. Nas primeiras versões desenvolvidas, sobrescrevíamos o valor atual em cada casamento. Posteriormente, utilizamos uma heurística para definir se o valor atual deve ser trocado. A ideia é que priorizar valores numéricos (mais prováveis a serem valores corretos) e valores que aparecerem primeiro. Sendo assim, a ordem que visitamos os nós na Seção 3.1 influencia o desempenho da abordagem. Para uma visão geral do processo de casamento dos atributos veja o Algoritmo 4.

CAPÍTULO 4

Avaliação

Para avaliação do REX, realizamos um estudo de caso para o domínio de imóveis. Nesse capítulo mostramos os resultados da avaliação experimental realizada, utilizando uma coleção de páginas HTML de diversos sites de imóveis. Realizamos dois tipos de avaliação. No primeiro deles, analisamos diversas versões do REX construídas ao longo do projeto, comparando o resultado da extração com um *golden set* construído a partir da criação de um extrator específico para cada site da coleção. No outro tipo, utilizamos uma abrangência maior de sites e uma quantidade menor de páginas por site, e testamos o desempenho da melhor versão do REX, verificando o resultado da extração manualmente.

4.1 Construção da Coleção de Páginas

Para desenvolvimento e avaliação do REX, coletamos páginas de diversos sites, que foram divididos em dois grupos:

- Conjunto de desenvolvimento: Utilizado para análise de características na forma da apresentação do conteúdo e experimentação de diferentes abordagens. Para esse conjunto buscamos utilizar alguns sites populares e também sites que representassem a heterogeneidade encontrada na forma de exibição do conteúdo. Esse conjunto é composto por 14 sites.

- Conjunto de teste: Utilizado apenas para avaliação do REX, ou seja, não analisamos o conteúdo encontrado nas páginas para melhoria da solução. A maior parte dos sites desse conjunto representa sites clientes de um software de geração de sites de imobiliárias, como imobex.com.br, flex49.com.br e tecimob.com.br. Dessa forma, há outros sites (também clientes dos mesmos softwares) que compartilham da mesma estrutura. Esse conjunto é composto por 6 sites.

Para todos os sites considerados, coletamos 10 páginas para avaliação e criamos um extrator específico para rotulação automática da informação contida nessas páginas, utilizando essa informação como *golden set*. Em geral, há pouca variação entre páginas de um mesmo site, já que elas normalmente utilizam o mesmo *template*, de modo que mesmo um número relativamente baixo (10 páginas) já é capaz de fornecer uma estimativa razoável da qualidade da extração para aquele site.

Para implementação, tanto do extratores específicos quanto do REX, utilizamos Jsoup[1], uma biblioteca em Java utilizada para facilitar o trabalho com documentos HTML. Utilizando

[1] https://jsoup.org

Site	*Tag* Raiz	Caminho Atributo	Caminho Valor	Mesmo nó
zapimoveis	ul	li > span li > h3 > span	li li > h3	Não
vivareal	div	dl > dt dl > dd	dl > dd dl > dl > dd > span	Não Não
olx	ul	li > p > span	li > p > strong	Não
aluguetemporada	tbody	tr > td	tr > td	Não
viaimoveis	tbody	tr > td > span	tr > td > span	Sim
redeimoveispe	div	div	div > strong	Não
classificadosdegraca	div	div > span	div > span	Não
novaera	div	section	section	Sim*
movingimoveis	ul	li > div li > div > span	li > div li > div > span	Não Sim

Tabela 4.1: Formas de apresentação do conteúdo em sites. Os caminhos indicam a sequência de *tags* entre a raiz e o nó final. Mesmo nó indica se o atributo e o valor fazem parte do mesmo nó (um * indica que pode não haver separador).

essa biblioteca, é possível criar uma árvore DOM a partir do texto da página, navegar pela árvore, consultar atributos de um elemento, selecionar todos os descendentes de um nó de acordo com algum seletor, e mais.

Características do conjunto de desenvolvimento

No restante dessa seção mostramos algumas características da coleção formada pelo conjunto de desenvolvimento. Essas características buscam ilustrar a dificuldade em realizar a extração automática do conteúdo encontrado nessas páginas.

Na Tabela 4.1 apresentamos a forma como as informações estruturadas estão organizadas nas páginas HTML em alguns sites desse conjunto. Percebemos que há uma grande variabilidade, e casos que exigiram atenção especial como: (i) mesmo caminho para atributo e valor (Classificados de Graça e Alugue Temporada); (ii) atributo e valor no mesmo nó (Via Imóveis e Nova Era); (iii) sub-região aninhada (Moving Imóveis); (iv) mais de um caminho para atributos e valores (Zap Imóveis e Viva Real).

No caso do Moving Imóveis, os atributos relacionados ao valor (condomínio, iptu etc) encontram-se em uma sub-estrutura contida dentro da estrutura que contém as informações para os outros atributos. Já para o Zap Imóveis, há variabilidade intra-site (há um *layout* especial para páginas na categoria lançamento), mas não intra-página. Para o Viva Real, além da variabilidade intra-site, diferentes atributos em uma mesma página possuem caminhos diferentes.

Há uma grande quantidade de atributos distintos existentes nas páginas do domínio. Para as 140 páginas do conjunto de desenvolvimento, encontramos 89 atributos distintos, dentre os 926 pares atributo-valor existentes no *golden set*. Isso mostra que é importante uma forma automática de detecção desses atributos. Na Figura 4.1 mostramos um histograma contendo a quantidade de ocorrências dos atributos mais frequentes. Percebemos que há variações no texto

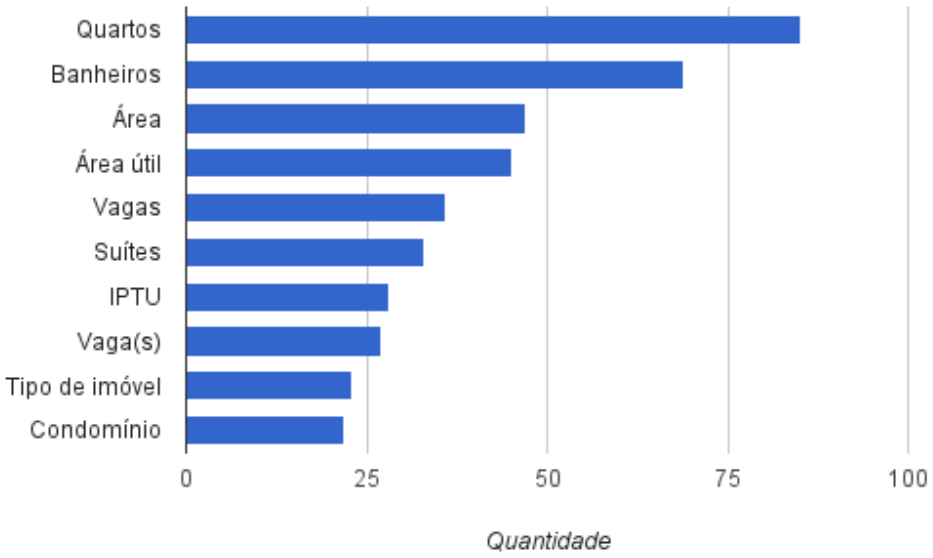


Figura 4.1: Histograma com a quantidade de ocorrências dos atributos mais frequentes para o conjunto de desenvolvimento.

de alguns atributos, como área e vagas, o que mostra a importância de uma fase de integração de atributos a ser realizada após a extração, com o objetivo de agrupar os dados extraídos para diferentes sites em uma única base de dados estruturada.

Na coleção utilizamos a versão antiga do site Moving Imóveis na versão de desenvolvimento e a versão nova (lançada no decorrer do projeto) na versão de teste. Embora tenha sido necessário refazer o extrator específico, podemos ver na Seção 4.4 que o REX funcionou muito bem para a versão nova. Esse é um exemplo real de uma das dificuldades em trabalhar com extratores específicos, o custo manual de atualização dos extratores.

4.2 Versões do REX

Durante o Capítulo 3 falamos em alguns momentos sobre diferentes estratégias desenvolvidas para alguns passos. Com isso, resolvemos exibir os resultados encontrados para quatro versões distintas da abordagem, de forma a exibir o impacto causado pelas melhorias. A seguir listamos as partes do REX que não fazem parte de todas versões:

- RT: Remoção de nós referentes a outras entidades, utilizando sequência de *tags*. Seção 3.1.
- RTC: Remoção de nós referentes a outras entidades, utilizando sequência de *tags* e atributo HTML *class*. Seção 3.1.
- HR: Heurística para avaliar qualidade de regiões, utilizando a média do tamanho dos textos das subárvores filhas. Seção 3.2.

- IDT: Identificação de atributos DT, utilizando RaizDoc como estratégia de nó inicial e CamTag como estratégia de construção de caminho. Seção 3.3
- IST: Identificação de atributos ST, utilizando RaizSub como estratégia de nó inicial e CamTag como estratégia de construção de caminho. Seção 3.3.
- ISTC: Identificação de atributos STC, utilizando RaizSub como estratégia de nó inicial e CamTagClass como estratégia de construção de caminho. Seção 3.3.
- AV: Identificação de atributo quando atributo e valor estão no mesmo nó sem separador. Seção 3.3.
- HS: Heurística para definir sobrescrita no casamento de atributo e valor. Seção 3.4.

Na Tabela 4.2 mostramos quais dessas partes opcionais são utilizados em cada uma das quatro versões do REX. A segmentação é a mesma em todas versões.

	Versão 1	**Versão 2**	**Versão 3**	**Versão 4**
Detecção de nós		RT	RTC	RTC
Detecção de região de atributos		HR	HR	HR
Identificação de atributos	IDT	IST	ISTC	ISTC + AV
Casamento de atributo e valor		HS	HS	HS

Tabela 4.2: Componentes opcionais utilizados em cada versão avaliada.

4.3 Métricas

Nessa seção descrevemos as métricas que foram utilizadas para avaliar o desempenho do REX. O uso dessas métricas é fundamental para que possamos ter uma noção da qualidade da abordagem, além do impacto das diferentes mudanças implementadas ao longo do desenvolvimento. Seja N o número total de extrações possíveis, E o número de pares extraídos pelo REX e C o número de pares extraídos corretamente, utilizamos três métricas:

$$\text{Revocação} = \frac{C}{N}$$

$$\text{Precisão} = \frac{C}{E}$$

$$\text{F1} = \frac{2 * \text{Revocação} * \text{Precisão}}{\text{Revocação} + \text{Precisão}}$$

Como utilizamos diversos sites na avaliação, calculamos as métricas para cada um desses sites, e combinamos os resultados calculando a média micro do conjunto, que considera a soma de cada fator (N, E e C) combinado para os diferentes sites, e calcula as métricas.

Site	Versão 1	Versão 2	Versão 3	Versão 4
Zap Imóveis	0.96	0.96	0.96	1.00
OLX	1.00	1.00	1.00	1.00
Lugar Certo	0.78	0.97	0.97	0.97
Imóvel Top	0.62	0.99	0.99	0.99
Classificados de Graça	0.81	0.81	1.00	1.00
Portal do Proprietário	1.00	1.00	1.00	1.00
Alugue Temporada	0.5	0.9	0.99	0.97
Viva Real	0.76	0.94	0.95	0.94
Lar Imóveis	0.63	1.00	1.00	1.00
Moving Imóveis (versão antiga)	0.91	0.95	0.94	0.94
Via Imóveis	0.95	0.95	0.95	0.95
Rede Imóveis	0.22	0.67	0.85	0.84
Mercado Livre	0.89	0.75	0.75	0.75
Nova Era	0.72	0.73	0.73	0.94
Média micro	0.73	0.91	0.94	0.95

Tabela 4.3: Valores de F1 para os sites do conjunto de desenvolvimento, em cada uma das versões desenvolvidas.

Na comparação das chaves dos pares, realizamos uma comparação ignorando a diferença de maiúsculas e minúsculas, e.g., “Quartos” é equivalente a “quartos”. Para comparação dos valores, realizamos uma comparação ignorando espaços em brancos, e.g., “10 m^2” é equivalente a “10m^2”.

4.4 Resultados

Para avaliação da solução, executamos cada versão definida na Seção 4.2 para cada página da coleção formada, e comparamos os resultados extraídos com os valores do *golden set*, calculando as métricas definidas na Seção 4.3. Os valores de F1 para cada versão estão disponíveis na Tabela 4.3 e Figura 4.2, para o conjunto de desenvolvimento, e na Tabela 4.4 e Figura 4.3, para o conjunto de testes. Os valores individuais de precisão e revocação para a versão 4 podem ser vistos na Tabela 4.5 (conjunto de desenvolvimento), Tabela 4.6 (conjunto de testes) e Figura 4.4.

Lembramos que os valores encontrados para o conjunto de desenvolvimento foram utilizadas para guiar a implementação e análise de novas ideias, e nos ajudar a escolher valores para parâmetros utilizados nas heurísticas. O conjunto de teste foi utilizado apenas para avaliação em sites não vistos, de forma que não realizamos mudanças baseadas nos resultados encontrados para esse conjunto.

Podemos notar que conseguimos alcançar um desempenho muito bom para a maioria dos sites. Apenas para dois sites do conjunto de desenvolvimento alcançamos F1 abaixo de 0.9, e

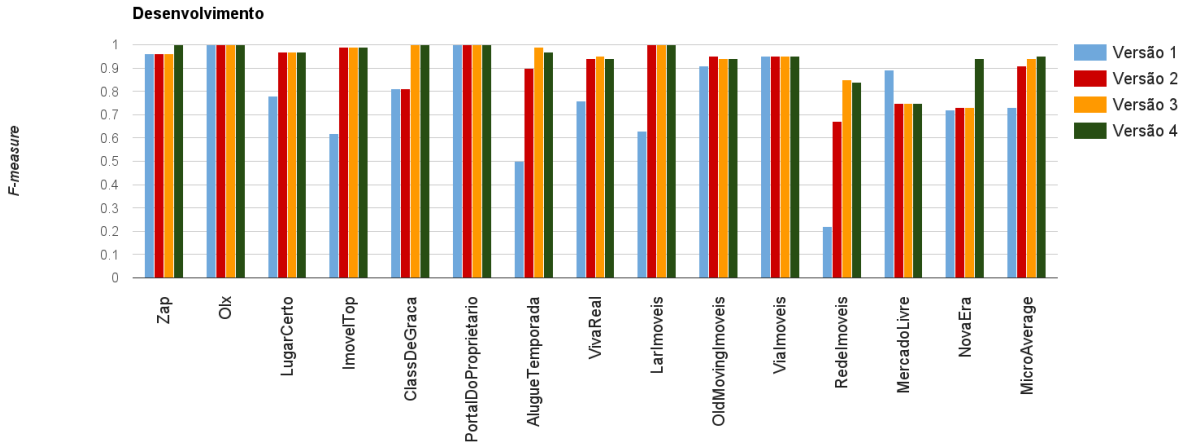


Figura 4.2: Valores de F1 para os sites do conjunto de desenvolvimento, em cada uma das versões desenvolvidas.

Site	**Versão 1**	**Versão 2**	**Versão 3**	**Versão 4**
Imóveis Grande SP	0.96	0.96	0.96	0.96
Unikka Acessoria	0.69	0.76	0.76	0.76
MFG Imóveis	0.62	0.92	0.95	0.95
Imóveis D'AGostini	0.44	0.65	0.95	0.95
ACRio Imóveis	0.51	0.51	0.51	0.52
Moving Imóveis (versão nova)	1.00	1.00	1.00	1.00
Média micro	0.71	0.83	0.90	0.89

Tabela 4.4: Valores de F1 para os sites do conjunto de testes, em cada uma das versões desenvolvidas.

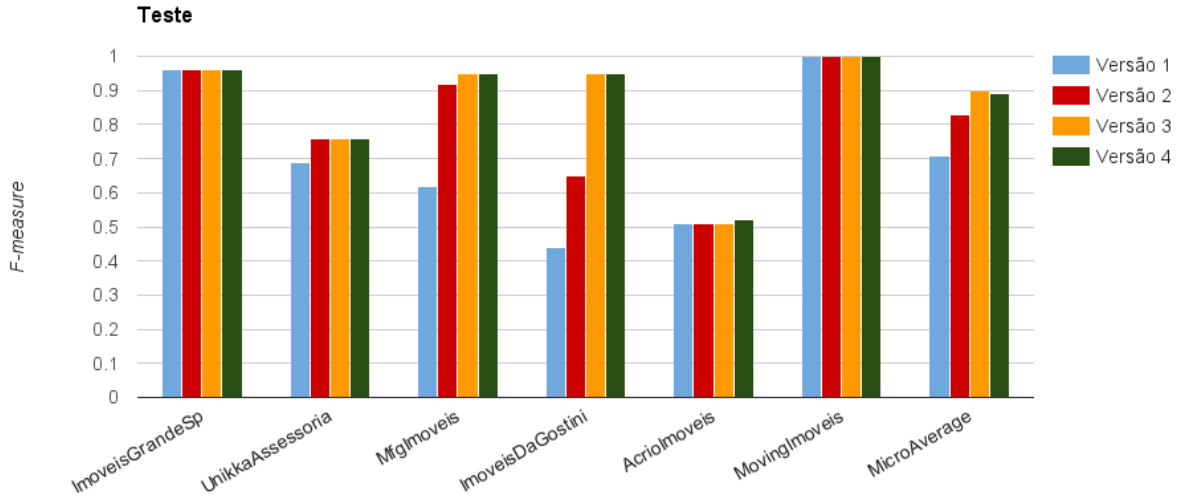


Figura 4.3: Valores de F1 para os sites do conjunto de testes, em cada uma das versões desenvolvidas.

Site	**Revocação**	**Precisão**	**F1**
Zap Imóveis	1.00	1.00	1.00
OLX	1.00	1.00	1.00
Lugar Certo	1.00	0.95	0.97
Imóvel Top	1.00	0.97	0.99
Classificados de Graça	1.00	1.00	1.00
Portal do Proprietário	1.00	1.00	1.00
Alugue Temporada	0.96	0.98	0.97
Viva Real	0.93	0.94	0.94
Lar Imóveis	1.00	1.00	1.00
Moving Imóveis (versão antiga)	0.93	0.95	0.94
Via Imóveis	0.91	1.00	0.95
Rede Imóveis	1.00	0.73	0.84
Mercado Livre	0.64	0.9	0.75
Nova Era	0.88	1.00	0.94
Média micro	0.94	0.96	0.95

Tabela 4.5: Valores de precisão, revocação e F1 para os sites do conjunto de desenvolvimento, para a versão 4.

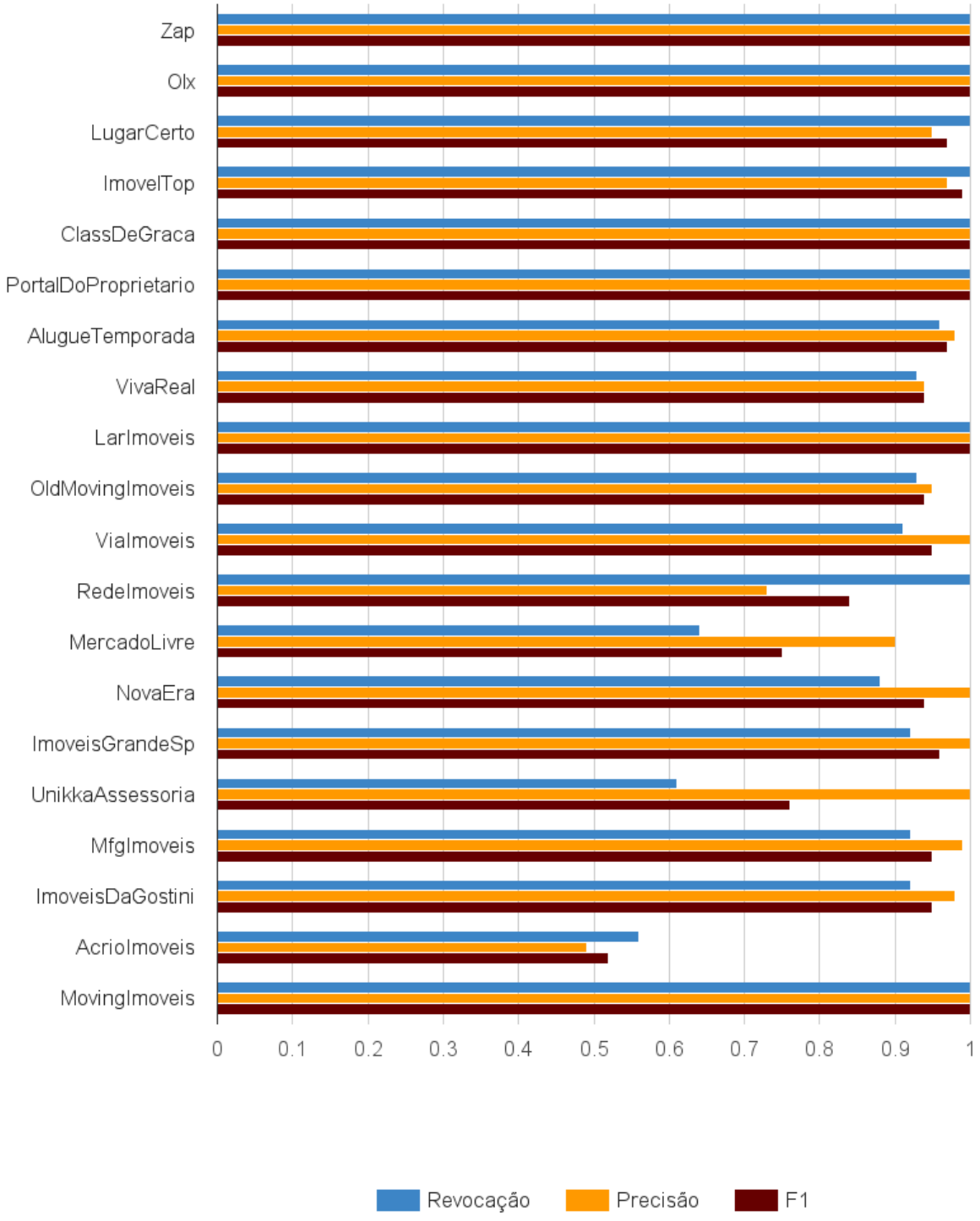


Figura 4.4: Valores de precisão, revocação e F1, para a versão 4.

Site	Revocação	Precisão	F1
Imóveis Grande SP	0.92	1.00	0.96
Unikka Acessoria	0.61	1.00	0.76
MFG Imóveis	0.92	0.99	0.95
Imóveis D'AGostini	0.92	0.98	0.95
ACRio Imóveis	0.56	0.49	0.52
Moving Imóveis (versão nova)	1.00	1.00	1.00
Média micro	0.85	0.93	0.89

Tabela 4.6: Valores de precisão, revocação e F1 para os sites do conjunto de testes, para a versão 4.

Figura 4.5: Exemplo de informação em página do Mercado Livre.

apenas para um site do conjunto de testes alcançamos precisão abaixo de 0.98. Conseguimos um aumento de 0.73 para 0.95 na média micro do F1 para o conjunto de desenvolvimento, e 0.71 para 0.89 no conjunto de testes. No resto da seção, detalhamos alguns problemas identificados.

O único site que obteve um desempenho pior na versão 4 que a versão 1 foi o Mercado Livre. Isso ocorreu por causa da heurística de remoção de regiões. Nesse site há uma grande quantidade de atributos, divididos em colunas. No exemplo da Figura 4.5, a coluna da esquerda possui atributos mais comuns, e a da direita possui alguns atributos que normalmente não fariam parte de uma tabela ou lista de destaque, como disposição e idade do imóvel. Para versão 1, seriam identificadas duas regiões relevantes: a da coluna da esquerda (menor ancestral comum dos atributos quartos e banheiros) e a região contendo todos atributos (menor ancestral comum dos atributos quartos e condomínio). A partir da versão 2, a região contendo todos atributos é desconsiderada, devido ao tamanho do texto das suas subárvores (cada uma dessas subárvores contém um texto grande já que contém vários pares atributo-valor). O problema não aconteceria se mais algum atributo base estivesse presentes na coluna da direita (além de condomínio), já que nesse caso as raízes das duas colunas seriam identificadas corretamente. Se não fosse esse problema, esse site provavelmente se beneficiaria do uso do atributo *class* adotado a partir da versão 3, já que o caminho para o atributo e para o valor tem a mesma sequência da *tags*, mas os nós possuem valores diferentes para o *class*. Uma outra observação é que muitos pares foram identificados na primeira versão pelo fato de que muitos valores são numéricos, e por isso não são identificados erroneamente como atributos (na Figura 4.5 apenas "Norte" seria identificado como atributo).

O site que apresentou pior desempenho foi o ACRio Imóveis. Na Figura 4.6a podemos ver

(a) ACRio Imóveis (b) Unikka Acessoria

Figura 4.6: Exemplo de informação exibida em páginas do ACRio Imóveis e Unikka Acessoria.

um exemplo de informação contida em uma página desse site. Podemos ver que textos como "1 vaga", "Sendo 1 suíte", "Com elevador" e "120 m² área útil" apresentam formas variadas de apresentação dos dados para a mesma região. REX identifica corretamente todos os atributos válidos, porém também identifica "apartamento", "com elevador" e "m²". Os dois primeiros não causam problema, já que não há casamento com um valor. Porém o "m²" faz com que ocorra e extração do par "m² = 120" ao invés de "área total = 120 M²". Outro erro que ocorre é para suíte, onde o valor extraído é "Sendo 1" ao invés de apenas "1".

O site Rede Imóveis foi o site que o REX obteve a menor precisão para o conjunto de desenvolvimento. Como podemos ver no exemplo exibido na Figura 4.7, esse site possui regiões com algumas informações pouco comuns relacionadas ao imóvel, como interfone e portão eletrônico, e também informações que em outros lugares estariam em destaque, como número de vagas e área, apresentada em listas. Por normalmente haver uma predominância de atributos considerados pouco comuns, escolhemos não adicionar informações disponíveis apenas nessa região no nosso *golden set*. Para a extração genérica, essa região é identificada como relevante em algumas páginas, causando problemas para os passos de segmentação e casamento, que geram pares como "infraestrutura = 1". Mesmo casos possivelmente corretos como "andar(es) = 10" seriam considerados incorretos devido à escolha de não inserir essa região no *golden set*.

Identificamos dois problemas na detecção de regiões relevantes que prejudicaram a revocação. O primeiro deles foi a ocorrência de páginas onde não são detectados pelo menos dois nós de uma mesma região, de forma que não é possível utilizá-los para encontrar o ancestral comum. O outro problema foi que não consideramos combinações de nós detectados para o mesmo atributo para encontrar o ancestral comum. Isso foi um problema especialmente no site Unikka Acessoria, que exibe as informações para área em uma região separada, como pode ser visto na Figura 4.6b, e com isso não fomos capazes de combinar pares de nós detectados para encontrar o ancestral comum.

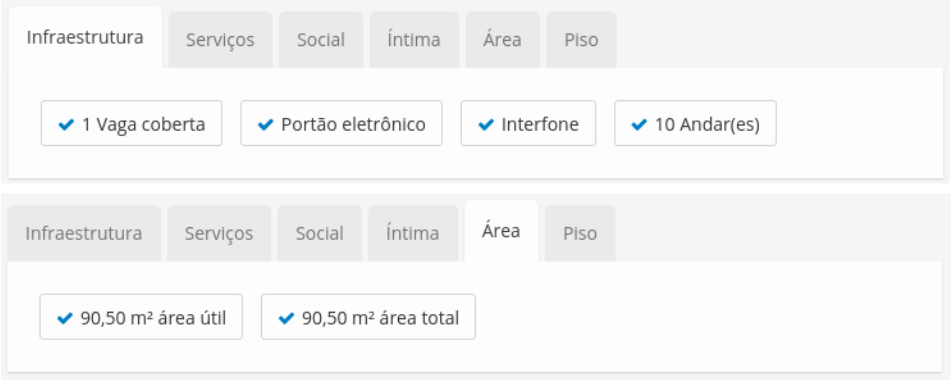


Figura 4.7: Exemplo de informação extra em página do Rede Imóveis.

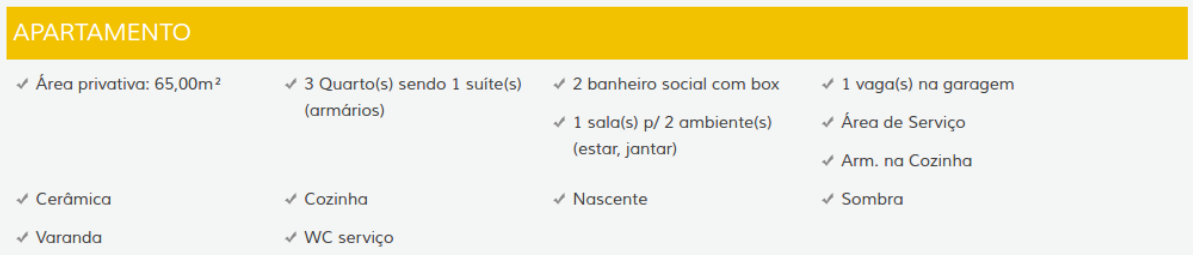


Figura 4.8: Exemplo de informção de difícil extração.

4.5 Avaliação Abrangente

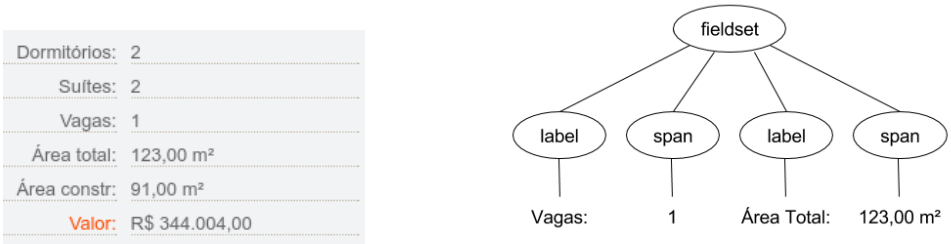
Julgamos necessário complementar nossa avaliação englobando uma quantidade maior de sites, e assim avaliar de forma mais concreta o desempenho do algoritmo em face de uma maior heterogeneidade de sites. Para isso, escolhemos uma página de entidade de cada site para um conjunto de 50 sites não presentes nos conjuntos de desenvolvimento ou de testes, executamos o REX para extrair as informações contidas na página e analisamos manualmente os pares extraídos. No cálculo das métricas, descartamos 5 desses sites, já que em 4 deles a informação estava disponível apenas em texto e o outro exibe os atributos como ícones ao invés de texto, como pode ser visto na Figura 4.9.

Na avaliação, REX alcançou 0.96 de precisão e 0.78 de revocação (0.86 F1). Para os 80% sites de melhor desempenho, a precisão foi de 0.97 e a revocação 0.91 (0.94 F1). Em 25 desses sites REX conseguiu extrair perfeitamente o conteúdo da página.

Acreditamos que esse seja um bom desempenho, já que para alguns sites a informação é exibida de forma que dificulta bastante a extração, como no exemplo da Figura 4.8, onde o texto é escrito de forma parecida com texto livre. Além disso, a precisão é a métrica mais crucial nesse tipo de avaliação.

Um dos casos que causou problemas foi a presença de textos como “3 quartos sendo 4 suítes” e “140m² de área”, onde a forma que utilizamos para identificar o atributo no texto não foi eficaz. Outro problema encontrado foi a possibilidade de um valor ser identificado como

Figura 4.9: Exemplo de ícones sendo utilizados para representar atributos.



(a) Informação exibida no site

(b) Parte da árvore HTML

Figura 4.10: Exemplo de informação onde todos atributos e valores são filhos do mesmo nó raiz.

atributo, por exemplo pela presença de um par como “tipo = condomínio”, fazendo com que outros valores também sejam identificados como atributos. Uma mudança simples que teria trazido um resultado melhor para um site seria um melhoramento no detector de atributos base, substituindo o padrão da regra de “condomínio” para “cond”, para lidar com a abreviação.

Para uma das páginas onde nenhum par foi extraído, apenas os atributos suítes, vagas e vendas estavam presente, de forma que apenas o nó contendo vagas foi detectado, e por isso REX foi incapaz de encontrar a região de atributos. Utilizando outra página do mesmo site contendo mais atributos base teríamos sido capazes de extrair corretamente todos os pares. Já para o outro site, a informação é apresentada espalhada em diversas regiões, de forma que não foi possível combinar dois nós para achar a região relevante. Para o terceiro site, todos atributos e valores são filhos diretos do nó raiz, e com isso as subárvores filhas da raiz não contêm os pares atributo-valor (veja um exemplo na Figura 4.10). Para resolver esse problema, poderíamos considerar o texto da subárvore da própria raiz ao invés do texto de seus nós filhos, e assim extrair corretamente todos os pares. Para não prejudicar o desempenho em outros sites, poderíamos adotar essa técnica apenas quando nenhum par fosse extraído para página.

CAPÍTULO 5

Conclusão

Nesse trabalho, apresentamos REX, um extrator automático capaz de extrair dados de entidades estruturadas de diversos sites de um certo domínio. Para iniciar o processo de extração, REX utiliza conhecimento de domínio, detectando nós contendo atributos base na árvore DOM da página. Em seguida, detecta regiões dessa página contendo informações a respeito de atributos e valores da entidade. Para identificar quais atributos estão presentes na página, REX seleciona nós em estrutura parecida com a dos nós detectados no primeiro passo. Por fim, segmenta o texto das regiões detectadas, e extrai os pares atributo-valor a partir do texto segmentado. Em uma avaliação experimental utilizando uma coleção de páginas em 20 sites de imóveis, alcançamos 0.95 para a F1 em um conjunto de sites usado para desenvolvimento, e 0.89 para a F1 em um conjunto de sites não antes visto. Em outra avaliação, utilizando uma página por site para um conjunto de 45 outros sites, REX alcançou uma precisão de 0.96.

Na avaliação experimental, percebemos que um dos problemas encontrados é a precisão na detecção de regiões relevantes. Embora tenhamos resolvido parte do problema utilizando uma heurística baseada no tamanho do texto em subárvores, seria interessante explorar o uso de uma solução que utilize outras propriedades e seja capaz de aprender a classificar uma região como relevante. Outras possibilidades seriam explorar o uso de várias páginas geradas para o mesmo *template* para guiar o processo de extração, além de utilizar a informação contida em páginas de lista. Além disso, seria interessante a partir da extração de informação em algumas páginas, induzir seletores para outras páginas de um mesmo *template*. Também poderia ser feita a extração de outros tipos de informação, como preço e localização, que acreditamos que pode ser feito utilizando técnicas como expressões regulares em conjunto com os outros atributos extraídos atualmente. Por último, a solução poderia ser configurada para outro domínio, de forma a testar a generalidade da mesma.

Referências Bibliográficas

[1] N. Dalvi, A. Machanavajjhala, and B. Pang, "An analysis of structured data on the web," *Proc. VLDB Endow.*, vol. 5, pp. 680–691, Mar. 2012.

[2] E. Crestan and P. Pantel, "Web-scale table census and classification," in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 545–554, ACM, 2011.

[3] M. J. Cafarella, A. Halevy, and J. Madhavan, "Structured data on the web," *Commun. ACM*, vol. 54, pp. 72–79, Feb. 2011.

[4] K. Lerman, L. Getoor, S. Minton, and C. Knoblock, "Using the structure of web sites for automatic segmentation of tables," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, (New York, NY, USA), pp. 119–130, ACM, 2004.

[5] H. G.-M. Arvind Arasu, "Extracting structured data from web pages," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, Inc., June 2003.

[6] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual web information extraction with lixto," in *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, (San Francisco, CA, USA), pp. 119–128, Morgan Kaufmann Publishers Inc., 2001.

[7] N. Kushmerick, *Wrapper induction for information extraction*. PhD thesis, University of Washington, 1997.

[8] V. Crescenzi, G. Mecca, P. Merialdo, *et al.*, "Roadrunner: Towards automatic data extraction from large web sites," in *VLDB*, vol. 1, pp. 109–118, 2001.

[9] B. Liu, R. Grossman, and Y. Zhai, "Mining data records in web pages," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), pp. 601–606, ACM, 2003.

[10] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava, "Dexter: Large-scale discovery and extraction of product specifications on the web," *Proc. VLDB Endow.*, vol. 8, pp. 2194–2205, Sept. 2015.

[11] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang, "Diadem: Thousands of websites to a single database," *Proc. VLDB Endow.*, vol. 7, pp. 1845–1856, Oct. 2014.

[12] L. Barbosa and G. Ferreira, *Extracting Records and Posts from Forum Pages with Limited Supervision*, pp. 233–240. Cham: Springer International Publishing, 2015.

[13] N. Derouiche, B. Cautis, and T. Abdessalem, "Automatic extraction of structured web data with domain knowledge," in *2012 IEEE 28th International Conference on Data Engineering*, pp. 726–737, IEEE, 2012.

[14] A. Arasu and H. Garcia-Molina, "Extracting structured data from web pages," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 337–348, ACM, 2003.

[15] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 10, pp. 1411–1428, 2006.

[16] C.-H. Chang and S.-C. Lui, "Iepad: Information extraction based on pattern discovery," in *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, (New York, NY, USA), pp. 681–688, ACM, 2001.

[17] B. Liu and Y. Zhai, *NET – A System for Extracting Web Data from Flat and Nested Data Records*, pp. 487–495. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).