AMBER: A Domain-Aware Template Based System for Data Extraction

Cheng Wang

Trinity College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2015

Contents

Acknowledgements

It is my honor to spend five years of my life in Oxford, a place full of knowledge, wisdom and inspiration. The four years working towards to a D.Phil of computer science has been a wonderful journey in my life. I would like to devote the following words to gratefully thank those who have inspired me and helped me to get this far.

First of all, I want to thank my supervisor, Georg Gottlob, for offering me the wonderful opportunity to study at Oxford, and for all his guidance, advice, support and encouragement. I could not have done it without him believing in me. His precious guidance, inspiration and life philosophy will stay with me for the rest of my life.

It has been a privilege to be involved in DIADEM group, a group full of brilliant researchers. I want to express my gratitude to five extraordinary computer scientists in DIADEM: Tim Furche, for mentoring me through all my D.Phil study, providing inspirations when I felt lost, and encouraging me when I felt depressed and low self-confident; Giorgio Orsi, for all the inspirations and enlightenments, also for all your valuable time and patience during our countless discussions in the four years; Giovanni Grasso, for guiding my master thesis, helping me getting involved in the DIADEM group always cheering me up when I was upset; Christian Schallhart, for helping me improving coding skills and getting started in research; Andreas Pieries, for sharing your experiences and feelings of research and always cheering me up. Thank you all for your patience and valuable time, I could not get my D.Phil without your help.

I would like to express special gratitude to Xiaonan Guo, for all her kindly help, patience for my endless questions, and for all the funny stories and joyful moments we shared. I could not imagine what my PhD life would be without Xiaonan. My profound thanks to other DIADEM members: Andrew Sellers, Andrey Kravchenko, Omer Gunes, Stefano Ortona, and Jinsong Guo. I will always remember when you gave me a surprise cake decorated with blueberries as two rings to celebrate my engagement, our "DIADEM world cuisine tour", chocolate rabbits for easter and tons of cakes in Christmas season. Your friendship is very precious, and I could not be more luckily to join such a warm and friendly group.

Special thank to Thomas Lukasiewicz, my transfer, confirmation and final dissertation examiner. Thomas has provided great suggestions for my research in every stage. Thank you to Georg Lausen for being my dissertation examiner and helped me improving my thesis.

Thank you very much for my friends who made my Oxford life wonderful: Xiaochong

Zhang, Luyun Jiang, Haiyun Gong, Lin Xiao, Lu Feng, Jue Wang, Binlei Sun, Weiming Zhu, Baoqiang Xu, Yinpu Guo, Yuan Xia, Huajun Zhang, Stela Maricic, Andrea Sellers and Mariaelena Casciaro.

I am very grateful to the Department of Computer Science and Trinity College for offering me a great environment to study.

No words can express my gratitude to my family. Dear Mom and Dad, thank you for your unconditional love, for educating me when I was little, for showing me how to be a honest, dignified and respectable person, for encouraging me pursuing a D.Phil degree and supporting me traveling around the world. Dear Grandma, I am deeply grateful for all your understanding of me being far away from home, and I am extremely proud that I have the coolest ever grandma who knows how to use iPad since 2011. Dear Grandpa, thanks very much for all your blessing, I know you are always here with me, being a great person like you is my goal for life.

Last but not least, my husband Wenyuan Yu, thank you so much for all your love, encouragement and accompany. Your optimism, erudition and sense of humor enriched my life significantly, I am so lucky to have you in my life.

Abstract

The web is the greatest information source in human history, yet finding all offers for flats with gardens in London, Paris, and Berlin or all restaurants open after a screening of the latest blockbuster remain hard tasks – as that data is not easily amenable to processing. Extracting web data into databases for easier processing has been a resource-intensive process, requiring human supervision for every source from which to extract. This has been changing with approaches that replace human annotators with automated annotations. Such approaches could be successfully applied to restricted settings such as single attribute extraction or for domains with significant redundancy among sources.

Multi-attribute objects are often presented on **(i)** Result pages, where multiple objects are presented on a single page as lists, tables or grids, with most important attributes and a summary description, **(ii)** Detail pages, where each page provides a detailed list of attributes and long description for a single entity, often in rich format. Both result and detail pages are having their own advantages. Extracting objects from result pages is orders of magnitude faster than from detail pages, and the links to detail pages are often only accessible through result pages. Detail pages have a complete list of attributes and full description of the entity.

Early web data extraction approaches requires manual annotations for each web site to reach high accuracy, while a number of domain independent approaches only focus on unsupervised repeated structure segmentation. The former is limited in scaling and automation, while the latter is lacked in accuracy. Recent automated data extraction systems are often informed with an ontology and a set of object and attribute recognizers, however they have focused on extracting simple objects with few attributes from single-entity pages and avoided result pages.

We present an automatic ontology-based multi-attribute object extraction system AMBER, which deals with both result and detail pages, achieves very high accuracy (>96%) with zero site-specific supervision, and is able to solve practical issues that arise in real-life data extraction tasks. AMBER is also applied as an important

component of DIADEM, the first automatic full-site extraction system that is able to extract structured data from different domains without site-specific supervision, and has been tested through a large-scale evaluation (>10, 000) sites.

On the result page side, AMBER achieves high accuracy through a novel domain-aware, path-based template discovery algorithm, and integrates annotations for all parts of the extraction, from identifying the primary list of objects, over segmenting the individual objects, to aligning the attributes. Yet, AMBER is able to tolerate significant noise in the annotations, by combining these annotations with a novel algorithm for finding regular structures based on XPATH expressions that capture regular tree structures. On the detail page side, AMBER integrates boilerplate removal, dynamic lists identification and page dissimilarity calculation seamlessly to identify data region, then employs a set of fairly simple and cheaply computable features for attribute extraction. Besides, AMBER is the first approach that combines result page extraction and detail page extraction by integrating attributes extracted from result pages and the attributes found on corresponding detail pages.

AMBER is able to identify attributes of objects with near perfect accuracy and to extract dozens of attributes with > 96% across several domains, even in presence of significant noise. It outperforms uninformed, automated approaches by a wide margin if given an ontology. Even in absence of an ontology, AMBER outperforms most previous systems on record segmentation.

Chapter 1

Introduction

While only a decade ago digital information was often sparse, today, the rapid increase in the amount of published information and the effects of this abundance of data has transformed the world-wide web into the largest repository of data, spread over hundreds of thousands of sites. In the United States alone, the number of online shopping sites with $10k+ revenue is estimated in excess of a hundred thousand [61], with a long-tail of several hundred thousand smaller shops. Unfortunately, all of the data in these shops are only available as HTML with visions of the "web of data'"" remaining elusive. Extracting data from as many sources as possible is crucial for business, ranging from price comparison engines, where shops supplied data are enhanced by crawled results, to DBLP[10], where wrappers extract additional information to enhance partial records provided by publishers.

Reverse engineering the structured data underlying HTML product listings or single product pages or other HTML-only interfaces is the purview of (structured) *data extraction*. This contrasts to text-based information extraction [29] which extracts entities and facts about entities from only flat text. Commercial data extraction systems are able to reach high accuracy, but through supervised extraction [5, 53] where humans must provide training data for each site or template. Almost since its inception, the need for automated data extraction has been acknowledged, but uninformed automated data extraction systems [1, 17] have been lacking in accuracy and have been limited to simple objects and structures [85]. More recently, *informed, but automated approaches* have been investigated, e.g., for single-attribute extraction [23] and for single-entity pages [25, 43], where a single object is presented in detail. These approaches are informed by the expected schema and recognizers for the entities and attributes in that schema. Recognizers come either in the form of a background instance database [42] for relevant objects, or are ontology-based, such as an annotator for individual attributes in the form of dictionaries, regular expressions, or similar facilities.

Though these informed approaches have considerably improved accuracy over previous

uninformed, automated approaches, they have been focused on fairly simple objects. To gather complete information of objects, the extraction should consider the following two categories of pages:

(1) *Result Pages*, where summaries of multiple objects including their most important attributes, in an organized structure such as list, grids or tables.

(2) *Detail pages*, which are single-entity pages that provide the most complete list of attributes with various format and rich text, but might be lacked entirely on some websites.

Object extraction from result pages and detail pages are both crucial. By exploiting result pages, the number of pages to consider and thus the resource cost for the data extraction can often be reduced by an order of magnitude, and one can acquire the most important attributes of objects, especially the links to the detail pages which are usually only accessible through result pages. By exploiting detail pages, we can obtain a full, complete list of attributes of objects.

1.1 AMBER

This thesis presents AMBER, the first ontology-based, automated *multi-attribute object extraction* system that extracts data from both result pages and detail pages at *very high accuracy* ($> 96\%$), yet requires no site-specific supervision. It is able to automatically identify, segment, and extract data from both result and detail pages for any site in a domain, once provided with a simple ontology of that domain. Specifically,

(1) AMBER focuses on both result pages and detail pages, rather than informed approaches who only deal with one kind of page.

(2) AMBER requires no site-level supervision, whether with a background labeled training set or manually annotate a template for each site.

(3) AMBER only requires a domain-specific knowledge base, and then is able to extract multi-attribute objects from most websites of the domain.

(4) AMBER combines result page and detail page extraction process, utilize the data extracted from result page to boost detail page extraction process.

These properties set AMBER apart from previous data extraction systems. Section 1.2 further illustrates the space of result pages and detail pages covered by AMBER.

AMBER achieves high accuracy on *object extraction from result page* through a novel domain-aware, path-based template discovery. Previous informed approaches have employed separate pre-processings of the input to find relevant subsets of generated annotations [23] or a post-processing to filter relevant rules [7]. AMBER, on the other hand, informs all parts of the extraction with its ontology of relevant entities and their attributes. In particular, AMBER uses

the concept of *pivot attributes* to identify the primary list of objects (or data area) and segment data areas into individual objects. *Pivot attributes* are easily recognisable attributes that are likely to occur in every object of a domain, e.g., the price in almost all product domains, and are used to recognize the basic template structure. The integration of annotations into the template discovery makes the template discovery more resilient to noise within the template (e.g., interspersed advertisement) and guides the search for regular structures.

On *object extraction from detail page*, AMBER first locates the data area by combining three components: **(1)** *Boilerplate removal*, which removes the static non-informative parts, **(2)** *Dynamic list identification*, which applies AMBER's ability to extract lists of objects from result pages to extract content-related object lists such as advertisements, reviews, or feature lists, and **(3)** *Page dissimilarity calculation*, that computes dissimilarities between different detail pages from the same website (if more than one given) to distinguish static and non-static elements of a templates. AMBER then computes a fairly simple feature model for each potential attribute and determines the best alignment according to an easy to compute, yet effective objective function. Typically features range from features local to an individual record to features correlating the candidate attributes on the detail pages to attributes in the corresponding result page.

AMBER is a major component of the DIADEM system [32, 33], the first full-site data exploration system for complex structured data. DIADEM explores entire web sites fully automatically to find relevant data and induce OXPath [35] wrappers. In DIADEM, AMBER provides template discovery that identifies and segments relevant data and provides input for the separate wrapper induction component. The architecture and components of DIADEM are briefly sketched in [32], however based on the earlier version of AMBER [34]. A more extensive discussion of the work flow and principles of DIADEM is presented in [33], with focuses on the integration and inter-component communication by means of a network of relational transducers. This thesis also includes an extensive evaluation of the aspects of DIADEM relevant to AMBER on over $10k$ web sites from UK and US real-estate and used cars. For each of these websites, if DIADEM can locate one or more result pages, AMBER is employed to detect the template structure of those pages and provide such information to the wrapper induction component. Extracting data from detail pages requires roughly one order of magnitude more resources (as most extraction systems, including DIADEM and AMBER scale with the number of visited pages) and was thus not feasible at this large scale. Section 7 shows the results of this large-scale evaluation.

1.2 AMBER by Examples

1.2.1 Result Page Extraction

We illustrate the process of result page extraction through an example result page from Rightmove, the biggest UK real-estate aggregator. Figure 1.1 shows the typical parts of such pages: On top, **(1)** some featured properties are arranged in a horizontal block, while directly below, separated by an advertisement, **(2)** the properties matching the user's query are listed vertically. Finally, on the left-hand side, a block **(3)** provides some filtering options to refine the search result. At the bottom of Figure 1.1, we zoom into the third record, highlighting the identified attributes.

After annotating the DOM of the page, AMBER analyses the page in three steps: data area identification, record segmentation, and attribute alignment. In all these steps we exploit annotations provided by domain-specific annotators, in particular for pivot attribute types, here PRICE, to distinguish between relevant nodes and noise such as ads. AMBER compensates the inherent noise in these annotations by comparing different records and their attributes to find missing attributes and filter out wrongly annotated attributes.

For Figure 1.1, AMBER identifies *price* annotations (highlighted in green, e.g., "£995 pcm"), most locations (purple), the number of bedrooms (orange) and bathrooms (yellow). The price on top (with the blue arrow), the "1 bedroom" in the third record, and the crossed out price in the second record are three examples of false positives annotations, which are corrected by AMBER subsequently.

Data area identification. First, AMBER detects which parts of the page contain relevant data. In contrast to most other approaches, AMBER is able to deal with web pages displaying multiple, differently structured data areas. For instance, in Figure 1.1 AMBER identifies two data areas, one for the horizontally repeated featured properties and one for the vertically repeated normal results (marked by red boxes). Then, AMBER is able to detect that the horizontal one is a secondary data area, and the vertical one is the main data area.

Where other approaches rely solely on repeated structure, AMBER first identifies *pivot nodes,* i.e., nodes on the page that contain annotations for regular attribute types, here *price*. Second, AMBER obtains the data areas as clusters of continuous sequences of pivot nodes which are evenly spaced at roughly the same DOM tree depth and distance from each other, and with a sufficient number of characteristic attributes, i.e., domain-specific attributes that could characterize the domain and distinguish it from others. For example, AMBER does not mistake the filter list (3) as a data area, despite its large size and regular structure. Approaches only

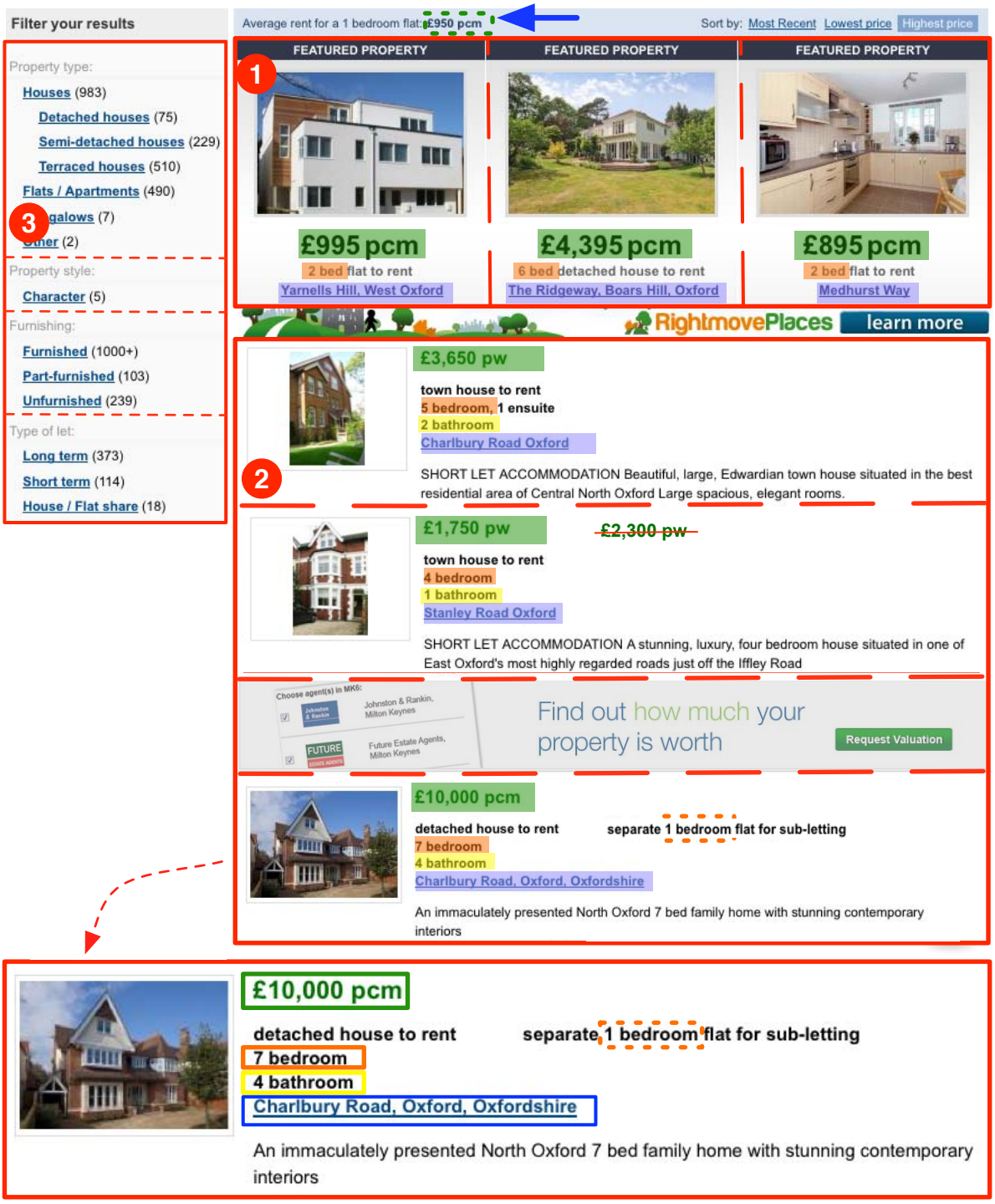


Figure 1.1: Result Page on `rightmove.co.uk`

analyzing structural or visual structures may fail to discard this section. Also, any annotation appearing outside the found areas is discarded, such as the price annotation with the blue arrow atop of area (1).

Record segmentation. On the second stage of result page analysis, AMBER needs to segment the data area into "records", each representing one multi-attribute object, rooted at the same level and with the same length. Unlike most of other approaches which only consider records arranged as lists, AMBER is able to deal with more complex structures of records, such as grid structure.

To this end, AMBER employs a hierarchical segmentation process, executes same-level segmentation from top to bottom, starts from the children of the data area root level, and keeps validating the segmentation with the assists of the annotation process to avoid over-segmenting.

On each level, it is possible that some pivot nodes are actually noisy. AMBER utilizes a dynamic programming process to calculate the maximum and continuous pivot sequence and remove the noisy nodes. The remaining pivot nodes segment the data area into fragments of uniform size, each with a highly regular structure, but additional shifting may be required as the pivot node does not necessarily appear at the beginning of the record. Among the possible record segmentations the one with highest regularity among the records is chosen.

After segmenting data areas using pivot nodes, AMBER generalize the template of existing records (tag similarity and annotation types similarity), matches the template to the rest of nodes in the data area and checks if there are records without regular attributes left.

In our example, AMBER correctly determines the records for the data areas (1) and (2), as illustrated by the dashed lines. It is worth noticing that AMBER prunes the advertisement in area (2) as inter-record noise, since it does not match record pattern, and would lower the segmentation regularity.

Attribute alignment. Finally, AMBER aligns the found annotations within the repeated structure to identify the record attributes. The analysis outcome is twofold, providing first the attribute values of the objects described on the result page, and second, the locations of the DOM nodes containing these attributes. Thereby, AMBER requires that each attribute occurs in sufficiently many records at corresponding positions. If this is the case, it is *well-supported,* and otherwise, the annotation is dropped. Conversely, a missing attribute is inferred, if sufficiently many records feature an annotation of the same type at the position in concern. For example, all location annotations in data area 2 share the same position, and thus need no adjustment. However, for the featured properties, the annotators may fail to recognize "Medhurst Way" as

a location. AMBER infers nevertheless that "Medhurst Way" must be a location (as shown in Figure 1.1), since all other records have a location at the corresponding position. For data area 2, bathroom and bedroom number are shown respectively at the same relative positions. However, the third record also states that there is a separate flat to sublet with one bedroom. This node is annotated as bedroom number, but AMBER recognizes it is false positive due to the lack of support from other records.

To summarise, AMBER addresses low recall and precision of annotations in the attribute alignment, as it can rely on an already established record segmentation to determine the regularity of the attributes. In addition it compensates for noise in the annotations for regular attribute types in the record segmentation by majority voting to determine the length of a record and by dropping irregular annotations (such as the crossed out price in record 2). AMBER also addresses noise in the regular structure on the page, such as advertisements between records and regular, but irrelevant areas on the page such as the refinement links. All these comes at the price of requiring some domain knowledge about the attributes and their instances in the domain, that can be easily acquired from just a few examples, as discussed in Section 4.7.

1.2.2 Detail Page Extraction

We illustrate the detail page extraction process of AMBER in this section with a detail page from a UK real-estate agency website http://www.oliverjames.co.uk in Figure 1.2 and its corresponding record from the result page in Figure 1.3. First of all, the link to the detail page is extracted from the record, on the bottom right, "View Full Details".

Data Area Identification Similar to result page extraction, AMBER detects which part of the detail page is the data area, defined in section 6.3 as the smallest data region containing all attributes of the object. However, unlike result pages, detail pages do not have a repeating structure and hence could not utilize pivot attributes. AMBER combines three components, boilerplate removal, dynamic list identification and page similarity computation to detect the data area.

A boilerplate removal component detects the non-informative parts, such as header, footer, banner, navigation menus. In Figure 1.2, the data area is highlighted in pink. The navigation menu above the data area can be eliminated from the data area through boilerplate removal process. On some websites, a list of "nearby properties" (or advertisements, recommendations, etc.) are included in the detail page. Since they contain objects from the same domain, they can be a source for confusion in extraction. AMBER utilizes the result page extraction process to

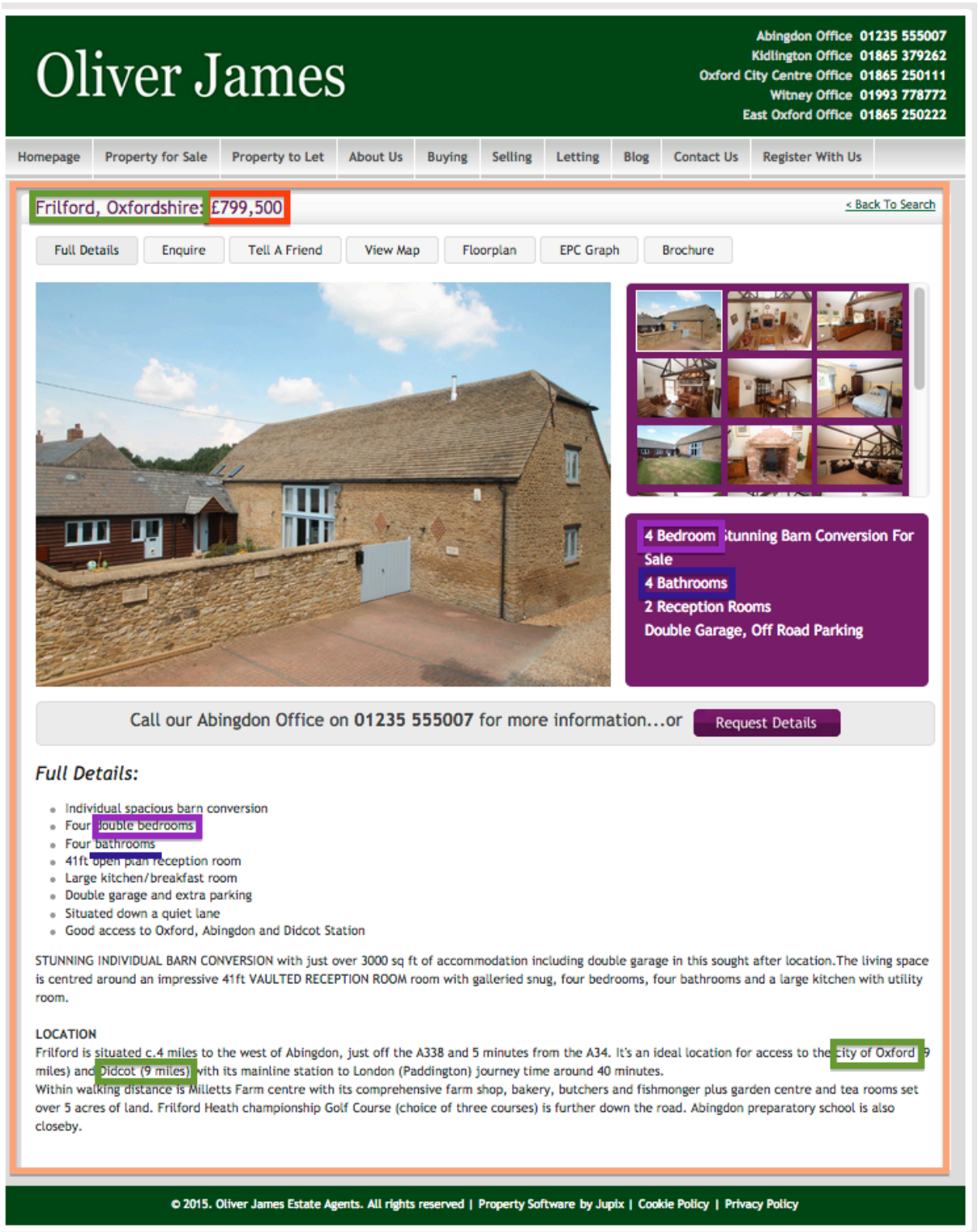


Figure 1.2: A detail page from Oliverjames

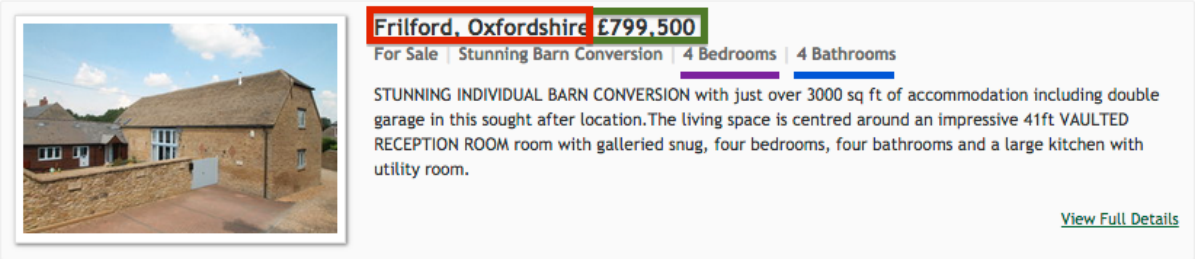


Figure 1.3: Corresponding record to detail page 1.2

identify those dynamic lists and excludes them (or reports them as nested records, if useful). Finally, if more than one detail page from the same website are given as input, AMBER computes the similarity of the pages based on both dom structure and textual content, then distinguishes non-static nodes that are more likely to contain attributes of the entity from static nodes, which are only relevant as context (e.g., labels) for the non-static nodes. In the example detail page, the list of office addresses on the top right is an example of a static node. With those three components, AMBER is able to detect the pink area as the data area.

Attribute Extraction With the data area identified, AMBER needs to extract the optimal attribute for each type, defined via a simple objective function on top of a feature model for attributes. We can see that all attributes contained in the result page record (Figure 1.3) are also present in the corresponding detail page (Figure 1.2), such as price, location, bedroom and bathroom. However, the detail page contains more attributes, such as "Double Garage", "Off Road Parking", and the branch location "Abingdon Office" with the telephone number "01235 555007". AMBER uses the correlation to the result page to disambiguate candidate attributes of the same type. In this example, there are many locations in the data area (highlighted in green). Only the one in the title is the actual location of the property. Through correlation to the major attributes extracted from result page, AMBER is able to figure out which is the optimal location. For the bathroom attribute type, both attribute instances have the same value. However, the one in the purple box has larger font size, and has support by other detail pages in that they all have a bathroom attribute in the same position. The feature selection is described in more detail in Section 6.5.

1.3 Contributions

This section presents AMBER's main contributions as following:

(1) AMBER is the first informed, automated data extraction system focused on both result pages and detail pages with complex objects. It achieves very high ($> 96\%$) accuracy, signif-

icantly outperforming previous (uninformed) automated approaches for object extraction, see Section 7.1.3.

(2) AMBER is able to extract complex objects with many attributes ($>$ 10 per domain). Attributes may be structured (e.g., locations with possibly optional sub-components of street address, city, or postcode), optional (where only some of the records or pages contain the attribute, e.g., an indicator of new or sold products), or may have overlapping instances where labels are necessary to distinguish instances of different attributes (e.g., bedroom and bathroom numbers), as well as overlapping instances where formatting and relative placement in the record are necessary for disambiguation (e.g., multiple prices for special offer or lease-related prices), see Section 1.2.

(3) AMBER requires no per-site supervision whatsoever, neither manually labeled training corpus nor a generated template. All AMBER needs is a domain-specific knowledge base, then it could extract multi-attribute objects from both result and detail pages on most sites (>95%) of the domain.

(4) AMBER achieves high precision and recall on result page extraction through novel domain-aware template discovery. Attribute types occur in most objects of the domain, such as the price in most product domains, are called "pivot nodes". Those instances are served as "pivots" for the template discovery algorithm, significantly boosting its ability to distinguish relevant data from irrelevant, but regular structures. Even without a domain ontology provided, AMBER could still use generic attributes such as urls, images as pivot.

(5) AMBER reaches high precision and recall on object extraction from detail pages thanks to a combination of three components for data area identification with a feature model similar to the one used for result pages: **(a)** Dynamic list identification, which identifies dynamically generated lists similar to result page lists. **(b)** Boilerplate removal, removes static non-informative parts, such as navigation, menus, headers, footers and sidebars. **(c)** Page dissimilarity calculation, computes page dissimilarities with the assist of domain annotations to distinguish static parts of the page and boosts the accuracy of attribute extraction.

(6) On both result and detail page extraction, AMBER adopts a set of fairly simple and cheaply computable features that are computed for each DOM node, ranged from page-level to node level, to assist optimal attribute selection. AMBER is also able to infer unknown attribute instances based on support from repeated structures (records, detail pages) and enlarge the gazetteer in the ontology.

(7) AMBER is the first approach that integrates result page and detail page extraction. Result page provides links to detail pages and major attributes that could be used as a key feature in the detail page attribute selection process. Besides, the result page extraction process is also

applied for extracting dynamic lists from detail pages.

(8) In an extensive evaluation, we substantiate these claims and demonstrate that AMBER is indeed able to extract data automatically from a large number of websites. Our evaluation (Chapter 7) shows that **(a)** AMBER is able to extract attributes with $> 96\%$ from modern web sites in multiple domains. **(b)** AMBER outperforms previous uninformed approaches by a large margin on modern web sites. **(c)** AMBER is also able to identify and segment objects from result pages effectively without any domain knowledge but only using generic attributes such as the detail page URL or aligned images as pivot attributes. **(d)** AMBER's record segmentation outperforms the reported performance of previous approaches on a benchmark dataset even without any domain knowledge in most cases. It matches the performance of the best performing previous approach [6], without any training (as employed by [6]) or other adaptation. **(e)** AMBER is able to distinguish result pages with relevant data from other pages as well as advertisements of such objects with very high accuracy. **(f)** We carefully validate the impact of various design decisions in AMBER, e.g., the choice of the pivot attribute for different domains, the choice of features for attribute scoring, and all validation thresholds. **(g)** AMBER continues to perform at or above 90% accuracy even in presence of 50% noise in the annotations.

(9) Finally, AMBER is a crucial component of DIADEM, the first automatic full-site extraction system that is able to extract structured data from different domains without site-specific supervision. **(a)** DIADEM reaches accurate extraction (97%) of highly structured data on most sites, demonstrated by an evaluation over a diverse set of 10602 websites from UK and US real-estate and UK used car (Section 7.2). **(b)** DIADEM achieves this automation with the assist of a novel kind of domain knowledge, which is provided once for the entire domain and is the only form of supervision in DIADEM. **(c)** DIADEM integrates several different components such as exploration, identification and a self-adaptive synchronized network of relational transducers seamlessly.

1.4 Organization of the Thesis

AMBER as presented in this thesis has grown from earlier work by the authors on rule-driven result page analysis [34]. This work introduced the idea of pivot attributes, but used a considerably simpler template discovery and ad-hoc detection of regularities (rather than a systematic characterisation through XPath expressions). It also used a less capable attribute alignment and presented only a preliminary evaluation focused on regular attributes in list-structured result pages. The self-boosting part of AMBER is summerized in [37].

AMBER is a major component of the DIADEM system [32, 33], the first full-site data ex-

ploration system for complex structured data. DIADEM explores entire web sites fully automatically to find relevant data and induce OXPath [35] wrappers. In DIADEM, AMBER provides template discovery that helps to identify and segment detect relevant data and provides input for the separate wrapper induction component. The architecture and components of DIADEM are briefly sketched in [32, 36], however based on the earlier version of AMBER [34]. A more extensive discussion of the workflow and principles of DIADEM is presented in [33], with focuses on the integration and inter-component communication by means of a network of relational transducers. This paper also includes an extensive evaluation of DIADEM on over 10k web sites from UK and US real estate and used cars. For each of these websites, if DIADEM can locate a result page, AMBER is employed to detect the template structure of those pages and provide such information to the wrapper induction component.

In Chapter 2, we formulate the object extraction problems and provide preliminaries. Chapter 3 discusses existing research approaches related to AMBER. Chapter 4 provides a detailed description of the algorithms AMBER employs for object extraction from result page. Followed by a brief introduction of DIADEM system where AMBER acts as a major component, discussed in chapter 5. Chapter 6 explains how AMBER extract attributes from detail pages. Chapter 7 lists evaluation results of AMBER, on result pages, detail pages and as part of DIADEM. At the end of the thesis, the conclusion and future work are discussed in Chapter 8.

Chapter 2

Multi-Attribute Object Extraction Problem

The web now offers complete and precise market data from entire consumer domains, such as the UK real-estate market with its approximately 17,000 different agencies. From international large aggregator, to small local business, object is a common way to present data. To integrate those enormous amount of objects into comprehensive and continuously maintained domain databases, whether for archiving, search or analysis, we need to automatize web data extraction without compromising accuracy.

We divide the multi-attribute object extraction problem into two parts, **(1)** object extraction from result pages and **(2)** object extraction from detail pages. This chapter defines the problem of object extraction, discussed in section 2.1 and section 2.2 respectively.

2.1 Object Extraction From Result Pages

A result page is a page containing at least one repeated structure of complex multi-attribute records, often in the form of a sequence but arranged in different formats, such as lists, grids or tables. On many sites, a large number of result pages are returned as a response to a form query on a web site, and they can only be accessed through a query form.

Figure 2.1 shows four result pages where records are aligned in different formats, namely list, table, list and table combined, grid. We observe that:

(1) A set of records may contain interspersed separators, advertisements, section headers or other information inserted between objects.

(2) A set of records may appear as a list, where objects are aligned either horizontally or vertically, or as a grid, where objects are arranged both horizontally and vertically, or as a table, where some parts of the objects, typically the label of attributes, are rendered only once in the header or footer.

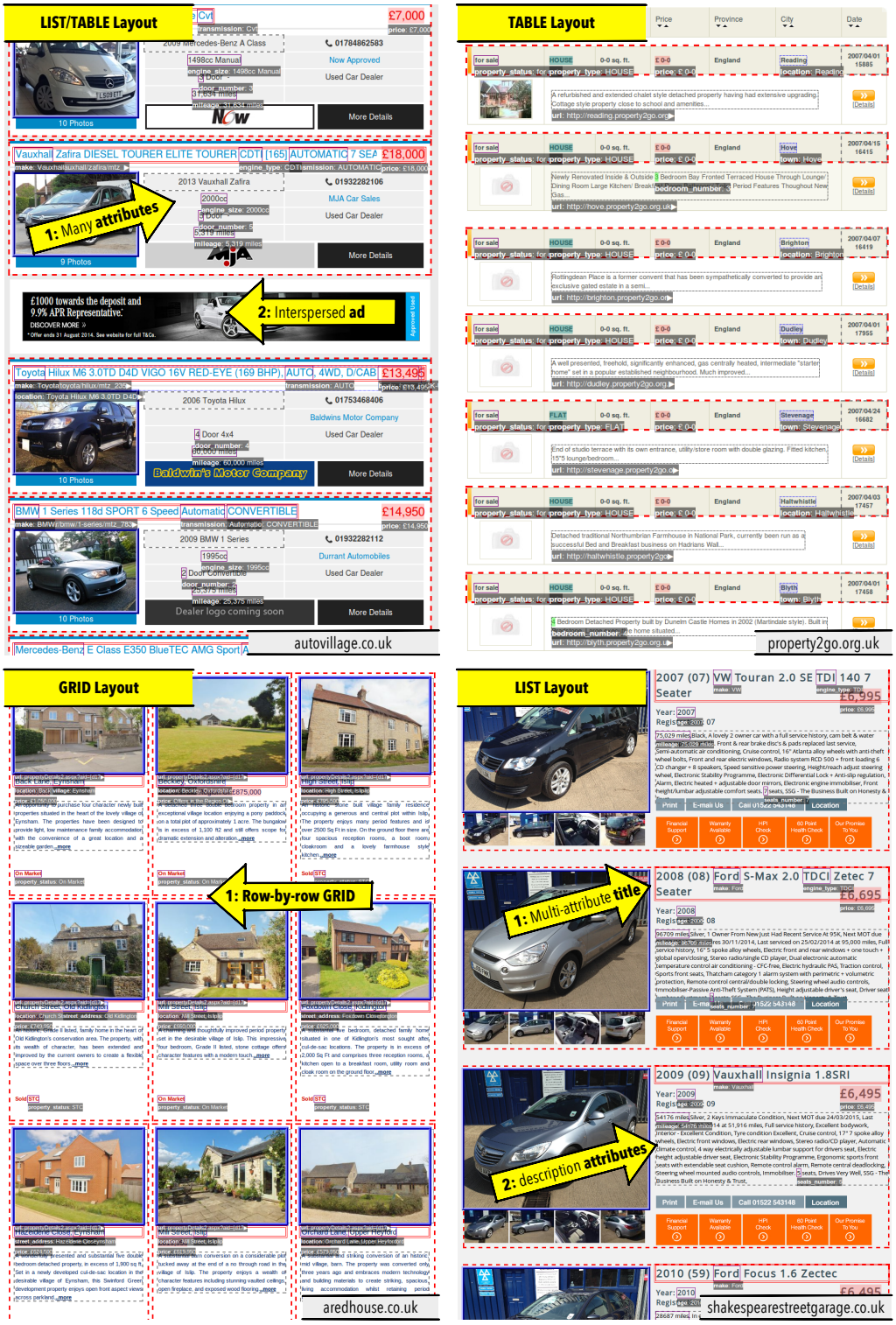


Figure 2.1: Various Result Pages

(3) A large number of attributes are optional, i.e., only some of the objects on the result page contain the attribute. Attributes are noisy, i.e., the same type of attributes may appear in a single object description more than once, either repeating with the same value or with different values, (e.g., "2 bedrooms" v.s. "There is one bedroom located on the second floor").

(4) It is possible that one attribute spans several DOM nodes, or one DOM node contains multiple attributes of the objects.

To scale result page analysis to thousands of sources, no per source supervision is feasible to be used. In particular, we can not assume a collection of pages of a specific type or following a specific template as input, or assume each attribute occur in each object description once and only once. To solve the scalable result page object extraction problems, dealing with the multitude of different templates and tolerating optional attributes or slight irregularities in records are key factors.

Therefore, the problem of object extraction from result page has been defined as following:

Definition 1. Given a result page P and an annotation schema $\Sigma = \langle \Sigma_C, \pi, \Sigma_O \rangle$ as input, where Σ_C is the set of *characteristic* attribute types, representing attributes that characterize the entity that is targeted by the extraction, Σ_O is the set of *optional* attribute types, representing attributes that can be optional in a record and for which no minimal support is expected, and π represents the pivot attribute type, the result page extraction problem has been divided into three steps:

(1) *Data area identification*, which locates the data areas of the result page, each data area is a smallest area covering a maximum set of objects with repeating structures.

(2) *Record segmentation*, segments each data area into separate records, each associated to one object.

(3) *Attribute alignment*, extract relevant attributes from the records, exclude noises and duplications, and locate those attributes explicitly in the DOM nodes.

2.2 Object Extraction From Detail Pages

A detail page is designed for displaying a single object with a full list of attributes and detailed descriptions, but sometimes also include related advertisements, related products, or reviews. On many sites, detail pages are hidden deeply, only accessible through links from corresponding records of result pages. Object extraction from result pages is often significantly faster, however the attributes extracted from result pages may not be complete. To extract a complete list of attributes, extracting from detail pages is the only way.

Figure 2.2 illustrates an example of detail page. Each detail page only illustrates a single object, but surrounded by many other blocks:

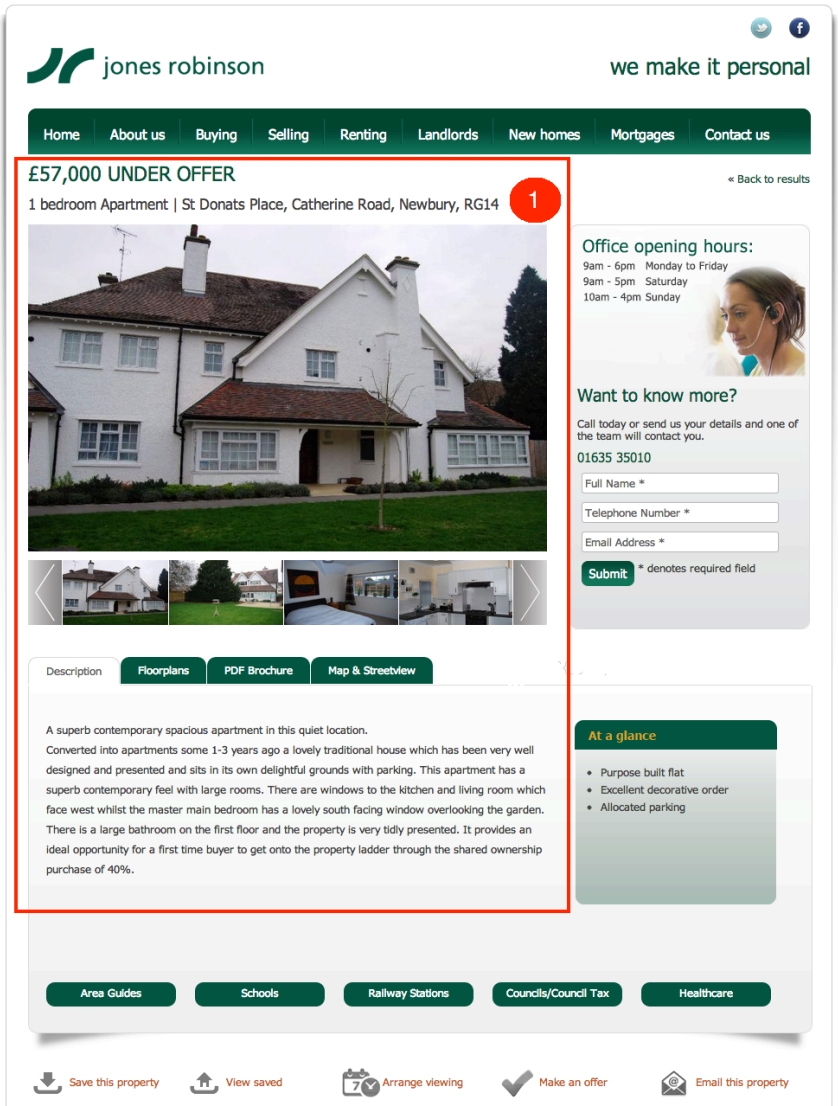


Figure 2.2: An example detail page from http://www.jonesrobinson.co.uk

(1) Static blocks, usually generated by the template of the website, providing exactly the same information for each detail page, such as the banner, header, footer, menu, navigation links, etc.

(2) Non-informative functional blocks, which are generated dynamically and may vary on different detail pages, such as the refinement form, and advertisements.

(3) Content-based dynamic blocks, usually showing content-related objects as in-site promotions, such as "properties near by", or advertisements from third party, such as Google AdSense, which allow publishers to show automatic text, image, video, or interactive media advertisements.

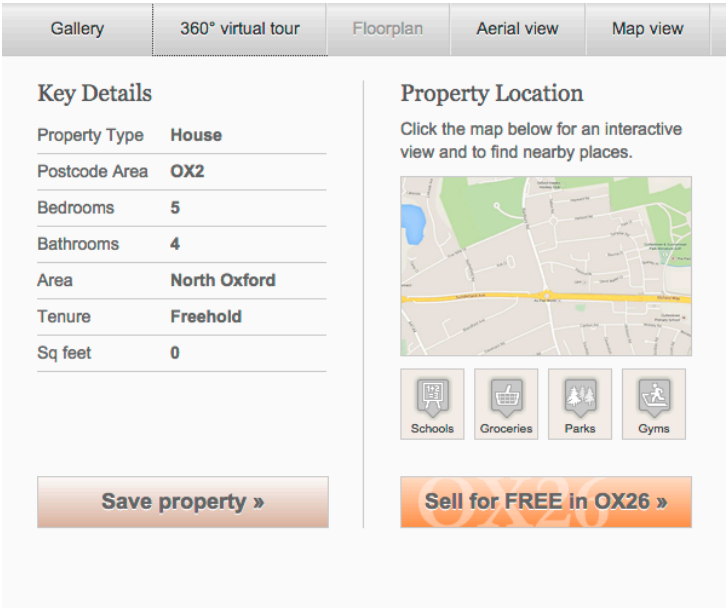


Figure 2.3: Detail page data area from http://www.wwagency.co.uk

On the attribute level, we observe:

(1) Structurally, attributes occurring on a detail page may vary in format, both in their textual presentation and in how they appear in complex structures such as images, tables, list structures, hyper links, animations, or even third-party widgets. In Figure 2.3, the left column illustrates a list of attributes, however the coordinates of the property is shown through Google maps.

(2) Textually, detail pages often provide long descriptions, which are full of attribute duplicates and noises. Similar to result pages, attributes on detail pages are also often optional and irregular, but having even more duplicates, noise, and irregularities.

The *input* for the detail page extraction problem is slightly more involved than in the result page case, most importantly as we assume the presence of a result page with records linked to the detail pages. While AMBER is able to extract attributes from detail pages also without

this information, it can significantly improve the quality in particular for sites with sparse or irregular detail pages. The inputs are:

(I1) A set of detail pages $\mathcal{P}$, $|\mathcal{P}| \geq 1$;

(I2) A set of result page records $\mathcal{R}$, where each R_i in $\mathcal{R}$ corresponds to one detail page P_i in $\mathcal{P}$ given, and this is optional;

(I3) A domain *annotation schema* $\Sigma = \langle \Sigma_C, \Sigma_O \rangle$ where Σ_C is the set of *characteristic* attribute types, representing attributes that characterize the entity that is targeted by the extraction and Σ_O is the set of *optional* attribute types, representing attributes that can be optional in a detail page and for which no minimal support is expected.

(I4) A set of pre-defined features $\mathcal{F}$ for attribute alignment.

Definition 2. Given input **[(I1)-(I4)]**, the detail page extraction problem has been divided into two steps:

(1) For each detail page P in $\mathcal{P}$ (**I1**) and the record in the corresponding R_i (**I2**), locate the data area D, which is the least area P containing all attributes of the object.

(2) For each data area D, extract the relevant attributes of the object based on the annotation schema Σ **(I3)** and align those attributes using features $\mathcal{F}$ (**I4**) given.

Chapter 3

Related Work

In this chapter, we consider related researches and contrast them to AMBER, outlining our original contributions to this field. Section 3.1 introduces three categories of result page analysis related approaches, **(1)** wrapper induction approaches, **(2)** fully automatic approches and **(3)** combined approaches. Section 3.2 discusses detail page analysis related approaches through **(1)** structure analysis approaches, **(2)** instance-based approaches, **(3)** automatic annotation-based approaches and **(4)** model-based approaches.

3.1 Result Page Analysis

Web data extraction is the task of recognizing and extracting data on the web that is structured by regular HTML markup and visual styling. This is often the case when web content is automatically generated by populating templates with data from underlying databases [10]. The use of structural features sets web data extraction apart from traditional information extraction [29] where entities and their relations are extracted from free text. It also sets it apart from web page segmentation [30] and block classification [11, 24] where the goal is to recognize the logical structure of the page, e.g., to distinguish content areas from advertisements and navigational blocks such as forms, menus, and pagination links. See [71] for a recent survey on this problem.

Early web data extraction approaches address data extraction via manual wrapper developments [16, 46] or through visual, semi-automated tools [5, 47, 54] (still commonly used in industry). Modern web data extraction approaches, on the other hand, overwhelmingly fall into one of two categories (for recent surveys, see [13, 55]): *Wrapper induction* [22, 31, 42, 48, 49, 52, 53, 64] starts from a number of manually annotated examples, i.e., pages where the objects and attributes to be extracted are marked by a human, and automatically produce a wrapper program which extracts the corresponding content from previously unseen pages. *Unsupervised wrapper generation* [18, 50, 59, 60, 70, 73, 79, 84] attempts to fully automate the extraction

process by unsupervised learning of repeated structures on the page as they usually indicate the presence of content to be extracted.

Unfortunately, where the former are limited in automation, and the latter are lack in accuracy. This has caused a recent flurry of approaches [23, 25, 69] that like AMBER attempt to automatize the production of examples for wrapper inducers through existing automatic annotators. Where these approaches differ most is how and to what extend they address the inevitable noise in these automatic annotations. AMBER relies on a structured-named-entity recognizer SNER , a surrogate for both template-based [41, 57] and instance-based [7, 9] redundant information as it is implicitly encoded in the method each SNER uses to produce annotations. On the other hands, Skoga [6] is a distant supervised approach that based on global analysis of DOM structure knowledge for data region and record extraction.

3.1.1 Wrapper Induction Approaches

Wrapper induction can deliver highly accurate results which provide correct and complete input annotations. The process is based on the iterative generalization of properties (e.g., structural and visual) of the marked content on the input examples. Some labeled pages or sites are required as input of the domain to perform the induction, then the wrapper induction systems automatically learn the pattern and features, finally apply them to extract records from new pages or websites.

The earliest web data extraction approaches address the data extraction problem via manual wrapper development. Languages, visual, and semi-automated tools were designed to assist user in constructing wrappers manually. Different approaches have been pursued to this end: TSIMMIS [46] provides specification language for user to write wrappers that work on the text, Minerva [16] presents a formalism for writing wrappers, attempts to combine the advantages of a declarative grammar-based approach with the flexibility of procedural programming.

These manual-coding style of the wrapper is costly, since it is time consuming and requires a non trivial previous knowledge of web technologies. Furthermore, it is neither scalable nor robust, but brittle and difficult to maintain when facing the changes of web sites.

Due to such limitations, efforts have been made to automate the wrapper generation process. Later, *semi-automated* tools have been introduced to reduce the manual effort required from wrapper developers. These systems replace most of the low-level coding with a GUI-supported or interactive wrapper definition that automatically produces parts of the extraction code from manually marked *examples*, leaving to the developer only the refinement of the produced wrapper. Some best known semi-automated approaches are listed as follows.

Lixto [5] helps users to create wrappers interactively by providing a GUI and letting users

select relevant pieces of information visually. DEByE [54] drops the fashion top-down extraction strategy, provides a GUI for users to mark attribute values only, then adopts a bottom-up extraction strategy to assemble the marked atomic components(attribute) into objects. Thresher [47], which allows users to highlight and label examples of semantic content, and uses tree edit distance to generate a wrapper. SoftMealy [48] is based on a finite-state transducer and contextual rules, STALKER [64] introduces a embedded catalog formalism to describe web pages and performs hierarchical data extraction.

The structure of the required example annotations differs across different tools, impacting the complexity of the learned wrapper and the accuracy this wrapper achieves. Approaches such as SRV [31], a top-down greedy rule learner that generates single-slot extraction rules, and Kosala et al. presented [52] which operates on *single attribute* annotations, i.e., annotations on a single attribute or multiple, but a-priori unrelated attributes. As a result, the wrapper learns the extraction rules independently for each attribute, but, in the case of multi-attribute objects, this requires a subsequent reconciliation phase. SoftMealy [48], WIEN [53], STALKER [64] are based on *annotated trees*. The advantage w.r.t. single-attribute annotations is that tree annotations make the nested structures recognition task much easier.

Some modern wrapper induction systems integrate machine learning techniques to improve the quality of wrappers. The learning algorithms infer generic and possibly robust extraction rules in a suitable format. For instance, Dalvi et al. [22] presented an probabilistic tree-edit model to generate robust XPath wrappers. Vertex [42] uses a greedy algorithm on picking structurally diverse sample pages for annotation , clusters similar structured pages, and produces XPath expression for extracted attribute values.

Semi-automated wrapper induction approaches smoothed the wrapper engineering process, and represented a noticeable advancement in the field from a technological point of view. However, they did not solve the underlying issues: wrapper design is not yet scalable since the wrapper is still built on a per-site basis, and the wrapping refinement still requires considerable work and knowledge of the technologies.

By itself, wrapper induction is incapable of scaling to the web. Because of the wide variation in the template structures of given web sites, it is practically impossible to annotate a sufficiently large page set to cover all relevant combinations of features indicating the presence of structured data. More formally, the *sample complexity* for web-scale supervised wrapper induction is too high in all but some restricted cases, as in Wang et al. [79] which extracted news titles and bodies. In case of multi-attribute extraction, the number of relevant subsets may quickly become intractable. Crowdsourcing wrapper validation such as Crescenzi et al. [19] is effective but still requires a large number of workers to obtain accurate wrappers. Furthermore,

traditional wrapper inducers are very sensitive to incompleteness and noise in the annotations thus requiring considerable human effort to create such low noise and complete annotations.

3.1.2 Automatic Web Data Extraction Approaches

The completely unsupervised record level object extraction has been based on discovering regularities on pages presumably generated by a common template. Works such as [18, 50, 58, 59, 63, 70, 84, 85] discuss *domain-independent* approaches that only rely on repeated HTML markup or regularities in the visual rendering.

Two early approaches RoadRunner [18] and ExAlg [1] extracts attributes only, however, they do not identify records or data areas. RoadRunner takes multiple HTML pages as input, discovers patterns based on the similarities and dissimilarities between pages and identifies relevant attributes. EXALG also requires a collection of web pages generated from a common template as input, discovers unknown template and extract attributes.

The most common task that can be solved by these tools is *record segmentation*, where an area of the page is segmented into regular blocks each representing an object to be extracted. One of the earliest data record extraction approach OMINI [8] is based on a set of heuristics to identify record separator tags between objects. Those heuristics limits OMINI such that it is only applicable to simple, well-formed (e.g., tags must be in pairs, every start tag must have a corresponding ending tag). If the separator tags occur in the middle of the record, such as paragraph breaker or placeholder, that may cause a problem for OMINI. IEPAD [14] discovers record boundary by repeated pattern mining and multiple sequence alignment. It encodes all HTML tokens into a binary string sequence and find frequent patterns through a data structure called PAT tree. Similar to OMINI, those early approaches are limited by the heuristics, and some heuristics are not applicable on modern web pages. DeLa [78] gives a attempt on allowing nested repetition and employs regular expression wrappers to extract data records and attributes. However, it assumes data objects are generated by common templates with repeated structure. That assumptions leads to very low tolerance for optional attributes, which occur frequently in modern result pages. MDR [58] is based on the assumption that a large majority of web data records are formed by tables or form related tags such as table, form, tr, td, etc, and mines data record from such structures. However, along with the development of web design, the structure of data records turned to be more complicated and the assumption does not hold on a large number of cases. FiVaTech [50] formulates the page generation model using an encoding scheme based on tree templates and schema. It takes several result pages as input, merge their DOMs into a structure called fixed/variant pattern tree to identify the template and detect data schema of a website. However, they assume templates are fixed for the same data item, which

reduce its flexibility, and may not be able to deal cases having many variants, such as manually edited pages.

Different to OMINI, IEPAD, MDR and FiVaTech which only rely on the HTML tag structure, ViPER [70], VIDE [59], TPC [63], DEPTA [84] and ViNTs [85] extract data records with the help of visual information. DEPTA is an improvement of MDR, enables visual features to extract data records, and adopts a partial tree alignment method to align attributes. ViPER incorporates users' visual perception of the web page for data records extraction and also adopts a global multiple sequence alignment technique for attribute alignment. VIDE utilizes only the visual content features on the result page to extract data records. VIDE's feature selection is based on a series of assumptions, however, after several years, some of them are not always hold, e.g., they claim data areas are always centered horizontally. TPC utilizes some visual information to assist record segmentation, but mainly focuses on identifying repeated occurrence of distinct tag path (i.e., tag path from root to leaf), and is able to extract non-consecutive and nested data records. ViNTs combines both visual content features and HTML tag structures and identifies data area and records from query pages. However, ViNTs reports only one data area of each page, and could only extract records that are separated horizontally. Unfortunately, these systems are quite susceptible to noise in the repeated structure as well as to those regular but irrelevant structures such as navigation menus. This limits their accuracy severely, as also demonstrated in Chapter 7.

After the record identification process, DeLa, ViPER, DEPTA, VIDE and MDR also extract attributes from data records. However, the attributes extracted by these fully automatic systems are not accurate. A significant number of garbage information might be contained in the extracted attributes, such as meaningless pieces of text ("click here"), static information occurring in all the records (static images, texts). Therefore, a posteriori cleanup process is required before applying the extracted attributes, a task that is surely challenging at web scale. In addition, most of the extracted attributes are not classified or semantically labeled, and could not be turned into knowledge for further usage. Two recent approaches [60, 74] attempt to circumvent these problems by employing generic annotators capable of recognizing common data types, e.g., integers and urls, to identify data-rich structures. However, the use of generic annotations create several problems as different attributes might have the same datatype. In particular, being based on tag and value similarities, Su et al. [74] are unable to distinguish between attribute labels and values, while Lu et al. [60] require that the record structure has been already identified by other means.

A complementary line of work deals with specifically stylized structures, such as *tables* Cafarella et al. [9] and Gatterbauer et al. [38], and *lists* Elmeleegy et al. [27]. The more clearly

defined characteristics of these structures enable domain-independent algorithms that achieve fairly high precision in distinguish genuine structures with relevant data from structures created only for layout purposes. They are particular attractive for use in settings such as web search that optimize for coverage over all sites rather than recall from a particular site.

Instead of limiting the structure types to be recognized, some approaches such as Zhu et al. [87] exploit statistical models and machine learning techniques. [87] proposed a Hierarchical Conditional Random Field (HCRF) model integrates record extraction and attribute labeling, which relies on VIPS [11] to construct a visual tree for the web page. However, one limitation is that HCRF assumes every record corresponds to one block in the visual block tree. The assumption may not hold on modern web pages. Furthermore, HCRF also requires domain knowledge to train more specific models, such as product-oriented labels. Besides the difficulty of choosing the features to be considered in the learning algorithm for each domain, changing the domain usually results in at least a partial retraining of the models if not an algorithmic redesign.

In AMBER, having domain specific annotators at hand, we also exploit the underlying repeated structure of the pages, but guided by occurrences of regular attributes which allow us to distinguish relevant data areas from noise, as well as to address noise among the records. This allows us to extract records with higher precision. In addition, we only require one result page as input, and AMBER is able to extract record separated horizontally or vertically, even in grid structure. Moreover, with the help of domain specific annotators, AMBER is completely tolerable to optional and disjunctive attributes, attributes extracted by AMBER are accurate and classified into corresponding category, can be applied as knowledge directly.

Surprisingly, also one of the oldest web extraction tools by Embley et al. [28] used an ontology for record segmentation.

More recent approaches are, like AMBER and the approaches discussed in Section 3.1.3, *domain-parametric*, i.e., they provide a domain-independent framework which is parameterized with a specific application domain. For instance, ODE [73] constructs an ontology for a domain based on information matching between query interfaces and query results, then apply the ontology on aligning and labeling data values in the extracted records. However, constructing ontology through query interfaces and results may significantly limit the completeness of the ontology. There are many types of attributes normally will not be presented in the query forms, but remain an important role in describing the objects.

3.1.3 Combined Approaches

Besides AMBER, we are only aware of three other approaches [23, 25, 69] that exploit the mutual benefit of unsupervised extraction and induction from automatic annotations. All these approaches are a form of *self-supervised* learning, a concept well known in the machine learning community and that has already been successfully applied in the information extraction setting Rozenfeld and Feldman [68].

In Senellart et al. [69], web pages are independently annotated using background knowledge from the domain and analyzed for repeated structures with conditional random fields (CRFs). The analysis of repeated structures identifies the record structure in searching for evenly distributed annotations to validate (and eventually repair) the learned structure. Conceptually, [69] differs from AMBER as it initially infers a repeating page structure with the CRFs independently of the annotations. AMBER, in contrast, analyses only those portions of the page that are more likely to contain useful and regular data. Focusing the analysis of repeated structures to smaller areas is critical for learning an accurate wrapper since complex pages might contain several regular structures that are not relevant for the extraction task at hand. This is also evident from the reported accuracy of the method proposed in [69] that ranges between 63% and 85% on attributes, which is significantly lower than AMBER's accuracy.

This contrasts also with Dalvi et al.[23], which aims at making wrapper induction robust against noisy and incomplete annotations, such that fully automatic and cheaply generated examples are sufficient. The underlying idea is to induce multiple candidate wrappers by using different subsets of the annotated input. The candidate wrappers are then ranked according to a probabilistic model, considering both features of the annotations and the page structure. This work has proven that, provided that the induction algorithm satisfies few reasonable conditions, it is possible to produce very accurate wrappers for single-attribute extraction, though sometimes at the price of hundreds of calls of the wrapper inducer. For multi-attribute extraction, [23] reports high, if considerably lower accuracy than in the single-attribute case. More importantly, the wrapper space is considerably larger as the number of attributes acts as a multiplicative factor. Unfortunately, no performance numbers for the multi-attribute case are reported in [23]. In contrast, AMBER fully addresses the problem of multi-attribute object extraction from noisy annotations by eliminating the annotation errors during the attribute alignment. Moreover, AMBER also avoids any assumptions on a subsequently employed wrapper induction system.

A more closely-related work is ObjectRunner [25], a tool driven by an intensional description of the objects to be extracted (a SOD in the terminology of [25]). A SOD is basically a schema for a nested relation with attribute types. Each type comes with associated annotators

(or recognizers) for annotating the example pages to induce the actual wrapper from by a variant of ExAlg [1]. The SOD limits the wrapper space to be explored ($\leq$20 calls of the inducer) and improves the quality of the extracted results. This is similar to AMBER, though AMBER not only limits the search space, but also considers only alternative segmentations instead of full wrappers (see Section 4.4). On the other hand, the SOD can seriously limit the recall of the extraction process, in particular, since the matching conditions of a SOD strongly privilege precision. The approach is furthermore limited by the rigid coupling of attribute types to separators (i.e., token sequences acting act as boundaries between different attribute types). In fact, attribute types appear quite frequently together with very diverse separators (e.g., caused by a special highlighting or by a randomly injected advertisement). The process adopted in AMBER is not only tolerant to noise in the annotations but also to random garbage content between attributes and between records as it is evident from the results of our evaluation: Where ObjectRunner reports that between 65% and 86% of the objects in 5 domains (75% in the car domain) are extracted without any error, AMBER is able to extract over 95% of the objects from the real estate and used car domain without any error.

Skoga [6] introduces a DOM structure-knowledge-oriented global analysis framework, which conduct a global analysis on the DOM structure and detect data region and records based on the DOM structure knowledge. Skoga first conducted a global analysis on the DOM structure, consists of background statistical knowledge which capturing the characteristics of record regions and encodes the semantics of record/data region with 8 types of labels. Then a series of features such as single-node features, sibling features and parent-children features are designed for identification. The weight of features is determined by a parameter estimation algorithm based on a structured output supporting vector machine on a training dataset. However, Skoga only focuses on record region (data area) and records, and is not able to extract attributes. Different from Skoga, AMBER does not require a learning process and the record identification accuracy is not rely on a selection of training corpus. We compare AMBER with skoga on a benchmark dataset TBDW in section 7.1.3, we ran AMBER on all 51 websites and reported a slightly lower precision (0.15%) and recall (2%) compared to Skoga, while in the data Skoga reported, 2 of 51 websites were excluded.

3.2 Detail Page Analysis

The sheer amount and complexity of information published on the web leads to web designs which disperse the attributes of a single object over several pages. To retrieve full details of a given object, it is often necessary to extract these data from the corresponding detail page,

typically reached through a search form and a sequence of paginated result pages. AMBER combines the information extracted from result and detail pages into a coherent output.

The analysis of different page types necessitated the incorporation of different features, such that AMBER relies mainly upon four different strands in research:

(1) *Boilerplate removal*, as in [3, 51], distinguishes the main content of a detail page from other non-informative parts, such as the navigation link, menus, headers, footers, sidebars, related articles, banners and advertisement. **(2)** *Repeated pattern analysis,* as in [59, 70, 73, 84], deals with result pages generated from templates by searching for repeated structures either within the page or between different pages generated from the same template. **(3)** *Instance-based approaches,* as in [43, 66] identify already known instances on the analyzed pages in lieu of human provided annotations to induce wrappers for extraction. **(4)** *Automatic annotation-based approaches,* as in [23, 69], use automatic annotations engines to replace the human annotator, either for direct extraction or to induce a wrapper. The approaches differ in the way they deal with the inevitable noise in these annotations. **(5)** *Table and list based approaches,* as in e.g. [27, 77], analyze domain independent structures which were already used in traditional documents to format structured content.

Aside those approaches, *wrapper induction,* as in [22, 31, 42, 48, 49, 52, 53, 64], provides an important alternative to web data extraction: It relies on example pages annotated by humans to drive a machine learning algorithm generating a small program, the wrapper, to extract from new pages those values which correspond to the values annotated on the training pages. The majority of the pre-existing approaches (for surveys, see [13, 55]) falls into **(2)** or wrapper induction. On the other hand, the more modern approaches in **(3)** and **(4)** compensate for human supervision needed in wrapper induction and for inaccuracies occurring in repeated structure analysis, partly incorporating those older approaches to improve upon them. In general, the combination of different approaches is becoming more important, as monolithic techniques appear to be limited in the range of pages they can deal with successfully. Early versions [34] of AMBER belonged to **(4)**, building upon **(2)**, while the current version of AMBER incorporates techniques from **(1-5)**. Although AMBER does not use wrapper induction internally, it can generate example pages annotated at almost human accuracy.

Aside these broad categories, there have been some other approaches utilizing further sources of information. For example, DeLa [78] already matched form filling values with extracted attribute values to assign form labels as attributes labels.

3.2.1 Boilerplate removal

On most web pages, the main content is accompanied by non-informative parts referred as boilerplate, such as the navigation link, menus, headers, footers, sidebars, related articles, banners, forms, copyright notice, advertisement, etc. According to Gibson et al. [39], the volume of templated material is 40-50% of the total bytes on the web. To solve the problem, the CleanEval competition [4] has the goal of preparing a representative corpus with a gold standard for linguistic and language technology research and development on boilerplate removal.

The existing work for boilerplate removal could be divided into two main catigories, site-level approaches [3, 83] and page-level approaches [11, 12, 24, 40, 51].

Site Level Approaches Most site level approaches are based on the assumptions that noise blocks from different pages of the same website share a common template, usually machine generated and have highly similar characteristics. They could finish the boilerplate removal task by exploiting the similarties of pages. Yi et al. [83] proposed a data structure named style tree, which captures the common presentation styles and actual contents of the pages of a given site, and applies a measure to determine the noisy part of the tree. Bar-Yossef and Rajagopalan [3] segments web pages from the same website into coherent blocks which named pagelets by applying a simple heuristic based on the number of contained links, and mark boilerplate by grouping pagelets that are similar.

Although site-level boilerplate removal approaches report some promising result, however, the common template of boilerplate of the same site may still contain variations. In addition, boilerplate context on modern web pages may not be templated, such as "nearby properties" in real-estate domain, "similar cars" in used car domain, and some advertisements.

Page Level Approaches With the limitations of site-level approaches, more researchers propose approaches that are more flexiable and takes only one page as input. On page level, boilerplate detection problem are turned into a classification problem by classify text nodes into different catigories (e.g., main content or boilerplate). Three categories of features are commonly used, **(1)** Structural features, which are the structural information of the dom tree, e.g., tag density, tag arrangement and frequency of tag occurrence. **(2)** Textual features, that are distinguishable text pieces representing properties of the current text, such as keyword, header and footer class name. **(3)** Visual features, such as css block position, block size, font size, etc.

Page-level approaches either set up some herustics or adopt machine learning techniques using one or more catigories of features we listed above. Chakrabarti et al. [12] trained a Logistic regression classifiers, Debnath et al. [24] ran a Supporting Vector Machine to segment

blocks and then identifies the informative block, Gibson et al. [40] used a Conditional Random Field sequence labeling model to extract the whole content of news from pages, Pasternack and Roth [67] extracts article text by training a naive bayes model as a local classifier, and Kohlschutter et al. [51] analyzed a set of shallow textual features which are at token-level and independent (e.g., average length of sentence), then combines with some visual block features such as text block position, and employs a decision tree model to classify individual text element.

3.2.2 Repeated Structure Analysis

Boilerplate detection approaches are often used as a pre-processing step for detail page analysis. A vast number of detail page based approaches are focusing on extracting relevant data. There are several unsupervised wrapper generation approaches which are based on discovering repeated structures on pages presumably generated by a common template. For example, RoadRunner [18], FiVaTech [50], MDR [58], VIDE [59], ViPER [70], ODE [73], DEPTA [84], Trinity [72] and ViNTs [85] rely on repeated HTML markup or regularities in the visual rendering. The most common task that can be solved by these tools is *record segmentation*, where an area of the page is segmented into regular blocks each representing an object to be extracted. Otherwise domain independent, among recent approaches, only [73] employs a domain ontology for data area identification, ignoring it during record segmentation. More detailed discussion about those repeated structure based record segmentation tools are in section 3.1.2. In contrast, the repeated structure analysis of AMBER, providing a general but domain parameterized framework, searches for repetition not within the plain but the annotated DOM. This improves AMBER in distinguishing relevant data areas from noise and also addresses noise among records.

In principle, repeated structure analysis can reveal templates that were instantiated multiple times on the same page, as in case of result pages, or templates that were instantiated on multiple pages, as in case of detail pages. Some repeated structure analysis approaches work with a single page, e.g., ViNTs [85], searching for intra-page repetition, while others, e.g., ExAlg [1], RoadRunner [18], Trinity [72], require a set of pages, searching for inter-page structures. Trinity builds on the hypothesis that pages generated from the same server-side using the same template may introduce some shared patterns that do not provide any relevant data and can thus be ignored. It takes two or more pages as input and learns a regular expression to model the page and extract data.

In contrast to those systems, AMBER is not only able to search for both intra-page and inter-page repetition. AMBER is also able to follow links from result pages to detail pages and integrate the results obtained from all explored pages.

3.2.3 Instance-Based Approaches

Instance-based approaches identify already known instances on the analyzed pages in lieu of human provided instance annotations to induce wrappers for extraction.

Gulhane et al. [43] present an approach which exploits instances occurring redundantly among several sites to learn templates of new sites for extracting further instances data subsequently. Given an initial set of instances, and a set of detail pages from the same site, it identifies occurrences of known instances on the given detail pages to learn the template and to extract the data from the detail pages with hitherto unknown data. Similarly, the approach in [66] presented by Papotti et al. also targets detail pages by exploiting instances occurring redundantly on multiple sites. But differently to [43], this approach first performs a repeated structure analysis RoadRunner [18] for inducing initial wrappers. Subsequently these wrappers refined during integrating the data extracted by those wrappers.

Both approaches rely on template consistency – just as AMBER does – but they both assume a different notion of content consistency: While AMBER requires redundancy in the presented objects between different pages of *same* site, they exploit redundancy between *different* sites: The former might occur in many domains; however in domains with a strong segmentation, e.g., on a market where each participant offers different products, this assumption fails. On the other hand, the content consistency assumed by AMBER is inherent to today's web design patterns, and does not impose any assumption on the content distribution between different sites. The knowledge required by AMBER during initial annotation is *not* organized in instances, i.e., albeit AMBER requires gazetteer lists or patterns to recognize possible attribute values and labels, the attributes are kept completely independent.

3.2.4 Automatic Annotation-Based Approaches

Another strategy improving upon wrapper induction and repeated structure analysis is the utilization of automatic annotations. Besides AMBER, we are only aware of three data extraction approaches Dalvi et al. [23], Derouiche et al. [25] and Senellart et al. [69] incorporating automatic annotations into their analysis.

However, such a *self-supervision* has already been applied successfully in information extraction by Rozenfeld and Feldman [68].

In [69], automatic annotations are combined with repeated structure analysis: web pages are independently annotated using domain knowledge and analyzed for repeated structures with conditional random fields to integrate the results afterwards. In contrast, AMBER's repeated structure analysis is guided by annotations, enhancing performance and accuracy, as page por-

tions with irrelevant or irregular data are avoided. The lacking integration of annotations and repeated structure analysis in [69] leads to significantly lower accuracy, ranging between 63% and 85% on attributes.

This contrasts with Dalvi et al. [23], which harden wrapper induction against noisy and incomplete automatic annotations. The approach induces multiple candidate wrappers using different annotation subsets to choose one according to a probabilistic model, considering the likelihood that wrapper fits the annotations. This leads to very accurate wrappers for single-attribute extraction from noisy annotations, at the price of hundreds of wrapper induction invocations. For multi-attribute extraction, [23] reports high but lower accuracy than in the single-attribute case, and more importantly, requires a much larger wrapper space. In contrast, AMBER fully addresses the problem of multi-attribute object extraction from noisy annotations by eliminating the annotation errors during attribute alignment.

More closely-related, ObjectRunner [25] is driven by intensional descriptions of the objects to be extracted. Such descriptions are basically schemata for nested relations with attribute types, each associated with annotators to generate examples for wrapper induction. The approach is limited, because its matching conditions strongly privilege precision, and since attribute types are tightly coupled to separators, i.e., token sequences marking attribute boundaries. It fact, attribute types appear quite frequently together with very diverse separators, e.g., caused by special highlighting or randomly injected advertisements. In contrast, AMBER is not only tolerant to noisy annotations, but also tolerants to dynamic object-like advertisement (e.g., a list of "nearby property", "related property") by combining both result page analysis and detail page analysis.

3.2.5 Model-Based Approaches

While template based approaches work with arbitrary templates, model-based approaches locate specific data structures, such as grids, trees, tables, lists, and key-value maps, that are likely to contain structured data. Cafarella et al. [9] and Gatterbauer et al. [38] are focusing on extraction data from tables; Elmeleegy [27], Gupta and Sarawagi [44], and Weninger [81] are extracting information from lists.

Their clearly definable characteristics are easier to encode as features that domain-independent algorithms can exploit to achieve fairly high accuracy of the recognition. These structures are particularly useful to locate interesting areas on the page to focus the analysis [86]. They are also very useful for deriving gazetted knowledge such as label-instance pairs to support wrapper induction, such as Weninger et al. [81], Dalvi et al. [21], and Wang et al. [80]. On the other hand, due to their popularity, these regular structures are likely to be frequently used also

for layouting purposes and can therefore create a large number of false positives that need to be cleaned via a-posteriori repair or semantic interpretation [45, 77]. Moreover, such regular structures are rarely sufficient for the form of data extraction that AMBER is aiming for, i.e., a complete extraction of a multi-attribute object with high accuracy.

Chapter 4

Result Page Analysis

A result page is a page containing a repeated structure of complex entities, often in the form of a sequence but arranged in different formats, such as lists, grids or tables, and returned as a response to a form query on a web site, sometimes can only be accessed through a query form. Extracting objects from result pages is efficient, orders of magnitude faster than from detail pages, and the links to detail pages are often only accessible through result pages.

This chapter presents the result page extraction process of AMBER. Section 4.1 introduces the system architecture of AMBER, an automatic ontology-based multi-attribute object extraction system. Section 4.2 discusses AMBER's domain-parameterizable annotation schema SNER (Structured Named Entity Recognizer). Section 4.3 introduces how does AMBER utilize pivot nodes to identify the data areas of a given result page, followed by section 4.4 which shows the steps to segment the data areas into records, then section 4.5 describes the data area and record validation process in AMBER, including noise elimination and classification on primary and secondary data areas. Section 4.6 presents the attribute alignment process, based on computing a set of constituent scores characterizing the visual and textual features of attributes. Last but not least, section 4.7 demonstrates AMBER's self-boosting ability. AMBER is able to infer missed attributes and then add them back to the gazetteer, bootstraps the extraction performance.

4.1 System Architecture

Figure 4.1 shows AMBER's architecture composed mainly of three layers. The *Browser Layer* consists of a JAVA API that abstracts the specific browser implementation actually employed. Through this API, currently AMBER supports a real browser like Mozilla Firefox, as well as a headless browser emulator like HTMLUnit. AMBER uses the browser to retrieve the web page to analyze, thus having direct access to its DOM structure. Such DOM tree is handed over

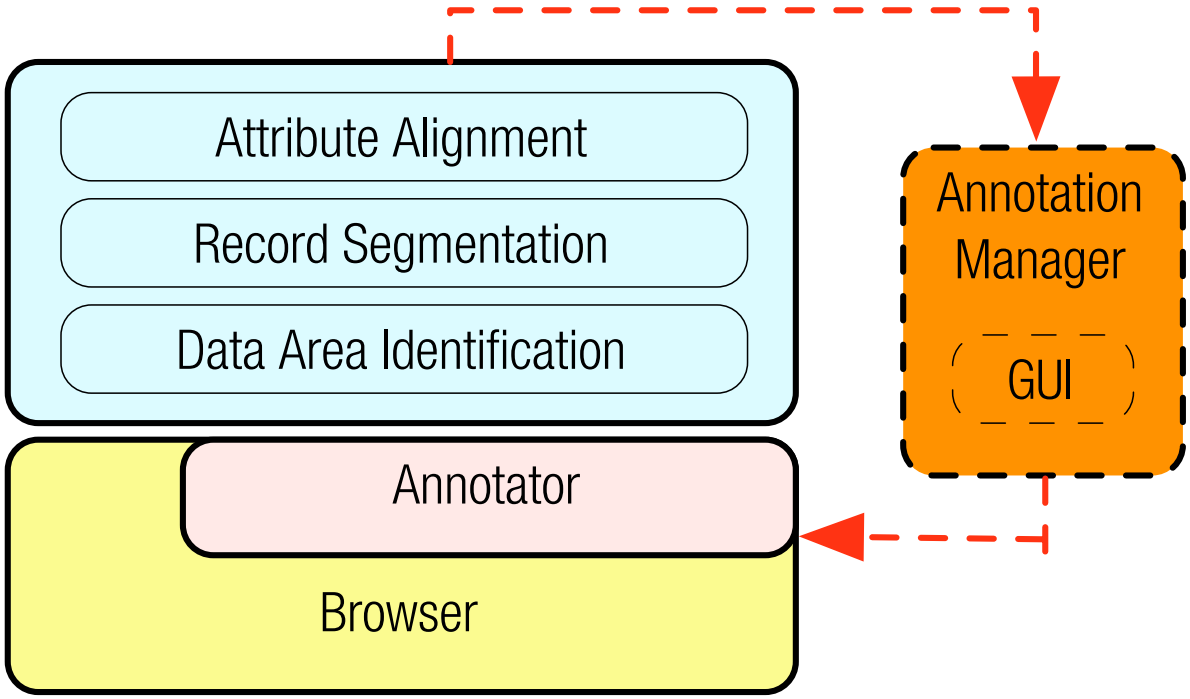


Figure 4.1: System architecture

to the *Annotator Layer*. This is implemented such that different annotators can be plugged in and used in combination, regardless their actual nature, e.g., web-service or custom standalone application. Given an annotation schema for the domain at hand, such layer produces annotations on the input DOM tree using all registered annotators. Further, the produced annotations are reconciliated w.r.t. constraints present in the annotation schema. Currently, annotations in AMBER are performed by using a simple GATE (gate.ac.uk) pipeline consisting of gazetteers of terms and transducers (JAPE rules). Gazetteers for real-estate and used cars domains are either manually-collect (for the most part) or derived from external sources such as DBPedia and Freebase. Note that many types are common across domains (e.g., price, location, date), and that the annotator layer allows for arbitrary entity recognizers or annotators to be integrated. With the annotated DOM at hand, AMBER can begin its analysis with data area identification, record segmentation and attribute alignments.

The outcome of AMBER's attributes alignment component is a set of attributes with relative support. During AMBER's bootstrapping, however, the outcome could be re-used as feedback to improve the gazetteer used in the annotation(Section 4.7), which is managed by the *Annotation Manager* module. The Annotation Manager is optionally complemented with a graphical user interface, implemented as an Eclipse plugin (eclipse.org) which embeds the browser for visualization.

```
<div>
 <img src="[..]/bed.jpg" alt="Bedrooms">
 <br/> 7
</div>
...
<div>
 <img src="[..]/house.jpg" alt="">
 <br/> 4,306 ft<sup>2</sup>
</div>
```

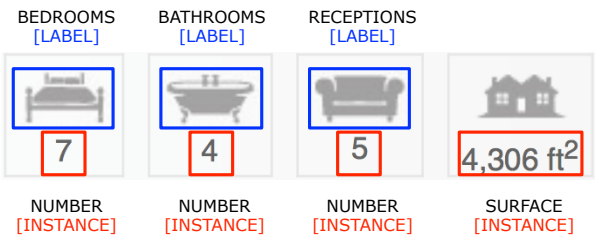


Figure 4.2: SNER at work.

4.2 Annotation

Besides their importance in information extraction, automatically generated annotations have proven useful in web data extraction as they replace tedious human supervision. Annotations are typically used to automatically generate examples for wrapper induction [23, 69] as they mark nodes in the DOM tree possibly corresponding to values to be extracted. In the same way, they are now frequently part of template discovery algorithms [60, 74] to limit the exploration to areas on the page deemed to contain interesting data. Annotations are also useful in a-posteriori validation of the templates [25]. Most of the approaches label the content of *visible* text nodes in the DOM with their data type, e.g., string, integer.

AMBER relies on a new class of annotators, dubbed *SNER* (Structured Named Entity Recognizer), that enhance classic NERs to leverage structural and rendering information of the page to identify entities of interest. The types of the entities recognized by a SNER include both *instances* of a certain concept, e.g., "449 Kelton Av", "£50", and *labels*, e.g., "Price", "Make". The scope of a SNER is not limited to visible content on the page, e.g., from text nodes. It extends the annotation to hidden structures such as values of DOM attributes, e.g., from `id`, `class` attributes, as they often contain useful clues for data extraction purposes. A SNER is also a surrogate for both template-based [41, 57] and instance-based [7, 9] redundant information as it is implicitly encoded in the method each SNER uses to produce annotations.

As an example, on `www.pennyandsinclair.co.uk` (Figure 4.2), the number of bedrooms in the property are plain numbers with attached icons to denote the type of room:

In this case, a traditional NER recognizes "7","4", and "5" as numbers, while a DIADEM SNER also recognizes the corresponding labels in the `img`'s `alt` texts allowing AMBER to exploit this information during template discovery. In the case of the house size, no useful text is provided by the `alt` attributes. However, the unit of measure "ft^2" is sufficient to label the text with SURFACE:INSTANCE.

Annotations are modeled as a relation ann where $\mathsf{ann}(t,n,v,s,e)$ is interpreted as follows:

t is the annotation *type*, e.g., PRICE:INSTANCE, MAKE:LABEL, n is an element node in the DOM, v is the annotation *value*, e.g., "Ford", "£500", and s, e are integer numbers representing the *span* $[s, e]$ (e.g., *start* and *end*) of the annotation within the textual content of n. Notice that the value of an annotation might not coincide with the actual string annotated on the page as many annotators are capable of reporting meta-data about the annotation, e.g., after conversion or normalization. Given a type t, a node n, and a span $[s, e]$, the value v is unique. On the other hand, an annotation is allowed to span multiple elements, e.g., for `<b>300</b><i>GBP</i>`, the PRICE:INSTANCE annotation spans the `<b>` and `<i>` elements. For simplicity, in the following we sometimes use abbreviated forms such as $\mathsf{ann}(t, n)$ to denote the fact that the other elements in the relation do not matter. In annotation types we sometimes omit the suffixes *label* and *instance* when their value does not matter.

A SNER takes into consideration both the HTML structure and CSS visual styling during the annotation. This is captured by the following definitions.

Definition 3. We say that a DOM element is *entity breaking* if it denotes visually or structurally a separate unit of information or a separator. Elements denoting information units are, e.g., block level elements (including inline elements with CSS `:display` property set to `block`, `list-item`, and `inline-block`). Separators include inline but empty, i.e., without child text, elements with non-zero height and width, such as `<br>`.

Definition 4. Given an annotation $\mathsf{ann}(t, n, v, s, e)$ we say that it is *redundant* if it spans across entity-breaking elements or there exists an annotation $\mathsf{ann}(t, n', v, s', e')$, where n is an ancestor of n' in the DOM and $e - s = e' - s'$, i.e., the same annotation provided on n is also fully carried by one of its descendants in the DOM.

The annotation process produces an annotated DOM tree and proceeds as follows: First, all visible text nodes in the DOM are concatenated into a single string (called the *element clob*) following the document order. This is done for two main reasons: Allow cross text-node annotations and support the recognition of label-value pairs. As an example, a price SNER applied to the HTML fragment `<div><span>Price:</span> 300 GBP</div>` yields the annotations $\mathsf{ann}(\text{PRICE:LABEL}, n_1)$ and $\mathsf{ann}(\text{PRICE:INSTANCE}, n_2)$, where n_1 is the `span` node and n_2 is the `div` node. The concatenation is not blind. A separator, currently $\sim@\sim$, preventing any annotation from spanning over it, is introduced in the string after the textual content of entity-breaking elements. In the example above, the concatenated string is "Price: 300 GBP $\sim@\sim$". All DOM attribute values of visible DOM elements are concatenated into a unique string (called *attribute clob*), where the safe separator is introduced between each text fragment. Attribute and element clobs are then annotated using standard text annotators. Safe separators guarantee that textual annotations on element and attribute clobs are non redundant.

Table 4.1: Annotation Schema.

		Annotation Schema (Σ)	
		Characteristic (Σ_C)	Optional (Σ_O)
Domain	REAL-ESTATE	BEDROOMS, LOCATION, BATHROOMS, PRICE	RECEPTIONS, TOWN, CITY, VILLAGE, COUNTY, POSTCODE,PROPERTY_TYPE, FURNISHING, STREET_ADDRESS, PROPERTY_STATUS, AVAILABILITY_DATE, BRANCH_LOCATION, MAP, IMAGE, URL, DESCRIPTION
	USED CARS	MAKE, FUEL_TYPE, TRANSMISSION, MILEAGE, ENGINE, PRICE	MODEL, ROAD_TAX, EMISSIONS, POWER, COLOUR, BODY_TYPE, MPG, AGE, SEATS, OWNERS, DOORS, REGISTRATION, ENGINE_SIZE, VILLAGE, TOWN, CITY, COUNTY, POSTCODE, STREET_ADDRESS, IMAGE, URL, DESCRIPTION
	SEARCH ENGINE	URL	-

Textual annotation in a SNER is implemented as gate [20] applications. They are either annie gazetteers semi automatically generated using sources such as dbpedia [2], jape rules encoding regular expressions for, e.g., prices and postcodes, or interfaces to existing NERs such as opencalais and stanfordner.

Once the clobs have been annotated, the annotations are reported on the DOM structure thus creating annotated DOM elements and text nodes. An annotation on the element clob is transferred to the corresponding text nodes and up to the first ancestor element in the DOM tree such that the annotation is no longer split across elements to guarantee non redundancy. An annotation on the attribute clob is transferred to the corresponding DOM element and is by construction non-redundant.

Creating a SNER is a relatively easy task as it mainly relies on crafting a suitable NER for which several implementations and APIs can be found online. A growing number of resources are available for extending documents with semantic annotations that label document snippets as referring to entities of particular semantic concepts. While originally focused on a few broad standard classes of annotations, e.g., people, locations, organizations, the annotator ecosystem is now increasingly diverse, with annotators, e.g., cicerolite,[1] capable of dealing with a great variety of concepts. When a suitable NER is not available, one can craft it by means of tools such as gate that greatly simplify this task. As also noticed in [60], to obtain a reasonable coverage of the entities multiple annotators, for the same entity have to be employed. This often introduces the problem of reconciling their opinions when contradictory annotations are returned. Although this is beyond the scope of this paper, in AMBER this is solved via the roseann annotation integration system [15]. On the other hand, AMBER's template discovery is robust to noisy and incomplete annotators. The effect of noise in a SNER output on template discovery is discussed in Section 7.1.2.

The annotation types provided by the SNER are interpreted according to a reference *annotation schema* $\Sigma = \langle \Sigma_C, \pi, \Sigma_O \rangle$ where Σ_C is the set of *characteristic* attribute types, represent attributes that characterize the entity that is targeted by the extraction, e.g., a property or a used-car, and occur with some minimal support on the listing page. Among these we distinguish the *pivot* attribute type π, that is used to focus the template discovery algorithm and is chosen as the attribute that is mandatory in each record. Finally, Σ_O is the set of *optional* attribute types, representing attributes that can be optional in a record and for which no minimal support is expected. AMBER uses a different annotation schema for each target domain. Examples for three different domains are shown in Table 4.1, where the pivot attribute type is underlined. It is worth noting that several attributes are often present in different domains, e.g., IMAGE,

[1] http://www.languagecomputer.com/products/text-annotation/cicerolite.html.

4.3 Data Area Identification

For the purpose of identifying and extracting structured objects with their attributes from a result page, AMBER firstly distinguishes data areas with relevant objects from non-informative part, such as advertisements or navigation menus.

A data area is an area rooted at node d_{root} in a DOM tree P together with its supporting pivot nodes PIVOTS, i.e., nodes on the page that contain annotations for regular attribute types, e.g., PRICE. AMBER first identifies pivot nodes, then obtains the data areas as clusters of continuous sequences of pivot nodes which are evenly spaced at roughly the same DOM tree depth and distance from each other.

We overcome the mutual dependency of data area, record, and attribute in approximating the regular records through instances of the pivot attribute type PIVOTS: for most of records, we aim to identify a *pivot node* contained in that record. Based on the assumption that all records are having repeating structures, a data area is then a cluster of pivot nodes appearing regularly, i.e., the HTML tags of pivot nodes should be exactly the same, the nodes occurring have roughly the same depth, and a pairwise similar distance.

Definition 5. Let N_π be a set of pivot nodes, $\mathsf{ann}(\pi, n, v)$ be an annotation type π on node n with value v, and for each $n \in N_\pi$ there is some v such that $\mathsf{ann}(\pi, n, v)$ holds. At the same time, N_π also holds for the following two conditions:

(1) Θ^{depth}*-depth consistent,* if $\forall n \in N_\pi$, $\max\{\mathsf{depth}(n)\} - \min\{\mathsf{depth}(n)\} \leq \Theta^{\mathsf{depth}}$;

(2) Θ^{dist}*-distance consistent,* if $\forall n_i, n_j \in N_\pi$, $\max\{|\mathsf{path}(n_i, n_j)|\} - \min\{|\mathsf{path}(n_i, n_j)|\} \leq \Theta^{\mathsf{dist}}$.

Therein, $\mathsf{depth}(n)$ denotes the depth of n in the DOM tree, and $|\mathsf{path}(n, n')|$ denotes the length of the undirected path from n to n'. Assuming some parametrization Θ^{depth} and Θ^{dist}, we derive our definition of data areas from these measures:

Definition 6. Let P be a DOM, A **data area** $D(d_{root}, \text{PIVOTS})$ is a subtree of P with a regular attribute type π that:

(1) D contains a set of pivot nodes N_π with $|N_\pi| \geq 2$;

(2) N_π is maximal;

(3) N_π is tag consistent;

(4) N_π is depth and distance consistent;

(5) D is rooted at the least common ancestor of N_π.

Algorithm 1: `identify`$(P, \pi, \text{ann}, \text{PIVOTS})$

input : P – DOM to be analyzed
input : π – the pivot attribute type $\pi \in \Sigma_R$
input : ann – annotations on P
output : DAs(d_{root}, D) – data area candidates with roots and pivot nodes

PIVOTS $\leftarrow \{n \mid \text{ann}(\pi, n, v)\}$;
DAs $\leftarrow \emptyset$;
while $(|\text{PIVOTS}| > 0)$ **do**
  CandDAs $\leftarrow \{\{a,b\} : a,b \in \text{PIVOTS}, a \neq b, \texttt{sameTag}(a,b), |\text{Depth}(a) - \text{Depth}(b)| < \Theta^{\text{depth}}\}$;
  if CandDAs $== \emptyset$ **then**
    break
  foreach $D \in$ CandDAs **do**
    foreach $k \in$ PIVOTS **do**
      if $|\texttt{sameTag}(a,k) : a \in D| = |D|$ **then**
        if $\max\{\text{Depth}(a) : a \in D \cup \{k\}\} - \min\{\text{Depth}(a) : a \in D \cup \{k\}\} \leq \Theta^{depth}$ **then**
          minDist $\leftarrow \min\{\texttt{treeEditDist}(a,b) \mid a,b \in D \cup \{k\}, a \neq b\}$;
          maxDist $\leftarrow \max\{\texttt{treeEditDist}(a,b) \mid a,b \in D \cup \{k\}, a \neq b\}$;
          if maxDist $-$ minDist $\leq \Theta^{dist}$ **then**
            $D \leftarrow D \cup k$
  maxDASize $\leftarrow \max\{|D| : D \in \text{CandDAs}\}$;
  foreach $D \in$ CandDAs **do**
    if $|D| =$ maxDASize **then**
      DAs $\leftarrow$ DAs $\cup \{\texttt{lca}(D), D\}$;
      PIVOTS $\leftarrow$ PIVOTS $\backslash D$;
      break

Algorithm 1 shows AMBER's approach on identifying data areas. The algorithm takes as input a DOM tree P, an annotation relation ann, and a pivot attribute type π. As a result, the algorithm extracts the data areas DAs, which represents the set of data areas and will also be used in further stage, e.g., record segmentation. Each data area consists of a root node d_{root} and a set of pivot nodes D *support* it, where $d_{root} \in P$ and $d_{root} = \texttt{lca}(D)$.

The algorithm recursively finds data areas. It first finds the group which has the largest number of unclustered pivot nodes having tag consistent, depth consistent and distance consistent properties. Then AMBER clusters these pivot nodes into one data area, rooted in their latest common ancestor. The algorithm terminates if and only if it is not able to find a group with ≥ 2 unclustered pivot nodes which is tag consistent, depth consistent and distance consistent.

During initialization, the algorithm resets the data areas of a given page P as empty, sets PIVOTS as N_π, for each $n \in N_\pi$ there is some v such that $\mathsf{ann}(\pi, n, v)$ holds and starts the recursive data area extraction process, either finds a data area or terminates the main loop.

In each iteration of the main loop, the algorithm initializes the data area candidates by turning all pairs of unclustered pivot nodes having tag and depth consistency into a candidate data area (Line 4). If no such pair of pivot nodes exists, the loop terminates. For each candidate, after initialization, the algorithm iterates in document order over all unclustered pivot nodes to expand the data area candidates. If after adding the pivot node into the data area candidate, the candidate still remains tag (Line 9), depth (Line 10) and distance (Line 11-13) consistent, the pivot node is added to the data area candidate. With a set of expanded data area candidates, it is possible that two candidates both have the largest amount of pivot nodes. The algorithm selects the one with the smallest document order of pivot nodes as the optimal candidate, mark all the pivot nodes in the selected group as clustered, and goes into the next iteration of the main loop.

To illustrate Algorithm 1, Figure 4.3 shows a fairly uniform (but common) case for data area identification: assume the pivot attribute is *price*, $\Theta^{\mathsf{dist}} = 2$ and $\Theta^{\mathsf{depth}} = 1$ (the standard setting in AMBER), there are two data areas, D_1 and D_2, shown in yellow diamond. The pivot nodes are shown in red triangles, labeled as $P_{i,j}$.

After initialization, for a data area candidate, if the pair of pivot nodes are $p_{1,1}$ and $p_{1,2}$, then $p_{1,3}$ will be added to the data area candidate since adding it remains tag, depth, distance consistent. Although $p_{4,1}$ has same depth, however the distance between $p_{4,1}$ and $p_{1,1}$ is 10, and distance between $p_{1,1}$ and $p_{1,3}$ is 6. The differences of pairwise distances prevented us to add $p_{4,1}$ into the data area candidate. Therefore, the data area candidate D_1 has size 3. For data area candidate $D_2 = \{p_{2,1}, p_{2,2}\}$, the size is 2. Consider pivot node $p_{3,1}$, AMBER is not able to find another pivot node that within consistent depth of $p_{3,1}$. Therefore it has not been added to any cluster.

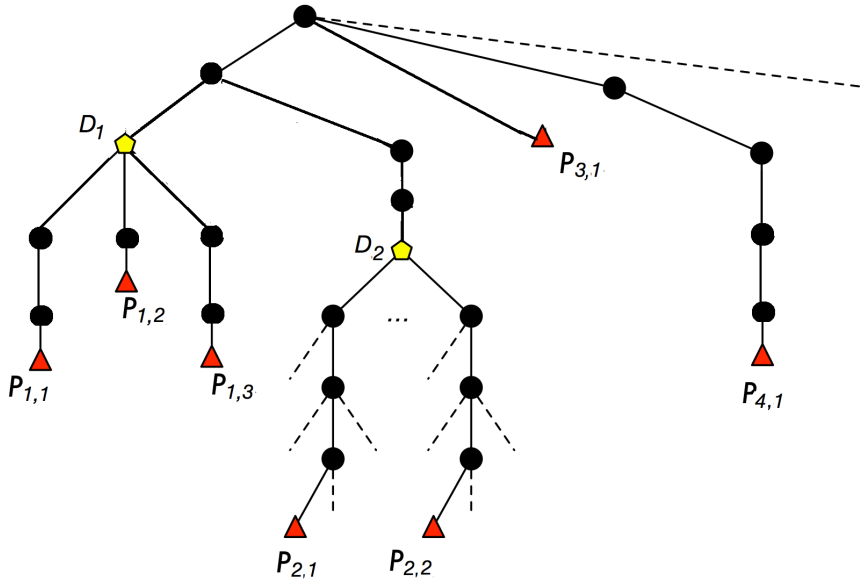


Figure 4.3: Data area identification

In the first iteration, there are two data area candidates D_1 and D_2, we pick D_1 since it contains more pivot nodes, then pick D_2 at the next iteration. If there is a tie, we took the first one according to the document order. In the third iteration, only two pivot nodes $p_{3,1}$ and $p_{4,1}$ left, no data area candidate can be formed, the main loop terminates.

4.4 Candidate Record Segmentation

The previous section discusses the data area identification process of AMBER. AMBER identifies data areas $\mathcal{D}$ of a given result page P_i, and for each data area $d \in \mathcal{D}$, marks its roots d_{root} and provides the pivot nodes $\textsc{pivots}(d)$ supporting the data area, with its pivot nodes occurring roughly at the similar depth and mutual distance. This section describes the record segmentation component of AMBER, which takes a data area $D = (d_{root}, \textsc{pivots})$ as input and then segments it into a set of records $\mathcal{R}$. Each record R_i in $\mathcal{R}$ is a set of HTML nodes that are rooted at same level and share the same template, i.e., positional expression e from d_{root} to root nodes of R_i and R_j should be the same.

AMBER is tailored to result pages with multiple records each representing an object, and following the assumption that the areas' records are instantiations of the same template, and arranged in some specific organized format, such as lists, grids, or tables. Despite the assumption, AMBER is able to deal with a large number of irregularities of the template:

(1) AMBER tolerates inter-record noises, such as advertisements inserted between records.

(2) AMBER tolerates most intra-record variances, such as optional attributes or multiple

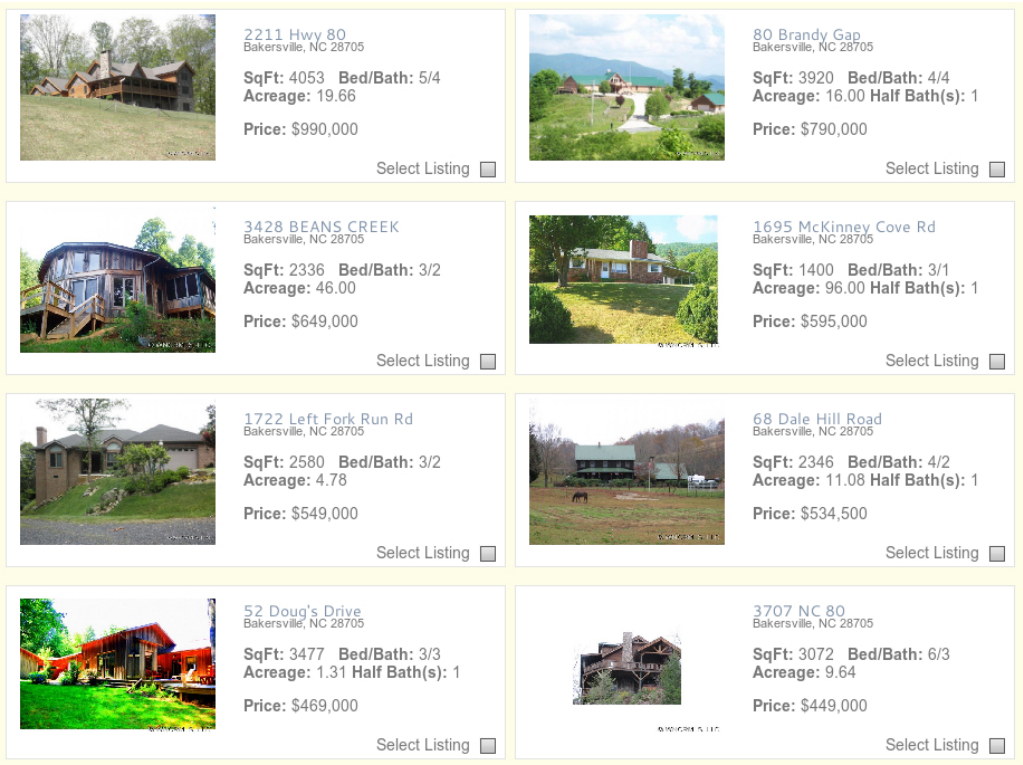


Figure 4.4: A grid result page.

entity types, by segmenting records purely based on regular attributes.

(3) Unlike most approaches who only extract records from lists, AMBER is able to extract records in different format such as grids (a list of lists), tables and lists.

(4) AMBER is able to infer new records which do not contain a pivot attribute or have irregular pivot attributes through generating a template for extracted records and then check if there exists any matches of the template in the data area that has not been extracted as records.

Figure 4.4 illustrates an example of grid result page. The data area contains four rows and each row contains two records. For most other approaches based on the assumption that records are arranged only as lists, they might extract each row as a record, which is clearly under segmentation, 50% data are lost. In Figure 4.5, each record contains a list of ways of mortgage attached at the bottom of the record. In this case, although AMBER uses price as pivot attribute, it is still able to extract the whole record and avoid over segmenting the record into two (one contains the attributes and the other is the price list).

Since AMBER is based on the assumption that all records of a data area are rooted at the same depth, the segmentation problems have been turned into two sub-problems:

(1) Given a data area D rooted as d_{root}, a list of pivot nodes PIVOTS and L, a list of D's descendant nodes at the same level l, get a segmentation $\mathcal{R}$ of D on level l.

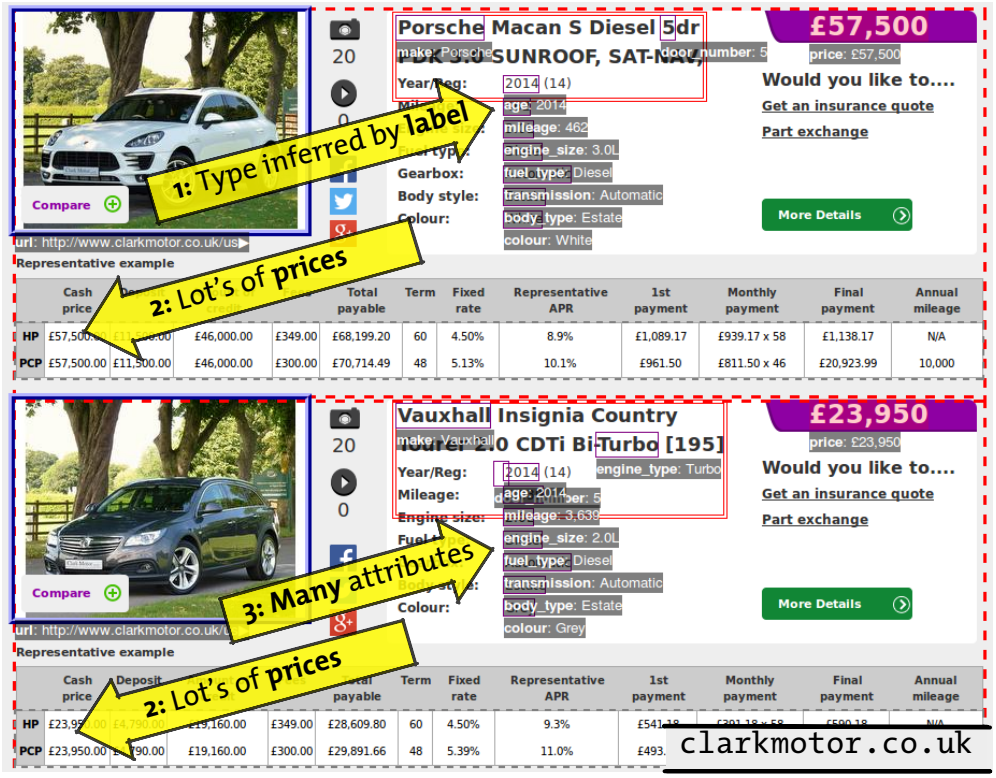


Figure 4.5: Multiple price record.

(2) Find the optimal level l, that neither over segments nor under segments the data area.

To better explain AMBER's same level segmentation process, we define a same level node distance in definition 7.

Definition 7. For a set of nodes L at the same level l in a DOM P and a node $n \in L$, we write $\mathsf{PREV}(n,L)$ for the node n' that is the closest previous node in L to n in document order. $\mathsf{PREV}_{\geq k}(n,L)$ ($\mathsf{PREV}_{=k}(n,L)$) denotes the closest previous node n' in L to n with at least (exactly) k nodes in P at level l between n' and n in document order. Similarly, $\mathsf{SUCC}(n,L)$ is defined to get the succeed of the node n.

We write $\mathsf{SHIFT}(n,s,len,L)$ to represent a set of continuous sibling nodes in L with length len, and node n is the s-th node ($0 \leq s \leq len$). $\mathsf{SHIFT}(n,s,len,L)$ is defined as $\{\mathsf{PREV}_{=s}(n,L) : \mathsf{SUCC}_{=len-s}(n,L)\}$

We use figure 4.6 to illustrate how $\mathsf{PREV}(n,S)$ are computed. If S is set to be all black, blue, yellow, and red nodes, and n is the red node "div" in the middle with a child "£900", then $\mathsf{PREV}(n,S)$ is the yellow node adjacent to it on the left, and $\mathsf{PREV}_4(n,S)$ is the red node with a child "£860". If we define S to be only red and black nodes, then $\mathsf{PREV}(n,S)$ is the red node with a child "£860". Assume $s = 2$, $len = 4$, $\mathsf{SHIFT}(n,s,len,L)$ returns 4 nodes that 2 are left to n, n, and the one right to n.

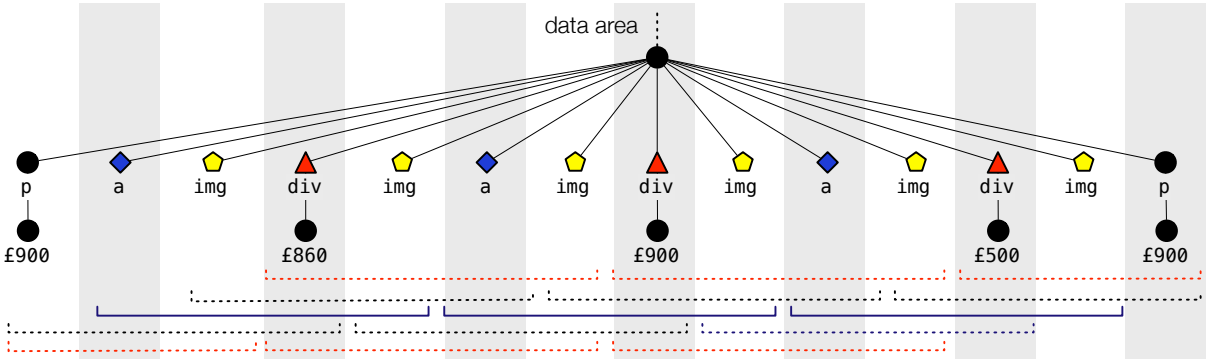


Figure 4.6: A DOM Tree Example.

Definition 8. Let P be a DOM, $D = (d_{root}, \text{PIVOTS})$ a candidate data area, L a list of descendant nodes of d_{root} all at level l, and N is the intersection of L and all ancestors of PIVOTS. Then, a level l **record segmentation** $\mathcal{R}_l$ is a set of records $\{R_i\}$, if there is a length len and shift factor k such that

(1) $|\{(n,n') \in N \times N : \text{PREV}_{=len}(n,L) = n'\}|$ is maximal among all such sets;

(2) $C = \text{SEQ}_{len}(last)$ where $last$ is the last node in N in document order;

$$\text{SEQ}_{len}(n) = \begin{cases} \text{SEQ}_{len}(\text{PREV}(n,L)) & \text{if } |\text{SEQ}_{len}(\text{PREV}(n,L))| > |\text{SEQ}_{len}(\text{PREV}_{\geq len}(n,L))| \\ \text{SEQ}_{len}(\text{PREV}_{=len}(n,L)) \cup \{n\} & \text{otherwise} \end{cases}$$

with $\text{SEQ}_{len}(\emptyset) = 0$.

(3) $C_k = \{\text{SHIFT}(n,k,len,L) : n \in C \text{ and } \text{SHIFT}(n,k,len,L) \text{ is valid } \}$ and $|C_k|$ is maximal among all such sets;

(4) $C_i = \{\text{SHIFT}(n',k,len,L) : \text{SHIFT}(n',k,len,L) \cap C_k = \emptyset \text{ and } Match(C_k, \text{SHIFT}(n',k,len,L))\}$;

(5) $\mathcal{R}_l = C_k \cup C_i$.

Algorithm 2 finds the maximal same level record segmentation for a data area (d,L) and a level l in a DOM P. The first step of this algorithm is to compute the optimal record length. Line 2 intersects all ancestors of pivot nodes given and all nodes on the root level L, gets a series of leading nodes as the base of records. However, there could be noises between those leading nodes. To get the optimal record length, we calculate sibling distances of all leading node pairs, and take the length with the maximal pairwise support as length (Line 3-5). Line 6 executes a dynamic programming algorithm to find the maximum matching leading nodes with optimal length given. Since for each node n, $\text{SEQ}_{len}(n)$ only depends on $\text{SEQ}_{len}(\text{PREV}(n,L))$ and $\text{SEQ}_{len}(\text{PREV}_{\geq len}(n,L))$, hence its time complexity is linear. With the maximum match, we need to shift each leading nodes with length len to a proper position(Line 7-12). After finding the best shift offset, AMBER scans those nodes that are not in any shifts, to check if there exists other record candidates which have been ignored previously due to missing of pivot node, and

Algorithm 2: same-level segmentation($D(d_{root}, \text{PIVOTS}), L$)

input : $D(d_{root}, \text{PIVOTS})$, Data area to segment
input : L, list of descendants of d_{root}
output : $\mathcal{R}_l$, segmentation

1 max $\leftarrow 0$
2 $N \leftarrow Ancestor(\text{PIVOTS}) \cap L$;
3 **foreach** $i \leq |L|$ **do**
4 $\quad c_i \leftarrow |\{(n,n') \in N \times N : n = \text{PREV}_{=i}(n', L)\}|$
5 $\quad$ **if** $c_i >$ max **then** max $\leftarrow c_i$; len $\leftarrow i$;
6 C as in Definition 8
7 $C_s \leftarrow \emptyset$; $C_i \leftarrow \emptyset$; maxshift $\leftarrow 0$
8 **foreach** $j \leq$ len **do**
9 $\quad C_s^j = \{\text{SHIFT}(n_s, j, \text{len}, L) : \exists n \in C : n_s = \text{PREV}_{=j}(n, P)\}$
10 $\quad$ **foreach** $s \in C_s^j$ **do**
11 $\quad\quad t_j \leftarrow |\{s \in C_s^j : s \text{ is valid}\}|$
12 $\quad\quad$ **if** $t_j >$ maxshift **then** maxshift $\leftarrow t_j$; $C_s \leftarrow C_s^j$;
13 **foreach** $n \in L$ **do**
14 $\quad s \leftarrow \text{SHIFT}(n, \text{maxshift}, len, L)$;
15 $\quad$ **if** $s \cap (C_s \cup C_i) = \emptyset$ ***and*** s *is valid* ***and*** $\exists s' \in C_s$: $sameTag(s, s')$ **then**
16 $\quad\quad C_i \leftarrow C_i \cup s$;
17 **return** $C_i \cup C_s$;

adds them back to the segmentation.

The solution of same level segmentation problem has been presented in algorithm 2. The remaining question is to find the optimal level to avoid under or over segmentation, where AMBER employs a top to bottom hierarchical segmenting strategy.

Algorithm 3 illustrates the process of the hierarchical record segmentation, i.e., to find the optimal level for record segmentation. It starts from the children of data area d_{root} level(Line 5), then for each possible level, runs a same-level segmentation as described in Algorithm 2. Line 6 checks if the newly segmented results are sufficient enough. There are two cases: **(i)** The very first segmentation, $\mathcal{R}$ is empty. **(ii)** A segmentation already exists, so we want to check if the new segmentation splits each of the old record into more than one new records, except for the one old record occurring at the last. (ii) is very important for result pages that are listed as grids or tables. In their last row, sometimes only one record exists, and splitting it means over segmenting. So here we only check if at least $|\mathcal{R}| - 1$ old records have been segmented into new records. If the new segmentation $\mathcal{R}'$ has a convincing number of records, we update existing $\mathcal{R}$, and prepare another list of nodes for segmentation on next level (Line 9-12). One possible case worth noticing is that each node in L has only one child, and no need to do only one level deeper since the result will be the same. To reduce those redundant runs, we go deeper until reaching a level that have more nodes. The loop terminates when no new lists of L are available

Algorithm 3: Hierarchical Record Segmentation

```
input  : D(d_root, PIVOTS), Data area to segment
output : R, segmentation
R ← ∅;
L ← AllChildren(d_root);
continue ← True;
while continue = True do
    R' ← SameLevelSegmentation(D(d_root, PIVOTS), L);
    if R = ∅ or |R'| ≤ 2 * (|R| − 1) + 1 then
        R ← R';
        do
            L ← {AllChildren(n) : n ∈ L ∩ R} ;
            if |L| < |R'| then
                continue = False ; break;
        while |L| = |R'|;
    else
        continue = False
return R;
```

or with given L, no segmentation is available.

Overall, with the two algorithms, for each data area given, AMBER is able to segment it properly. The evaluation result is presented in section 7.1.

4.5 Data Area and Record Validation

After extracting data areas from a result page given and segmenting each data area into a set of candidate records, AMBER distinguishes result page from other pages through validating the data areas and records. The validation process has three main purposes:

(1) Cleanup noises from record segmentation candidates, such as advertisements inserted in the data area which used a very similar template to regular records.

(2) Remove those candidate data areas that structurally like result page data areas but textually irrelevant to the domain.

(3) Distinguish between primary and secondary data areas and detect whether or not the given input page is a result page.

To achieve (1) and (2), we employ domain specific annotations acquired from SNER and a set of domain-parameterizable thresholds to tackle noisy records cleanup and domain-irrelevant data area removal problems.

The noisy record cleanup process starts with detecting individual irrelevant records. A record should have sufficient types of annotations to be domain relevant. We adopt a domain-parameterizable threshold $\min_{\text{COUNT}}^{\text{TYPE}}$ (typically 3) here, and compare domain-characterizable

annotation types (Σ) contained in each record with the threshold.

Definition 9. Let P be a DOM page, D a candidate data area, and R a candidate record segmentation for D, and for each $r_i \in R$, $c_i = |\{\tau \in \Sigma, \exists n \in r, v : \mathsf{ann}(\tau, n, v)\}|$, the number of annotation types contained in r_i. A record $r_i \in R$ is called *valid*, if $c_i > \min_{\text{COUNT}}^{\text{TYPE}}$.

Subsequently, number of removed noisy records might be huge and affected the segmentation schema. If the number of remaining records is insufficient, or the block sizes of remaining records vary significantly, the whole segmentation schema will be removed. Two domain-parameterizable thresholds $\min_{\text{COUNT}}^{\text{RECORD}}$ and $\max_{\text{BLOCK}}^{\text{RECORD}}$ are used here. The former controls the minimum record number each segmentation schema should contain, if below then drop the segmentation schema, typically set to 2. The latter controls the maximum allowed visual block size difference of records, typically set to 200%.

Definition 10. A candidate record segmentation R after removing all non-valid records is a *valid record segmentation*, if, $|R| > \min_{\text{COUNT}}^{\text{RECORD}}$ or for any two records in R,
$s_{\max} \leq \max_{\text{BLOCK}}^{\text{RECORD}} \cdot s_{\min}$, where s is the visual block size of the record.

After cleaning noisy records, we further pick up those domain irrelevant data areas from the set of data areas recognized for given input result page through distinguishing one or more characteristic annotation types for each domain. These correspond to attribute types that typically occur in this domain, but are rare in other domains, such as BEDROOM-NUMBER, FURNISHING, PROPERTY TYPE for real-estate and FUEL-TYPE, TRANSMISSION, MAKE for used cars. These are different from *pivot* annotation types which correspond to mandatory, but not necessarily characteristic. For instance, PRICE, are considered as *pivot* annotation since it is mandatory and capable to contribute on locate templates, however it is not characteristic since it is shared by nearly all product domains and is not domain recognizable. A data area must contain more than $\min_{\text{COUNT}}^{\text{CHARACTERISTIC}}$ (typically set to 3) of characteristic annotation types to prove itself as domain relevant, otherwise will be removed. Afterwards, we consider only *valid data areas*.

Definition 11. Let P be a DOM page and $\mathcal{D} = \{D_i\}$ the set of candidate data areas identified on P. Then, a data area $D_i \in \mathcal{D}$ is a *valid data area* if there is a valid record segmentation R for D_i and D_i contains at least $\min_{\text{COUNT}}^{\text{CHARACTERISTIC}}$ *characteristic* annotation types, if any characteristic annotation types is defined for the current domain.

For most of the object extraction approaches, the input is restricted to a specific type of pages, in this context, result pages. However, that requires pre-processing steps such as page classification. However, AMBER is capable to accept all kinds of pages as input and distinguish

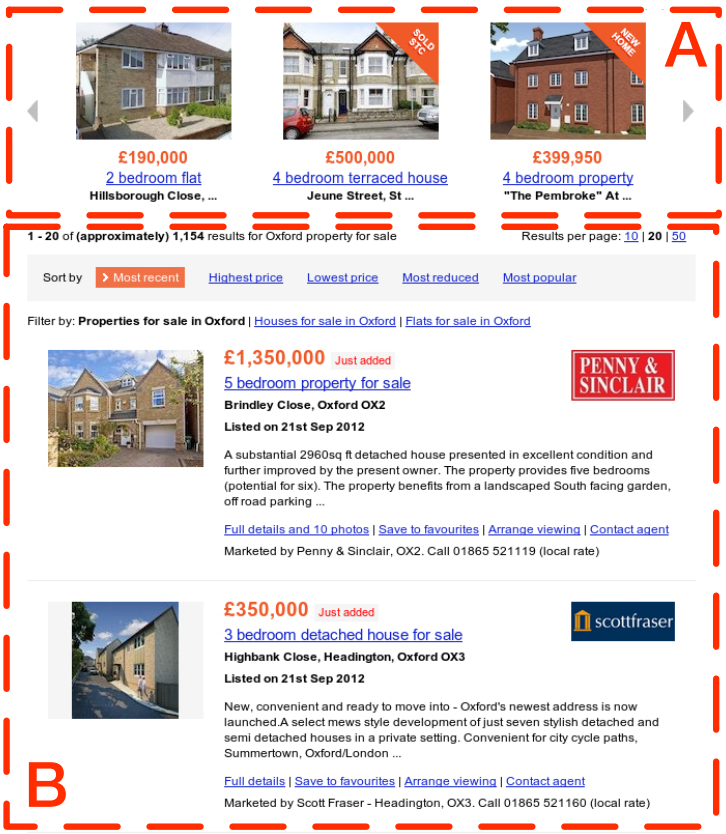


Figure 4.7: Multiple data areas.

result pages from them, hence significantly reduced the effort of applying AMBER to a new domain and ran AMBER on large scale evaluations.

Consider a typical result page from Zoopla.co.uk showed in Figure 4.7, here we have two distinct data areas where records are laid out using different templates. Records are aligned vertically in the data area (A), and aligned horizontally in data area (B). Apparently, data area (B) shows the main content of the result page and (A) is a list of advertisements. Through data area block position and block size, we could tell that (B) is the primary data area of the page and (A) is one secondary data area.

The final step of the validation process is to classify those data areas into two categories: (1) primary and (2) secondary, based on features like the position within the page, block size, and prominent nodes annotated with SECONDARY-TITLE. It is worth noticing that each page could only have at most one primary data area, on the other hand, no limitations are set on the number of secondary data areas.

Definition 12. A data area (D_i) is further considered *primary*, if it is in the most central position of the page, the block size of D_i exceeds a certain threshold $\min_{\text{FRACTION}}^{\text{VISUAL}}$ (typically 0.10) and does not contain prominent nodes annotated with SECONDARY-TITLE, such as "Featured",

"Advertisement", or "Links". Otherwise, it is called *secondary* and likely represents a listing of featured offers, advertisements, or other lists of records that are not the main content of the page.

AMBER is able to distinguish between primary data areas and secondary data areas, then based on whether or not the given result page contains a primary data area to judge if the given page is a result page, which reduced hugh amount of work when we apply AMBER as a key component of DIADEM for large scale evaluations through providing effective feedback for the exploration system (Chapter 5). In addition, the detail page extraction process of AMBER also runs result page extraction on each detail page to identify object content-based recommendation or advertisement lists that are generated dynamically since they will be identified as secondary data areas. The evaluation on distinguishing primary and secondary data areas is shown in section 7.1.2.

4.6 Attributes Alignment

Given a data area D with a record segmentation R, the remaining task is the identification and alignment of attributes in the records of R. AMBER first identifies candidate attributes. Each candidate attribute for type τ is a sequence of nodes a, one from each record r, characterized by a unique ancestor path e from each record r to the corresponding attribute node a, such that sufficient of the a's carry a τ annotation. AMBER combines these candidate attributes for every type in the final sequence of attribute nodes through a scoring that considers textual, structural, and visual aspects of the involved nodes.

AMBER also infers *visual attributes*, i.e., attributes that are not driven by annotations but by the visual and structural position in the records. These attributes are the main title, description, and image of the record. For each of these types, we annotate a unique node for each record individually before the alignment step, in which the same alignment algorithm as for proper annotations is used. Candidates for each of these must be visible and not part of a list of similar items in the record. Among those nodes, the (unique) candidate description is the largest element that contains the text node with the most characters in the record and does not contain significant gaps or changes in font size. Candidates for title must be visually above the center of the record and its children must all be horizontally-aligned inline elements. Among those candidates, nodes with larger font size that are links, occur early in the record, and contain annotations for any attribute of the domain, are preferred. Finally, the main image is simply the largest image in the record, preferring images to the top-left of the record.

Definition 13. Let R be a record segmentation for a data area D. Then, a set of nodes A is a

candidate attribute for R with type τ, if there is an XPath expression e such that

(1) for each $a \in A$, there is a *unique* $r \in R$ with $e(r) = a$ and e is the unique ancestor path from a to r (same relative position).

(2) $\frac{|\{a \in A : \mathsf{ann}(\tau, a, v)\}|}{|R|} > \theta$ where $\theta = \min^{\text{ATTRIBUTE}}_{\text{SUPPORT}}$ (typically, 25%) if all $a \in A$ are annotated with τ, otherwise $\theta = \min^{\text{ATTRIBUTE}}_{\text{INV-SUPPORT}}$ (sufficient annotation support, invented attributes require significantly higher support, typically 50%).

For an attribute A, we denote with $A[r]$ the attribute node in A for a record $r \in R$ (`NULL` if there is no such node). We also record the span and value of attribute a, where the span is either the span of the underlying annotation (see Section 4.2), or the entire node, if the attribute has no corresponding annotation. If there are multiple annotations for τ on the same node, the one with the largest span is chosen.

Definition 14. Let R be a record segmentation for a data area D and A a candidate attribute for R with type τ. Then, A is *optimal*, if A's attribute score $\mathsf{score}(A)$ is maximal among all other candidate attributes A' for R with type τ. The attribute score $\mathsf{score}(A)$ is a score for the regularity and prominence of A in R. $\mathsf{score}(A) = \sum_{r \in R, a \in r} \mathsf{score}(a, r)$. Finally, for an attribute $a \in r$ with value v (and τ_l the label annotation type for τ), we define its score as the sum of five constituent scores:

$$
\begin{aligned}
\mathsf{score}(a,r) &= \sum_{k \in \{\text{ANNOT},\text{REDUN.},\text{LABEL},\text{SPAN},\text{FONT}\}} \mathsf{score}_k(a,r) \\
\mathsf{score}_{\text{ANNOT.}}(a,r) &= \frac{|\{r' : \exists a' \in r' : \mathsf{ann}(\tau, a', v', s', e')\}|}{|R|} \\
\mathsf{score}_{\text{REDUN.}}(a,r) &= \frac{|\{(a', v') : \exists a' \in r : \mathsf{ann}(\tau, a', v)\}|}{\max_{v'}(|\{(a', v') : \exists a' \in r : \mathsf{ann}(\tau, a', v')\}|)} \\
\mathsf{score}_{\text{LABEL}}(a,r) &= 1, \text{ if } (\mathsf{ann}(\tau_l, a, v') \text{ or } (\mathsf{ann}(\tau_l, a', v') \text{ and } prec(a', a))) \\
&= 0, \text{ otherwise} \\
\mathsf{score}_{\text{SPAN}}(a,r) &= \frac{length(v)}{\mathsf{length}(\texttt{string}(a))} \\
\mathsf{score}_{\text{FONT}}(a,r) &= \frac{\mathsf{font\text{-}size}(a)}{\max_{a' \in r}\{\mathsf{font\text{-}size}(a') : \exists v : \mathsf{ann}(\tau, a', v)\}}
\end{aligned}
$$

For two nodes n, n', $\mathsf{proximity}(n, n')$ holds if n precedes n' in document order separated by only nodes containing no text or punctuation and whitespace only.

The five constituent scores are all ranged from [0,1], and are listed as follows:

(1) *ANNOT*: A data-area level constituent score, computes the percentage of records that contain the attribute type τ. The more records containing τ, the more likely that τ should be extracted.

(2) *REDUNDANCY*: A record-level score that represents how many times the attribute value v occurred within the record r. The more redundant of v, the more likely that v is the attribute we want. We use the maximum occurrence time of all v's with in this record r and having same annotation type τ as a baseline to compute frequency.

(3) *LABEL*: A node level score which reflects if current node a or the precedent node of a also has a label of τ annotated. An attribute with a label adjacent to it could prove to be more precise.

(4) *SPAN*: A node level score that computes the span rate of v with the length of the whole node a. If the span coverage rate of v is low, then a might be a long description and hence v is relatively less important than v' with a high span coverage rate.

(5) *FONT*: A record-level score that distributed attributes between interval $[0,1]$ by the font size of v. Larger font size means more eye-catching, therefore are relatively more important than smaller font size.

Definition 15. Let R be a record segmentation for a data area D and $\mathcal{A}$ the set of candidate attributes with (A) the type, and $\mathsf{score}(A)$ the score of attribute $A \in \mathcal{A}$. Then, for each $\tau \in \Sigma$, A_τ is the *aligned attribute* of $A_\tau[r] = A[r]$ if there is an A such that $\mathsf{score}(A) = \max\{\mathsf{score}(A') : A' \in \mathcal{A} \wedge A[r] \neq \texttt{NULL}\}$ or NULL otherwise (maximally scored candidate attributes, that have sufficient support by definition, may be merged to form the aligned attribute).

Algorithm 4 provides the pseudo-code for attribute alignment process. Line 1 reads the candidate types we want to extract provided by SNER . For each annotation, AMBER first calculates the support from annotations in other records which are having the same unique ancestor path(Line 3-6). Then through $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$ and $\min_{\text{INV-SUPPORT}}^{\text{ATTRIBUTE}}$ two thresholds, we filter out the too irregular ones (Line 7-8), and infer the missing ones, add the inferred ones into the candidate lists as well (Line 9-12). We do the whole process for each annotation type τ and each annotation a in each record r, once it is done, we got a set of candidate attributes. Then if a record has two or more candidate attributes with the same type τ, we calculate the five constituent scores of them and sum the results up, pick the highest one as the attribute (Line 14-17). The output of the whole process is a set of attributes, for each record and each type τ, we pick the one with enough supports from other records in the data area and gets the highest constituent score.

We use figure 4.8 as an example to demonstrate the process for attribute alignment. In the first record, "Banbury Road, NorthOxford" in the title and "North Oxford" at the bottom are annotated as location, "Bedrooms" and "Bathrooms" are annotated as bedroom label and bathroom label respectively. There are two property type attributes, "family house" in the description and "house" at the bottom attribute list, both of them are annotated as property type. In the second record, similar to the first one, bedroom and bathroom label are annotated

Algorithm 4: attributes(types Σ, record segmentation R, DOM P with ann)

candidateTypes $\leftarrow \{\tau : \exists n : \text{ann}(\tau, n)\}$;
foreach $\tau \in$ candidateTypes **do**
  foreach $e : \exists r \in R, n \in$ **descendant**::$*(r)$: $ann(\tau, n) \wedge$ *e unique ancestor path n to r* **do**
    foreach $r \in R$ **do**
      if $|e(r)| = 1 \wedge ann(\tau, e(r))$ **then**
        $A[r] \leftarrow e(r)$;
    if $\frac{|A|}{|R|} < \min^{\text{ATTRIBUTE}}_{\text{SUPPORT}}$ **then**
      continue;
    if $\frac{|A|}{|R|} > \min^{\text{ATTRIBUTE}}_{\text{INV-SUPPORT}}$ **then**
      foreach $r \in R$ **do**
        if $|e(r)| = 1 \wedge \neg(ann(\tau, e(r)))$ **then**
          $A[r] \leftarrow e(r)$
    candidateAttrs$(\tau) \leftarrow$ candidateAttrs$(\tau) \cup \{A\}$
foreach $\tau \in$ candidateTypes : $|\text{candidateAttrs}(\tau)| > 0$ **do**
  foreach r *in*R **do**
    $N \leftarrow \{A \in \text{candidateAttrs}(\tau) : A[r] \neq \texttt{NULL}\}$
    $A_\tau[r] \leftarrow \begin{cases} A[r] & \text{if } A \in N \text{ and score}(A) = \max\{\text{score}(A') : A' \in N\} \\ \texttt{NULL} & \text{otherwise} \end{cases}$
  alignedAttrs$(\tau) \leftarrow A_\tau$
return alignedAttrs

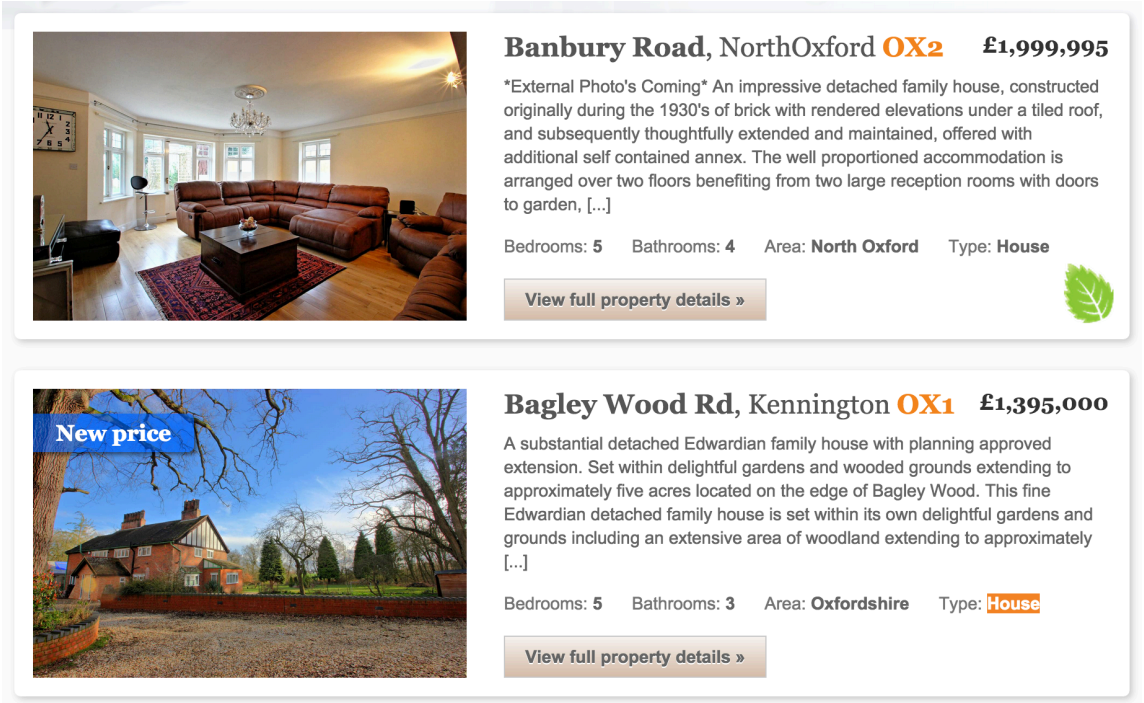


Figure 4.8: Examples on intuitive scores computation

properly, two "family house" appear in description and one "House" occurs the first sentence of the description, one in the fourth line and one at the bottom(attribute list), "Bagley Wood Rd, Kennington" in the title and "Oxfordshire" at the bottom are annotated as location. In addition, "Bagley Wood" in the third line of the description is also annotated as location, and "New price" on the image is annotated as property status. Since there are only two records, the support is at least 50%, so all those attributes are considered as candidate attributes. If we add four more records and neither of them are having "New price" attribute, then we will not add "new price" to the candidates list since the support rate is less than 25%.

For "Bedrooms: 5" and "Bathrooms: 3" attributes, although they are the only instance of their attribute types, but it is worth noticing that those attributes are having a label annotated in their precedent node. For location attribute, "Banbury Road, North Oxford" in the first record beats "North Oxford" on *SPAN* and *FONT*, and will become the location attribute. For the second records, "Bagley Wood Rd, Kennington" has the highest score due to the large font size and almost full span coverage rate, "Oxfordshire" beats "Bagley Wood" in the description since "Bagley Wood" has a even lower span coverage rate and the smallest font size. In addition, "North Oxford" and "Oxfordshire" will be annotated as "Area", "Oxfordshire" will also be annotated as county since it is in our county gazetteer. Even if we do not have "Area" label in our annotation schema, since it occurs frequently, regularly in different records and are label-shaped, AMBER is going to automatically recognize that and add it to the SNER . For property type attribute, the two "family house" in the description are redundancy for each other and provides support on *REDUNANCY*. However, they still got lower score than "House" since the latter has a label, slighly larger font size and full span coverage rate.

AMBER aligns attributes from records and extract > 10 attributes for each domain. The evaluation results are listed in Section 7.1. Besides, AMBER also have a module for gazetteer self-boosting, described in the next subsection.

4.7 Gazetteer Self Bootstrap

For monitoring all advertisements in the UK real-estate market, we have to wrap more than 5,000 different web sites, including sophisticated nation wide aggregators with thousands of offers as well as manually maintained sites of small agencies offering only a handful of properties.

Resorting to established technologies, we face a dilemma: We either rely on unsupervised but inaccurate extraction tools, e.g., [17, 70, 84], or we semi-manually induce wrappers for each of these sites, achieving accuracy through manual annotations, e.g., [22, 64]. The latter choice

requires the generation of example pages with human-quality annotations for all relevant data, infeasible for a domain of >5,000 different sites.

To overcome this dilemma between accuracy and scalability, we introduce AMBER for extracting data from an entire domain, providing extraction results which are highly accurate while requiring some domain knowledge but not human supervision per site.

AMBER analyzes the annotated DOM tree and thus searches for *repetitions in the annotated structure.* AMBER proceeds in the following three phases:

(1) *Page segmentation* identifies areas with relevant data and segments them into records based on the page's DOM, including the visual layout, augmented with both domain-dependent and domain-independent textual annotations generated from gazetteers such as UK locations.

(2) *Attribute alignment* matches the page structure against domain constraints in order to fix the attributes and to obtain a suitable record model.

(3) Finally, we extend the existing gazetteers, in the *gazetteer learning*: AMBER collects the terms occurring in apriori unannotated text nodes, splits them if necessary, and determines a confidence value for the suggested terms. Depending on the calculated confidence and configuration, AMBER adds the found terms to the gazetteers either automatically or semi-automatically. AMBER also validates the utility of formerly added terms in identifying new attributes. If a term never occurs again, AMBER lowers its confidence, optionally asking a human operator for a decision.

This section demonstrates AMBER's approach for bootstrapping gazetteers and its data extraction capabilities. The gazetteer learning proceeds through incremental runs of AMBER until no new terms can be learned, where each iteration involves the three steps as described above. Thus, in the beginning, annotations serve merely as a starting point for extrapolation, while over time, they turn into requirements for recognition. The bootstrapped gazetteers are valuable, not only for AMBER itself but for other tools as well.

As described in section 4.2, AMBER relies on a new class of annotators SNER (Structured Named Entity Rec- ognizer), which takes into consideration both the HTML structure and CSS visual styling during the annotation. The textual annotation in a SNER is implemented as gate [20] applications. They are either annie gazetteers semi automatically generated using sources such as dbpedia [2], jape rules encoding regular expressions for, e.g., prices and postcodes, or interfaces to existing NERs such as opencalais and stanfordner. Among all these ways, the gazetteer based annotation is relatively most costly.

To reduce the time of establishing a gazetteer, AMBER learns terms for its gazetteers starting with a small seed gazetteer and expands this gazetteer throughout a number of iterations over the following three phases. An simplified representation of the result is shown in Figure 4.9

with n records, highlighting occurrences of terms from the gazetteer on the left.

Page Segmentation This phase proceeds in three sub steps, turning a page into a set of data areas consisting of individual records:

(1) *page retrieval and annotation,* AMBER loads and renders a page, evaluates all embedded scripts, and annotates the contained domain terms, using GATE [20] as annotation engine. We provide the necessary terms as gazetteer lists, which are initially either manually assembled or automatically derived from external sources, such as DBPedia.

(2) *data area identification,* AMBER identifies the page areas containing relevant data, organized in structurally similar records.

(3) *record segmentation,* Having obtained the data areas, AMBER segments each data area into individual segments, each corresponding to a single record.

Attribute Alignment In this phase, we choose the attribute instances to be associated with each record. To check whether the identified attribute instances are coherent with the discovered repeated structure, AMBER compares for each attribute all *relative paths* leading to its instances as shown in Figure 4.9 where the green paths lead from the first node of the corresponding segment to the attribute instance in question, moving along first-child and next-sibling axes. We link semantic annotations with the DOM structure by considering *type-path pairs,* each consisting of an attribute type and such a relative attribute path. We call the instances with the same type-path pair the *support set* of that type-path pair. Intuitively, the larger the support sets, the more annotations are coherent with the repeated structure.

Having analyzed the type-pair paths and their support, the reconciliation phase proceeds with three sub steps.

(1) In *Attribute cleanup* process, AMBER (a) discards instances with a support below a threshold l, (b) prompts the user for a decision on instances with support between l and u, and (c) accepts all instances with support above threshold u. The parameters l and u are either specified directly or given as quota fractions l' and u'. In the latter case, AMBER discards the l' fraction of the most weakly supported instances and accepts the top u' fraction without user interaction, prompting the user for all remaining candidates. Thus, by setting $l = u$, AMBER cleans the attribute instances fully automatically. In practical usage, l and u are thresholds $\min^{\text{ATTRIBUTE}}_{\text{SUPPORT}}$ and $\min^{\text{ATTRIBUTE}}_{\text{INV-SUPPORT}}$ mentioned in section 4.6 respectively.

(2) During *attribute disambiguation,* AMBER considers those records which contain more than one instance for a single attribute and elects those instances with maximum support.

(3) In *attribute generalization,* AMBER checks records for missing mandatory attribute in-

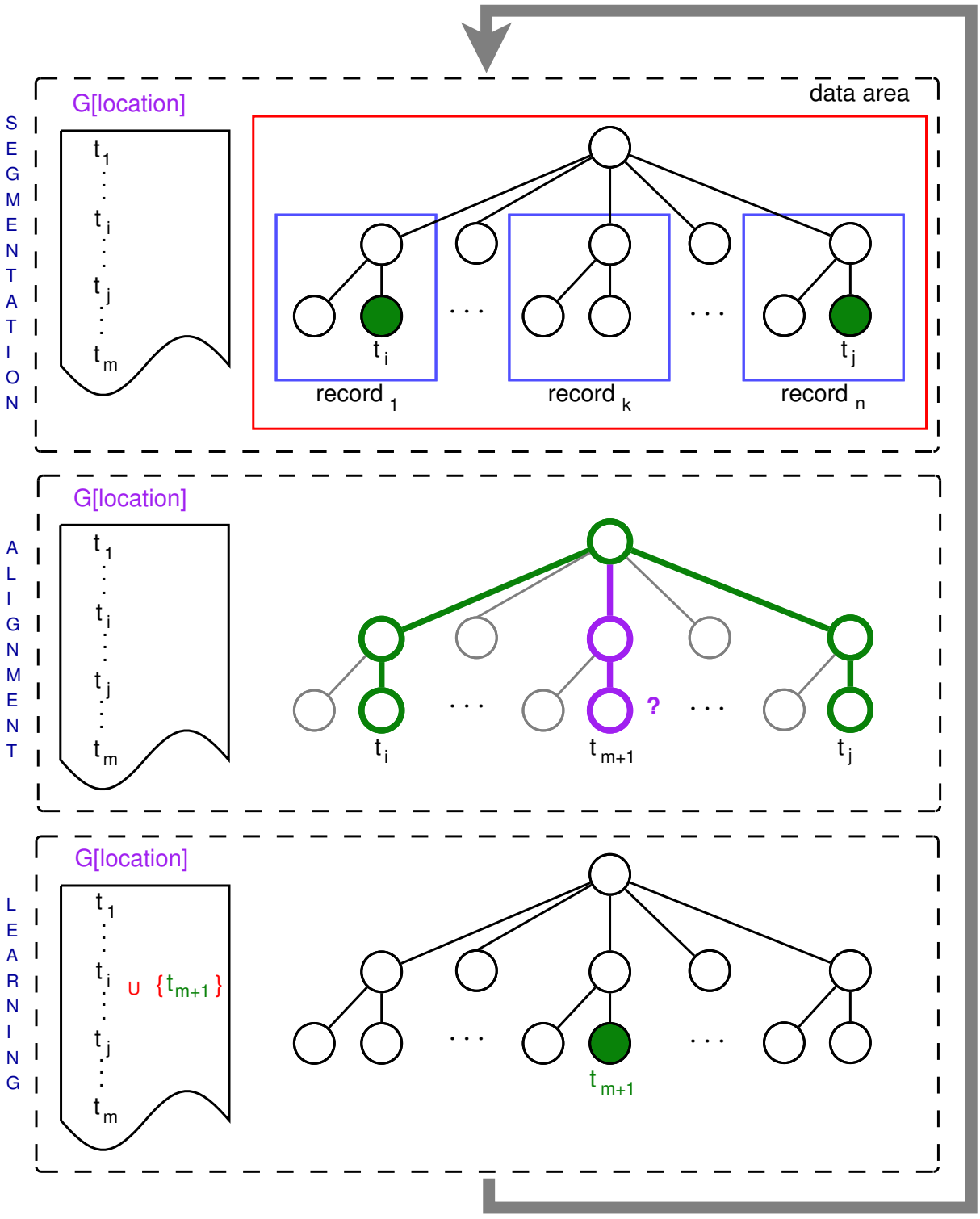


Figure 4.9: AMBER approach

stances and adds new instances at unannotated nodes, if the corresponding type-path pair has support larger than a threshold g, and the path in the pair leads to the text node in consideration. This is the case in Figure 4.9 for the node containing term t_{m+1} with a violet path from the segment root which is sufficiently supported by the green paths. If the support of a type-pair path is low, the user is prompted to check whether the instance belongs to the attribute.

The attributes obtained during generalization are candidates for learning, as they were missing in the original gazetteer and had to be inferred structurally.

Gazetteer Learning For learning, AMBER performs the following two steps to extract terms from the attribute instances obtained during the attribute generalization. During **(1)** *term formulation,* AMBER splits the text node identified as an attribute instance into relevant terms. For example, in the description "Oxford, Walton Street, top-floor apartment", a location gazetteer should contain "Oxford" and "Walton Street" as location terms, while the "top-floor apartment" should be ignored. After splitting the attribute slots into new terms, we remove all terms appearing in a blacklist of excluded terms. This list contains both terms already belonging to disjoint gazetteers and terms learned to be excluded. At last, a confidence value for each term is computed, involving the size of the support set and the size of the term as compared with the entire instance that contained it. If the resulting confidence is low, the user is optionally prompted. During **(2)** *term validation,* AMBER deals with false positives, i.e., incorrectly added terms. To this end, AMBER tracks the relevance of learned terms by checking in later iterations whether such a term occurs again in an attribute instance with sufficient support. If this is not the case, AMBER blacklists the term and removes it from the gazetteer.

The evaluations on AMBER's gazetteer self-boosting ability are shown in Section 7.1.4.

Chapter 5

AMBER in DIADEM project

The result page analysis part of AMBER presented in the previous section introduced the idea of using pivot attributes as a hint and adopting a considerably simpler template discovery and ad-hoc detection of regularities for extracting objects from result pages. Not only as an individual system, AMBER is also a major component of the DIADEM system [32, 33], the first full-site data exploration system for complex structured data, which explores entire web sites fully automatically to find relevant data and induce OXPath [35] wrappers.

In DIADEM, AMBER provides template discovery that helps to identify and segment detect relevant data and provides input for the separate wrapper induction component. The architecture and components of DIADEM are briefly sketched in [32], however based on the earlier version of AMBER [34]. A more extensive discussion of the work flow and principles of DIADEM is presented in [33], with focuses on the integration and inter-component communication by means of a network of relational transducers.

This chapter gives a brief idea on DIADEM system. Section provides a brief introduction of DIADEM system, section 5.2 illustrates the whole process through running examples, and section 5.3 states the approaches of DIADEM. For each of these websites, if DIADEM can locate a result page, AMBER is employed to detect the template structure of those pages and provide such information to the wrapper induction component. A large-scale evaluation of DIADEM on over 10k websites from UK and US real estate domain and used cars domain is included in the evaluation chapter, section 7.2.

5.1 Introduction

The automatic, yet accurate extraction of the structured data underlying web pages is a long standing challenge [10]. Semi-supervised data extraction approaches, such as [5, 22], have been investigated extensively, but require users to supervise the induction by navigating each

site and identifying relevant data. In contrast, **automatic full-site extraction** (AFE) operates *automatically* with no per-site supervision, navigates to all relevant data on the *full site*, yet *extracts* highly *structured data*.

Automatic full-site extraction involves three primary sub-problems, namely site exploration (with form understanding and filling), record and attribute identification, and wrapper induction. Each of these sub-problems is a significant challenge by itself, but worse, these problems have in the past been tackled in isolation with few exceptions. Successful applications of AFE have been limited to narrow settings with simple structure, such as title and body extraction for news articles [79] or search engine results (ViNTs [85]). For extracting highly structured data, these approaches are unsuitable. Furthermore, most of these approaches fail on modern sites. For example, ViNTs [85] full-site extraction identifies records (of title and body only) with only 83%-88% accuracy (Section 7.1.3), even when supervised by selecting negative and positive examples of result pages for each site.

Exploration includes focused crawling of sites for relevant data and automatic understanding and filling of forms. Both form labeling to associate text labels to fields [26, 65, 75] and form filling [62, 76] have been considered before, but mostly in isolation. Form filling approaches are generally supervised [62]; automatic approaches require some per site supervision (e.g., past submissions [76]) and suffer from low accuracy.

Record and attribute identification have proven difficult to automatize due to the high variability of web sites. Early automatic approaches [18, 58] suffer from low accuracy as they recognize any regular structure, including navigation menus and sidebars. This has long limited accurate data extraction to approaches that require per-site supervision and thus do not scale to large number of sites. The exceptions are approaches limited to specific domains (news [79]) or HTML structures (HTML tables [9]).

Wrapper induction exploits the fact that from few example pages a pattern applicable to all similar pages can be derived, yielding an efficient extraction program, called a wrapper. Automatic wrapper induction has in the past focused on induction for sequences of result pages, rather than on induction for full sites. In particular, most automatic wrapper induction approaches [22] do not consider the generalization of exploration sequences.

This need of accurate, yet automated solutions that can be employed at large-scale has lead to a revival in data extraction in the last few years, focused on record and attribute identification:

(1) Crowd-sourced training data has been shown [19] effective in reducing the cost of supervision via a reduction to simple, yes/no queries, answerable by untrained crowd workers. While this increases the scalability of supervised approaches, extraction from thousands of websites remains very costly.

(2) Several recent approaches [7, 43] exploit cross-site redundancy, typically found in some product domains such as electronics. Starting with manually extracted initial data sets, redundant objects on the remaining sites are identified as training data for those sites. To be effective, these approaches unfortunately require substantial redundancy between sites, which among others makes them unsuitable for sites with few objects.

(3) The most generally applicable approach relies on domain knowledge to identify relevant data. Such domain knowledge comes in two shapes, either as a schema of the expected data or as recognizers for domain entities (in form of dictionaries or regular expressions). However, existing approaches are limited in how they apply that domain knowledge: Some [73] only use it to classify attributes, thus are unable to compensate for errors in determining record and attribute boundaries; some [23] focus on single attribute extraction and are ill-suited for extracting complex records with many attributes.

This lack of integrated, full-site extraction approaches has significantly reduced the impact of data extraction—with some commercial applications such as Google Products moving away from extraction towards purely curated data collection. Yet, the need has only increased, in particular with the rise of big data analytics for competitive price intelligence or intelligent supply chain management. In many of these applications, the acquisition of accurate, large-scale data, e.g., about competitor's products, current deals, or supply levels are crucial.

In summary, automatic, accurate full-site extraction at web scale has remained elusive. Previous approaches have presented different trade-offs between required supervision and achieved accuracy. However, at the scale of thousands of sites, no previous approach provides accurate data extraction ($> 80\%$) without significant per-site supervision or strong limitations on the types of sites or data.

DIADEM is the first automatic full-site extraction system that is able to extract structured data from different domains, such as real estate or used cars, at very high accuracy. In contrast to previous AFE systems, such as ViNTs [85], DIADEM extracts, for over 90% of the 10602 evaluated websites, highly structured data with dozens of attributes at 97% accuracy. In contrast to existing approaches focusing on only one or two sub-problems of AFE, DIADEM is an integrated system that solves each of the sub-problems given just a list of sites. In contrast to most previous data extraction approaches, it requires no per-site supervision, not even the selection of result pages following the same template, as, e.g., in [23].

DIADEM achieves this thanks to two primary innovations addressing the most challenging issues in AFE:

(1) The automation of solutions for exploration, identification, and induction at high accuracy.

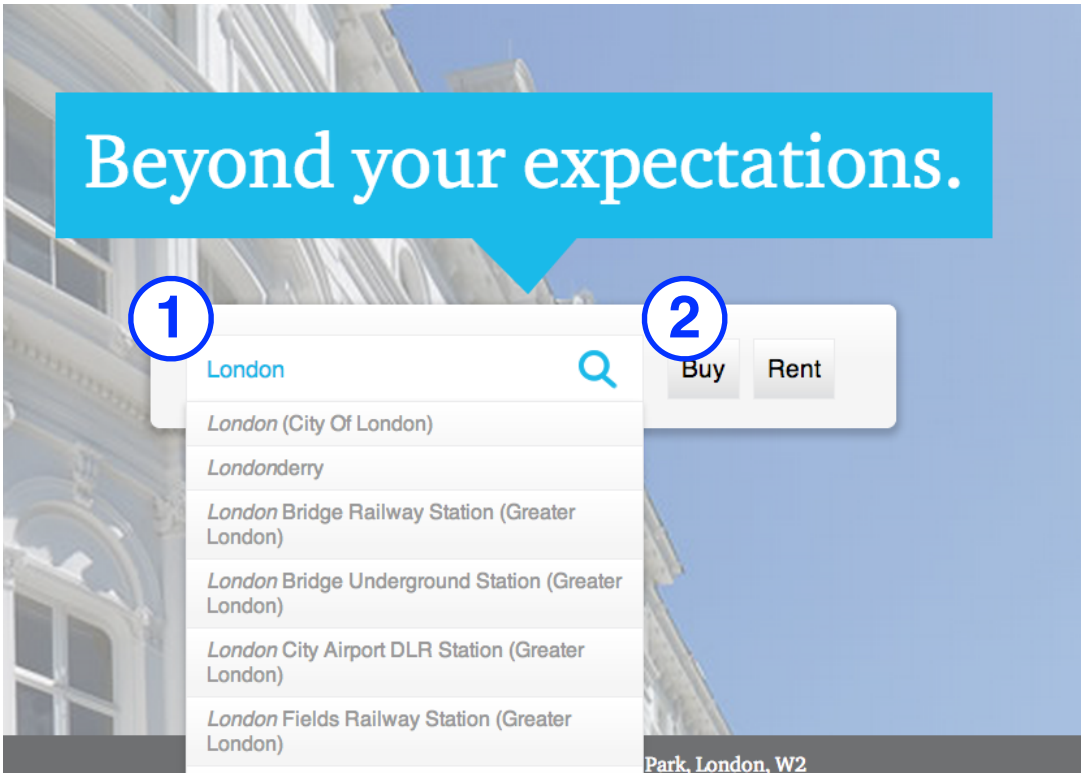


Figure 5.1: Hamptons form

(2) The integration of these components for dynamically adapting its exploration to cope with the wide variety of web sites.

Challenge **(1)** is addressed in DIADEM through domain knowledge. Domain knowledge has been used in existing extraction approaches [73, 23, 25], however, DIADEM uses knowledge in a unique fashion: Its domain knowledge, the **DIADEM ontology**, not only includes a schema of the domain and entity recognizers for the schema types, but also classifies and constrains these types with respect to their appearance on web sites. This *phenomenology* defines, e.g., that some attributes are mandatory, some only appear together, and some attribute values are wild-cards for form filling (e.g., an option "All"). DIADEM demonstrates that a small set of observations about web data in the phenomenology (about 100 per domain) suffices to significantly improve the accuracy of automated data extraction, both overall and in each component (Section 7.2).

Challenge **(2)** is addressed through a novel analysis and work flow engine realized as a **self-adaptive network of relational transducers**, each representing a component of DIADEM. The network adapts itself according to previously collected knowledge about a site, e.g., to rearrange the transducer execution order or to react to exceptions. For the 10602 sites of our evaluation, this yields thousands of different sequences of transducers, each adapted to a par-

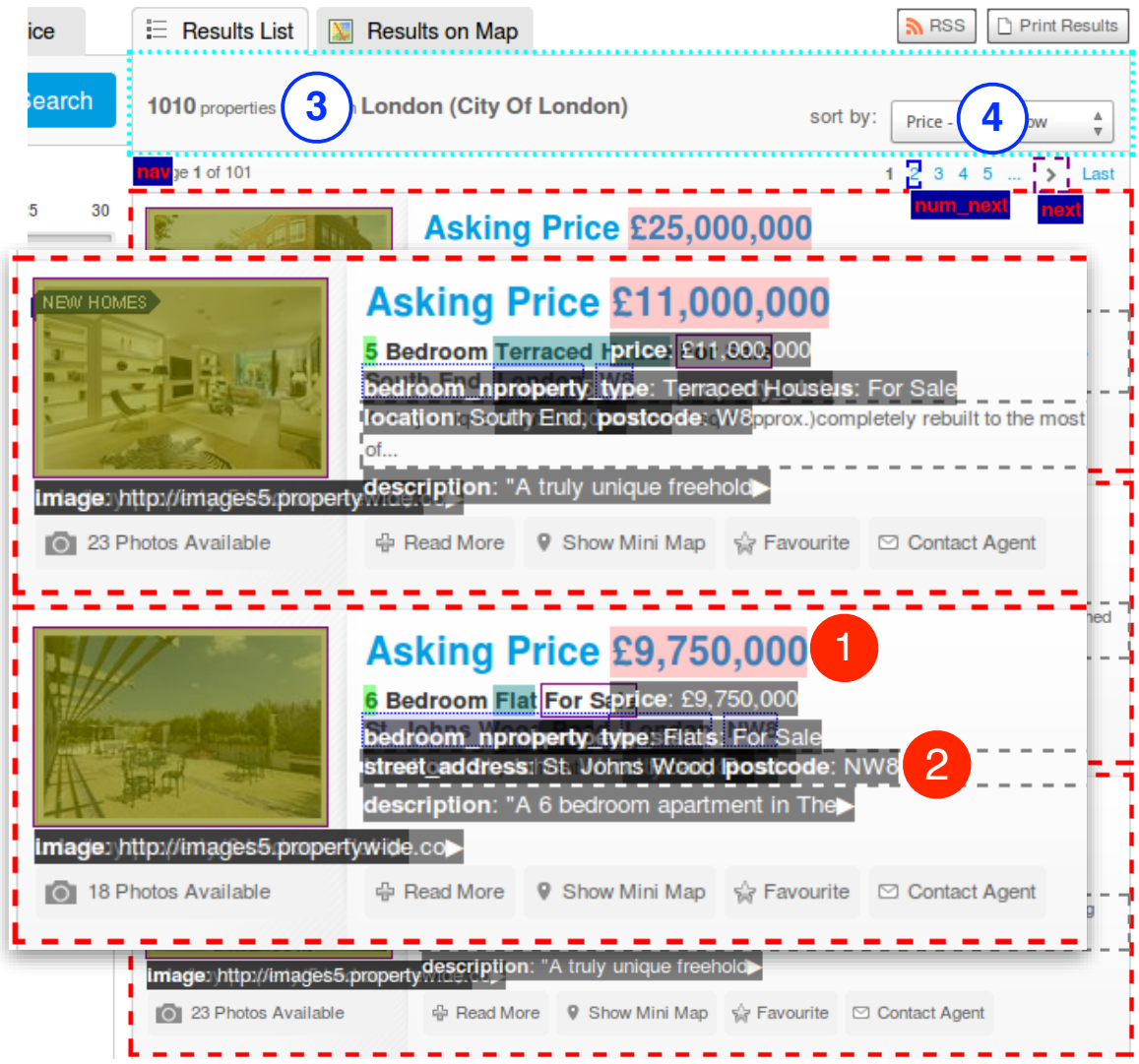


Figure 5.2: Hamptons listings page

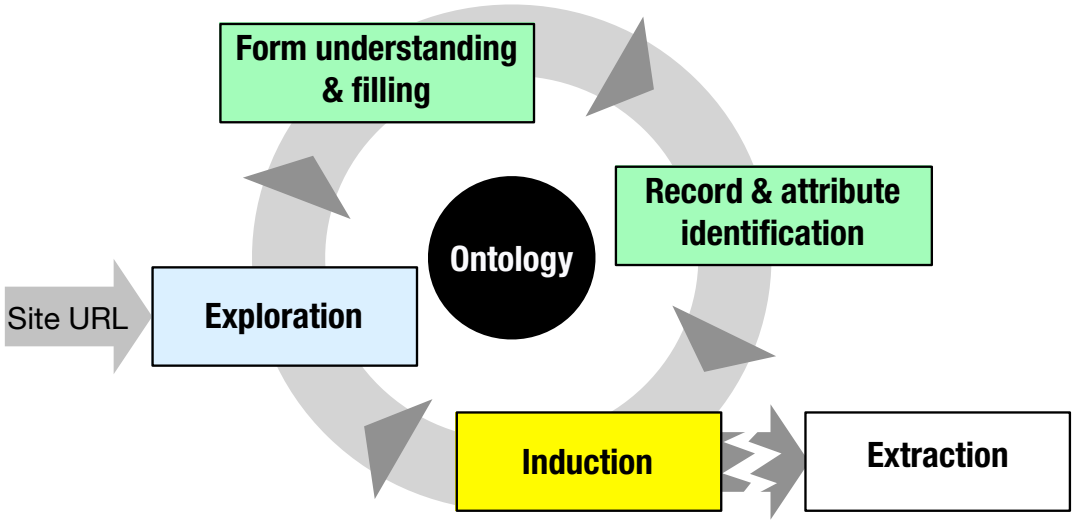


Figure 5.3: DIADEM architecture

ticular site's shape. To improve scalability and isolation of individual transducers, DIADEM enforces an access policy with fine granularity that dynamically defines the data accessible to each transducer and the control flow in the network.

DIADEM's automatic full-site exploration, schematically shown in Figure 5.3, is driven by the knowledge in this ontology. Starting with only a site's URL, DIADEM explores the website through a combination of focused crawling and form filling, identifies records and attributes for extraction, and induces a wrapper for eventual extraction of all relevant data. Each component is realized as a relational transducer, together forming a network of relational transducers that adapts itself to react to observations as necessary. In the following, we focus on the integration and communication of DIADEM's three major components:

(1) OPAL, *form understanding*, uses domain knowledge to generate fillings and to react to feedback from filling;

(2) AMBER, *record and attribute identification*, that encodes domain-independent rules for detecting patterns on web sites and applies domain knowledge to implement a domain-aware template discovery system for attribute extraction;

(3) OXPath, an extraction language for *wrapper induction*. The process accumulates information about all identified result pages and the navigation paths leading to them and integrates that information into a coherent wrapper.

Contributions The contributions of the DIADEM system are:

(i) DIADEM automates full-site extraction without site-specific supervision, yet achieves accurate extraction (97%) of highly structured data on most sites (90%). This is demonstrated by an evaluation over a diverse set of 10602 websites from UK and US real estate and UK used

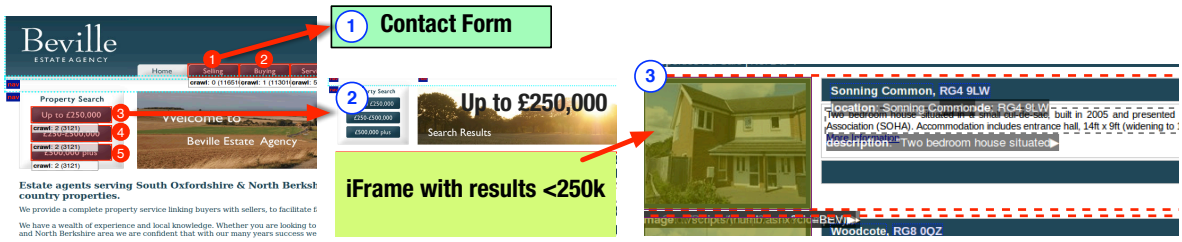


Figure 5.4: Beville exploration

car (Section 7.2). DIADEM outperforms, both as a whole and in its individual components, previous automated solutions, often by a significant margin.

(ii) DIADEM achieves this automation through a novel kind of domain knowledge, the DIADEM ontology. This information is provided once for the entire domain and is the only form of supervision in DIADEM (Section 7.2 outlines the needed effort).

(iii) DIADEM integrates solutions for exploration, identification, and induction into a self-adaptive synchronized network of relational transducers . Most transducers use specifications of phenomenological patterns (describing how objects appear on the web) or of finite state transducers with transitions guarded by first-order formulas.

5.2 Example for DIADEM

Figure 5.1 shows a modern form on the homepage of `hamptons.co.uk`, a UK real estate agent. Given this URL, DIADEM identifies the form, including its buttons and search field despite the fact that the form element covers nearly the whole page and contains no proper submit button. What appears as two buttons (❷) are in fact styled HTML links. To identify these, DIADEM's form understanding uses a combination of visual, structural, and textual clues, specifically the vicinity to the text field (❶) and their text labels and IDs. Using the entity recognizers from DIADEM's ontology, the form understanding picks up the example value in the text field. Thus, it classifies the text field as a location and the two links as a buy/rent switch according to the ontology.

DIADEM's form filling uses the information from the form understanding to fill the form and detects that the location is mandatory (as submissions without filling that field fail with a red highlight on the text field). It proceeds to fill the text field with "London", a location recognized on the page. This triggers an auto-complete list which is an expected behavior for a text input, according to DIADEM's ontology. DIADEM identifies the auto-complete list and iterates over it for submission.

The listings page (Figure 5.2) contains many regular structures. Fortunately, DIADEM's on-

```
doc('http://hamptons.co.uk/')//a[@id='Generic...ester_4']/prec-sibl::input[contains(@type,'text')]/{
    /} ❶
      //div[@id='SearchContainerMulti']/a[@id='saleSearchButton']/{click /} ❷
      /.:<data_area>[? .//h1/span/span[1]/text()[1]:<search_results_number=norm(.)>
          ] ❸
  /(//div[3]//li[1]/foll-sibl::li[@class='paging-next']/a[@class='pagingbutton']/{nextclick
      /})* ❹
      //ul/node()/div[@class='One']:<record>
    [? .//label/text():<price=norm(.)> ] ❶
    [? .e//span[@class='result-address']/text():<location=norm(.)> ] ❷
    [? .//a[contains(.,'Bedroom')]/text():<bedroom_number=substring-before(norm(.),"
       Bedroom")> ]
    [?
        .//span[@class='result-address']/prec-sibl::text():<property_status=substring(norm(.),
       len(norm(.)) - 7)> ]
    [? .//span[@class='result-address']/prec-sibl::text():<property_type=norm(.)> ]
    [? .//div/foll-sibl::p:<description=norm(.)> ]
    [? .//img[@class='PropertyMainImage']/@src:<image=norm(.)> ]
    [? ./div[@class='inner-photo-wrap']//@href:<url=norm(.)> ]
```

Figure 5.5: Hamptons wrapper

tology prescribes that real estate records must contain prices and DIADEM's domain-aware template discovery uses that to identify and segment the most likely records. Within these records it identifies and aligns the structural attributes such as PRICE (❶) and LOCATION (❷), maximizing inter-record regularity. For attribute identification, it again uses entity recognizers, recognizing both labels ("location") and instances ("London") of entities. It also identifies the two links (❹) and meta data about the number of returned records (❸)—useful to verify the quality of the induced wrapper (Section 7.2). DIADEM follows the next links to generalist the record and attribute structure from multiple result pages. Figure 5.5 shows the OXPath wrapper (with some abbreviations) that DIADEM induces from the identified records and the explored navigation paths. For brevity, it omits the symmetric part for rentals. With this wrapper, DIADEM manages to extract all properties for any location in the UK from `hamptons.co.uk` successfully.

To illustrate the variety of websites DIADEM is able to explore, we discuss an unusual exploration case. Figure 5.4 shows DIADEM's exploration sequence on `beville.co.uk`. This site does not use a form, but rather three pre-defined queries, such as "Up to 250,000", for accessing the properties. Furthermore, the properties are shown in an iFrame pointing to a real-estate hosting provider. DIADEM explores this site successfully: On the homepage, it ranks relevant links, here "Selling" (❶), "Buying" (❷), and then the three buttons for the pre-defined queries (❸–❺). Exploring the "Selling" and "Buying" links leads to no data, but a contact form (❶), which DIADEM correctly identifies and discards. DIADEM then explores each of the

links for the predefined queries (❸–❺). For each of these links, it reaches a page (❷) where the results are contained in an iFrame (shown for the first link in Figure 5.4). It identifies that the `iFrame` most likely contains the main content, switches to the frame (❸), and identifies the records.

This is just a brief foray into the exploration performed by DIADEM. It deals with a vast variety of forms, navigation structures, and result pages. Many more examples can be viewed in the automatically generated reports for the 10602 evaluated websites, see http://diadem.cs.ox.ac.uk/evaluation/14/02.

5.3 AMBER in DIADEM

To ensure efficiency on the large scale evaluation (>10, 000) sites, DIADEM uses an earlier version of AMBER, which only focuses on extracting objects from result pages. As a major component of DIADEM, AMBER contributes in the following three aspects:

(1) For each of these websites, if DIADEM is able to locate a page through a navigation link or a form query, AMBER is employed to detect the template structure, the result of AMBER determines the success of navigation or form query.

(2) AMBER provides template discovery that identifies the relevant data area, segments the primary data area into records, and extracts multiple attribute (>10 per domain) from each record.

(3) The output of AMBER has been utilized to feed the wrapper induction component, which automatically generalizes the template for each web site.

In general, for each website, AMBER validates the exploration process, extracts accuracy and complete data from the web, then provides those data to feed a wrapper induction system for extracting all objects from the website. Therefore, AMBER is the crucial component of DIADEM, determines the output quality of the whole system. The large scale evaluation shows that DIADEM obtains an effective wrapper that extracts all relevant data with 97% average precision.

A start-up company has been established with DIADEM system, under the support of the university of Oxford, to extract data based on requirement of clients. Collaborations have been built between the DIADEM start-up and some well-known industrial companies.

Chapter 6

Detail Page Analysis

In object extraction area, result page analysis is fast, effective and is able to provide major attributes. However, the object information listed on result page are still quite limited due to user interface design style and space limitation. Many websites would prefer illustrating lists of objects with a limited number of attributes, short summaries, and links to detail pages which illustrating complete long descriptions. Therefore, after the automatic result page analysis, AMBER collects all the detail page links from result page records and is able to navigate to detail pages one by one to gather full information. In general, AMBER is a domain-aware, self-supervised object extraction system which extracts information from both result and detail pages.

Figure 6.1 illustrates the work flow of AMBER on detail pages. In this chapter, section 6.1 gives an overall view of the approach, section 6.2 describes AMBER's annotation framework, section 6.3 shows how AMBER calculates the data areas of given detail pages based on boilerplate removal, dynamic list removal and page dissimilarity calculation, then illustrates the process through one running example in section 6.4. Section 6.5 describes the features AMBER employs in its attribute extraction part, and 6.6 finishes the chapter with the an example for attribute extraction.

6.1 The AMBER Approach

AMBER takes as *input* **(I1)** a URL of a result-page; **(I2)** a relation $R = (A_1, \ldots, A_n)$, where $A_1, \ldots, A_n$ are attributes of R, acting as a schema describing the entities of interest and attributes; **(I3)** a mapping $\mathcal{M} : A_i \mapsto T_{A_i}$ from attributes in R to annotation types defining which annotations AMBER should look for to locate and extract the attribute A. The analysis *outputs* for each detail page p, reached via the result page, a location vector $l_p = (l_1 \ldots, l_n)$, where l_i is the DOM node in page p containing the value for the attribute A_i in the relation R. In addition, AMBER provides

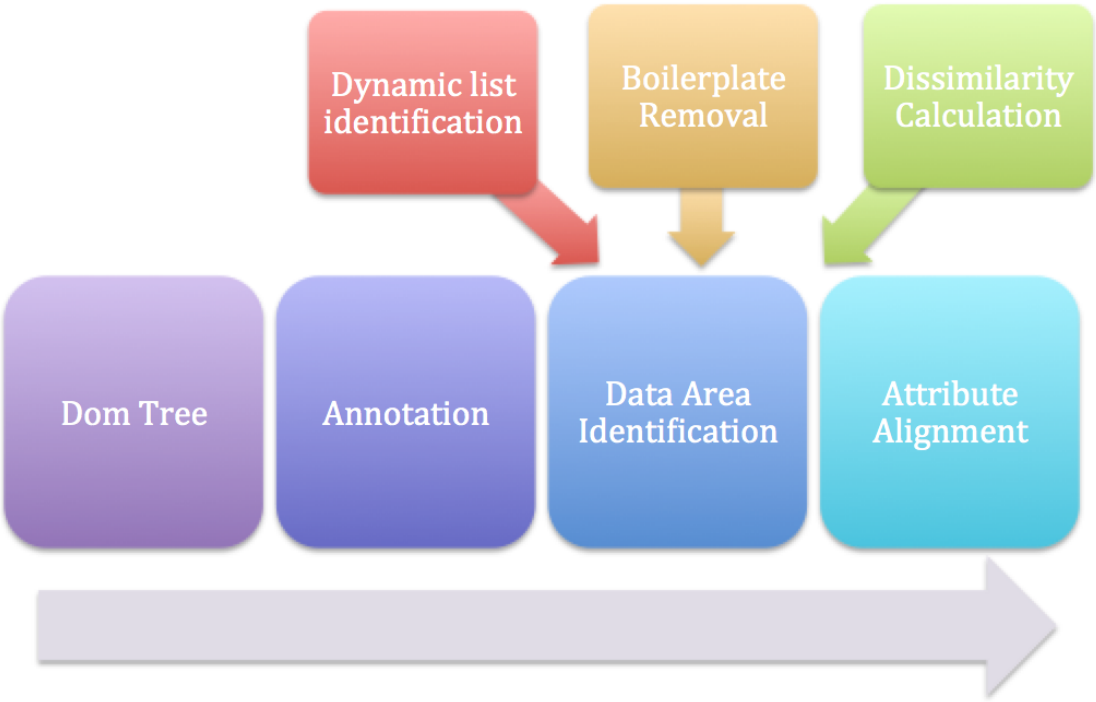


Figure 6.1: AMBER workflow on detail pages

for each detail page the DOM node on the result page which links to the detail page.

As in other self-supervised approaches (e.g., [23, 25, 69]), AMBER uses automatically-generated *annotations* to lower the manual effort required to construct examples for induction algorithms. For this, AMBER is designed to **(i)** robustly induce wrappers in presence of noisy annotations, and to **(ii)** mutually verify annotations from different sources to improve the immediate output and to learn from corrected inaccuracies for future applications of AMBER.

The verified annotations support AMBER's *repeated-structure analysis* in the following ways:

(i) Firstly, AMBER locates (possibly multiple) data areas on result pages, breaking them into regular record-like structures.

(ii) AMBER then recognizes, within each record, attributes of R by comparing the DOM structures of each record. In particular, AMBER locates the links to the detail pages.

(iii) AMBER contributes to the boilerplate removal process which distinguishes the main content of a detail page from other non-informative parts,

(iv) AMBER also applies result page analysis mechanism on detail pages for dynamically gen-

erated recommendation lists,

(v) At the same time, AMBER locates the data areas on detail pages by comparing the DOMs of different pages linked by the same result page.

(vi) Finally, AMBER locates DOM nodes representing attributes of R within data areas on detail pages and aligns them with attributes and labels located within the corresponding data areas on result pages.

6.2 Annotation

The detail page annotation process reuses the annotation module SNER , described in section 4.2, but with slightly different attribute types, listed in table 6.1.

Table 6.1: Annotation Schema.

		Annotation Schema (Σ)	
		Characteristic (Σ_C)	Optional (Σ_O)
Domain	REAL ESTATE	BEDROOMS, LOCATION, BATHROOMS, PRICE	RECEPTIONS, TOWN, CITY, VILLAGE, COUNTY, POSTCODE,PROPERTY_TYPE, FURNISHING, STREET_ADDRESS, PROPERTY_STATUS, AVAILABILITY_DATE, BRANCH_LOCATION, FLOOR_PLAN, MAP, IMAGE, TITLE, DESCRIPTION
	USED CARS	MAKE, FUEL_TYPE, TRANSMISSION, MILEAGE, ENGINE, PRICE	MODEL, ROAD_TAX, EMISSIONS, POWER, COLOUR, BODY_TYPE, MPG, AGE, SEATS, OWNERS, DOORS, REGISTRATION, ENGINE_SIZE, VILLAGE, TOWN, CITY, COUNTY, POSTCODE, STREET_ADDRESS, IMAGE, TITLE, DESCRIPTION

The detail page annotation schema used for two different domains are shown in Table 6.1, which are interpreted according to a reference *annotation schema* $\Sigma = \langle \Sigma_C, \Sigma_O \rangle$ where Σ_C is the set of *characteristic* attribute types, represent attributes that characterize the entity that is targeted by the extraction, e.g., a property or a used-car. Σ_O is the set of *optional* attribute types, representing attributes that can be optional in a detail page and for which no minimal support

is expected. The difference between detail page and results page here is that AMBER does not require a pivot attribute type for detail pages extraction. AMBER's annotation schema for different domains are configurable and parameterizable. Also, it is worth noting that several generic attributes are often present in different domains, e.g., IMAGE, DESCRIPTION, LOCATION. While adapt AMBER to a new target domain, we could always re-use some parts of existing domain schemas (such as generic attributes, prices, etc) to reduce adaption time.

6.3 Data Area Identification

AMBER carries out the analysis in two steps. First, AMBER locates the *data areas* of pages in $\mathcal{P}$, then it marks the nodes on these pages that are likely to contain values for the attributes of the entity of interest. The analysis takes as input a set of detail pages $\mathcal{P}$ and a set of annotations $\mathcal{A}$ and produces a set $\mathcal{L}_{\mathcal{P}}$ of location vectors for attributes found on pages in $\mathcal{P}$.

The data area of a detail page is defined as *the smallest data region containing all attributes of the main object*, so the data area identification is to distinguish the main content of a detail page from other non-informative parts, such as the navigation links, menus, headers, footers, sidebars, related articles, banners and advertisements.

Some previous approaches are based on the assumption that all detail pages from a same website typically share the same template, however they are varied by the content, e.g., the length of the description block, existence of optional attributes, more media information existing. Even the attribute types, number of pictures, description lengths of the two objects are exactly the same, the content of the attributes or the descriptions will not be identical. Even the irrelevant information, such as advertisement, merchant information, copyright information are always identical, both structural and textural. With these assumptions, the dissimilarities between detail pages are likely to be the content.

However, we found that those assumptions are not always hold. On modern detail pages, the advertisements and recommendation blocks are more intelligent and dynamic.

(1) *Recommendation system*. Recently it is common for websites to have a block showing related content. Some recommendations are static website highlights, but recent years dynamic recommendation lists provided by recommender systems through collaborative or content-based filtering became popular, and has been applied in a large number of domains, e.g., "related products" and "people bought this also bought" in product domains, "movies you might like" in movie domain, "properties nearby" in real-estate domain and so on.

(2) *Advertisement*. Many websites may insert a block of advertisement from third-party websites to make some extra revenue, for example, Google AdSense. Most of them allow

publishers of content sites to serve automatic text, image, video, or interactive media advertisements, that are targeted to site content and audiences.

Dissimilarities based algorithm could be fooled by these dynamically generated irrelevant information. In addition, different products in a same domain might have several attributes exactly the same. For example, 2 flats in a same building, the size, number of bedrooms/-bathrooms/reception rooms might be the same, the address is almost the same except the flat number, same postcode, same nearby places and so on. Computing dissimilarities only for content might lose lots of attributes.

With the help of domain knowledge, AMBER produces an algorithm to detect the data area by integrating boilerplate removal, dynamic list identification and dissimilarities between pages. To identify data areas, with a input of a set of detail pages from a same website, AMBER executes the following steps to identify the data area:

(1) *Boilerplate removal*: removes the static non-informative parts, such as the navigation links, menus, headers, footers, sidebars, banners and advertisements from each detail pages given,

(2) *Dynamic list identification*: Executes result page analysis on each detail page to find secondary data areas which are likely to be auto-generated dynamic related-recommendations and remove them,

(3) *Page dissimilarity calculation*: If multiple detail pages are given, computes page dissimilarities with the assist of domain annotations.

(4) Combines the results of the first three steps and gets the data area.

Therefore, even with only one page given, AMBER is still able to detect the data area of the detail page since step (3) will mark all nodes as dissimilar.

Algorithm 5 provides the pseudo-code for the data area identification based on boilerplate removal, dynamic list identification and dissimilarity calculation. Given a set of detail pages $\mathcal{P}$, the algorithm computes data areas $\mathcal{D}$ for each page P_i given in $\mathcal{P}$. To illustrate each component clearly, we separate the pseudo-code of boilerplate removal and page dissimilar calculation in algorithm 6 and 7 respectively, and pass their output as input of algorithm 5.

For each detail page P_i given in $\mathcal{P}$, line 3 applies result page analysis mechanism on each detail page given to locate dynamically generated lists. Then line 5 and line 4 are fetching boilerplate node lists and non-static(dissimilar) node lists of P_i respectively. The data area on page P_i is finally determined as the common ancestor of all nodes that are non-static, not in boilerplate or contained by dynamic generated list. (Line 6).

Algorithm 5: Data Area Identification

input : $\mathcal{P}$ – the set of pages
input : $\mathcal{DN}$ – the set of dissimilar nodes
input : $\mathcal{BP}$ – the set of boilerplate nodes
output : $\mathcal{D}$ – the set of data areas

$\mathcal{D} \leftarrow \emptyset$;
foreach $P_i \in \mathcal{P}$ **do**
    DLNodes $\leftarrow$ `AMBERResultPage`(P_i) ;
    DTNodes $\leftarrow P_i \cap \mathcal{DN}$;
    BPNodes $\leftarrow P_i \cap \mathcal{BP}$;
    $\mathcal{D} \leftarrow \mathcal{D} \cup$ `lca`(DTNodes $\setminus$ (BPNodes $\cup$ DLNodes));
return $\mathcal{D}$

Boilerplate Removal To detect whether or not a node belongs to a boilerplate, a top-to-bottom algorithm is applied. For each detail page $P_i \in \mathcal{P}$, we calculate if a node satisfies any of the boilerplate conditions, from the root node of HTML to its children and descendants. Once a node n_i satisfies all the conditions of one boilerplate type, then all the descendants of n_i will also be marked as boilerplate.

Algorithm 6: Computation of boilerplate nodes

input : $\mathcal{P}$ – the set of boilerplate types and conditions
input : $\mathcal{C}$ – the set of pre-defined boilerplates classes
output : $\mathcal{BP}$ – the set of boilerplate nodes

$\mathcal{BP} \leftarrow \emptyset$;
foreach $P_i \in \mathcal{P}$ **do**
    $List \leftarrow \{getRoot(P_i)\}$;
    while $|List| > 0$ **do**
        $n \leftarrow$ pop$(List)$;
        foreach $C_i \in \mathcal{C}$ **do**
            if *conditionsMatch*(n, C_i) **then**
                $\mathcal{BP} \leftarrow \mathcal{BP} \cup \{n,$ descendants$(n)\}$;
            else
                $List \leftarrow List \cup$ children(n)s ;
return $\mathcal{BP}$

Algorithm 6 demonstrates the boilerplate removal process. Given a set of detail pages $\mathcal{P}$, the algorithm computes the set of boilerplate nodes $\mathcal{BP}$ for each page P_i in $\mathcal{P}$ with assist of a set of pre-defined boilerplate class $\mathcal{C}$, such as header, footer, sidebar, advertisement, banner, menu, navigation links, etc. Each class C_i in $\mathcal{C}$ is constituted by a set of conditions, such as block size, block position, keywords contained in HTML class and ids, and annotation types contained in the text.

The algorithm executes computations on individual pages and do not require cross-page information(Line 2). For each page P_i, we maintain a top-to-bottom list of nodes by binary-first-traverse of the DOM tree stored in a first-in-first-out queue, and the first node of the queue was set to be the root of the HTML DOM. (Line 3-5). On each iteration, we pop n, the first node in the queue (Line 5), and examine if n satisfies the conditions of any boilerplate type through enumerating all the classes in $\mathcal{C}$ (Line 6-7). If n satisfies the conditions of one boilerplate class, then n and all descendants of n are considered as boilerplate nodes and will be added to $\mathcal{BP}$ (Line 8). Otherwise, we expand n to all its direct children and add those children to the queue. Once the queue is empty, the algorithm terminates.

Dynamic list identification AMBER is able to distinguish between the primary result page data areas, which occuring on the most central position of the result pages and secondary result page data areas, as well as regular structures which are not the main content of the page, and on pages which are not the result pages. Those lists are highly liky to represent a list of featured offers, advertisements, recommendations and are used in detail page extraction for identifying dynamic lists. AMBER achieves this with the assist of characteristic attributes and visual block information to validate the data area. The detailed description and evaluation of AMBER's secondary result page data area extraction ability are in section 4.5 and section 7.1.2 respectively.

Page Dissimilarity Calculation It is common that detail pages from the same website are inherited from the same template, those automatically generated static information contained in almost every detail page, lead to the fact that two detail pages from a same website may have high similarities on some parts, although they probably are describing completely different objects. Hence, we could get more explicit data areas from detail pages through calculating the dissimilarities between pages. If some nodes are dissimilar between different pages, those might be related to the objects described, otherwise are likely to be automatic generated static information that are non-relevant. AMBER computes the dissimilarities using a bottom-up algorithm.

Algorithm 7 demonstrates the process of computing page dissimilarities. Given a set of detail pages $\mathcal{P}$, the algorithm executes a pair-wise comparison between every two pages and locates all nodes that are not having full support from all other pages, which are more likely to be the description of the object.

In Algorithm 7, a set of detail pages (more than one) has been taken as input and the

Algorithm 7: Computation of page dissimilarities

```
   input  : 𝒫 – the set of boilerplate types and conditions
   output : 𝒟𝒩 – the set of dissimilar nodes
1  𝒟𝒩 ← ∅;
2  foreach P_i ∈ 𝒫 do
3      List ← bfs(getRoot(P_i));
4      List ← reverse(List);
5      while |List| > 0 do
6          n_i ← pop(List);
7          foreach ∃n_j ∈ P_j with P_i ≠ P_j, P_j ∈ 𝒫 do
8              if sameTagPath(n_i, n_j) then
9                  if isTextNode(n_i) and diffContent(n_i, n_j) then
10                     𝒟𝒩 ← 𝒟𝒩 ∪ n_i;
11                 if diffChildrenNumber(n_i, n_j) or |children(n_i) ∩ 𝒟𝒩| ≠ |children(n_j) ∩ 𝒟𝒩| then
12                     𝒟𝒩 ← 𝒟𝒩 ∪ n_i;
13         if ∄n_j(n_j ∈ P_j, P_j ∈ 𝒫, P_i ≠ P_j) then
14             𝒟𝒩 ← 𝒟𝒩 ∪ n_i;
15 return 𝒟𝒩
```

algorithm is page pair-wised and from bottom-to-top. Line 2 iterates on all pages in $\mathcal{P}$, followed by line 3 to generate a binary-first-traverse of the DOM tree from top-to-bottom, then the list is flipped in line 4 to reverse the top-to-bottom into bottom-to-top order. Next, for each node of page P_i, a page from input $\mathcal{P}$, we first check if there exists another page P_j and a node n_j in P_j, where n_i and n_j are having same tag-path support (Line 7-8). If yes, then node n_i will be considered as a non-static node if it satisfies one of the following two conditions: **(1)** n_i is a text node (leaf node), and n_i and n_j should have different content (Line 9). **(2)** n_i is a non-leaf node, since non-leaf nodes are not able to carry any text data, n_i and n_j should either have different numbers of children nodes, or different numbers of non-static children nodes (Line 11). Line 13 checks if n_i has got no similarity support from any other pages. If so, n_i will be marked as non-static node directly. Finally, we return $\mathcal{DN}$, containing all nodes that are non-static.

With the three components, AMBER is able to extract explicit data areas from detail pages.

6.4 Data Area Example

To illustrate the data area identification process, we randomly picked a detail page from the largest UK real-estate aggregator http://www.rightmove.co.uk as an example. Figure 6.2 and 6.4 are screen shots of two detail pages.

In both Figure 6.2 and 6.4, the area inside the green rectangle is the data area, defined in section 6.3 as the smallest data region containing all attributes of the main object. The three

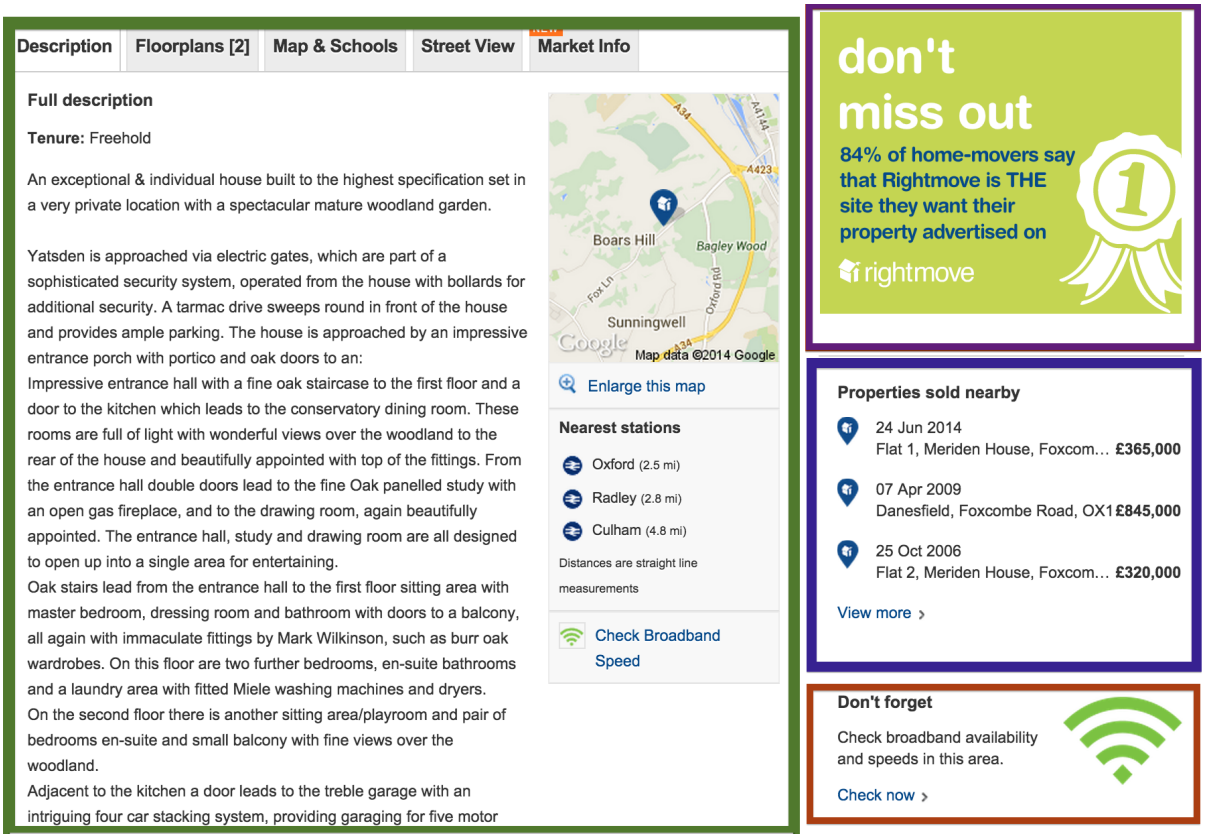


Figure 6.2: Detail page data area example I from rightmove

```
▼<div class="mpu-medium-rectangle inline-block">
  ▼<div id="propertyDetailsMPU">
    ▼<div id="google_ads_iframe_/5029762/Rightmove_Public_Site/Property-Details/firstMPU_0__container__" style="border: 0pt
    none;">
      ▼<iframe id="google_ads_iframe_/5029762/Rightmove_Public_Site/Property-Details/firstMPU_0" name="google_ads_iframe_/5029762/
      Rightmove_Pub[illegible]tails/firstMPU_0" width="300" height="250" scrolling="no" marginwidth="0" marginheight="0"
      frameborder="0" src="javascript:"<html><body style='background:transparent'></body></html>"" style="border: 0px; vertical-
      align: bottom;">
        ▼#document
          ▶<html>…</html>
        </iframe>
      </div>
    ▶<iframe id="google_ads_iframe_/5029762/Rightmove_Public_Site/Property-Details/firstMPU_0__hidden__" name="google_ads_iframe_/
    5029762/Rightmove_Public_Site/Property-Details/firstMPU_0__hidden__" width="0" height="0" scrolling="no" marginwidth="0"
    marginheight="0" frameborder="0" src="javascript:"<html><body style='background:transparent'></body></html>"" style="border:
    0px; vertical-align: bottom; visibility: hidden; display: none;">…</iframe>
    </div>
  </div>
```

Figure 6.3: HTML code for detail page data area example I

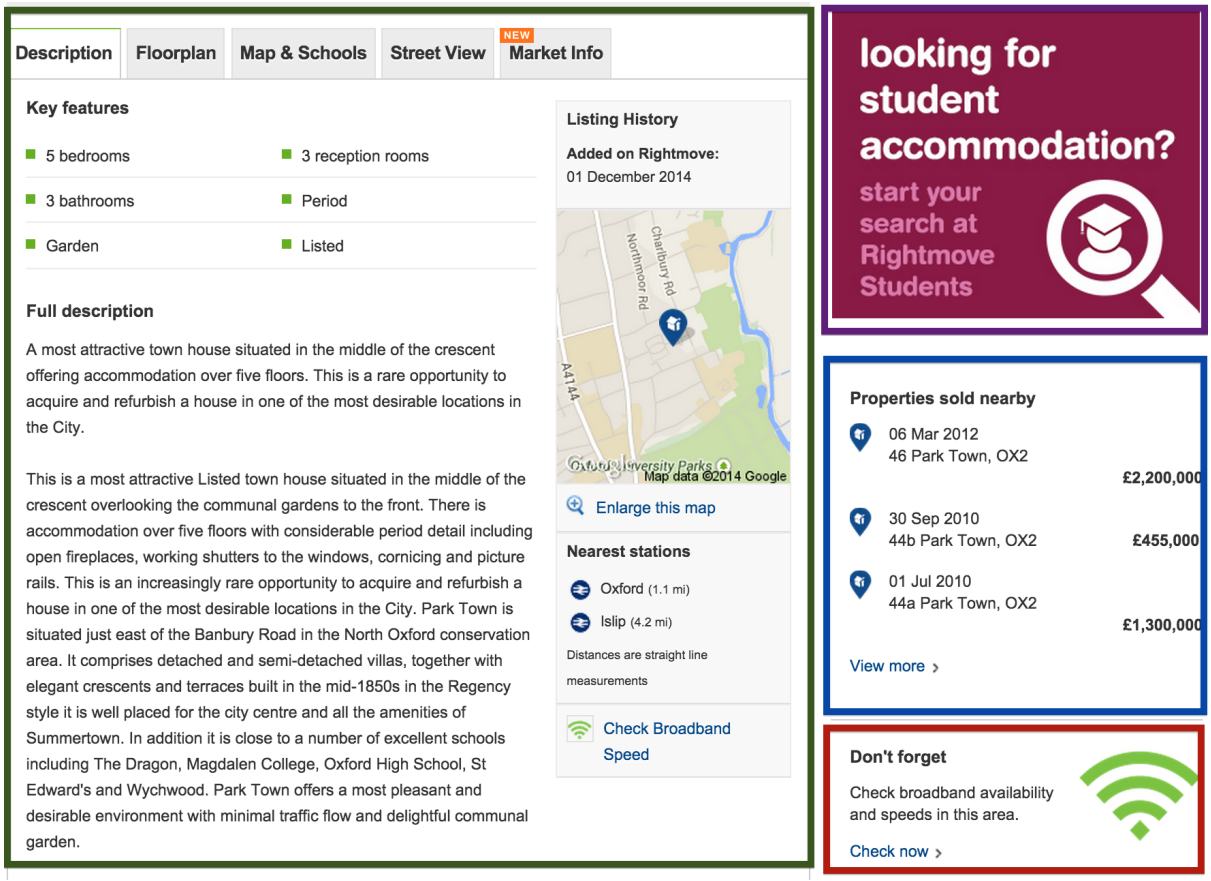


Figure 6.4: Detail page data area example II from rightmove

blocks on the right-hand side are irrelevant blocks automatically generated by the site. The block on the top right marked by the purple is a dynamic advertisement but not depends on the main object itself. The block marked by blue shows a list of nearby properties, which is also generated dynamically but highly depends on the content of the object. Since it is highly correlated with the main object, some template-only or annotation-only approaches might be fooled and misidentify it as a part of the data area. The block marked by red is a static advertisement, identical for all detail pages.

We could distinguish the advertisement block (purple) through boilerplate removal process. Based on the position (i.e., not covering the center, on top-right corner), it is highly likely to be a boilerplate. In addition, it contains no valid annotations at all. It is worth noticing that the HTML code of the block may also reveals some information. Figure 6.3 shows the HTML code, and by those key words stating "google ads" in class name or node id, AMBER is confident that this block is an advertisement block and should not be part of the data area.

By applying the result page object extraction part, AMBER extracts lists of products in the domain, therefore the list marked by blue is extracted and marked as a secondary result page data area. A secondary result page data area on a detail page usually means a list of recommendations, either website highlights or object-related, so these blocks will not be considered as part of the main data area as well.

The bottom-right block marked by red will also be identified as irrelevant by boilerplate removal process. In addition, the dissimilarity calculation process will find out that the block on both Figure 6.2 and 6.4 has no dissimilarities at all. On most of the times, static templates have no contributions on attribute completeness of the object and will also be excluded during the extraction process.

After removing the three irrelevant blocks, the remaining block marked by green is the data area. Although there are still some static templates inside, like the "Check Broadband speed" part, but we could not find a smaller block containing all attributes (e.g., floor plan links in the menu, map on the right and all attributes in the description), therefore the block marked by green satisfies our definition of the data area.

6.5 Features

AMBER's analysis is based on a set of fairly simple and cheaply computable *features* that are computed for each DOM node. Such features can be computed globally, w.r.t. a set of detail pages, at page-level, or at node-level.

(1) *Tag-path support* **(TPS)**. A simple global feature representing the fact that different

nodes carrying the same instance annotation for an attribute A share a common tag-path along the first-child/next-sibling axes. The number of such nodes represents the support for a given tag-path w.r.t. an attribute A.

(2) *Dynamic content* **(DC)**. A Boolean global feature that is positive when, given a set of pages $\mathcal{P}$ and a tag-path tp, the nodes on each page in $\mathcal{P}$ with tag-path tp carry the same instance annotation for an attribute A but have different textual content.

(3) *Instance redundancy* **(IR)**. This page-level feature is true if the value of an attribute occurs more than once on the page. Given a page P, AMBER checks whether there exists more than one node carrying an instance annotation for an attribute, AMBER also reconciles different representations of the same value with instance annotations. As an example, consider two HTML nodes `<span>Bedrooms: 4</span>` and `<li>4 beds house</li>` where the string `4` carries an instance annotation of type bedrooms_number_instance. Despite the two nodes having different text content, they both carry an instance annotation for the attribute bedrooms, supporting the conclusion that the attribute has value `4`.

(4) *In title attribute* **(TA)**. It is frequent for detail pages to have a title summarizing some of the features of the entity on the page. Given a page $P \in \mathcal{P}$ and a node carrying an instance annotation for an attribute, AMBER checks whether the page title also carries an instance annotation of the same type and agreeing on the value. However, a mandatory condition is to have non-static content of the title w.r.t all pages in $\mathcal{P}$. This is necessary to make sure that the attribute values carried by the title are truly related to the attribute values on the detail page. Notice that this is not the case for result pages, where the title often summarizes the search criteria.

(5) *Label proximity* **(LP)**. This node-level feature captures the fact that there is a node carrying a label annotation for an attribute A in visual or structural proximity of a node carrying an instance annotation for the same A.

(6) *Context equivalence* **(CE)**. This global feature is positive when, given a set of pages $\mathcal{P}$ and a tag-path tp, on all pages $P \in \mathcal{P}$ the nodes at tp also appear in the same type of data structure (e.g., in a list or key-value map) in the same position, e.g., in the first element of a list, or in the same role, e.g., as a value in a map.

(7) *Result-page redundancy* **(RPR)**. Given a node carrying an instance annotation for an attribute A, AMBER validates the value for A discovered on the detail page using the value for the same attribute (if present) located on the result-page. Since detail pages often provide a superset of the attributes already provided by result pages, **RPR** can be considered as a page-level feature.

Based on our observations, we can rank these features according to their reliability, where $F_1 \prec F_2$ means that the features in the class F_i are more reliable than the features in F_2. In particular, {**DC**} $\prec$ {**RPR**, **TA**} $\prec$ {**TPS**} $\prec$ {**LP**, **CE**, **IR**}. Extensive evaluation on the impact of each of the features will be provided in Chapter 7.

6.6 Attribute Extraction

Each feature indicates the presence of an attribute on a given DOM node, albeit at a low reliability when considered in isolation. However, as AMBER's features are easily computed for a large number of nodes and pages, we compute all of them and aggregate the individual judgments to eliminate false positives produced by isolated features. In particular, semantic and structural annotations are very unlikely to co-occur already on a single page, and even more so, within an entire group of detail pages. AMBER's attribute extractions and later its wrapper induction rely on this fact.

Once the result pages are identified, AMBER extracts attributes (i.e., marks the nodes containing attribute values) only from data areas and not entire pages. Algorithm 8 shows the pseudo-code for attribute extraction. For each identified data area D (Line 2) and each attribute type A of relation R (Line 4), AMBER computes the tag-paths of all nodes annotated with instance annotations of a corresponding annotation type for attribute A (Line 5). AMBER then produces a *scoreboard* for each tag-path, i.e., mapping a tag-path and a feature to a Boolean value (Line 6). The bit $\mathsf{score}[t][\mathsf{feature}]$ for a tag path t and a feature feature is set to 1 if t is likely to contain the value of attribute A according to feature type feature.

In iterating over all tag-paths $t \in T$ (Line 7) and all features $\mathsf{feature} \in \mathsf{FeatureSet}$ (Line 8), AMBER completes all scoreboards $\mathsf{score}[t]$ (Lines 9). Then, the algorithm turns $\mathsf{score}[t]$ into the score value $\mathtt{score}(\mathsf{score}[t])$ and chooses as locater path $l[A]$ for the attribute type A the tag-path t with the highest score (Line 10). In case of ties, AMBER resolves them using prior information about the reliability of features, i.e., tag-paths with higher scores for more reliable features are ranked higher. The locator vector is finally added to the set of locator vectors to be used for wrapper induction (Line 11).

6.7 Attribute Extraction Example

To illustrate the feature calculation and attribute extraction, we randomly picked an example from a real-estate website `http://www.oliverjames.co.uk`. Figure 6.5 is a screen shot of

Algorithm 8: Attribute extraction.

input : RP – a result page
input : $\mathcal{D}$ – the set of data areas
input : $\mathcal{A}$ – the set of DOM annotations
output : $\mathcal{L}$ – the set of location vectors

$\mathcal{L} \leftarrow \emptyset$;
foreach dataarea $D \in \mathcal{D}$ **do**
  $l \leftarrow$ new (attribute type $\rightarrow$ tag path);
  foreach attribute type $A \in R$ **do**
    $T \leftarrow \{\texttt{tagpath}(n) | n \in D$ annotated for $A\}$;
    score $\leftarrow$ new (tag path $\times$ feature $\rightarrow \{0,1\}$);
    foreach $t \in T$ **do**
      foreach feature feature $\in$ FeatureSet **do**
        score$[t]$[feature] $=$ feature$(t, RP, D, \mathcal{A})$;
    $l[A] \leftarrow \arg\max_{t \in T}$ score(score$[t]$);
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{l\}$;
return $\mathcal{L}$

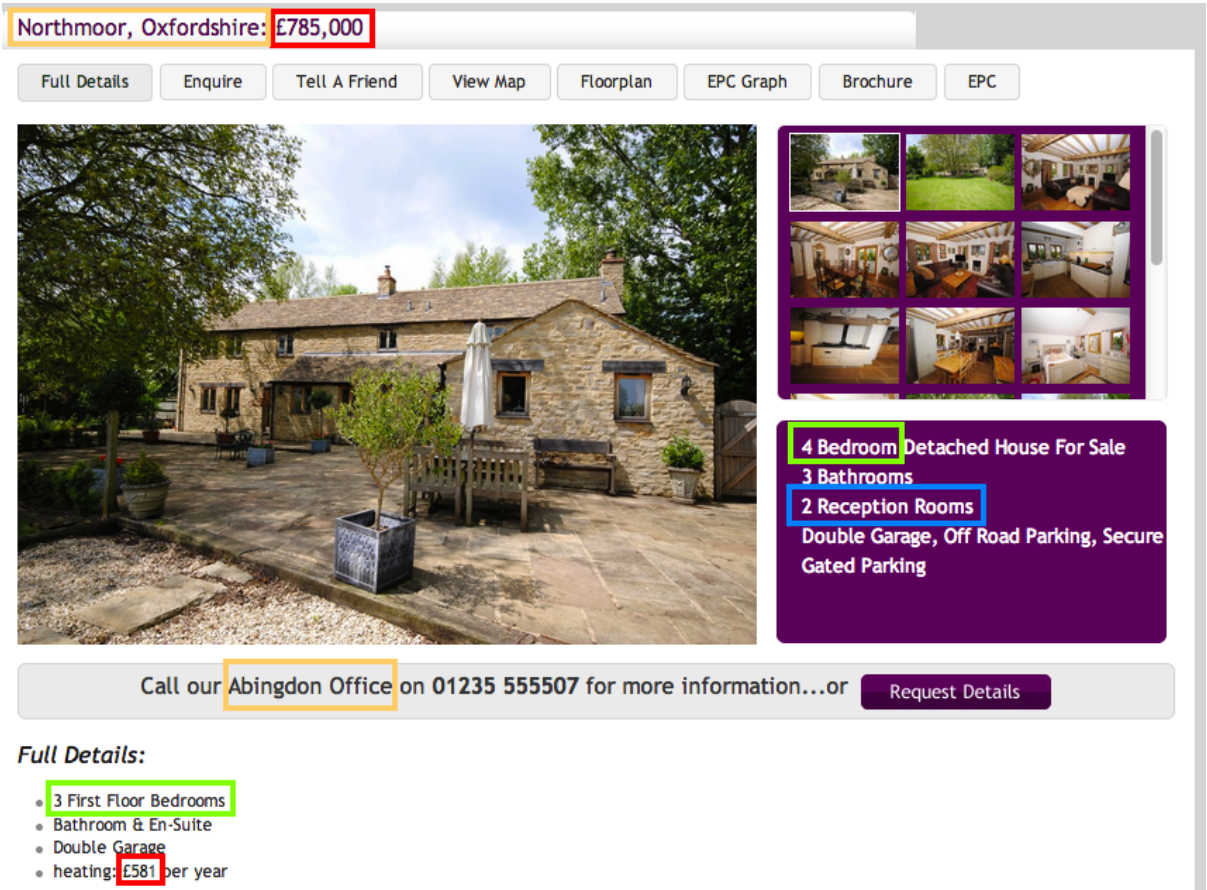


Figure 6.5: Detail page example I from Oliverjames

Figure 6.6: A record example from Oliverjames

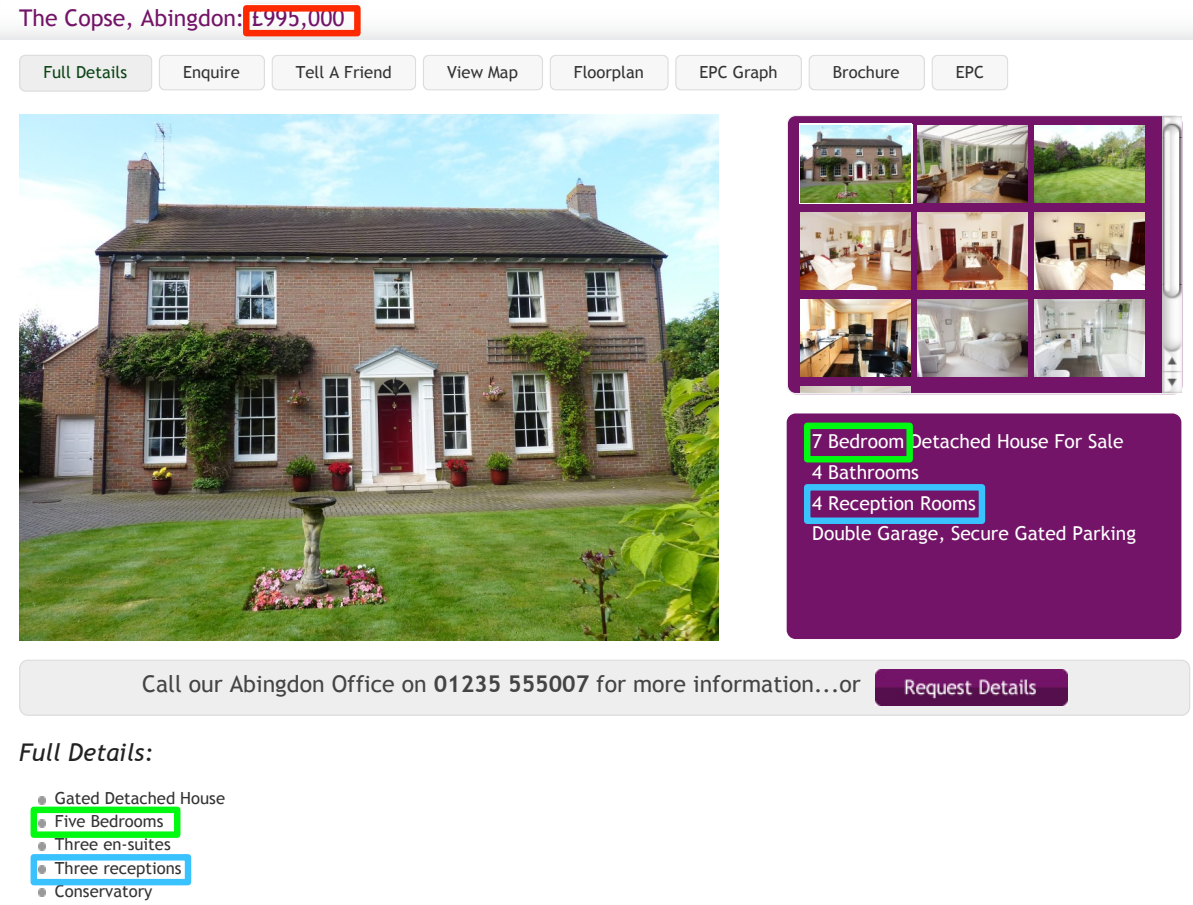


Figure 6.7: Detail page example II from Oliverjames

the data area of a detail page, Figure 6.6 is the screen shot of the corresponding record, and Figure 6.7 is the screen shot of the data area of another detail page from the same website.

In Figure 6.5, there are two price annotations, highlighted in red. The first one "£785,000" is the price of the property on the detail page (i.e., the object), and "£581" is the price of heating cost every year. To distinguish these two annotations, we calculate their features. "£785,000" has feature **TPS** (*tag-path support*) from the same tag path in Figure 6.7, **DC** (*dynamic content*) from the same tag path in Figure 6.7 but with a different price annotation content"£995,000", and it also has feature **RPR** (*result-page redundancy*). The other price annotation "£581" does not satisfy the conditions of any feature. Therefore, we pick the price annotation "£785,000" as the price attribute.

In addition, Figure 6.5 contains two number of bedrooms annotations, highlighted by green. The first one "4 bedrooms" is the number of bedrooms of the property, and the "3 First Floor Bedrooms" only describes the number of bedrooms on the first floor. Both two attributes satisfy the condition of feature **CE** (*context equivalence*), since they are both in a list, and also satisfy **TPS** and **DC**. However about **RPR**, the former annotation gets the support from the record and the latter does not. Therefore, we pick the former one, "4 bedrooms".

Moreover, the location annotations are highlighted by color yellow. The first one is the location of the object, "Northmoor, Oxfordshire", and the second one is the location of the real-estate agent, "Abingdon Office". They both satisfy the condition of feature **TPS**. However, the agent information on Figure 6.5 and Figure 6.7 are identical, which will be considered as *static content*, hence does not satisfy the condition of **DC**. Moreover, the first location also gets supports from the corresponding record. Therefore, the first location "Northmoor, Oxfordshire" is picked as the location of this property.

Last but not least, the example also shows that there are more attributes on detail pages than records from result pages. A good example is the attribute "number of reception rooms", which only occurs on detail pages.

Chapter 7

Evaluation

7.1 Result Page Evaluation

We carried out a number of experiments to evaluate AMBER's template discovery performance. In the following we discuss: **(i)** An introspective evaluation on three domains: real estate, used cars, and general search-engine results, where AMBER's output is compared against a manually curated gold-standard. The evaluation includes an ablation study on different pivot type selection, the impact of different constituent scores and threshold settings for data area and record validation as well as for attribute alignment. We also study the robustness of AMBER to noisy annotations. **(ii)** A comparative evaluation against state-of-the-art template-discovery systems for listing websites, include RoadRunner [17], MDR [58], ViNTs [85], and DEPTA+ [84].

7.1.1 Experimental Setup

Before discussing the outcomes of the experiments, first we introduce the experimental setup in terms of datasets and evaluation metrics.

Datasets The first dataset (ukre) consists of a random sample of 200 real estate websites from a corpus of 3,404 real estate websites (re.full) collected from a combination of results from UK yellow pages, real estate verticals such as RightMove and Zoopla, and Google. The dataset as a whole contains 317 pages with a total of 3,618 records and 31,972 fields across 20 attribute types (Table 4.1).

The second dataset (ukuc) consists of a random sample of 150 UK used car dealers from a corpus of 7,089 websites (uc.full) collected in a similar way as for ukre, by using corresponding used car verticals. It contains 180 pages with 2,086 records and 22,986 fields across 27 attribute types (Table 4.1).

To assure diversity in the corpora, in the case of two sites sharing the same template, we

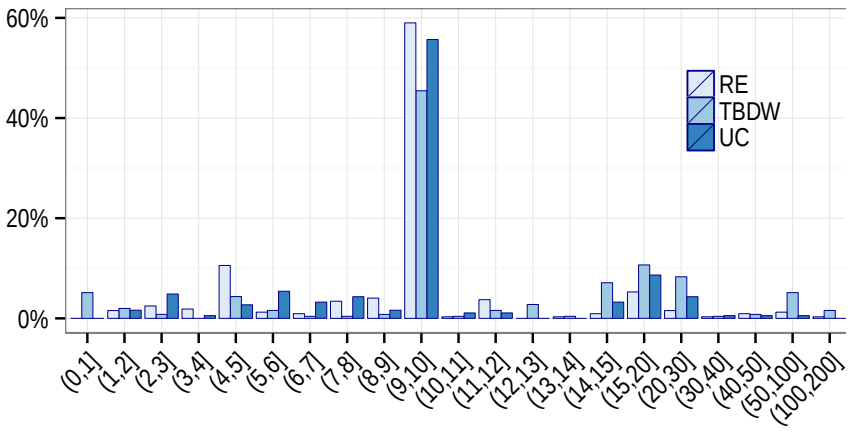


Figure 7.1: Records per page

replaced one of them with a new site picked randomly from the larger corpus. For both ukre and ukuc the ground-truthing has been done by manually annotating up to two result pages with at least two records in each page. Each record was manually annotated by three expert annotators, resulting in a *kappa* (i.e., inter-annotator agreement) of 92% for data areas and records and of 83% on attributes. Both datasets have been cleaned of websites that were unreachable or returned HTTP errors or redirects. We also removed all sub-domains of aggregators and car makers as these are all similarly structured and would have biased the evaluation.

The third dataset (tbdw)[1] is a well known dataset for record segmentation constructed by [82]. It includes 253 result pages (4,583 records) obtained by querying 51 search-engine like websites. tbdw has been previously used as a benchmark dataset by several systems, including skoga [6], viper [70], fivatech [50] and tpc [63].

The fourth dataset (ctrl) is a control dataset consisting of 300 non-result pages, e.g., home and detail pages, blog entries, ads collected from the same websites used to construct ukre and ukuc. The sampling has been done by randomly selecting 50 sites for each domain. The pages contain structured data and other regular structures that are meant to trick template discovery approaches into recognizing a data area where there is none.

Dataset characteristics AMBER is generally applicable to most product domains, where listing pages have at least some structured attributes in common, such as price or url, and the set of attributes is generally homogeneous, though their presentation may differ widely. Among product domains, we chose the domains of used car and real estate for two primary reasons:

(1) In both domains, a large number of sellers is active and products are often unique to a seller.

[1] http://daisen.cc.kyushu-u.ac.jp/TBDW/.

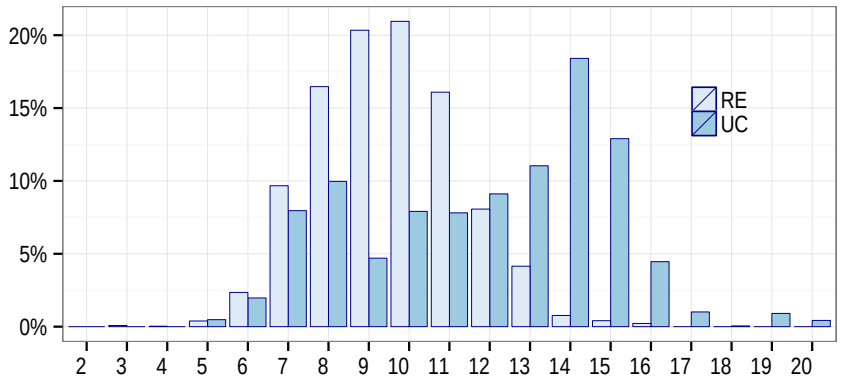


Figure 7.2: Attributes per record

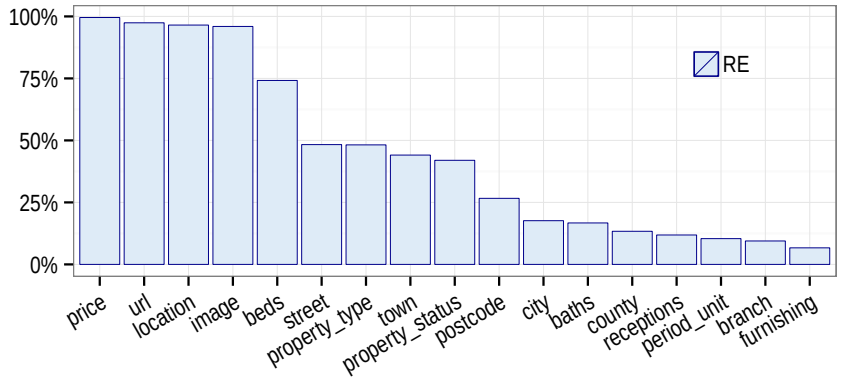


Figure 7.3: ukre attribute type frequency

(2) Furthermore, the two domains share very few attributes and differ quite notably with respect to average size and exploration of sites.

Figure 7.1 shows the distribution of records per page on ukre, ukuc and tbdw datasets. The distribution of records per page on real-estate is similar to used car domain. And in all three datasets, there are more than 40% pages containing 10 records. Figure 7.2 shows distribution of attributes per record on ukre and ukuc only, since tbdw does not have a gold standard for its attributes. On average, used car domain has 5 more attribute types per record than real estate sites, since the features of a used car is more systematic and parametrizable than a real estate property.

Figure 7.3 illustrates the distribution of attribute types in ukre and Figure 7.4 illustrates the distribution of attribute types in ukuc. In the real estate domain, we find that nearly all records contain “price”, “url”, “location” and “image”. Surprisingly, not all records clearly identify the “size” of the property, which in the UK might also be indicated by the number of bedrooms and the number of bathrooms. In the used car domain, price is the only attribute occurring in almost every single record. “make” is the second most common attribute type, some used car records listed “model” only since they were sufficiently distinguishable and “make” appear in

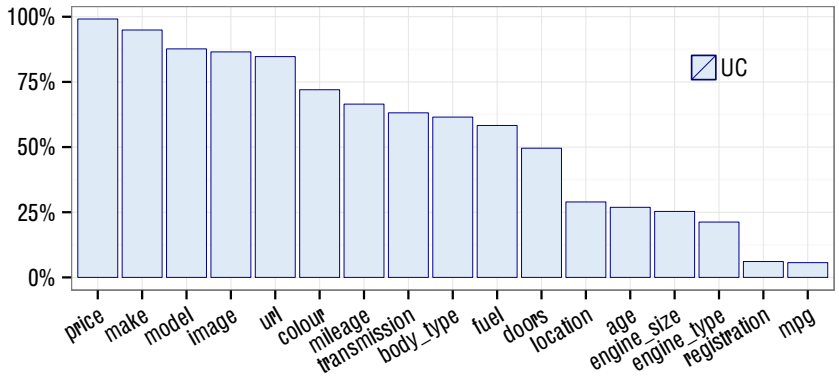


Figure 7.4: ukuc attribute type frequency

only 95% of records. In addition, "model", "image" and "url" also occurring in more than 75% of records.

Evaluation Metrics Evaluating data extraction systems is a challenging task since different works may have different understandings of what is relevant for the evaluation. In this work we evaluated the ability to recover:

(i) The data area, in terms of identifying the root node.

(ii) The correct segmentation of the records (including the delimiting nodes).

(iii) The value of each attribute within each record.

The metrics used are precision, recall, and F_1-score computed against the gold standard, and defined as follows, where S denotes one of the structures above:

$$Prec_S = \frac{|\text{correctly retrieved } S|}{|\text{retrieved } S|} \qquad Rec_S = \frac{|\text{correctly retrieved } S|}{|\text{correct } S|} \qquad F_1 = 2 \cdot \frac{Prec_S \cdot Rec_S}{Prec_S + Rec_S}$$

7.1.2 Introspective Evaluation

Overview The first experiment is concerned with an overall evaluation of AMBER on all three datasets and its outcome is shown in Figure 7.5. We evaluated data area identification (DA), record segmentation (Seg), and attribute alignment (Att). For ukre and ukuc datasets, the pivot attribute types were both PRICE while the sets of characteristic and for optional attribute types, we used those of Table 4.1. For the tbdw dataset, the pivot is the URL, which is also the only attribute provided by the gold-standard annotations. The performance of attribute identification in this case corresponded to the one for record segmentation and is therefore omitted from the chart. On average, AMBER was able to identify the data area correctly and segmented it into records correctly in 99% of the cases. Moreover, in domains having more robust pivot attribute, AMBER obtained better performance. The reason for the lower recall in record segmentation

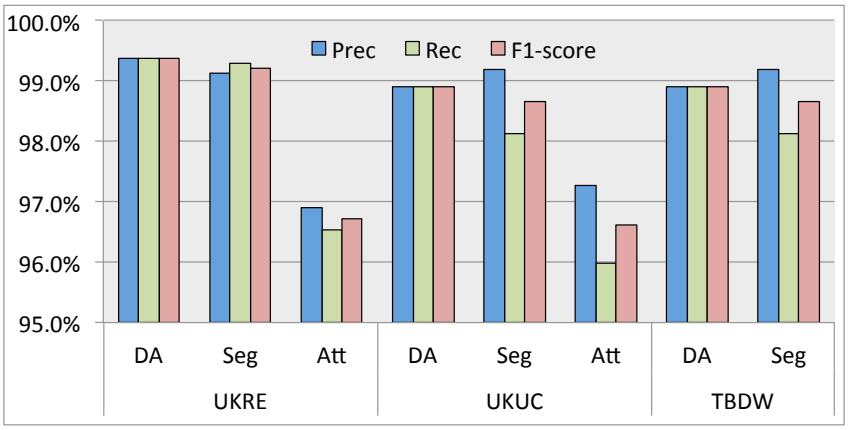
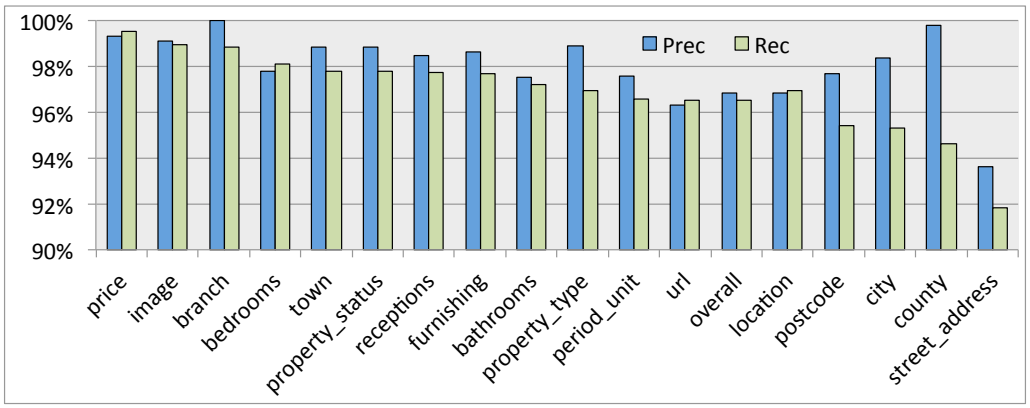


Figure 7.5: Overall performance on ukre, ukuc, and tbdw datasets.

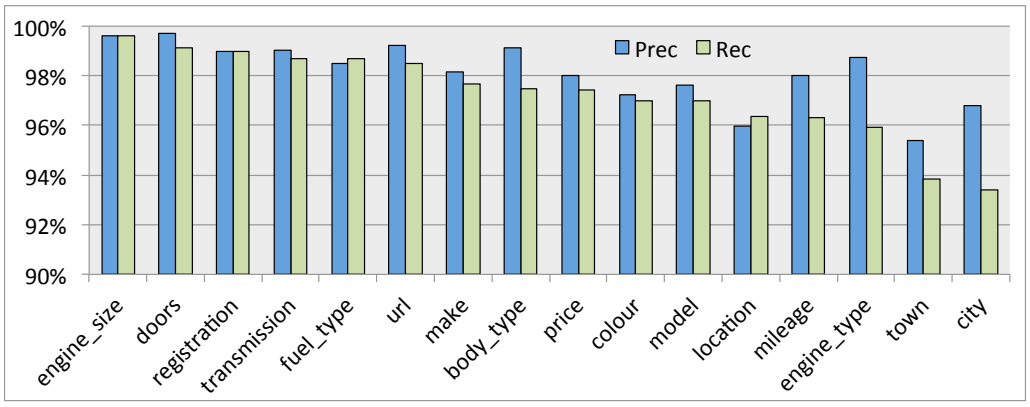
for tbdw was due to the fact that 13 pages (5% of the dataset) only contained one record, and were therefore not proper result pages according to our definition. It is worth mentioning that, by having access to at least one page from the same website with at least two records, AMBER would be able to recover the correct template that then can be applied to extract data from the single-record pages.

Regarding attribute alignment, AMBER achieves an average of $> 96\%$ F_1-score over ukre and ukuc datasets. This is an *unprecedented* result if we consider that we are targeting the extraction of attribute-rich objects with more than 15 different attribute types. The detailed performance for each attribute type in the two domains is shown in Figure 7.26. As expected, AMBER was able to identify more accurately for the characteristic attributes than the optional ones, since optionality influences features such as the redundancy resulting, e.g., in a lower effectiveness of attribute validation. It is also expected that the attributes with lower recall are related to geo-information such as STREET_ADDRESS. Location attribute is relatively harder to recognize than other product-related attribute as they are usually poorly structured and do not follow precise patterns. The extraction of location is often regarded as an independent research area.

Data Area Validation The second experiment aimed at evaluating the accuracy of data area validation. In particular, we want to evaluate how accurately can AMBER distinguish between the primary data area and the secondary ones, as well as, regular structures on pages which are not result pages, e.g., detail pages, blogs, tables, etc. The experiment has been carried out by running AMBER on the ctrl dataset and recording whether AMBER mistakenly recognized a data area. The *accuracy* in this case was computed as 1 minus the ratio of errors made. Figure 7.7 shows the results of the experiment broken down by domain and by the nature of the page. For the real estate domain, AMBER was able to correctly distinguish detail pages in 100% of the



(a) ukre



(b) ukuc

Figure 7.6: Attribute identification performance.

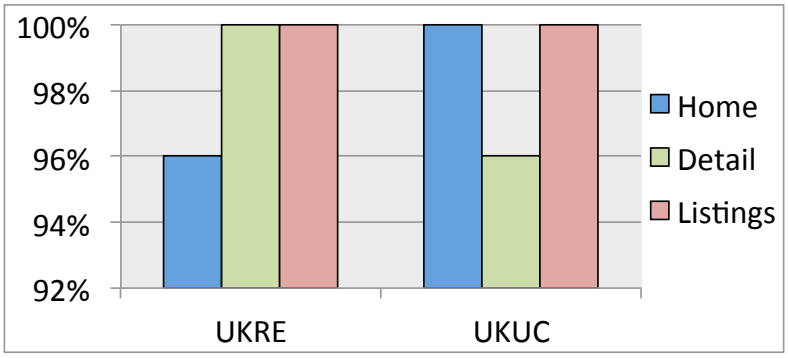


Figure 7.7: Data area validation performance on ctrl dataset.

cases but identified 2 home pages as result pages. Both of the pages included a large featured listing area (occupied more than half of the page), illustrating featured properties. AMBER failed to annotate them as featured listings due to their close similarities with result pages. In the used car domain, AMBER correctly distinguished all home pages from result pages but a data area on two detail pages. The reason in this case was a regular structure representing prices of car parts with corresponding images. The root of the problem was the co-occurrence of attributes.

Pivot Type Selection The third experiment devoted to understanding how do different pivot types affect the performance. We carried out the experiment by running AMBER on both ukre and ukuc datasets, but using different attribute types as pivot type. Firstly, for each attribute type, we calculated its frequency of occurrence in records, then picked several attribute types occurring most frequently in records as pivot nodes, i.e., "price", "location", "url" in ukre, and "price", "url", "make", "model" in ukuc, then run AMBER with those different pivot type settings on both ukre and ukuc dataset and report precision and recalls of both data areas and records. Figure 7.8 shows the results of the experiment by domain. In both domains, price occurs most frequently, more than 99%, and AMBER achieved the highest precision and recall with price as pivot attribute on both domains. For real estate domain, location occured in 97.43% records and url occured in 96.52%. For used car domain, "make" occured in 94.92% records and "model" occured in 87.68% records. AMBER reached high precision of data areas with all pivot types, but the recall of data areas was positively correlated with the occurrence of the attribute. The results on record extraction showed that noisy attributes such as location, might cause some misidentification and lower the precision of records. Similar to the recall of data areas, the recall of records also had positive correlations with the occurrence of pivot attributes. For those attribute types might miss in several records only but not the entire data area, AMBER was still able to extract those records without pivot nodes through the inference

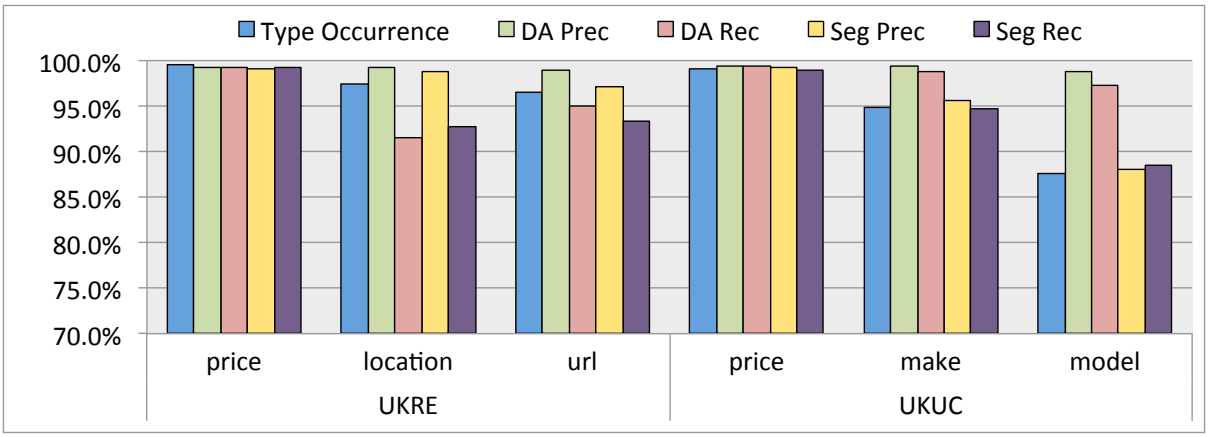


Figure 7.8: Performance of different pivot type selection.

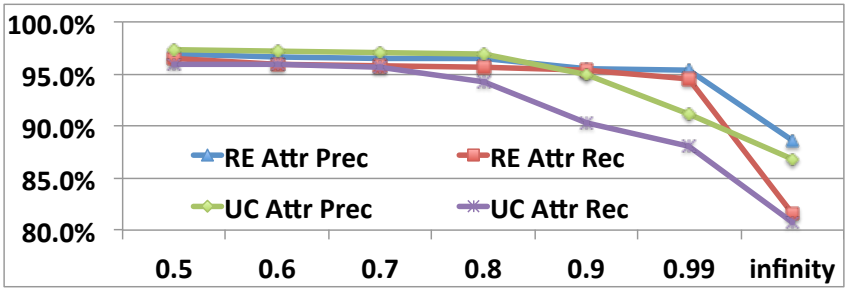


Figure 7.9: Impact of $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$

process and therefore the recall of records is a little bit higher than the occurrence of pivot node. In general, the evaluation result proves that AMBER's performance is stable with different pivot attribute types, however the noisiness of pivot attribute might still slightly affect the precision of records, and the recall of data areas and records were positively correlated to the occurrence of pivot attributes. Therefore, to reach the optimal results, we usually select a relatively frequent occurring and less noisy attribute type of the domain as the pivot attribute type.

Thresholds Analysis This experiment aimed at evaluating the impact of the two thresholds AMBER adopted in the attribute alignment process. Figure 7.9 shows the impact of inference threshold $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$ used in the attribute alignment process on both ukre and ukuc. If the rate of annotation support from other records were above the threshold, then triggered attribute inference and alignment process. $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$ lower than 0.5 means more than one potential attribute might occur at the same time and we needed to invent mechanisms for tie breaking, therefore we limited the lower bound of $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$ to 0.5. Higher thresholds means higher rate of annotating support from other records which is more difficult to reach. The result showed when the threshold increased from 0.5 to 0.7, the attribute precision and recall on both domains remained almost the same. When the threshold was above 0.7, the attribute precision

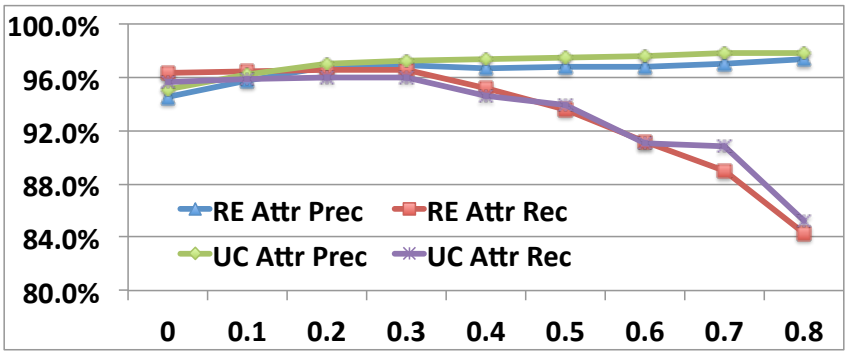


Figure 7.10: Impact of $\min_{\text{INV-SUPPORT}}^{\text{ATTRIBUTE}}$

and recall dropped when the threshold drops. Therefore, we picked 0.5 as the optimal value for $\min_{\text{SUPPORT}}^{\text{ATTRIBUTE}}$ and applied it in other evaluations.

Figure 7.10 shows the impact of the sufficient annotation support threshold $\min_{\text{INV-SUPPORT}}^{\text{ATTRIBUTE}}$ used in the attribute alignment on both ukre and ukuc. If the rate of annotation support from other records was below $\min_{\text{INV-SUPPORT}}^{\text{ATTRIBUTE}}$, the attribute would be removed. The results showed that if the threshold exceeded a certain value (0.3), attribute recall dropped when the threshold increased, but if the threshold was in the range of [0, 0.3], the attribute recall remained almost the same. In contrast, the attribute precision raised slightly when the threshold increased. By combining both attribute precision and recall, we picked 0.3 as the optimal value for $\min_{\text{INV-SUPPORT}}^{\text{ATTRIBUTE}}$ and applied it in other evaluations.

Impact of Constituent Scores This experiment aimed at evaluating the impact of each constituent score in the attribute selection process. The evaluation was carried on ukre and ukuc data set, we disabled each constituent score individually and report the precision and recall differentiation of attributes extraction. Figure 7.11 shows the result of the experiment. Each constituent score contributed to the selection process from 2% to 6% on both attribute precision and recall. In real estate domain, attributes were more likely to occur more than once in a record and noises were usually occurring in long text nodes such as the description, therefore node span and instance redundancy performed better than in used car domain, disabling each of them made AMBER lose 5% attributes precision and recall. On the other hand, the records in used car domain were more likely to have a list of attributes in the format of labels followed by instances, therefore the label proximity constituent score was more effective than in the real estate domain, disabling it made us loss 6% precision and recall. Other two constituent scores worked almost equally well in both domains. We also tried to increase the weight of each constituent score to find out if one or more of them dominants the selection process. When we increased the weight of each constituent score individually, the differentiation of precision

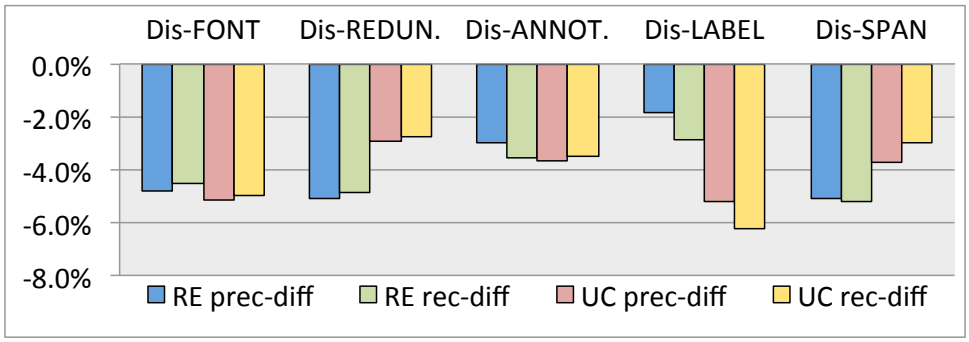


Figure 7.11: Impact of Constituent Scores.

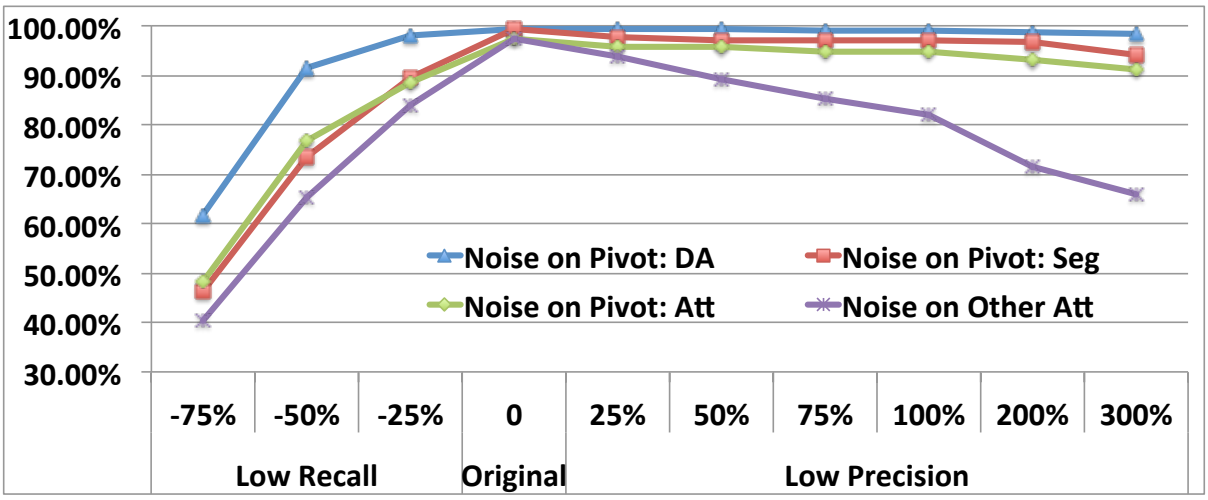


Figure 7.12: Robustness evaluation on ukre

and recall on both ukre and ukuc was less than 1%. In general, each constituent score contributed to the selection process, and our evaluation proves that assigning the same weight to each constituent score is simple but effective.

Robustness This experiment was devoted to understanding the effect of noisy annotations on AMBER's data extraction approach. We configured the SNER to:

(i) introduce dirty annotations for both instances and labels of any given attribute type, by randomly distributing them on text nodes in the DOM, thus simulating a loss in precision in the SNER . The amount is referred to as a percentage on the original annotations.

(ii) randomly remove annotation instances for any given attribute type, thus simulating a loss in recall in the SNER . Again, the number of removed annotations is expressed as a percentage of the original ones.

While stimulating a low recall process, we kept the labels of attribute types since in reality it was very easy to establish a relatively complete label gazetteer, but establishing an instance gazetteer for annotations requires much more work.

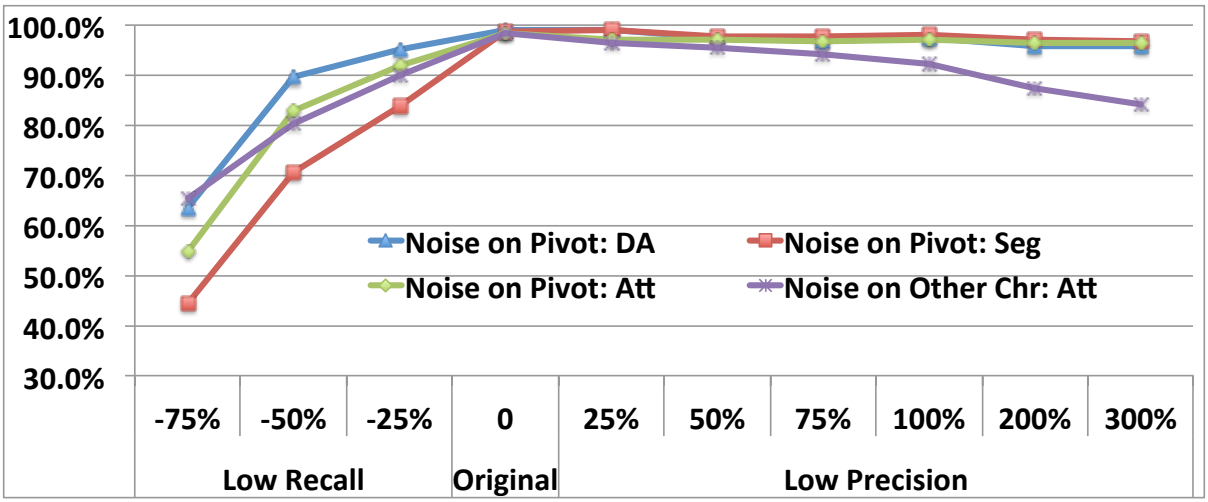


Figure 7.13: Robustness evaluation on ukuc

To better understand the effect of noisy annotations, the experiment targeted the pivot and other attributes separately. Biasing annotations corresponding to the pivot attribute would likely affect data area identification and record segmentation, while a bias in other attributes would mostly affect attribute identification.

The charts of Figure 7.12 and Figure 7.13 show how the amount of noise affects AMBER in terms of F_1 score in each domain respectively. The numbers on the left hand side shows the effect of noisy annotations for the pivot attribute types while no noise was added to the characteristic attributes. Dually, on the right-hand-side we show the effect of adding noise to the annotations for the characteristic attribute types while keeping the pivot annotations unchanged.

As expected, even a substantial loss (300%) in precision in the annotations corresponding to the pivot attribute affects the performance on data area identification only by $\sim$1% and by$\sim$ 5% on records. The reason was that the correct annotations still appeared on the nodes representing the regular pivots inside the data area and since the noise is not systematic, the chance of inducing regular but incorrect structures is negligible. On the other hand, a loss in recall for pivot annotations did affect the performance of both data area identification and record segmentation. AMBER was able to tolerate up to 25% loss in recall for the pivot without substantial decrease in performance, but if we only provided 25% accuracy (75% recall lost) of the original pivot , AMBER losed 50% of its original accuracy instead of losing all 75%. The data shows that the inference ability of AMBER boosted the performance even with very low recall of pivot nodes.

As all other attributes were concerned, without noise in the pivot attribute, even a substantial bias (both in precision and recall) in the annotations only marginally affected ($\sim$ 1%) data area

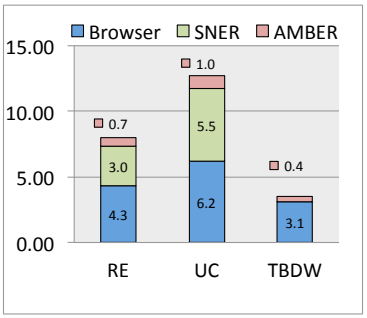


Figure 7.14: Average running time per page

identification and record segmentation. For attribute alignment, a loss in precision was less harmful than a loss in recall. On real estate domain, the effect of the low recall for other attributes is slightly dramatic than low recall for pivot attribute types. The significant drop occured when we lower the recall from 50% to 75%, where the annotations are too sparse and could not provide global support for inferring missed attributes. In used car domain, since a large number of attributes have a label, even when we lowered the recall of annotations into 75%, AMBER still reached a 65% F_1 score of attributes with the inference ability.

Running Time Statistic AMBER is running on Ubuntu 14.04 with 2 cores of 2.5 GHz Intel Core i5 and 8GB memory. Browser interaction is achieved via Selenium WebDriver 2.35 and Firefox 28. We report the running time of browser, SNER and AMBER separately on ukre, ukuc and tbdw datasets in Figure 7.14. tbdw is domain independent and extracts one attribute (url) only so SNER is not applied. The average running time per page is: 8.0 seconds on ukre, 12,7 seconds on ukuc and 3.5 seconds on tbdw. In general, browser interaction including page retrieval and rendering is the most expensive, the SNER is the second, and the result page analysis part of AMBER takes less than one fifth of the browser time.

7.1.3 Comparative Evaluation

We compare AMBER against state-of-the-art data extraction systems which support extraction from result-pages, namely: RoadRunner [17], MDR [58], ViNTs [85] [2], and DEPTA+ [84]. Due to the differences in the output of each of the systems, we also describe, for each experiment, the pre- and post-processing steps undertaken to obtain a comparable output.

RoadRunner The first experiment was carried out against RoadRunner, an unsupervised web data extraction system that, given a set of input pages (examples), distinguishes invariants on

[2]The DEMO of ViNTs is available at http://www.data.binghamton.edu/vints.

the page (i.e., the template) from variables (i.e., the data). RoadRunner does not extract data areas and records explicitly but produces an HTML output describing, as a relation, the variable parts of the page. In order to have a fair comparison, we only compared against the extracted data, and ignored data area identification and segmentation. This is also the reason why we did not run experiments on tbdw as they would not be significant. Being unsupervised, RoadRunner tends to extract too many of the structures on the page as it is not able to distinguish good variables from noise such as navigation menus, urls and links. For example, RoadRunner reports on some pages more than three times the attributes present in the gold-standard. To avoid biasing the evaluation against RoadRunner, we filtered its output by **(i)** removing the description blocks, as RoadRunner often splits them in several attributes when descriptions are structured with multiple nodes, and **(ii)** duplicate URLs, as RoadRunner recognizes, e.g., "breadcrumbs" and other navigation structures. On the other hand, to avoid biasing the evaluation against AMBER, we removed all attributes from the output of RoadRunner which are not contained in the gold standard, such as page or telephone numbers.

Another difference between AMBER and RoadRunner is in the granularity of the output. RoadRunner only extracts entire text nodes while AMBER locates finer-granular attributes within a text node by leveraging the information provided by the SNER . For example, RoadRunner often extracts entire key-value pairs, such as "Price £114,995", as attributes whereas AMBER produces only "£114,995". For a fair comparison, we evaluated RoadRunner in two ways. A first method counts an attribute as correctly extracted if the gold standard value is *contained* in one of the attributes extracted by RoadRunner, i.e., RR (cont.) in Figure 7.15. The second one, counts an attribute as correct only if the extracted value is an exact match in the gold-standard, i.e., RR (exact) in Figure 7.15. Finally, as RoadRunner produces a better output when provided with more than one input page, we excluded sites with a single result page from this comparison. In summary, AMBER outperforms RoadRunner by a wide margin, which reaches only 49% in precision and 66% in recall compared to almost perfect scores for AMBER. As expected, RoadRunner expresses a higher recall than precision.

MDR The second experiment was carried out against MDR, an unsupervised system for segmentation of web listings. MDR is purely a segmentation system and it does not produce attributes as output. We therefore only considered the tasks of data area identification and record segmentation. Being also an unsupervised system, MDR is fooled by regular but unrelated content on the page, as a consequence, we corrected its output by ignoring those parts that could be easily recognized as regular garbage by means of simple post-processing heuristics, such as menus, footers, pagination links. Figure 7.16 summarizes the results. In all cases, AMBER out-

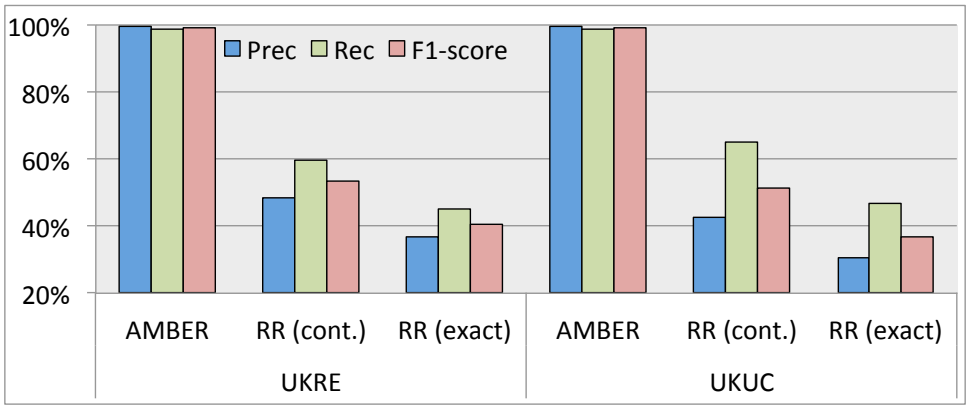


Figure 7.15: Comparison with RoadRunner.

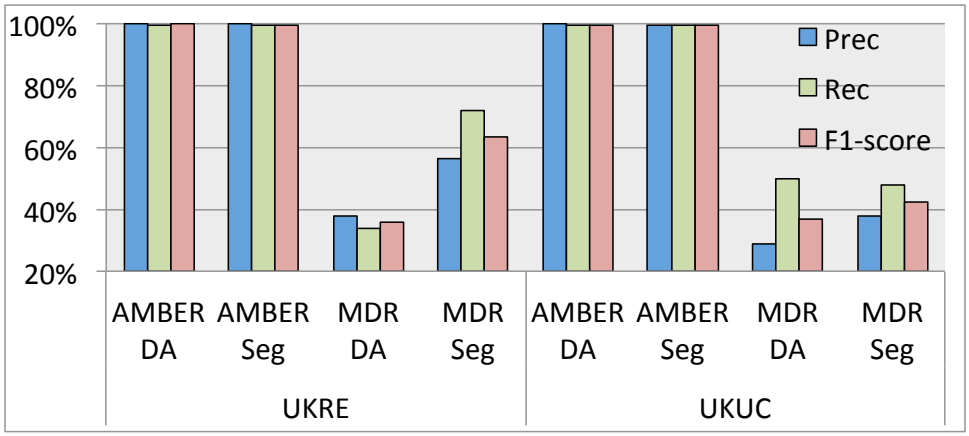


Figure 7.16: Comparison with MDR.

performed MDR which reports 57% in precision and 72% in recall, on the used car domain, as its best performance. MDR is particularly affected by optional nested structures in data records, which are often recognized by MDR as independent records, thus over segmenting the data area.

ViNTs The third experiment targets ViNTs, a tool which automatically produces wrappers for search engines and extracts search result records from dynamically generated result pages based on both visual and structural information. We used the online ViNTs demo for our tests. For each web site, ViNTs requires 1 to 5 result pages plus an optional control page with no records in it. As MDR, ViNTs also does not report individual attributes but only data areas with the corresponding segmentations. We therefore evaluated its performance accordingly. Moreover, ViNTs reports always a single data area for each result page. For this reason, we compared its output only against the corresponding data area in our gold standard. As ViNTs's paper claims that the number of result pages provided as input correlates positively with the accuracy of the results, we provided, whenever possible, 5 pages as input plus the corresponding control page. We also removed 22 websites from ukre and 4 from ukuc for which ViNTs reported rendering

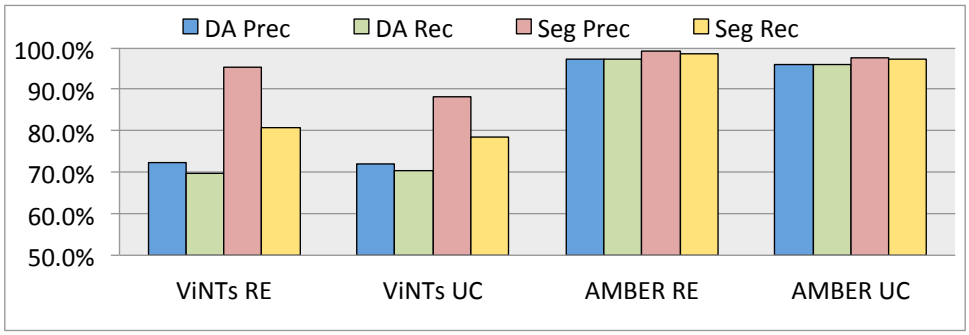


Figure 7.17: Real-estate domain Evaluation

problems or exceptions where thrown.

Figure 7.17 summarizes the results. On data area identification, AMBER outperformed ViNTs on all three corpora. The difference was more noticeable on ukre and ukuc, where AMBER outperformed ViNTs by between 15% to 25% in accuracy. On tbdw (shown in Figure 7.19) the improvement was lower (6%) but still significant. In 13% of the cases, ViNTs segmentation was only partially correct, e.g., ViNTs located the correct data area and some of the records, introducing either false positives (usually in the head or tail of the data area) or missing some records (typically the featured listings). In another 13% of the cases, ViNTs either reported the wrong data area or the segmentation was incorrect beyond repair. ukuc was similar to ukre with a slight increase in partially incorrect segmentations. This is due to the higher number of featured listings and interspersed ads in the used car domain. ViNTs performed much better in its natural environment, namely on simpler search-engine like result pages which are much less structured than those found in the used car and real estate domains. Here ViNTs reached an accuracy of 91%.

As far as record segmentation is concerned, ViNTs delivered high precision, but relatively low recall. On ukre, ViNTs reaches 95% in precision and 80% in recall whereas AMBER achieves 99% and 98% respectively. On ukuc, ViNTs reports 88% in precision and 78% in recall as opposed to AMBER with 97% in both precision and recall. The reason why the poor performance on data area identification was not reflected into the segmentation is that ViNTs often introduces spurious records in the data area, e.g., navigation menus, without affecting the segmentation of the other records.

DEPTA+ The fourth experiments compared AMBER with DEPTA+, an improvement of MDR, which automatically identifies data records in a page then aligns and extracts data items from records using partial tree alignment technique. In our evaluation, we used the most recent implementation of DEPTA+[3].

[3]http://seagatesoft.blogspot.co.uk/2012/05/structured-data-extractor.html

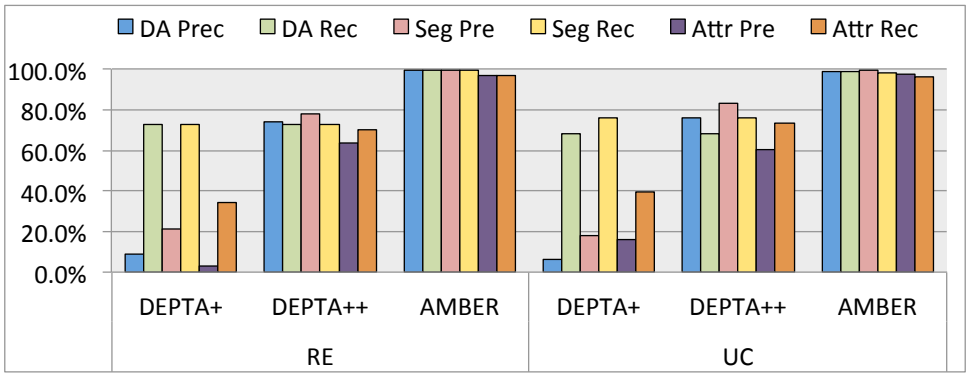


Figure 7.18: Real-estate domain Evaluation

DEPTA+ has two parameters which strongly affect the performance of record identification, the threshold of the normalized tree distance between generalized nodes and the maximum generalized node length. We set different thresholds of DEPTA+ and run it repeatedly, finding one combination that produces the highest accuracy, namely: normalized tree distance 0.20 and maximum generalized node length 12.

DEPTA+ extracted all list like structures (e.g., page navigation block, menu, drop down menu of forms) as data areas and segments them, then extracts attributes from each record. In ukre and ukuc, the average data area numbers per page DEPTA+ reported are: 7 in ukre and 9 in ukuc. The average record number per data area were 6 in ukre and 5 in ukuc.

To optimize the performance of domain independent tool DEPTA+, we used our annotator as a plug-in component to clean noise data areas DEPTA+ produces. For each result page, only the data area having the largest amount of annotations was kept, all others are dropped. With the help of our annotations, DEPTA+ reported only one data area - the most domain relevant one, which is comparable with our gold standard. On attribute extraction level, similar to RoadRunner, DEPTA+ also extracted entire text nodes while AMBER located finer-granular attributes within a text node. Here we applied the same cleaning method on DEPTA+ and RoadRunner, which counted an attribute as correctly extracted if the gold standard value was *contained* in one of the attributes extracted by DEPTA+. In the following evaluation, DEPTA+ optimized with our annotator plug-in and attributes cleaning method are referred as DEPTA++.

Figure 7.18 showed the results of DEPTA+, DEPTA++ and AMBER. With our cleaning, DEPTA++ reaches more than 70% data area precision on both domains, 78% record precision on ukre and 82% record precision on ukuc, while DEPTA+ only got less than 10% data area precision and roughly 20% record precision on both domains, and both DEPTA++ and DEPTA+ report the same data area recall and record recall. Higher precision and same recall indicated that plugging in our annotator for cleaning is reliable and sufficient. With the attribute *contained* cleaning, the attribute recall reported by DEPTA++ reaches 75% on both domains, while the orig-

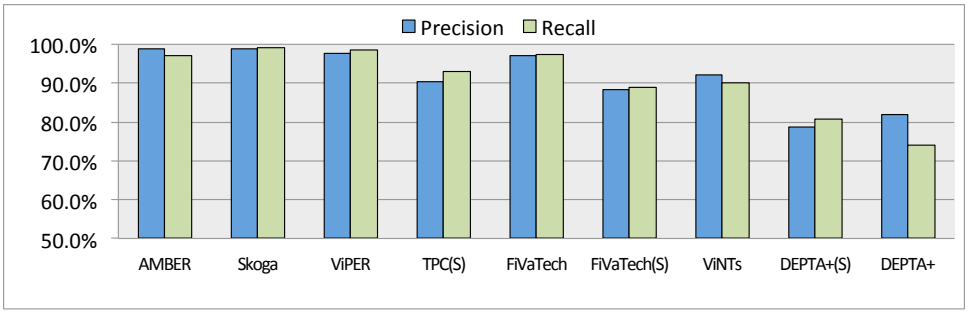


Figure 7.19: Real-estate domain Evaluation

inal attribute recall reported by DEPTA+ are only 40%. Finally, although DEPTA++ performed much better than DEPTA+, AMBER still outperformed DEPTA++ by a wide margin.

Other tools There are a number of other systems in the literature but for which we were not able to obtain a working prototype for testing. As a consequence, we considered only those who reported results for the tbdw dataset, thus enabling at least a distant comparison using the performance numbers published in the available literature.

To test AMBER's performance on this dataset, we selected the URL as pivot attribute type. This is located by considering every HTML <a> with a valid `href` attribute as a pivot node.

We compared AMBER against Skoga [6], TPC [63], FiVaTech [50], and ViPER [70]. In addition, we also run DEPTA+ and ViNTs on tbdw for comparison.

The result of this comparison is summarized in Figure 7.19. The performance results for Skoga, TPC, FiVaTech, and ViPER are taken directly from the corresponding papers. For completeness, we also report the numbers for FiVaTech and DEPTA+ on tbdw as reported by [6] and denoted by curly brackets(S).

The difference in numbers for DEPTA+ between our run and the one of [6] is easily explained by the different settings adopted. We set normalized tree distance to 0.20. The settings used in [6] were 0.36 for the normalized tree distance, reaching 3% lower precision but 6% higher recall compared to our results.

It is worth mentioning that in [6], 2 of the 51 websites were excluded because of ambiguous record annotations and in [63] only 43 websites were used. Also, in ViPER's evaluation, only one page per website was considered, thus reporting only 693 records out of 4,583. In the evaluation of DEPTA+, The excluded 2 out of 51 websites may cause the recall difference, and different threshold setting and cleaning process may reflect on the precision difference.

From Figure 7.19, one can see that AMBER still outperforms TPC, FiVaTech, ViNTs and DEPTA+ on both precision and recall, It outperforms ViPER on precision only, reporting a slightly lower precision (0.15%) and recall (2%) compared to Skoga.

7.1.4 AMBER Self-boosting Ability Evaluation

In Section 4.7, we learn additional gazetteer entries from AMBER's result-page analysis to enrich a small seed gazetteer. In particular, it shows how AMBER bootstraps a realistic gazetteer from seed gazetteer in a few initial learning rounds and then continuously improves this gazetteer when it analyses more result pages.

The evaluation of AMBER's learning capabilities is done with respect to the upfront learning mode discussed in Section 4.7. In particular, we want to evaluate AMBER's ability of constructing an accurate and complete gazetteer for an attribute type from an incomplete and noisy *seed* gazetteer. We show that at each learning iteration the accuracy of the gazetteer is significantly improved, and that the learning process converges to a stable gazetteer after few iterations, even in the case of attribute types with large and/or irregular value distributions in their domains.

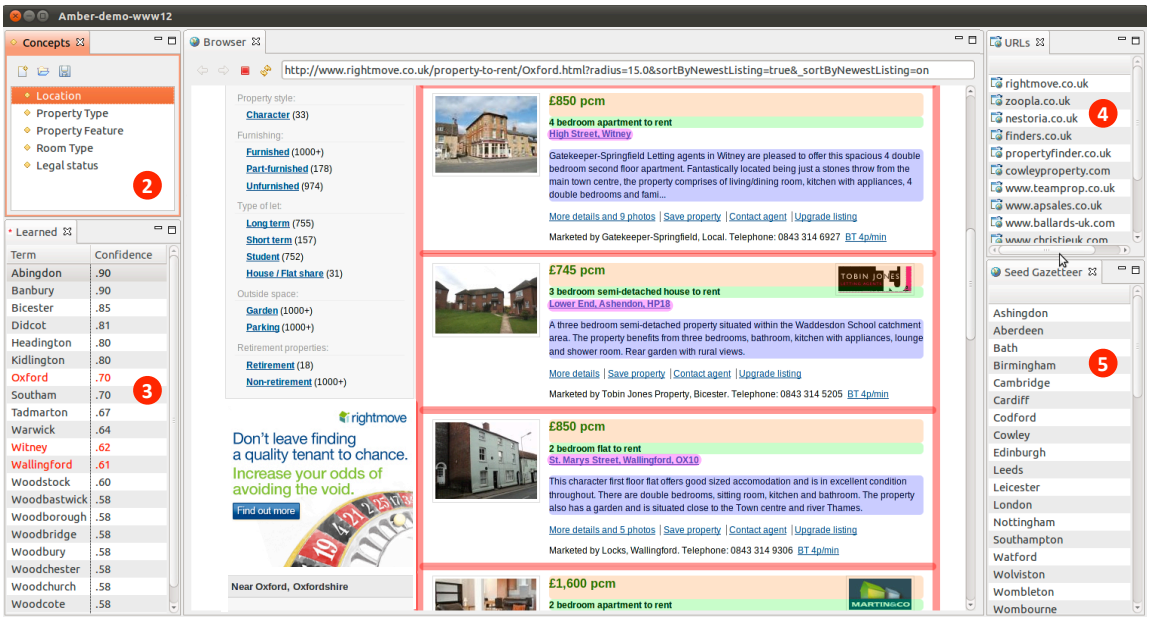


Figure 7.20: AMBER Interface

Demonstration We have created a demonstration for AMBER's gazetteer self-boosting ability [37]. Starting by considering the gazetteer for the attribute LOCATION in the context of the UK real estate domain, for which a reference gazetteer of 14,484 locations is available. Locations are terms referring to entities such as towns, counties and regions, e.g., *Oxford*, *Hampshire* and *Midlands*. The initial gazetteer consists of a random sample of 3621 location terms corresponding to 25% of the reference gazetteer.

The demonstration proceeds as follows: we pick a random website from a list of 150 UK real estate agencies, execute a "broad" search such as *flats to rent in the UK* on this site, and

run AMBER on the set of result pages. We then visualize the effect of AMBER's segmentation algorithm for individual pages; in particular, as shown in Figure 7.20, AMBER highlights the identified records and attributes (1). The panel on the left-hand-side of the GUI shows: (2) the concepts of the domain schema, e.g., LOCATION and PROPERTY-TYPE, and (3) the discovered terms with the corresponding confidence value (highlighting in red those terms occurring on the current page). The panel on the right-hand side contains (4) the list of URLs that are used for the analysis and the terms that have been identified on the current page, and (5) the current content of the gazetteer.

Setting. In the evaluation that follows we show AMBER's learning behaviour on the LOCATION attribute type. In our setting, the term location refers to formal geographical locations such as towns, counties and regions, e.g., "Oxford", "Hampshire", and "Midlands". Also, it is often the case that the value for an attribute type consists of multiple and somehow structured terms, e.g., "The Old Barn, St. Thomas Street - Oxford". The choice of LOCATION as target for the evaluation is justified by the fact that this attribute type has typically a very large domain consisting of ambiguous and severely irregular terms. Even in the case of UK locations alone, nearly all terms from the English vocabulary either directly correspond to a location name (e.g., "Van" is a location in Wales) or they are part of it (e.g., "Barnwood", in Gloucestershire). The ground truth for the experiment consists of a clean gazetteer of 2010 UK locations and 1,560 different terms collected from a sample of 235 web pages sourced from 150 different UK real-estate websites.

As an example, consider the webpage of Figure 7.20 taken from `rightmove.co.uk`. The page shows properties in a radius of 15 miles around Oxford. AMBER uses the content of the seed gazetteer to identify the position of the known terms such as locations. In particular, AMBER identifies three potential new locations, videlicet. "Oxford", "Witney" and "Wallingford" with confidence of 0.70, 0.62, and 0.61 respectively. Since the acceptance threshold for new items is 50%, all the three locations are added to the gazetteer.

We repeat the process for several websites and show how AMBER identifies new locations with increasing confidence as the number of analyzed websites grows. We then leave AMBER to run over 250 result pages from 150 sites of the UK real estate domain, in a configuration for fully automated learning, i.e., $l = u = 50\%$, and we visualize the results on sample pages.

Execution. We executed the experiment on two different seed gazetteers G_{20} (resp. G_{25}) consisting of a random sample of 402 (resp. 502) UK locations corresponding to the 20% (resp. 25%) of the ground truth.

Table 7.1: Learning performance on G_{20}.

rnd.	L^E	C^E	P^E	R^E	L^L	C^L
1	1009	763	75.65%	37.96%	169	147
2	1300	1063	81.77%	52.89%	222	196
3	1526	1396	91.48%	69.45%	224	205
4	1845	1773	96.10%	88.21%	59	52
5	1862	1794	96.35%	89.25%	23	19
6	1862	1794	96.35%	89.25%	0	0

Table 7.2: Learning performance on G_{25}.

rnd.	L^E	C^E	P^E	R^E	L^L	C^L
1	1216	983	80.84%	48.91%	289	248
2	1538	1334	86.74%	66.37%	225	204
3	1717	1617	94.18%	80.45%	57	55
4	1960	1842	93.98%	91.64%	44	35
5	1960	1842	93.98%	91.64%	0	0

By taking as input G_{20}, the learning process saturated (i.e., no new terms are learned or dropped) after six iterations with a 92.66% accuracy (F_1-score), while with G_{25}, only 5 iterations were needed for an accuracy of 92.79%. Note that at the first iteration the accuracy was 50.54% for G_{20} and 60.94% for G_{25} Table 7.1 and Table 7.2 show the behavior for each learning round. We report the number of locations extracted (L^E), i.e., the number of attribute nodes carrying an annotation of type LOCATION; among these, C^E locations have been correctly extracted, leading to a precision (resp. recall) of the extraction of P^E (resp. R^E). The last two columns show the number of learned instances (L^L), i.e., those added to the gazetteer and, among these, the correct ones (C^L).

It is easy to see that the increase in accuracy is stable in all the learning rounds and that the process quickly converges to a stable gazetteer.

7.2 Diadem Evaluation

This section discusses the large scale evaluation we have conducted on DIADEM, and reports the AMBER related evaluation results. We evaluate DIADEM on 10602 websites from the UK and US real estate (3404 and 109 sites) and the UK used car domains (7089 sites). For each of these websites, if DIADEM can locate a result page, AMBER is employed to detect the template structure of those pages and provide such information to the wrapper induction component. Section 7.2.1 describes how we setup the evaluation, including the infrastructure of the eval-

uation and the characteristics of the datasets used in our evaluation. Section 7.2.2 shows the effectiveness of the induced wrappers which was fed by AMBER and the quality of the extracted attributes. And section 7.2.3 presents the performance of the DIADEM system and individual component.

The primary finding is that DIADEM produces effective wrappers for $> 90\%$ (Table 7.3a) of the 10602 sites with a 97% average quality for attributes. Further findings include:

(1) Given only a site's URL as an input, in more than 95% of the cases, DIADEM produces a wrapper within 10 minutes.

(2) DIADEM as a whole as well as individual components outperform existing solutions, where available for comparison.

(3) The execution of the induced wrappers produced more than $900,000$ records.

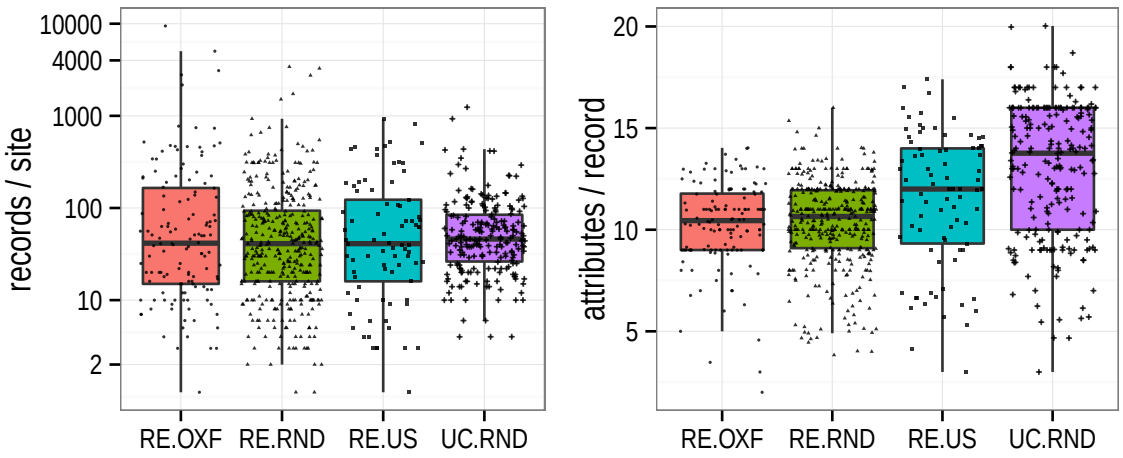
(4) To perform this task, DIADEM's transducers produce in total over 8.4 billion facts, download over 212 GB of web site data, and document the extraction with over $101,000$ screenshots. The entire evaluation was completed in 2.3 days on a 45 node cluster.

On http://diadem.cs.ox.ac.uk/evaluation/14/02, we provide the full datasets with links to the automatically generated DIADEM reports including fully-annotated screen shots, all induced wrappers, and statistics about the extracted data.

7.2.1 Evaluation Setup

Infrastructure For the evaluation, DIADEM is deployed on a Hadoop cluster of 45 m2.xlarge Amazon EC2 instances. Each instance is running Ubuntu 12.04 with 17 GB main memory and 2 cores of 6.5 elastic compute units, where each unit is equivalent to a 1.0-1.2 GHz 2007 Opteron or Xeon processor. We only employ one of the cores and limit memory use to 10GB, which suffices even for the largest websites. The current version of DIADEM uses DLV [56] as Datalog$^\neg$ engine for most of the transducers, except some parts that communicate with the browser or external tools, that are implemented in Java. Browser interaction is achieved via Selenium WebDriver 2.35 and Firefox 20.

Datasets We apply DIADEM to 5 datasets, three for real estate, one for used car, and one benchmark dataset for comparing form understanding. URLs of the sites in the full datasets (RE-FULL and UC-FULL) are extracted from a combination of yell.com and listings of real-estate or used car traders on aggregators, e.g., http://www.rightmove.co.uk for real-estate domain and http://www.autotrader.co.uk for used car. Both have been cleaned of websites that were unreachable or returned HTTP errors or redirects. We also removed all sub-domains of aggregators and car makers as these are all similarly structured and would have biased the



(a) Records per site

(b) Attributes per record

Figure 7.21: Dataset characteristics

evaluation. We do so even though DIADEM is able to generate effective wrappers for all major real-estate aggregators. We also sample both UK data sets randomly to 500 (RE-RND) or 250 (UC-RND) websites. A third data set used by us consists of 172 sites in the real-estate domain, RE-OXF, that is sampled regionally instead of randomly, including all real estate websites for Oxford. This is an interesting case, as Oxford is generally more affluent and has a higher portion of rental properties than most other UK cities. To provide insight into DIADEM's ability to adapt to different countries, we use a sample of US real estate sites (RE-US). The ICQ dataset is a benchmark dataset for form understanding and only used for comparing that component of DIADEM to other approaches. To demonstrate that DIADEM can handle noisy input, we use the UK Top-100 shopping websites UK-100 as reported by http://www.hitwise.com/.

DIADEM is generally applicable to most product domains, where listing pages have at least some structured attributes, such as the price, in common and the set of attributes is generally homogeneous, though their presentation may differ widely. Among product domains, we chose the domains of used car and real estate for two primary reasons: In both domains, a large number of sellers is active and products are often unique to a seller. Furthermore, the two domains share very few attributes and differ quite notably with respect to average size and exploration structure of sites: Figure 7.21a and 7.21b show box plots[4] for the distribution of records per site and attributes per record, omitting sites with more than 10000 records where we avoid extracting all data. RE-FULL and UC-FULL have similar characteristics to their random sample. Sites in the used car domain has a much narrower range of records per site, yet a much

[4]The line inside the box is the median, its bottom and top the first and third quartile. The line extending from the box ends at the highest (lowest) value whose distance from the top (bottom) of the box is $\leq 1.5 \cdot$ IQR (IQR: interquartile range).

wider range of attributes per record. On average they have more attributes (14) than real estate sites (11).

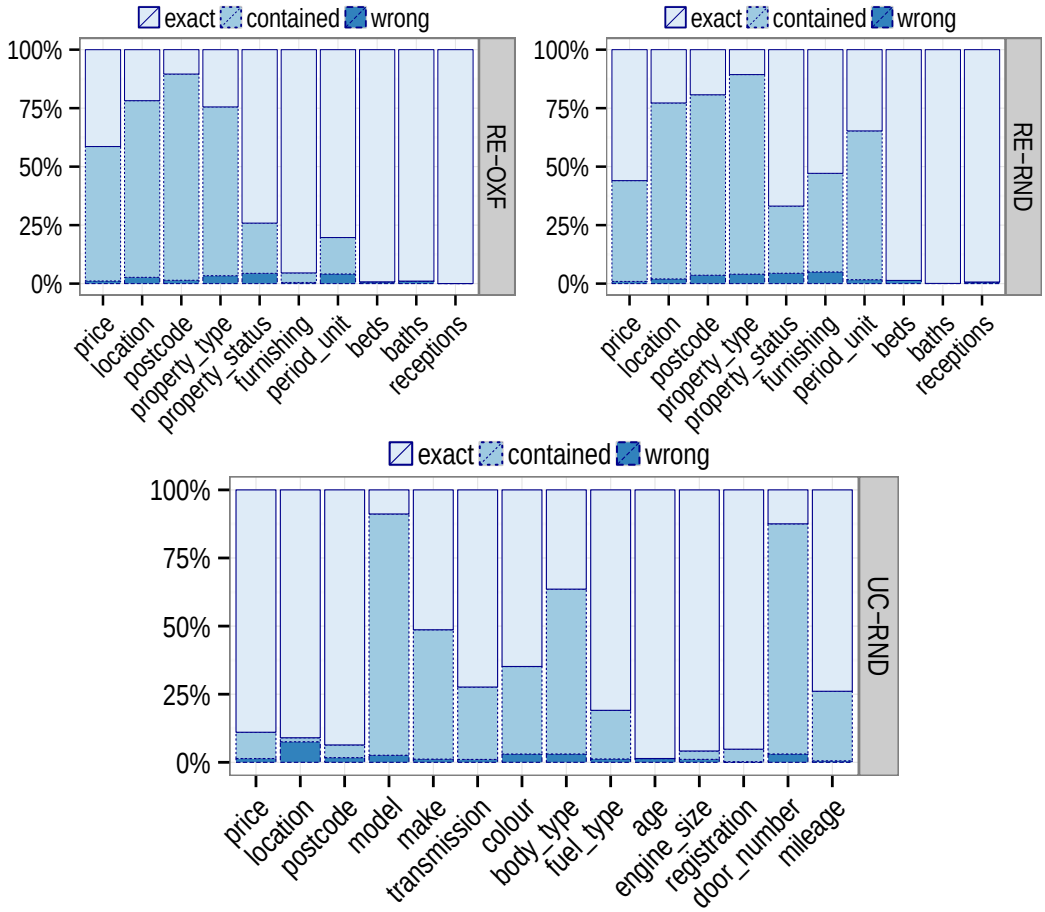


Figure 7.22: Attribute quality

7.2.2 Large Scale Wrapper Quality Evaluation

We evaluate the quality of DIADEM's full-site extraction and its components in four steps:

(1) *Wrapper effectiveness*, measures the portion of sites for which an effective wrapper is returned. Recall, that an effective wrapper returns all expected records from the page, possibly with duplicates, and thus has perfect recall.

(2) *Attribute quality*, measures the precision of the extracted attributes.

(3) *Form labeling accuracy* measures the accuracy of the assigned text labels for form fields and allows us to compare to existing approaches that only perform form labeling.

(4) *Record and attribute identification accuracy* measures the accuracy of the identified records and attributes on individual pages (rather than on the level of the generalised wrapper used for the extraction and evaluated in step (2)).

We do not evaluate DIADEM's wrapper execution individually for space reasons, but refer to [35].

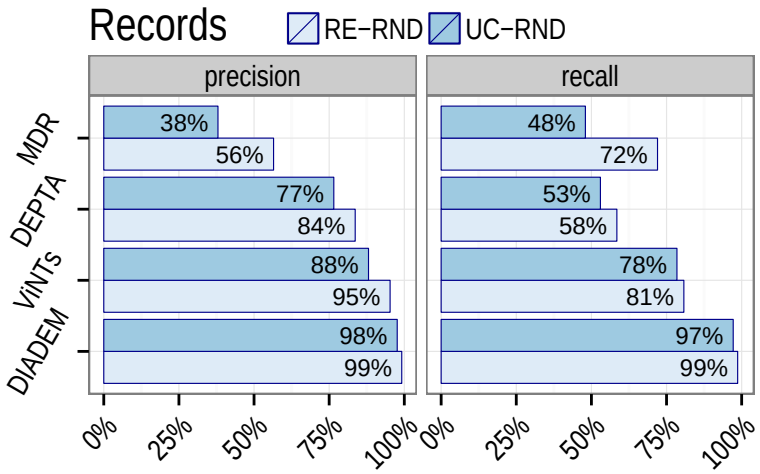
Table 7.3a shows the assessment of the **wrapper effectiveness** induced by DIADEM for RE-RND, UC-RND, RE-US, and RE-OXF. For the latter, we also show the corresponding numbers for ViNTs [85]. We assume that the random samples are representative for the full datasets, as indicated by the highly correlated characteristics. The primary result is that over 90% of the wrappers are effective in each datasets, with 91% average effectiveness. To avoid bias, we use a two step verification of the wrappers: Each wrapper is manually verified by one person. If a wrapper is considered effective, the actual extracted records are automatically compared to the search results number identified on the first listings page, if present. If not present, we use uniqueness of URL's and images and identical record numbers from different fillings. If this automatic verification fails, two more persons are asked to verify the wrapper and the aggregated result is reported.

		effective	incorrect	none
RE-RND	DIADEM	91%	7%	2%
UC-RND	DIADEM	93%	4%	3%
RE-US	DIADEM	90%	5%	5%
RE-OXF	DIADEM	90%	6%	4%
RE-OXF	ViNTs	4%	5%	91%

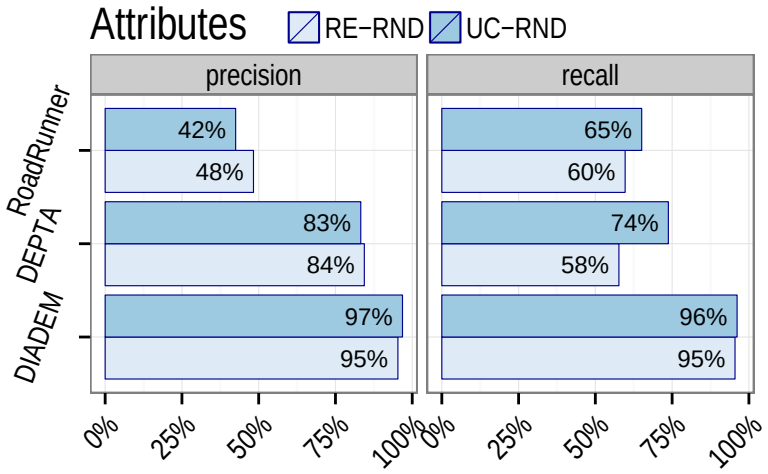
(a) Wrapper quality

Table 7.3: Datasets and wrapper quality

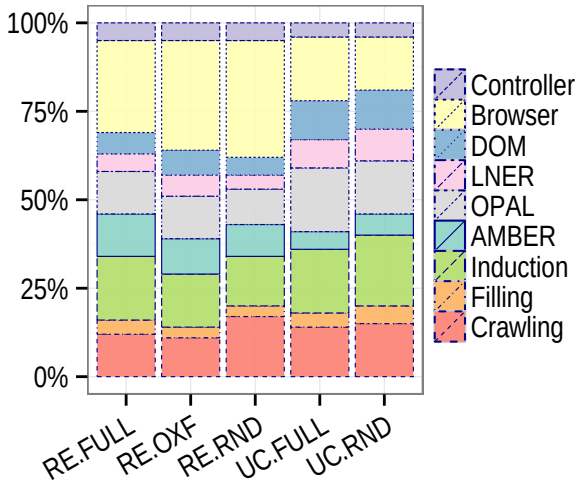
Contrast this to ViNTs, a system for fully automatically generating wrappers for search engines. It provides only few attributes that are common to many search engines, namely the title of the result and its textual description and thus we consider a wrapper effective if it extracts the right records (where for DIADEM we also inspect the attributes). ViNTs performs quite well if supervised by providing a few result pages and a non-result (control) page (Figure 7.23a). As DIADEM, ViNTs is also able to automatically extract a full-site wrapper starting from a form. However, this part of ViNTs has been specifically engineered for simple search forms. Thus, we remove all sites with no search forms, iFrames, or too few properties from the evaluation. Even in this case, ViNTs still only manages to produce a wrapper in 9%. Only for 4% of the sites it produces an effective wrapper. In all other cases, ViNTs only extracts part of the data, e.g., no rentals.



(a) Record identification (comparative)



(b) Attribute identification (comparative)



(c) Relative time for transducers

Figure 7.23: Comparative evaluation (identification) and transducer time

Among the most common causes for a DIADEM wrapper to be non effective are misaligned attributes, e.g., in presence of multiple pivot attributes or rare optional attributes, and sites that list related products more prominently. E.g., on a few sites that offer also new cars, DIADEM may extract those rather than the used cars, if neither listing contains many used car specific attributes and the new cars are more prominently placed on the site. There are about 3% of sites where no wrapper can be induced, typically as they contain no properties, all properties are on aggregators, or they contain no pivot attribute. For these sites, DIADEM correctly detects that there is no effective wrapper. The final case is that DIADEM fails to produce an effective wrapper, yet one exists. The most common reasons for these failures are dynamic forms (15%), result pages with dynamically rendered prices (12%), forms located in sidebar `iframe` s (15%), prices without currencies (6%), or sites which contain only a single property (6%).

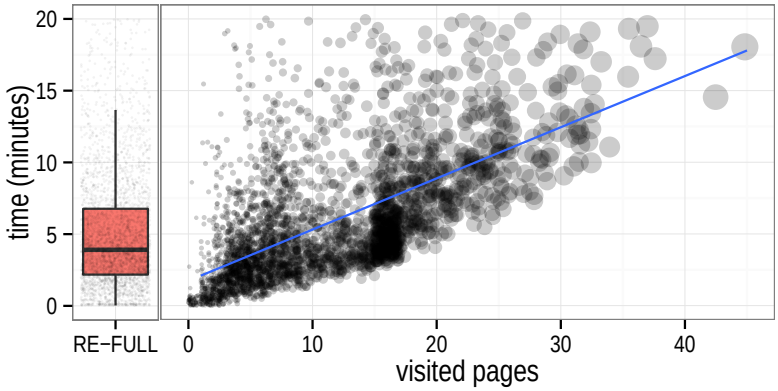
To demonstrate, that DIADEM does not produce a wrapper for sites that are not in the target domain, we also run DIADEM on the set of top UK shopping websites UK-100. On this set, DIADEM induces a wrapper for only 5 sites, confusing toy cars on Amazon and Toys-R-Us for used cars.

To determine the **attribute quality** of the extracted data, we perform a manual evaluation on the RE-OXF, RE-RND, and UC-RND datasets. Again, we use a two step verification, both manual and automatic with DIADEM's LNERs or Bing (for locations). Attributes are either exact matches, contain the intended value, or wrong. Figure 7.22 summaries the results. Overall, the attribute quality $> 97\%$ in the two random datasets (and even higher in RE-OXF). The attribute with the highest error rate is the location in UC-RND. In the real estate cases this attribute has a rather low error rate. The reason is that in UC-RND, location is not a very common attribute (below 20% of the records). It typically appears only on sites of dealers with multiple offices, indicating the cars position.

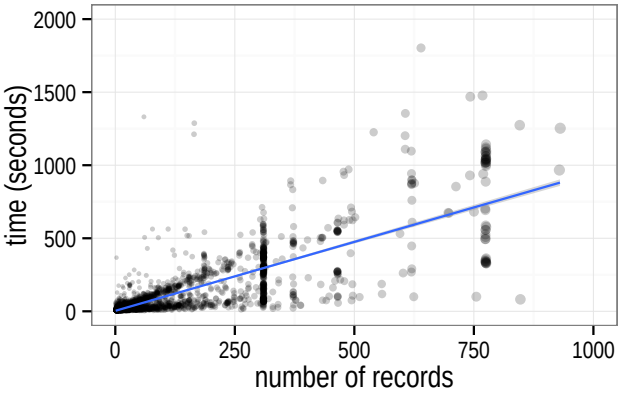
In addition to the overall performance of the full-site extraction considered so far, we also evaluated the AMBER components separately. For the **record and attribute identification accuracy**, we compare DIADEM with RoadRunner [18], MDR [58], ViNTs [85] (in supervised mode where we provide a set of result pages) and DEPTA+ [84]. These were the only systems available to us for evaluation. For this experiment, we further reduce the two random datasets RE-RND and UC-RND. We eliminated all sites with only one result page, where RoadRunner and ViNTs either fail or perform significantly worse than with multiple result pages. We also eliminate all sites where more than one of these systems fails to render the page, where one of them crashes, or that use iFrames, all not supported by one or more of these systems. For each of the remaining sites, we created a gold standard covering 2–5 result pages per site. Figure 7.23a reports the resulting record and attribute identification accuracy. For ViNTs and MDR, we only report

the record case, as they return no or very limited (title and body) attributes. For RoadRunner, we only report the attribute case, as it returns no records per se. Where for DIADEM, we require exact matches for both attributes and records, the other systems are evaluated using best case metrics, where possible: For all systems, we consider any value that contains the gold standard attribute a match. For MDR, we pre-filter navigation menus, footers, headers, pagination links, and other regular structures which otherwise are returned by MDR as records. For DEPTA+, we only consider identification accuracy for attributes in records that are perfectly segmented by DEPTA+.

Nevertheless, DIADEM outperforms all these systems by a wide margin. They particularly suffer from identifying nested structures (e.g., from repeated attributes). ViNTs performs quite well in record segmentation when provided with enough result pages, however, often recognizes adjacent pagination links as records and fails to deal with most grid layout pages.



(a) Overall time w/o extraction



(b) Extraction time

Figure 7.24: Performance of DIADEM

7.2.3 Performance Evaluation

DIADEM's automatic full-site extraction is able to produce an effective wrapper for most sites on a single core machine in just a few minutes. Figure 7.24a shows the time for the entire process for RE-FULL. The left hand shows a box plot for the distribution of runtime. The median is at 3.9 mins. For most websites, a wrapper is returned in $\leq$ 10 mins. Performance is correlated with the number of pages visited on a site (right hand side). It also depends on the size of these pages and the number of form fillings necessary. Figures 7.24b shows the time for evaluating all wrappers on the 3404 site RE-FULL dataset. These extract in total 352k records (before de-duplication) in 2.5hs on a 45 nodes cluster.

In addition to overall performance, the relative performance of DIADEM's transducers gives a better insight into its processing, as seen in Figure 7.23c. We have grouped some of the smaller transducers by function. Specifically, it shows that browser interaction, including page retrieval and rendering, as well as execution of form actions, dominates the runtime. There is a slight variance in the relative time for browser interaction between the two smaller real-estate datasets and the full one. This is caused by a larger number of non-form websites (of small agencies) in the full dataset. Among the other transducers, the wrapper induction takes up to 18%, followed by the crawler, AMBER, DIADEM's result and attribute identification, and OPAL, the form understanding transducer.

7.3 Detail Page Evaluation

We evaluate AMBER on real-world detail pages taken from the real-estate domain. In the following we discuss:

(i) The experimental setup in section 7.3.1.

(ii) An introspective quality evaluation on two domains: real estate and used car, where we compare AMBER's output with manually curated gold-standard in section 7.3.2.

(iii) Evaluations on feature effectiveness and component effectiveness in section 7.3.3.

7.3.1 Experimental Setup

Our real-estate data set **RE-D** consists of a random sample of 410 pages taken from 50 randomly picked UK real-estate websites out of a corpus of 3,404 real-estate websites (re.full), collected through a combination of results from UK yellow pages, real estate verticals and contains websites ranging from large aggregators such as RIGHTMOVE.CO.UK to very small city-wide agencies.

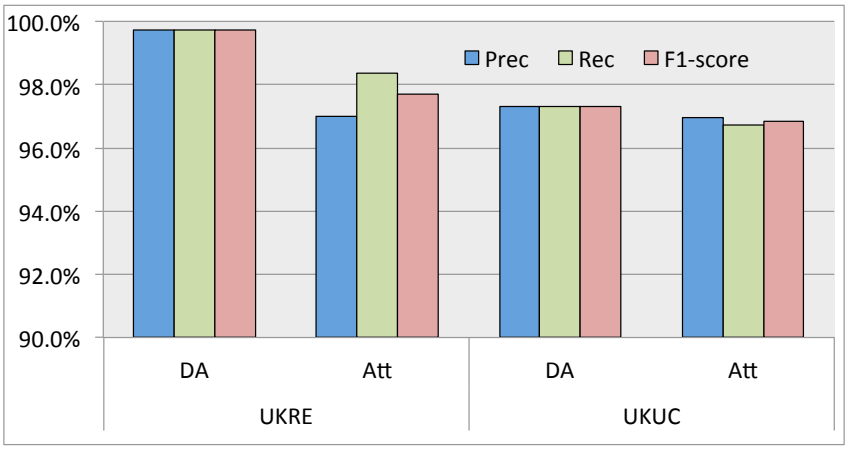


Figure 7.25: Overall performance on **RE-D** and **UC-D** datasets.

The second dataset **UC-D** consists of a random sample of 450 pages taken from 50 UK used car dealers out of a corpus of 7,089 websites (uc.full) collected in a similar way as for ukre, by using corresponding used car verticals.

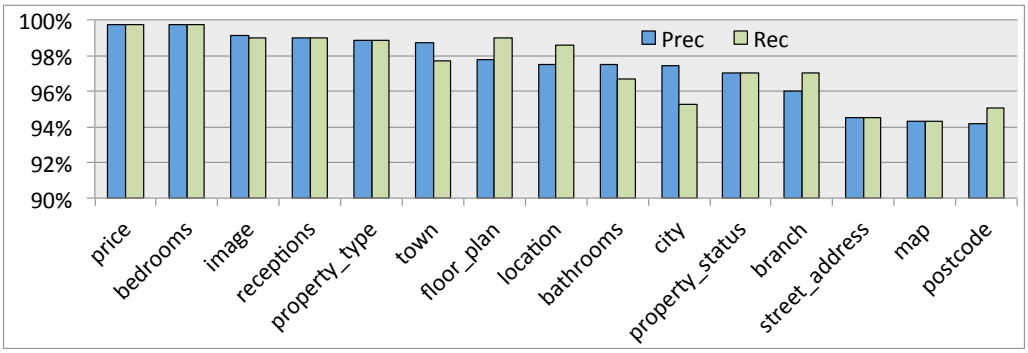
To ensure diversity in the corpora, if two sites are sharing the same template, we replace one of them with another new site from the larger corpus. For both **RE-D** and **UC-D** the ground-truthing has been done by manually annotating up to two result pages with at least two records in each page. Each record was manually annotated by three expert annotators, resulting in a *kappa* (i.e., inter-annotator agreement) of 92% for data areas and records and of 83% on attributes. Both datasets have been cleaned of websites that were unreachable or returned HTTP errors or redirects. We also removed all sub-domains of aggregators and car makers as these are all similarly structured and would have biased the evaluation.

The evaluation metrics we used are the same with section 7.1.1, precision, recall, and F_1-score computed against the gold standard.

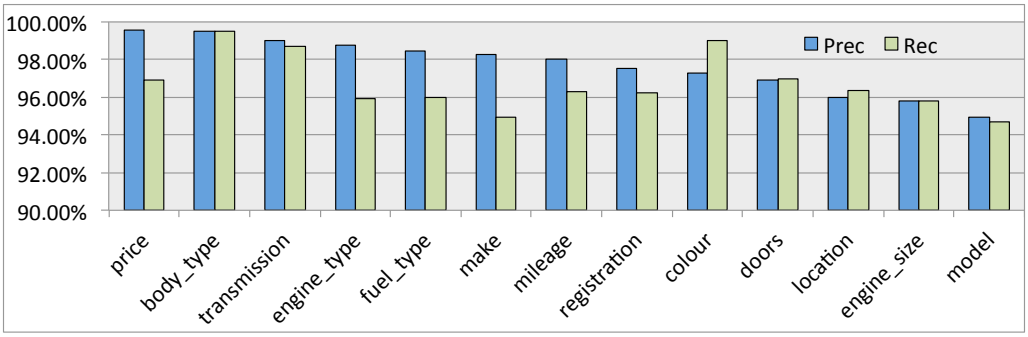
7.3.2 Attribute Extraction Quality Evaluation

The first experiment is concerned with an overall evaluation of AMBER on two datasets, and the outcome is shown in Figure 7.25. We evaluate data area identification(DA), and attribute extraction(Att). For the **RE-D** and **UC-D** datasets, the attribute types annotated and extracted are those listed in Table 6.1. In average, AMBER can identify the data area correctly in 97% of the cases, and reaches >96% precision and recall on overall attributes extraction.

Regarding attribute extraction, AMBER achieves an average of >96% F_1-score over the **RE-D** and **UC-D** datasets. This is an unprecedented result if we consider that we are targeting the extraction of attribute-rich objects with more than 15 different attribute types. The detailed performance for each attribute type in the two domains is summarized in in Figure 7.26. As



(a) ukre



(b) ukuc

Figure 7.26: Detail page attribute extraction performance.

expected, AMBER is able to identify more accurately the regular attributes than the optional ones. This is mainly due to the optionality that influences features such as the redundancy resulting and global tag path support. It is also expected that the attributes with lower recall are related to geo-information or complicated gazetteer such as STREET_ADDRESS and MODEL. Location information is harder to recognize than other product-related information as they are usually poorly structured and do not follow precise patterns. The extraction of this information is often regarded as an independent research area. MODEL is a special attribute since it depends on the quality of gazetteer. The gazetteer is large and complicated, with many duplications, and also has a high overlap with other attributes or basic text pieces such as numbers.

Extraction Quality Figure 7.27 summarizes the quality of the extracted instances. Our corpus contains a total of 410 distinct objects in **RE-D** and 450 distinct objcts in **UC-D**, both with values for all the attributes targeted by our extraction. By compare with the manually curated gold standard, we find that in 65% of the cases AMBER is able to extract the complete entity, with all attributes correctly identified. And in 24% of cases, only 1 type of attribute is incorrect, and only 4% of pages are having 3 and more types of attributes that are incorrect. All instances of the target entity have been (at least partially) recognized and no false positives have been

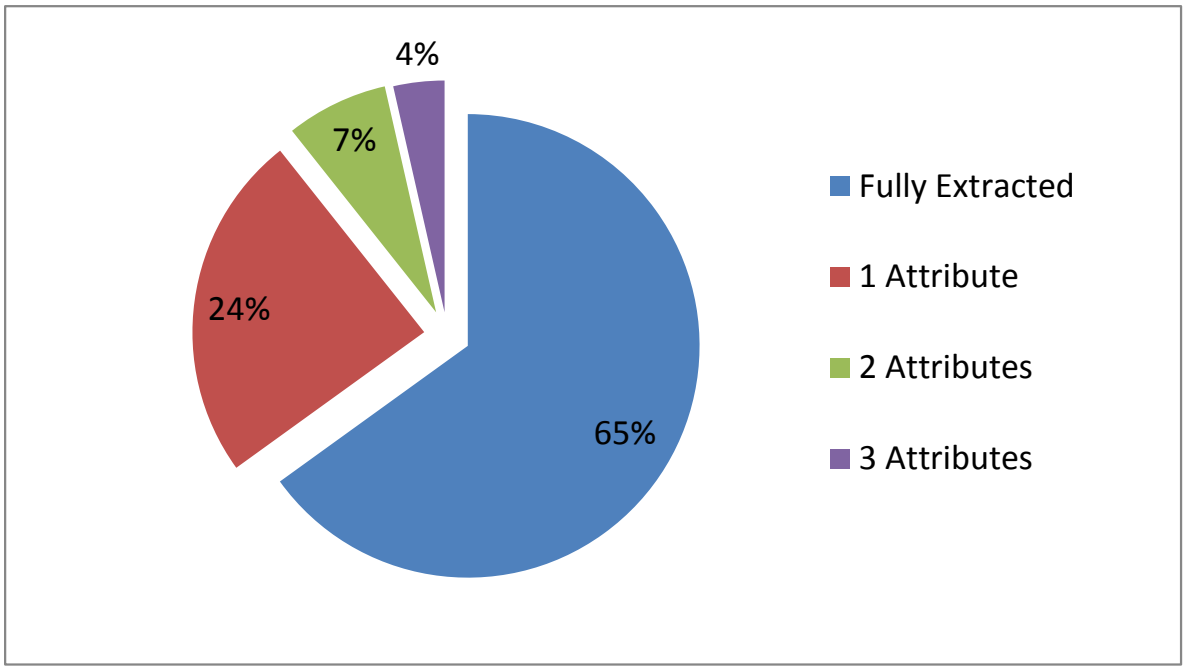


Figure 7.27: Extraction Quality.

registered.

7.3.3 Introspective Evaluation

We have done a series of introspective evaluation to demonstrate that AMBER is stable, robustness on detail page extraction.

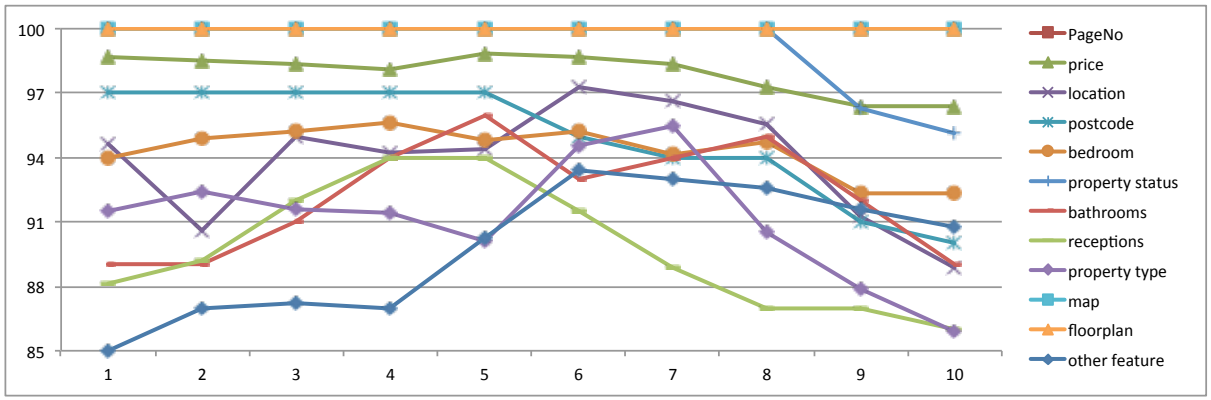


Figure 7.28: Robustness evaluation

Input Size Evaluation The efficiently-computable features and the simplicity of the scoreboard mechanism allow us to push the analysis time on average in the range of 2 seconds per detail page with peaks of 12 seconds on large pages. We tested on **RE-D** data sets that how different input size (number of detail page given as input) affect AMBER's result. The analysis process is also robust to the number of detail pages taken as input. To evaluate this, we randomly picked 30 websites, collected 10 pages from each of them. Then for each site, we repeatedly tested AMBER on random subsets of different size. The F1-score results are shown in Figure 7.28 for samples of size 1 to 10. AMBER is able to produce accurate attribute extraction for most of the attributes using only few pages, however, an increasing number of input pages often results in better performance until reaches the peak of 5-6 pages, then it slightly goes down again.

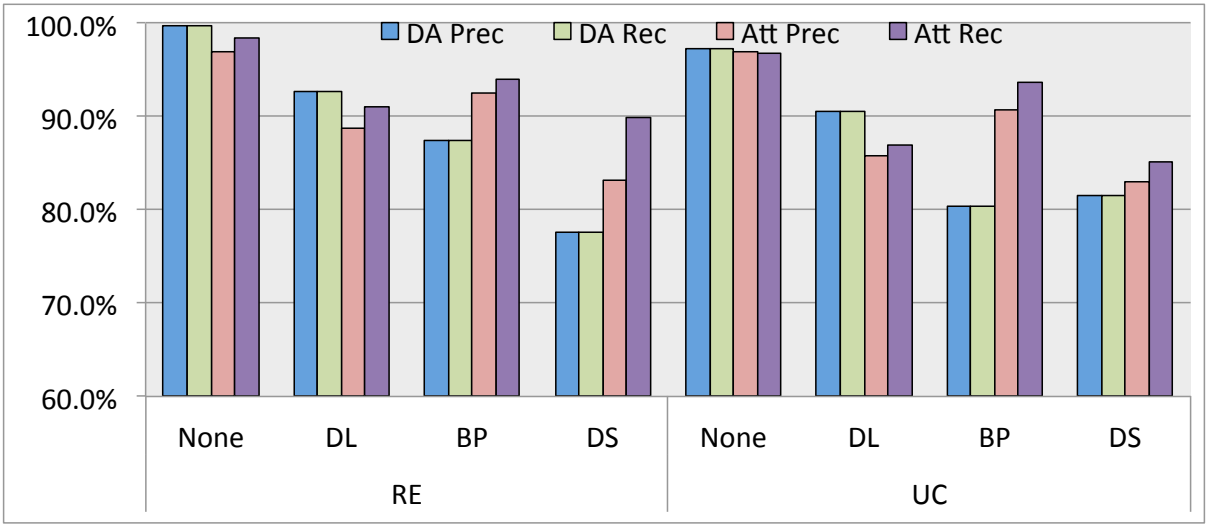


Figure 7.29: Component Effectiveness

Component Effectiveness In AMBER's data area identification part, we have three components **(i)** boilerplate removal, **(ii)** dynamic list identification, and **(iii)** page dissimilarity calculation, that contribute to getting the final result. To demonstrate the effectiveness of each component, we disable one component at a time and run AMBER on both entire **RE-D** and **UC-D** corpus to determine the contribution. Figure 7.29 shows that **(1)** When we disabled *boilerplate removal* component (BP), although the data area precision and recall may drop significantly (13% on RE and 16% on UC), the attributes precision and recall are less affected (5% on RE and 6% on UC). The result indicates that boilerplate blocks usually do not contain much useful information. **(2)** When *dynamic list identification* component (DL) is disabled, attribute precision and recall drop (9% on RE and 10% on UC) more than data area precision

and recall(7% on both RE and UC). That shows although only a small fraction of websites are having dynamic lists, however the content of those lists consists of real attributes and hence might mislead object extraction systems. **(3)** When we disabled *page dissimilarity calculation* component (DS), precision and recall on data area had a huge drop, (20% on RE and 16% on UC), however the attributes precision and recall has been affected less (13% on both RE and UC). Some static attributes might got a high vote since they may occur more than once and highly regular. Without calculating page dissimilarity, the feature dynamic content(DC) is also disabled, which reduced AMBER's performance. Overall, this evaluation demonstrates that each component has its own contribution and irreplaceable.

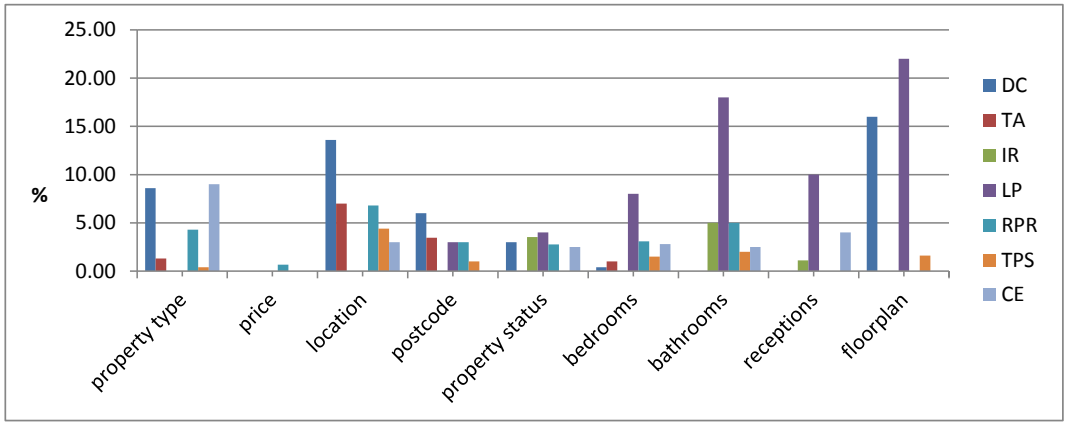


Figure 7.30: Precision loss vs features.

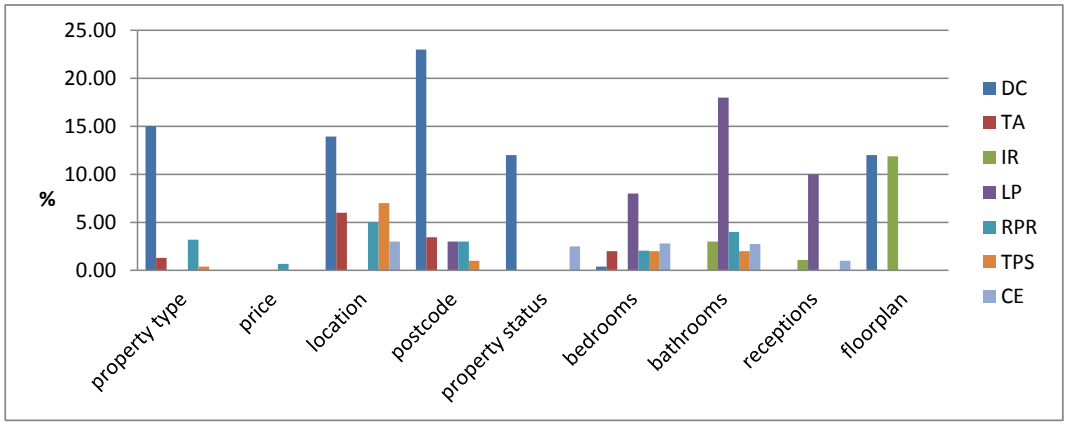


Figure 7.31: Recall loss vs features.

Feature Effectiveness In this experiment, we disable one feature at a time and run AMBER on the entire **RE-D** corpus to determine the contribution of each feature. Figure 7.30 and

Figure 7.31 show the precision and recall loss in attribute recognition once a feature is disabled. The most decisive feature is without doubt **DC**, identifying nodes with same tag-path but different content. Disabling this feature in the scoreboard leads to a 25% precision loss in the recognition of the POSTCODE and PROPERTYSTATUS attributes and a recall loss of 18% in the recognition of LOCATION. Unsurprisingly, the attributes suffering more from the absence of at least one of the features are PROPERTYTYPE, LOCATION, and PROPERTYSTATUS due to variations in the attribute values and noise in the annotations. Other attributes, such as PRICE or BATHROOMS can be recognized with either regular expressions or annotations with very small gazetteers and are therefore less sensitive to the absence of one of the features. Attributes that appear in proximity and whose domains have similar values, e.g., BEDROOMS, BATHROOMS, and RECEPTIONS suffer from the absence of structural annotations (**CE**), label proximity (**LP**), and result-page redundancy (**RPR**). These features are in fact fundamental to disambiguate multiple annotations overlapping on the same text nodes.

Chapter 8

Conclusion and Future Work

8.1 Summary

AMBER pushes the state-of-the-art in extraction of multi-attribute objects from the deep web, through a fully-automatic approach that combines the analysis of the repeated structure of the web page and automatically-produced annotations. AMBER's main contributions are:

(1) AMBER is the first informed, automated data extraction system focused on both result pages and detail pages with complex multi-attribute objects.

(2) AMBER is an important part of DIADEM, the first system for automatic full-site extraction capable of producing highly accurate, effective wrappers for thousands of websites in several application domains.

(3) AMBER requires no per-site supervision but a small amount of domain knowledge which is easily obtainable from just a few example instances and pages.

(4) Without domain knowledge given, AMBER is still able to extract and segment records from result pages through generic attributes such as images or urls, which made AMBER easily adaptive to other domains.

(5) AMBER compensates for both structurally and textually noises and reaches high accuracy(>95%) on extracting complex multi-attribute objects from both result and detail pages.

8.2 Future Work

Though AMBER is outperforming existing approaches by a notable margin for multi-attribute object extraction on product domains, there still remains a number of open issues in AMBER and DIADEM. This section discusses future plans of AMBER and DIADEM.

(1) We are confident that AMBER and DIADEM can be applied to most if not all product domains, such as hotel, electronic, fashion, restaurant, book, etc. However, the performance on several domains such as event announcement, where objects are having a significant variability and may not share any common attribute, is an open question.

(2) We want to integrate the form filling result with AMBER to improve the accuracy of extracted attributes.

(3) We also want to improve AMBER to work on irregular, modern designed detail pages and run a large scaling evaluation on detail page extraction.

(4) Last but not least, a start-up company has been established with the DIADEM system, under the support of the University of Oxford. Collaborations have been built recently between the DIADEM start-up company and well-known industrial companies.

Bibliography

[1] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348, San Diego, California, USA, 2003. ACM.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. DBPedia: A nucleus for a web of open data. In *Proceedings of International Semantic Web Conference (ISWC)*, pages 11–15, 2007.

[3] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 580–591, New York, NY, USA, 2002. ACM.

[4] Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a competition for cleaning web pages. Marrakech, Morocco, may 2008. European Language Resources Association.

[5] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB 01, pages 119–128, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[6] Lidong Bing, Wai Lam, and Tak-Lam Wong. Robust detection of semi-structured web records using a dom structure-knowledge-driven model. In *ACM Transactions on the Web*, volume 7, pages 21:1–21:32, New York, NY, USA, November 2013. ACM.

[7] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proceedings of the 39th Conference on Very Large Databases*, 6(10):805–816, 2013.

[8] David Buttler, Ling Liu, and Calton Pu. A fully automated object extraction system for the world wide web. In *Proceedings of the 21st International Conference on Distributed*

Computing Systems, pages 361–370, Phoenix (Mesa), Arizona, USA, 2001. IEEE Computer Society.

[9] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. In *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, volume 1, pages 538–549, Auckland, New Zealand, August 2008. VLDB Endowment.

[10] Michael J. Cafarella, Alon Y. Halevy, and Jayant Madhavan. Structured data on the web. In *Communications of ACM*, volume 54, pages 72–79, New York, NY, USA, February 2011. ACM.

[11] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Block-based web search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 456–463, New York, NY, USA, 2004. ACM.

[12] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. Page-level template detection via isotonic smoothing. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 61–70, New York, NY, USA, 2007. ACM.

[13] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled Shaalan. A Survey of Web Information Extraction Systems. In *IEEE Transactions on Knowledge and Data Engineering*, volume 18, pages 1411–1428, Piscataway, NJ, USA, October 2006. IEEE Educational Activities Department.

[14] Chia-Hui Chang and Shao-Chen Lui. Iepad: Information extraction based on pattern discovery. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 681–688, New York, NY, USA, 2001. ACM.

[15] Luying Chen, Stefano Ortona, Giorgio Orsi, and Michael Benedikt. Aggregating semantic annotators. In *Proceedings of the 39th Conference on Very Large Databases*, volume 6, pages 1486–1497, Hangzhou, China, August 2013. VLDB Endowment.

[16] Valter Crescenzi and Giansalvatore Mecca. Grammars have exceptions. In *Information Systems - Special issue on semistructured data*, volume 23, pages 539–565, Oxford, United Kingdom, 12 1998. Elsevier Science Ltd.

[17] Valter Crescenzi and Giansalvatore Mecca. Automatic Information Extraction from Large Websites. In *Journal of ACM*, volume 51, pages 731–779, 2004.

[18] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Automatic Data Extraction from Data-Intensive Web Site. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, page 624, Madison, Wisconsin, USA, 2002. ACM.

[19] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 261–272, Rio de Janeiro, Brazil, 2013.

[20] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. The University of Sheffield, Department of Computer Science, 2011.

[21] Bhavana Bharat Dalvi, William W. Cohen, and Jamie Callan. Websets: extracting sets of entities from the web using unsupervised information extraction. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 243–252, Seattle, Washington, USA, 2012. ACM.

[22] Nilesh N. Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of ACM Symposium on Management of Data (SIGMOD)*, SIGMOD '09, pages 335–348, Providence, Rhode Island, USA, 2009. ACM.

[23] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. Automatic wrappers for large scale web extraction. In *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, volume 4, pages 219–230, The Westin, Seattle, Washington, USA, 2011. ACM.

[24] Sandip Debnath, Prasenjit Mitra, Nirmal Pal, and C. Lee Giles. Automatic identification of informative sections of web pages. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, pages 1233–1246, Piscataway, NJ, USA, September 2005. IEEE Educational Activities Department.

[25] Nora Derouiche, Bogdan Cautis, and Talel Abdessalem. Automatic extraction of structured web data with domain knowledge. In *Proceedings of* 28^{st} *International Conference on Data Engineering*, pages 726–737, Washington, DC, USA, 2012. IEEE Computer Society.

[26] Eduard C. Dragut, Thomas Kabisch, Clement Yu, and Ulf Leser. A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, volume 2, pages 325–336, 2009.

[27] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. In *Proceedings of 35th Confercence on Very Large Databases*, volume 2, pages 1078–1089, Lyon, France, August 2009. VLDB Endowment.

[28] David W. Embley, Douglas M. Campbell, Y. Jiang, Stephen W. Liddle, Dallan Quass, Yiu-Kai Ng, and Randy D. Smith. Conceptual-model-based data extraction from multiple-record web pages. In *Journal of Data and Knowledge Engineering*, volume 31, pages 227–251, Amsterdam, The Netherlands, The Netherlands, 1999. Elsevier Science Publishers B. V.

[29] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. In *Communication of the ACM*, volume 51, pages 68–74, New York, NY, USA, December 2008. ACM.

[30] David Fernandes, Edleno Silva de Moura, Altigran Soares da Silva, Berthier Ribeiro-Neto, and Edisson Braga. A site oriented method for segmenting web pages. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 215–224, Beijing, China, 2011. ACM.

[31] Dayne Freitag. Machine Learning for Information Exrtraction in Informal Domains. In *Machine Learning*, volume 39, pages 169–202, Hingham, MA, USA, May 2000. Kluwer Academic Publishers.

[32] Tim Furche, Georg Gottlob, Giovanni Grasso, Omer Gunes, Xiaoanan Guo, Andrey Kravchenko, Giorgio Orsi, Christian Schallhart, Andrew Sellers, and Cheng Wang. Diadem: Domain-centric, intelligent, automated data extraction methodology. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 267–270, New York, NY, USA, 2012. ACM.

[33] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. DIADEM: Thousands of Websites to a Single Database. In *Proceedings of 40th Conference on Very Large Databases*, volume 7, 2014.

[34] Tim Furche, Georg Gottlob, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Little knowledge rules the web: Domain-centric result page extraction. In

Proceedings of the 5th International Conference on Web Reasoning and Rule Systems, RR'11, pages 61–76, Galway, Ireland, 2011. Springer-Verlag.

[35] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Sellers. OXPath: A language for scalable data extraction, automation, and crawling on the deep web. *VLDB Journal*, pages 47–72, 2013.

[36] Tim Furche, Georg Gottlob, Xiaonan Guo, Christian Schallhart, Andrew Sellers, and Wang Cheng. How the minotaur turned into ariadne: ontologies in web data extraction. In *International Conference on Web engineering (ICWE)*, volume 246858, 2011.

[37] Tim Furche, Giovanni Grasso, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Automatically learning gazetteers from the deep web. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 341–344, Lyon, France, 2012. ACM.

[38] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 71–80, Banff, Alberta, Canada, 2007. ACM.

[39] David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of web page templates. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, WWW '05, pages 830–839, New York, NY, USA, 2005. ACM.

[40] John Gibson, Ben Wellner, and Susan Lubar. Adaptive web-page content identification. In *Proceedings of the 9th Annual ACM International Workshop on Web Information and Data Management*, WIDM '07, pages 105–112, New York, NY, USA, 2007. ACM.

[41] Aman Goel, Craig A. Knoblock, and Kristina Lerman. Using conditional random fields to exploit token structure and labels for accurate semantic annotation. In *Proceedings of Association for the Advancement of Artificial Intelligence*, San Francisco, California, USA, 2011.

[42] Pankaj Gulhane, Amit Madaan, Rupesh R. Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H. Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1209–1220, Washington, DC, USA, 2011. IEEE Computer Society.

[43] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. Exploiting content redundancy for web information extraction. In *Proceedings of the* 36^{th} *Confercence on Very Large Databases*, volume 3, pages 578–587, Singapore, Singapore, September 2010. VLDB Endowment.

[44] Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. In *Proceedings of the* 35^{th} *Confercence on Very Large Databases*, volume 2, pages 289–300, Lyon, France, August 2009. VLDB Endowment.

[45] Rahul Gupta and Sunita Sarawagi. Joint training for open-domain extraction on the web: Exploiting overlap when supervision is limited. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 217–226, Hong Kong, China, 2011. ACM.

[46] Joachim Hammer, Jason McHugh, and Hector Garcia-Molin. Semistructured data: the TSIMMIS experience. In *Proceedings of* 1^{st} *East-European Symposium on Advances in Databases and Information Systems*, ADBIS'97, pages 22–22, Swinton, United Kingdom, United Kingdom, 1997. British Computer Society.

[47] Andrew Hogue and David Karger. Thresher: Automating the unwrapping of semantic content from the world wide web. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 86–95, Chiba, Japan, 2005. ACM.

[48] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semistructured data extraction from the web. In *Information Systems*, volume 23, pages 521–538, Oxford, United Kingdom, December 1998. Elsevier Science Ltd.

[49] Nitin Jindal and Liu Bing. A generalized tree matching algorithm considering nested lists for web data extraction. In *SIAM International Conference on Data Mining*, pages 930–941, Columbus, Ohio, USA, 2010. ACM.

[50] Mohammed Kayed and Chia-Hui Chang. FiVaTech: Page-Level Web Data Extraction from Template Pages. In *IEEE Transactions on Knowledge and Data Engineering*, volume 22, pages 249–263, Piscataway, NJ, USA, 2010. IEEE Educational Activities Department.

[51] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[52] Raymond Kosala, Hendrik Blockeel, Maurice Bruynooghe, and Jan Van den Bussche. Information extraction from structured documents using *k*-testable tree automaton inference. In *Data & Knowledge Engineering (DKE)*, volume 58, pages 129–158, Amsterdam, The Netherlands, August 2006. Elsevier Science Publishers B. V.

[53] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI)*, pages 729–735, NAGOYA, Aichi, Japan, 1997. University of Washington.

[54] Alberto H. F. Laender, Berthier Ribeiro-Neto, and Altigran S. da Silva. DEByE – Data Extraction by Example. In *Data & Knowledge Engineering (DKE)*, volume 40, pages 121–154, Amsterdam, The Netherlands, The Netherlands, February 2002. Elsevier Science Publishers B. V.

[55] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. In *Proceedings of ACM Symposium on Management of Data (SIGMOD)*, volume 31, pages 84–93, New York, NY, USA, June 2002. ACM.

[56] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *TOCL*, 7(3):499–562, 2006.

[57] Kristina Lerman, Lise Getoor, Steven Minton, and Craig A. Knoblock. Using the structure of web sites for automatic segmentation of tables. In *Proceedings of Special Interest Group on Management of Data*, pages 119–130, Paris, France, 2004.

[58] 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, pages 601–606, Washington, D.C., USA, 2003. ACM.

[59] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. In *IEEE Transactions on Knowledge and Data Engineering*, volume 22, pages 447–460, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[60] Yiyao Lu, Hai He, Hongkun Zhao, Weiyi Meng, and Clement Yu. Annotating search results from web databases. In *IEEE Transaction on Knowledge and Data Engineering*, volume 25, pages 514–527, Piscataway, NJ, USA, March 2013. IEEE Educational Activities Department.

[61] MarketLine. Online retail in the united states, 2013. On-line at http://www.marketresearch.com/MarketLine-v3883/Online-Retail-United-States-7760207/.

[62] Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the WEB*, 6(1):3:1–3:30, 2012.

[63] Gengxin Miao, Jun'ichi Tatemura, Wang-Pin Hsiung, Arsany Sawires, and Louise E. Moser. Extracting data records from the web using tag path clustering. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the* 18^{th} *International World Wide Web Conference*, pages 981–990, Madrid, Spain, 2009. ACM.

[64] Ion Muslea, Steven Minton, and Craig A.Knoblock. Hierarchical wrapper induction for semistructrued information systems. In *Autonomous Agents and Multi-Agent Systems*, volume 4, pages 93–114, Hingham, MA, USA, March 2001. Kluwer Academic Publishers.

[65] Hoa Nguyen, Thanh Nguyen, and Juliana Freire. Learning to Extract From Labels. In *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, pages 684–694, 2008.

[66] Paolo Papotti, Valter Crescenzi, Paolo Merialdo, Mirko Bronzi, and Lorenzo Blanco. Redundancy-driven web data extraction and integration. In *Procceedings of the 13th International Workshop on the Web and Databases*, WebDB '10, pages 7:1–7:6, New York, NY, USA, 2010. ACM.

[67] Jeff Pasternack and Dan Roth. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 971–980, New York, NY, USA, 2009. ACM.

[68] Benjamin Rozenfeld and Ronen Feldman. Self-supervised relation extraction from the web. In *Knowledge and Information Systems*, volume 17, pages 17–33, New York, NY, USA, October 2008. Springer-Verlag New York, Inc.

[69] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM Workshop on Web Information and Data Management*, WIDM '08, pages 9–16, Napa Valley, California, USA, 2008. ACM.

[70] Kai Simon and Georg Lausen. ViPER: Augmenting Automatic Information Extraction with visual Perceptions. In *Proceedings of the 14th ACM International Conference on*

Information and Knowledge Management, CIKM '05, pages 381–388, Bremen, Germany, 2005. ACM.

[71] Hassan A. Sleiman and Rafael Corchuelo. A survey on region extractors from web documents. In *IEEE Transaction on Knowledge and Data Engineering*, volume 25, pages 1960–1981, Piscataway, NJ, USA, September 2013. IEEE Educational Activities Department.

[72] Hassan A. Sleiman and Rafael Corchuelo. Trinity: On using trinary trees for unsupervised web data extraction. In *IEEE Transaction on Knowledge and Data Engineering*, volume 26, pages 1544–1556, 2014.

[73] Weifeng Su, Jiying Wang, and Frederick H. Lochovsky. Ode: Ontology-assisted data extraction. In *ACM Transactions on Database Systems*, volume 34, pages 12:1–12:35, New York, NY, USA, July 2009. ACM.

[74] Weifeng Su, Jiying Wang, Frederick H. Lochovsky, and Yi Liu. Combining tag and value similarity for data extraction and alignment. In *IEEE Transaction on Knowledge and Data Engineering*, volume 24, pages 1186–1200, Piscataway, NJ, USA, July 2012. IEEE Educational Activities Department.

[75] Weifeng Su, Hejun Wu, Yafei Li, Jing Zhao, Frederick H. Lochovsky, Hongmin Cai, and Tianqiang Huang. Understanding query interfaces by statistical parsing. *ACM Transactions on the WEB*, 7(2):8:1–8:22, 2013.

[76] Guilherme A. Toda, Eli Cortez, Altigran S. da Silva, and Edleno de Moura. A probabilistic approach for automatically filling form-based web interfaces. In *Proceedings of 36th Conference on Very Large Databases*, volume 4, pages 151–160, 2010.

[77] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. In *Proceedings of the* 37^{th} *Confercence on Very Large Databases*, volume 4, pages 528–538, The Westin, Seattle, WA, USA, June 2011. VLDB Endowment.

[78] Jiying Wang and Fred H. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 187–196, Budapest, Hungary, 2003. ACM.

[79] Junfeng Wang, Chun Chen, Can Wang, Jian Pei, Jiajun Bu, Ziyu Guan, and Wei Vivian Zhang. Can we learn a template-independent wrapper for news article extraction from a

single training site? In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1345–1354, Paris, France, 2009. ACM.

[80] Richard C. Wang and William W. Cohen. Character-level analysis of semi-structured documents for set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1503–1512, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[81] Tim Weninger, Fabio Fumarola, Rick Barber, Jiawei Han, and Donato Malerba. Unexpected results in automatic list extraction on the web. In *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations*, volume 12, pages 26–30, New York, NY, USA, March 2011. ACM.

[82] Yasuhiro Yamada, Nick Craswell, Tetsuya Nakatoh, and Sachio Hirokawa. Testbed for information extraction from deep web. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters*, WWW Alt. '04, pages 346–347, New York, NY, USA, 2004. ACM.

[83] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 296–305, New York, NY, USA, 2003. ACM.

[84] Yanhong Zhai and Bing Liu. Structured Data Extraction from the Web Based on Partial Tree Alignment. In *IEEE Transaction on Knowledge and Data Engineering*, volume 18, pages 1614–1628, Piscataway, NJ, USA, December 2006. IEEE Educational Activities Department.

[85] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully Automatic Wrapper Generation For Search Engines. In *Proceedings of the* 14^{th} *International World Wide Web Conference*, WWW '05, pages 66–75, Chiba, Japan, 2005. ACM.

[86] Xiaoqing Zheng, Yiling Gu, and Yinsheng Li. Data extraction from web pages based on structural-semantic entropy. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 93–102, Lyon, France, 2012. ACM.

[87] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. Simultaneous record detection and attribute labeling in web data extraction. In *Proceedings of the 12th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, pages 494–503, Philadelphia, PA, USA, 2006. ACM.