DEXTER: Large-Scale Discovery and Extraction of Product Specifications on the Web

Disheng Qiu
Università degli Studi Roma Tre
disheng@dia.uniroma3.it

Luciano Barbosa
IBM Research - Brazil
lucianoa@br.ibm.com

Xin Luna Dong
Google Inc.
lunadong@google.com

Yanyan Shen
National University of Singapore
shenyanyan@comp.nus.edu.sg

Divesh Srivastava
AT&T Labs – Research
divesh@research.att.com

ABSTRACT

The web is a rich resource of structured data. There has been an increasing interest in using web structured data for many applications such as data integration, web search and question answering. In this paper, we present DEXTER, a system to find product sites on the web, and detect and extract product specifications from them. Since product specifications exist in multiple product sites, our focused crawler relies on search queries and backlinks to discover product sites. To perform the detection, and handle the high diversity of specifications in terms of content, size and format, our system uses supervised learning to classify HTML fragments (e.g., tables and lists) present in web pages as specifications or not. To perform large-scale extraction of the attribute-value pairs from the HTML fragments identified by the specification detector, DEXTER adopts two lightweight strategies: a domain-independent and unsupervised wrapper method, which relies on the observation that these HTML fragments have very similar structure; and a combination of this strategy with a previous approach, which infers extraction patterns by annotations generated by automatic but noisy annotators. The results show that our crawler strategy to locate product specification pages is effective: (1) it discovered $1.46M$ product specification pages from $3,005$ sites and 9 different categories; (2) the specification detector obtains high values of F-measure (close to 0.9) over a heterogeneous set of product specifications; and (3) our efficient wrapper methods for attribute-value extraction get very high values of precision (0.92) and recall (0.95) and obtain better results than a state-of-the-art, supervised rule-based wrapper.

1. INTRODUCTION

The big data era is the consequence of two emerging trends: first, our ability to create, collect and integrate digital data at an unprecedented scale, and second, our desire to extract value from this data to make data-driven decisions. A significant source of this data is the web, where the amount of useful structured information has been growing at a dramatic pace in recent years.

A recent study [10] showed that *specifications* (set of attribute-value pairs for a single entity) are among the most common forms of structured data available on the web, e.g., infoboxes in wikipedia pages. Specifications are widely available for people (e.g., entertainers, politicians), products (e.g., cameras, computers), organizations (e.g., restaurants, hospitals), etc. Figure 1 shows example product specifications from different websites. Specifications are typically represented as HTML tables or lists, though only a small fraction of these tables and lists are specifications. There has been considerable interest in using specifications that can be collected from web data sources for a variety of applications, such as data integration, faceted search and question answering [4].

In this paper, we present a scalable focused-crawling technique to obtain specifications in a domain of interest. Domains are represented by categories of products (e.g., cameras, computers). We focus this study on product specifications for three reasons. First, there is much interest in using product specifications for comparison shopping, faceted search, etc. [21, 25]. Second, product specifications are available from a large variety of web data sources (e.g., e-commerce retailers, local stores). Third, despite their availability, efficiently obtaining a large set of high-quality product specifications in a given category comes with many challenges.

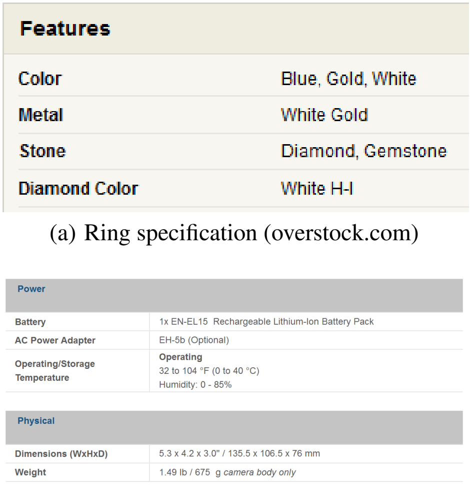
Efficiency: The number of data sources (websites) for any given product category can be in the thousands, but these sources are spread out on the web. Further, only a small fraction of pages in a relevant website have product specifications. This problem of sparsity makes it challenging to *efficiently* obtain a large set of product specifications without exploring, unproductive regions of the web.

Quality: There is considerable variety among product specifications in terms of their attributes, sizes, format and content. Further, even though product specifications are usually represented using HTML tables and lists, only a small fraction of tables and lists are specifications. This problem of identifiability makes it challenging to obtain a large set of *high-quality* product specifications.

A possible strategy to collect product specifications from the web is to run a general crawler, and then extract the data from the gathered pages. This approach has been investigated in the literature (see, e.g., [13]), and has its limitations: very few groups have access to up-to-date web crawls, and initiating such crawls is extremely resource intensive. An alternative is to use a publicly-accessible snapshot of the web, e.g., Common Crawl. We examined this possibility and verified that on Common Crawl many pages are out-of-date, some large sites are crawled only to a shallow depth (thus missing many product specifications present deep in the site), and some small websites are not even indexed.

A second strategy to obtain many product specifications is to

Proceedings of the VLDB Endowment, Vol. 8, No. 13
Copyright 2015 VLDB Endowment 2150-8097/15/09.



Features	
Color	Blue, Gold, White
Metal	White Gold
Stone	Diamond, Gemstone
Diamond Color	White H-I

(a) Ring specification (overstock.com)

Power	
Battery	1x EN-EL15 Rechargeable Lithium-Ion Battery Pack
AC Power Adapter	EH-5b (Optional)
Operating/Storage Temperature	**Operating** 32 to 104 °F (0 to 40 °C) Humidity: 0 - 85%
Physical	
Dimensions (WxHxD)	5.3 x 4.2 x 3.0" / 135.5 x 106.5 x 76 mm
Weight	1.49 lb / 675 g *camera body only*

(b) Camera specification (bhphotovideo.com)

Figure 1: Examples of specifications of different products.

use existing product aggregator sites, such as Google Products.[1] While Google Products does contain many product specifications, it has its own limitations. First, Google Products adopts a pay-to-play model, where merchants have to pay a subscription fee to have their products be listed. This limits the overall coverage in terms of available products, as we will show empirically. Second, Google Products can be used to obtain specifications for (many, but not all) known products ids, but cannot be used to discover new products since their catalog is not crawlable.

To effectively address the challenges of efficiency and quality, we propose an end-to-end system, DEXTER, that consistently uses the principles of *vote-filter-iterate*. From a small set of seed web pages from large, popular web sites and product specifications in a given category, DEXTER iteratively obtains a large set of product specifications. Specifically, in each iteration, DEXTER uses voting and filtering to prune potentially irrelevant websites and web pages in a site, reduce the noise introduced in the pipeline, and efficiently obtain a large number of high-quality product specifications.

In this paper, we make the following contributions: (i) an original approach to efficiently discover websites that contain product specifications; (ii) an adaptation of an existing in-site crawling approach [20] designed for forums to work for product websites; (iii) an effective technique to find and extract attribute-value pairs from product specifications; (iv) an end-to-end system to efficiently and accurately build a big collection of product specifications. This collection consists of specifications from instances of a given product from each visited website. It is important to point out that DEXTER does not perform any semantic integration of product specifications across websites. However, the product specifications collected by DEXTER can certainly be used as input to data integration studies. For this purpose, we make the product specifications collected by DEXTER publicly available.[2]

We have performed an extensive experimental evaluation with nine different product categories on the web. Our results show that (1) our website discovery and in-site crawling strategies efficiently identified 3,005 websites and 1.46M HTML pages that contain product specification pages in the nine product categories; (2) our specification detector obtains a high value of F-measure (close to 0.9) over a large variety of product specifications; and (3) our wrapper (attribute-value extractor) gets very high values of precision (0.92) and recall (0.95).

Our solution discovered an order of magnitude more sources than the website discovery techniques of [2, 16]. We also compared our extracted dataset with Common Crawl and two product aggregator sites: Google Products and Semantics3.[3] The results show that: (1) Common Crawl only covers 32% of the product sites discovered by DEXTER and (2) Google Products only covers 61% of DEXTER's products and Semantics3 only 41%.

The rest of this paper is organized as follows. Section 2 defines our problem and presents an overview of DEXTER. In Section 3, we introduce our strategies for website discovery and in-site crawling to locate product specification pages. Section 4 describes the approach we used in generic specification detection, and presents our approach to extract attribute-value pairs. In Section 5, we present our extensive experimental evaluation results. Related work is discussed in Section 6, and we summarize in Section 7.

2. PROBLEM AND SOLUTION OVERVIEW

We define a product specification as follows.

Definition 1: [Product Specification] A product specification SP is a set of attribute-value pairs $< a_i, v_i >$. ■

The goal of this work is to build a big collection of product specifications. More formally, we state our problem as follows:

Definition 2: [Problem Definition] Given seed products P and product websites S in a specific category C, we aim to efficiently crawl a comprehensive set of product specifications in C. ■

To deal with this problem, we propose DEXTER. Initially, from seed products chosen from large, popular product websites, DEXTER locates new product websites, which contain specifications, such as shopping and company websites (Website Discovery). Second, DEXTER crawls those websites to collect product specification pages (In-site Crawling). From those pages, DEXTER detects the HTML portion of the pages that contains the specification (Specification Detection) and, finally extracts the attribute-value pairs from the specification (Specification Extraction). At the end of this pipeline, DEXTER produces for each visited site a set of product instances with their respective attributes and associated values. Figure 2 shows the architecture of DEXTER. In the rest of this section we provide an overview of each step and the main challenges.

Website Discovery: The goal of Website Discovery is to locate candidate product websites. Since these are sparsely distributed on the web, DEXTER uses two strategies to deal with that: (i) it queries a search engine with known product ids, and (ii) it identifies hubs with links to known product websites. Voting is used to generate a ranking of the candidate websites to select websites for further exploration. Since some of the selected websites might not be relevant, we built a product website classifier to quickly filter out irrelevant websites. Iteration is subsequently used to obtain a large number of relevant websites. (Section 3.1)

In-site Crawling: Within a potentially relevant product website, we aim to efficiently locate the product specification pages. For that, we use a number of classifiers to discover product category entry pages and index pages, filtering out web pages on the website that are unlikely to lead to product specification pages. Voting is used to score the links from product specification pages in a website, discovered using a search engine, to aggregate the classifier scores for identifying promising category entry pages. (Section 3.2)

Specification Detection: Once product specification pages are identified, the goal is to detect the HTML fragments on those pages that correspond to specifications. For that, we develop a product

[1] http://www.google.com/shopping
[2] http://github.com/disheng/dexter
[3] http://www.semantics3.com

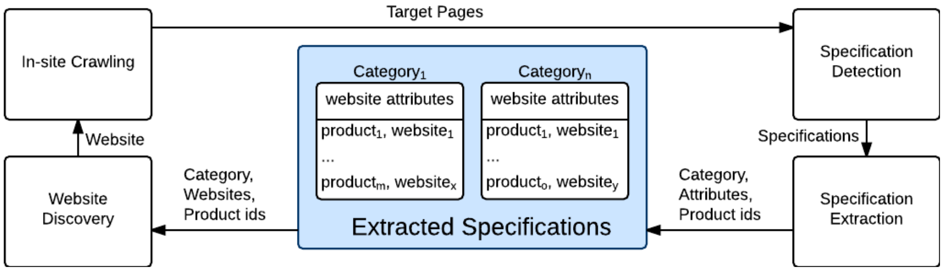


Figure 2: Architecture of DEXTER, with Sites Discovery, In-site Crawling, Specification Detection, and Specification Extraction.

category detector, to avoid labeling data for each considered category. The specification detection classifier looks at HTML fragments (e.g., tables and lists) in these pages, and makes use of various structural features such as the number of links, number of items, text size, etc., that distinguish high-quality specifications from non-specifications. (Section 4.1)

Specification Extraction: The final step in the DEXTER pipeline is to extract attribute-value pairs in the detected specifications. Since a large number of specifications can be detected, it is important for attribute-value pairs to be extracted efficiently. For that, we implemented two strategies: (1) a heuristic lightweight approach that takes into consideration HTML structural commonalities between specifications across sites, and (2) a hybrid method that combines the approach of [12] of inferring extraction patterns based on noisy annotations with our heuristic approach. (Section 4.2)

3. DISCOVERY AND CRAWLING

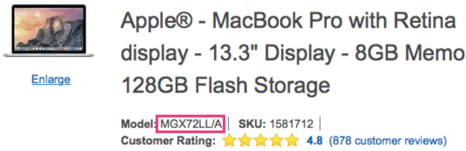
In this section, we describe the first part of our pipeline in more detail: the discovery of product websites and the crawling of the product specification pages within the discovered sites.

3.1 Website Discovery

The first task of DEXTER is to find websites with product pages. Since these websites are sparsely distributed on the web, the main challenges are: to efficiently find such websites while avoiding visiting unproductive regions on the web, and to discover a comprehensive catalog of them with a high quality. To achieve these goals, we implemented four different strategies: (1) Search: the crawler issues queries based on known products to a search engine in order to discover other websites that publish information about these products; (2) Backlink: from known relevant sites, the crawler explores their backlinks to find other relevant sites; (3-4) Merge: the system defines two ranking strategies based on a combination of the discovered websites adopting Union and Intersection. We give further details about these strategies in the rest of this section.

3.1.1 Search

In our domain, we expect that multiple websites publish specifications of the same product. Taking advantage of this high redundancy, searching for known products on a search engine would return pages of these products in different sites. Note that the search engine can return non-product-specification websites, e.g., a forum with a discussion of a product. To efficiently discover relevant websites without penalizing the overall recall, DEXTER searches for multiple products and provides an ordered ranking of the returned websites from the search engine considering the number of times a website is present in the results, i.e., a forum or other non specification websites are less likely to have pages in the results for many products. Based on the ranking, in each iteration, DEXTER selects the top K websites.



Apple® - MacBook Pro with Retina display - 13.3" Display - 8GB Memory - 128GB Flash Storage

Enlarge

Model: MGX72LL/A | SKU: 1581712 |

Customer Rating: ★★★★★ 4.8 (878 customer reviews)

Figure 3: A product detail page with the product id.

The idea of using search engines to help collect pages has been previously explored [2, 16, 13]. For instance, [13] analyzed the distribution of entities for a set of domains by searching for entities on the web using some unique identifiers, e.g., restaurant phone numbers. Similarly, we extract ids of given products to discover new product sites that contain the products. In Figure 3 we show an example extracted product id used for the search step.

Our method works as follows. Given a set of product ids $\mathcal{K}$ obtained from seed websites S, the crawler uses a search engine API to discover new websites S' that publish information for products in $\mathcal{K}$. More specifically, S' is built considering all the websites returned by the search engine over all queries in $\mathcal{K}$. S' is then ranked considering the score according to the following equation:

$$sSearch(\mathcal{K}, s_j) = \frac{\sum\limits_{k_i \in \mathcal{K}} search(k_i, s_j)}{|\mathcal{K}|} \tag{1}$$

$$search(k_i, s_j) = \begin{cases} 1 & \text{if } s_j \text{ returned searching for } k_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Using the function $sSearch$ for all the websites $s_j \in S'$, we generate an ordered ranking R'. Based on the ranked list of websites we can adopt a threshold to select the top websites and consider them for further steps in the pipeline.

The main limitations of the search approach are the restrictions usually imposed by search engine APIs such as number of results per query, number of results per user and total number of queries in an interval of time. This is a significant issue if the goal is to collect an unbounded number of product websites. Our next approach tries to deal with this problem of scalability since it is less dependent on search engine restrictions.

3.1.2 Backlink

A useful source for finding relevant websites are pages that point to multiple sites, so-called hub pages. As an example, previous work [1] uses backlinks of multilingual websites to find hub pages that point to other multilingual sites, and then restricts the crawler to the web region defined by the bipartite graph composed of the pages pointed by the backlinks of relevant sites and the outlinks of these hub pages. A backlink is just a reverse link so that from the initial website we can discover hubs that point to it.

Efficiency, recall and quality are challenging goals. In fact, backlinks can lead to non-relevant hubs, subsequently leading to non-relevant websites. Common examples are generic hubs that point to multiple websites like popular websites in a country, websites where the name of the domain starts with an 'A' and so on.

We adopted a *vote-filter-iterate* strategy to locate product websites and to address our challenges. Non-relevant hubs are less likely to point to many relevant websites, while relevant websites are more likely to be pointed by many relevant hubs. Based on this intuition using backlinks, we score hubs that point to many relevant websites, and we compute an ordered ranking of the new websites pointed by the hubs. As with search, we generate an ordered ranking and for each iteration we select the top K websites.

The approach works as follows. Given the initial set of websites S, we want to discover a new set of websites S'' and an ordered ranking R'' of S'' so that S'' are all pointed by hubs discovered from S. To provide a ranking and prune the enormous number of websites and hubs discovered by backlinks we first search for the hubs H using backlinks for each website $s_i \in S$. Each hub $h_j \in H$ is scored considering the following formulation:

$$sHub(S, h_j) = \frac{\sum\limits_{s_i \in S} hub(s_i, h_j)}{|S|} \tag{3}$$

$$hub(s_i, h_j) = \begin{cases} 1 & \text{if } h_j \text{ in backlink for } s_i \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

To improve the performance, we prune those hubs that are pointed only by a single website. We expect that if S is big enough, hubs discovered only by one website are not likely to lead to interesting websites. From the hubs H, we follow the forward links to discover new websites S''; for each new website s_j we score the website as the weighted average of all the hubs that point to s_j:

$$sForward(H, s_j) = \frac{\sum\limits_{h_i \in H} forward(h_i, s_j) * sHub(S, h_i)}{\sum\limits_{h \in H} sHub(S, h_j)} \tag{5}$$

$$forward(h_i, s_j) = \begin{cases} 1 & \text{if } s_j \text{ in forward links of } h_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Using the function $sForward$ for all the websites $s_j \in S''$ we generate a second ordered ranking R''. We adopt the same threshold of the search approach to select the top K websites for further processing in the pipeline.

3.1.3 Merge

An interesting observation is that the backlink and the search are two independent approaches and they can be combined in many ways to define a new ranking algorithm.

In this work, we consider two merging algorithms: the first one based on the union and the second based on the intersection of the two generated rankings. The intuition behind the union is that if the two rankings generate two disjoint but good ordered sets, considering the union of the top results from each ranking can provide better results. For the intersection we expect that if there is an overlap among the results from the two distinct rankings, a good website is likely to be in the overlap and ranked better by the new ordering.

Given two rankings R' and R'', each value in R^x is made of a pair (s_i, p_i^x) where $s_i \in S' \cup S''$ and p_i^x is the position assigned to s_i by the ranking R^x. We score a site s_i for a ranking R^x considering the following formulation:

$$rScore(s_i, R^x) = \frac{1}{|R^x|}||R^x| + 1 - p_i^x|$$

i.e, we score a source s_i for a ranking R^x considering the position of the source inside the ranking.

We define the score of the union as follow:

$$sUnion(s_i) = max(rScore(s_i, R'), rScore(s_i, R''))$$

From this function, we generate a new ranking R_{union} so that the score of the website s_i is defined by $sUnion$.

We define the score of the intersection as the harmonic mean among the rankings as follow:

$$sIntersection(s_i) = 2 * \frac{rScore(s_i, R') * rScore(s_i, R'')}{rScore(s_i, R') + rScore(s_i, R'')}$$

From $sIntersection$ we generate ranking $R_{intersection}$.

Note that we do not merge the rankings considering the scores provided by the source discovery strategies. An issue that we observed is that often the scores are not comparable because one ranking often leads to scores close to 1 while the other has lower scores. A score-based merging could penalize one of the rankings.

Since there can still be many non-relevant sites returned by the voting and iteration steps, the final step in the Website Discovery is to efficiently detect product websites. It is based on a classifier trained to recognize if a website is a product website considering the features in the root home page (Home Page Classifier). More specifically, we trained the classifier to recognize websites that publish product specifications. We trained the classifier with the anchor text of the links in the home page.[4] The words inside the anchor text are good features because the home page provides links to multiple categories, and links that lead to the same category are likely to be the same, e.g., for TV common anchor texts are TV, Television, Home, etc. While we expect to find many specifications from shopping websites, in our evaluation we discovered several company websites as well including asus.com, dell.com, lg.com, samsung.com, usa.canon.com.

3.2 In-site Crawling

Having discovered a new product website, the next step in the pipeline is to crawl the website and discover product specification pages. To avoid visiting unproductive regions of a website, it is important to have a strategy that collects as many product pages as possible, visiting as few non-product-pages as possible.

DEXTER's In-site Crawling is inspired by [20], which focused on forum crawling. We use a similar approach to crawl generic product websites. The main assumption is that, similar to forum sites, product-based sites have a well-defined structure consisting of an entry page to the product content in a given category, index pages that point to product pages and, finally, the product pages themselves. Based on that, we implemented the following strategy, depicted in Figure 4. First, the In-site Crawling discovers the entry page related to a category (Entry Page Discovery). Normally, product websites organize their catalogs in categories and subcategories. This is expected because the website administrator wants the customer to have a good navigation experience inside the website. Next, the crawler performs the Index Page Detection. The category entry page leads to index pages or nested index pages. An index page is a structured page inside the website that lets the customer search, filter and select the product that she wants to purchase. The Index Navigation Detector discovers the pagination structure inside an index or a nested index and finally, a classifier is trained to detect pages of a given product. The category that we assign to the crawled pages is inferred considering multiple steps: the

[4] We use 50 relevant product websites and 50 non-relevant websites as training data for all the categories.

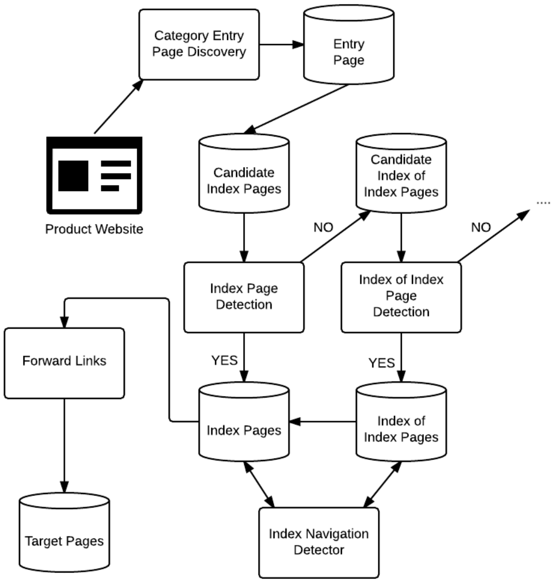


Figure 4: The pipeline to crawl a new website for target pages.

keywords used to find the website are category specific, the entry page is category specific and the features that we adopt to recognize the target pages are category specific. The category is automatically assigned to the target pages based on the category of the crawling step. We next provide details of the Entry Page Discovery and the Index Page Detection.

3.2.1 Entry Page Discovery

To discover the entry page of a given category, we defined three distinct strategies:

- **From the home page**: the first approach starts from the site's home page and uses the Entry Page Classifier to detect links to the entry page of a given category. For that, it uses as features words in the anchor texts. In this strategy we crawl the website from the home page and we score candidate entry pages as the product of the score returned by the Entry Page Classifier on the anchor text of each crawled link.
- **From the target pages**: The second approach follows the intuition that the product page often has references to the product's category. We use the confidence score from the Entry Page Classifier to score the links on the given page. This is repeated for every candidate target page and the final score is the average score obtained from each page.
- **By search**: The last approach uses a search engine to directly find the entry page. We search within the website using the category name as query terms. We score the candidate results by considering the ranking of the search engine.

The three strategies return independent scores, and DEXTER considers the webpage that gets the best score from all three scores using a harmonic mean formulation.

3.2.2 Index Page Detection

To recognize generic index pages, we make the assumption that the anchor text that points to product pages has some regularity. Under this assumption, we trained several classifiers, one for each category, using as features the words in the anchor text of the link that points to the target pages. Since index pages usually contain groups of links to product pages, to improve classification accuracy, we group links and then score the groups as the average score of the links that compose the group. To group links, we make the observation that regions in the index pages that point to target pages have the same layout and presentation properties. This motivates the usage of link collections [9], a group of links grouped by an extraction rule (XPath), like lists. The same technique is applied to recognize a nested index structure. We recognize a nested index that points to multiple index pages by scoring each page as an index page. We empirically observed that a complex nested structure is rare and that most of the cases are managed by a two level structure. We say that a page is an index page if the average score is greater than a given threshold that we set empirically.

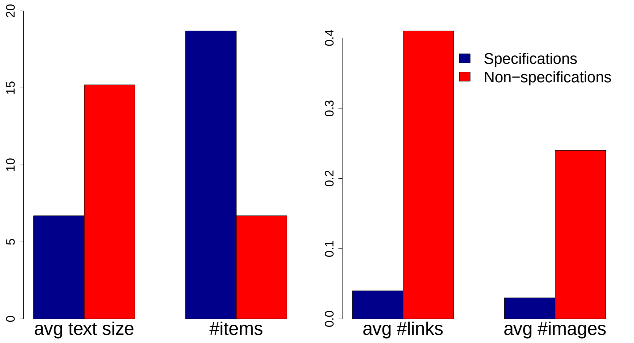


Figure 5: Avg. text length, avg. number of items, links, images for specifications and non-specification.

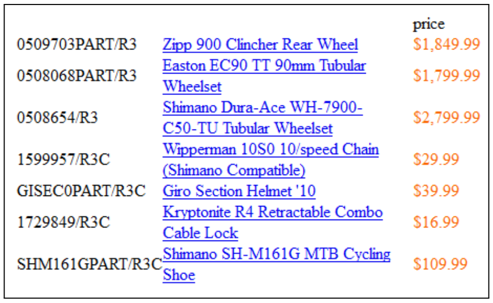
4. DETECTION AND EXTRACTION

In this section, we describe the second part of our pipeline. Given the product specification pages discovered by the previous steps, our goal is to automatically extract the attribute-value pairs and the product keys required by the search. For the specifications, we split the problem in two separate problems: the detection of the specifications inside product pages and the extraction of the attribute-value pairs given the specification.

4.1 Specification Detection

The process of automatically extracting specifications in many websites for different categories is a challenging task. For each website, the template based HTML pages are characterized by some local variability. This variability is related to the script used to generate them. The script is site specific, thus to accurately extract the data, we have to generate a specific wrapper for each website. A possible way to address this issue is to recognize the variability in a website by using domain knowledge to recognize a specific category, like in [18]. DEXTER adopts a different approach, which does not require any domain knowledge and is not specific to few categories. The detection addresses the local variety inside the websites, recognizing product specifications with a machine learning based solution. The extraction is then applied to a regular portion of the HTML page, thus simplifying the extraction task.

Previous approaches have been proposed to detect tables/lists that consist of structured data [19, 28]. More specifically, Wang and Hu [28] use machine learning algorithms to detect tables with relational information, and Gupta and Sarawagi [19] proposed some heuristics to remove useless and navigational lists. Similar to [28], we also use machine learning techniques but our task is not to detect relational tables on the web, but to detect specifications.



		price
0509703PART/R3	Zipp 900 Clincher Rear Wheel	$1,849.99
0508068PART/R3	Easton EC90 TT 90mm Tubular Wheelset	$1,799.99
0508654/R3	Shimano Dura-Ace WH-7900-C50-TU Tubular Wheelset	$2,799.99
1599957/R3C	Wipperman 10S0 10/speed Chain (Shimano Compatible)	$29.99
GISEC0PART/R3C	Giro Section Helmet '10	$39.99
1729849/R3C	Kryptonite R4 Retractable Combo Cable Lock	$16.99
SHM161GPART/R3C	Shimano SH-M161G MTB Cycling Shoe	$109.99

Figure 6: Example of non-specifications.

Specifications can be contained in different HTML structures. But they are primarily found within tables and lists (ul tag). We empirically verified that by manually inspecting 301 specifications from a great variety of products. Among them, 62% were inside tables whereas 31% were inside lists. The remaining 7% were in miscellaneous structures (e.g., HTML div tag). By also annotating tables and lists that are not specifications (304 instances), we observed that some structural characteristics of specifications can be useful as a good indicator of whether a table or list is a specification or not, independent of product or site. To illustrate that, we present numbers in Figure 5 from our sample set containing both specifications and non-specifications. As one can see from these numbers, specifications contain far fewer links and images than non-specifications, and more items and smaller texts. Figure 6 presents a concrete example of a table that does not contain product specifications. As one can see, it has many links and larger text size compared with the specifications presented in Figure 1. Other features that we use to differentiate specifications from non-specifications include average node depth of the items and its standard deviation, standard deviation of the text size, overall frequency of the word "specification", HTML type and average number of the pattern of two upper cases followed by a number (e.g., "Weight Max 40 pounds").

As we present in Section 5, DEXTER not only can effectively detect specifications, but it is also able to pinpoint them on the page because it classifies not the entire page but the tables or lists which contain specifications. As we show in Section 4.2, this is very helpful for the extraction of the specification's set of attribute-value pairs from these HTML fragments.

4.2 Specification Extraction

The final tasks of DEXTER are: the extraction of the attribute-value pairs from the HTML fragments provided by the generic specification detector (see Figure 2) and the extraction of the product ids $\mathcal{K}$ to feed as keywords to the search engine.

According to [17], designing an automated procedure to extract web data remains challenging due to its large volume and variety. To achieve very accurate performance, existing tools [22, 26, 27] always ask human experts to design the extraction pattern or prepare labeled data for training, which are labor-intensive. Other platforms such as RoadRunner [8] avoid human engagement by learning patterns from unlabeled examples automatically. However, they suffer in poor resilience by trading off performance against the power of automation. In our context, we can do better because the data to extract is no longer the raw data on the web, but some "clean" HTML fragments supplied by our generic specification detector. We adopted other techniques [8] for our task but we obtained poor performance. Therefore, instead of using an existing automatic wrapper generation system, we implemented two different wrapper strategies: the first completely unsupervised that exploits the regularity of the specification structure and the second that uses automatic annotations to generate the wrapper.

The first strategy is based on the observation that the structure of these fragments containing the specifications are very homogeneous. By inspecting these tables and lists, we came up with the following heuristic for the extraction. For HTML tables, we first assume each attribute-value pair of the specification is contained in a table row tag (tr). Subsequently, we parse the DOM subtree, in which tr is the parent node, and extract the text nodes in this subtree. The first text node is considered as attribute and the remaining ones as concatenated values. With respect to HTML lists (ul), we consider that each item in the list (li) contains an attribute-value pair, and the token that separates the attribute from the value is the colon character. In addition to its simplicity, this wrapper is domain-independent and does not require any training. We show in Section 5 that it obtains very high values of recall and precision over a set of heterogeneous sites with specifications.

The second strategy is based on the technique proposed in [12]. Dalvi et al. defined an approach for inferring extraction patterns by annotations generated by automatic but noisy annotators. For our purpose, we train two simple annotators considering the attribute-values pairs from the extracted websites. The first annotator annotates nodes in an HTML fragment if there is a perfect match between the string contained in the fragment with one of the values in the training, the second annotator is similar to the previous one but it is trained to annotate only attribute names. Notice that our implementation infers two extraction rules, one for all the values and the other for all the attribute names published in the specifications. A straightforward implementation would be to infer an extraction rule for each attribute in the specifications. As observed by [23], the attributes published by multiple websites of the same domain are skewed, in fact 86% of the attributes are published only by a small percentage of the sources. Adopting a per attribute inference would lead to successfully extracting only overlapping attributes that are just a fraction of the total number of published attributes.

Notice that while the previous heuristic is completely domain independent, the technique based on annotators is domain dependent. [12] requires training related to a specific category and an a-priori distribution to improve the generation of the extraction rule.

4.2.1 Extraction of Product ids

A similar technique to the previous specification extraction by annotators is adopted to incrementally extract product ids. From an initial seed set of large, popular websites S, we manually define the precise extraction rules to extract product ids $\mathcal{K}$. From $\mathcal{K}$ we define an annotator that annotates nodes that publish information that match any keyword k_j in $\mathcal{K}$. For each new website the system infers a new extraction rule and extracts new product ids that are added to the initial seed set $\mathcal{K}$. Extraction rules are generated by generalizing XPath expressions over annotated values. The generation process follows the technique proposed by [12]. For the robustness of our wrapper, our extraction rules are defined following principles described in [11] such as rules that exploit invariants close to the target values. In Figure 3 the expression //*[text()="Model:"]/following-sibling::*[1]/text() is a possible extraction rule that exploits the label Model: as invariant inside the page to extract the product id. These products ids are then used for querying the search engine and discovering new product websites.

We observe that even with an accurate wrapper generation system, the noise, iteration after iteration, can affect the quality of the generated $\mathcal{K}$. To address this issue we follow the intuition of voting that ids extracted by multiple websites are more likely to be relevant ids. We provide a ranking of the ids and we set a threshold τ

cat.	# sites	# pages
camera	1,107	278k
cutlery	339	103k
headphone	475	142k
monitor	941	184k
notebook	589	272k
software	419	129k
sunglass	471	182k
toilets	82	36k
tv	841	132k
all	3,005	1.46M

Table 1: Number of sites and pages per category.

# cat.	% websites
1	69.9%
2	15.7%
3	6.8%
4	3.4%
5	2.1%
6	1.3%
7	0.4%
8	0.2%
9	0.3%

Table 2: Percentage of websites with multiple categories.

to select only the ids that are extracted from multiple websites.

Each id $k_i \in \mathcal{K}$ is scored following this formulation:

$$sKey(k_i, S) = \sum_{s_j \in S} key(k_i, s_j) \qquad (7)$$

The key function is a binary function that scores an id k_i with 1 if and only if k_i is extracted from s_j. Ids are considered for search only if $sKey(k_i, S) > \tau$.

5. EXPERIMENTAL EVALUATION

In this section, we initially assess the site discovery and crawling, then the specification detection and extraction, and we conclude the section with a summary of the evaluation.

5.1 Product Sites Discovery and Crawling

Data Collection and Description. We discovered and collected $1.46M$ pages from $3,005$ online product websites related to 9 categories (camera, cutlery, headphone, monitor, notebook, software, sunglass, toilet, tv). We considered 5 electronics catgories and 4 non-electronics categories. The corpus was collected running our pipeline with different configurations from an initial set of 12 large, popular product websites (abt, adorama, amazon, bestbuy, bhphotovideo, cdw, newegg, overstock, pcrichard, staples, target, tigerdirect). We manually wrote the wrappers to extract the product ids $\mathcal{K}$ and the set of attribute-value pairs from the specifications for the initial seed set. We ran the pipeline several times with different configurations: we considered all ranking strategies (backlinks only, search only, union and intersection) and different setup of I, K and S. We used the Bing Search API to search new product websites with a limit of 250 results for each query and a public API[5] to find backlinks from known relevant websites, here also with a limit of 250 backlinks per website.

Overall, we discovered $193,169$ websites. Over those sites, we ran our home page classifier to detect product sites, resulting in $23,877$ detected sites. We then crawled these sites and manually checked the candidate target pages of $3,005$ sites, which were used as a golden set G for evaluating the site discovery strategies.

For each website on average we have collected 484 product pages for a total of $1.46M$ pages. The biggest websites contain more than $40k$ pages while the smallest websites have 5 product pages. We show in Table 1 the number of sites per category and in Table 2 the percentage of sites with the number of categories. Only 0.3% of the websites cover all the categories, while the majority of 69.9% are related to a single category. Camera is the most frequent category with $1,107$ websites while toilet is the least frequent with 82 websites. The number of pages is very uneven across sites, in fact, popular websites provide a huge catalog with many products. The $3,005$ websites are collected from $5,264$ websites/categories.

Manual Effort and Tuning. The manual effort required to trigger the complete pipeline is limited to: the seed set and the classifiers (Home Page, Entry Page, Index Page and Specification Detection). For the initial seed set of 12 websites, we manually wrote wrappers and crawlers to collect the pages and extract specifications for a given category. For each category in those websites we also found manually the Entry Page related to the category and the Index Pages that point to target pages. For the Discovery of new websites we set the minimal redundancy of $\mathcal{K}$ to 4 and we trained the Home Page classifier with a manually labeled list of 50 relevant and 50 non-relevant websites. For the In-site Crawling we trained two Classifiers for the discovering of the Index Pages (IP) and Entry Pages (EP). We trained IP considering the anchor text that pointed to the target pages, and EP using the anchor text that pointed to the entry pages from the root of the website. The training of IP and EP were limited to the initial seed set. For the specification extraction we trained a classifier based on the features from 37 websites in G.

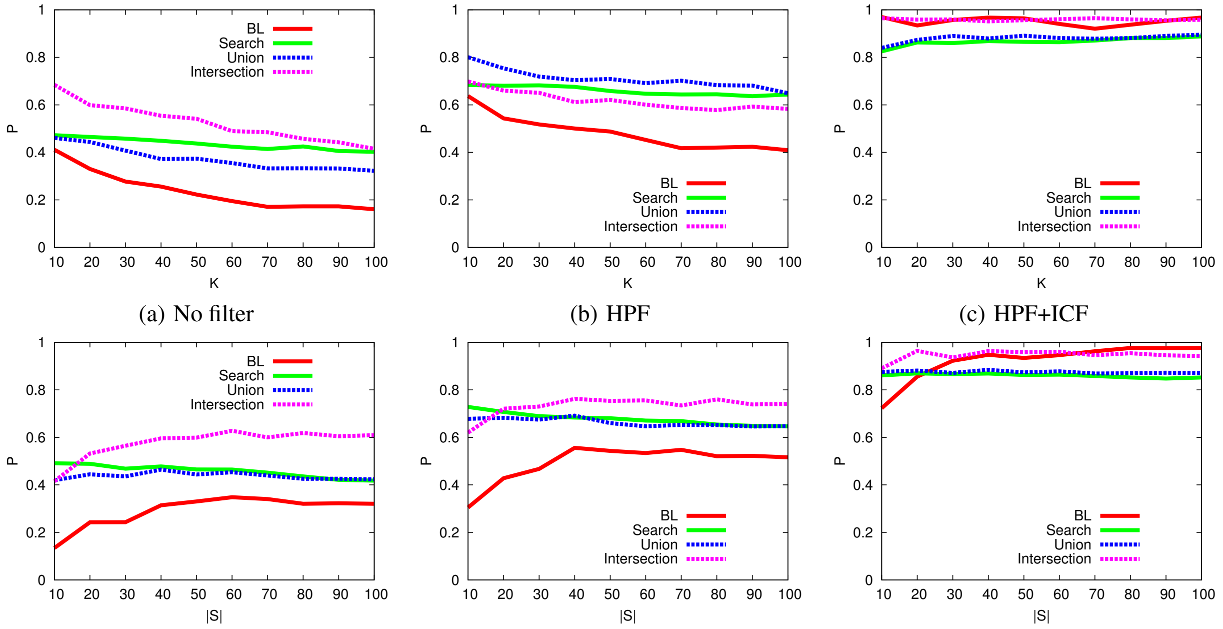
Strategies. To assess the discovery of new product websites, we evaluated the ranking strategies considering different parameters: the top websites returned from the ranking K, the size of the initial seed set $|S|$, the quality control filters and the number of iterations I for which the pipeline is executed. The quality control filters are based on the home page classifier, Home Page Filter (HPF), which discards evaluation websites that are not recognize by the classifier trained on the home pages of online product websites, and on the In-site Crawling Filter (ICF), which discards websites where the crawling returned no product specification pages.

For this evaluation, we take an initial seed set of websites S from our golden set G, from which our system extracts $\mathcal{K}$ and computes different rankings of new websites based on different ranking strategies. It then retrieves the top K sites in the ranking and passes them to the filters. The websites that successfully pass the filters are then considered relevant sites and are added to the initial seed set S. We score the precision considering the intersection between the updated S with our golden set G.

Ranking Results. In Figures 7(a) and 7(d), we evaluate the ranking strategies when we increase the top K (with a fixed $S = 50$) and the size of the seed set S (with a fixed $K = 20$). For this experiment, $I = 1$ (number of iterations) and no filter is applied to increase the precision. Overall, increasing K reduces the precision of all the ranking strategies: the Search Only strategy is slightly more resilient. Intersection achieves the best quality obtaining almost a 0.68 in precision when $K = 10$; the precision drops to 0.40 when $K = 100$. Increasing the seed set improves the quality of the ranking strategies: also in this case intersection achieves the best scores, precision goes from 0.41 to 0.61. The quality of the search-only strategy is not affected by the size of seed set. Explanations are (1) we have to consider also the number of product ids we used to search new sites (the average number of pages per site is 484) (2) we expect that the top product websites (amazon, bestbuy etc) are easily ranked even with few product ids. From Figure 7(a), we can also observe that the quality of the combined ranking strategies is affected by the quality of the single ranking strategies: the quality of Intersection drops because the quality of the Backlinks Only drops, the quality of Union is lower than the quality of Search Only because generally Backlinks Only is lower than the Search Only. The loss in precision for the Backlinks Only ranking when the initial seed set is 10 is due to non-popular websites.

Filter Results. The previous ranking results were obtained without any filter. But before discussing how our filters affect the ranking results, we present an evaluation of the quality of our filters. The

[5] Link Metrics from `apiwiki.moz.com`



(a) No filter (b) HPF (c) HPF+ICF

(d) No filter (e) HPF (f) HPF+ICF

Figure 7: Precision of the ranking algorithms, $I = 1$: (a,b,c) fixed $|S| = 50$ and variable K (d,e,f) fixed $K = 20$ and a variable $|S|$.

biggest challenge in this evaluation is to feasibly make an estimation of the overall recall. To craft a golden set a manual effort is required but the number of relevant websites is just a small fraction of all the non-relevant websites (not conditioned by a ranking strategy), making the manual evaluation difficult. To design an evaluation to estimate the recall, we randomly collected $1,000$ websites from the list of all the websites that are discovered from all the combinations of our techniques. We then applied our HPF to discover 145 candidate websites, and our ICF, selecting 40 relevant websites among this smaller filtered set. On these 40 candidate results, we manually checked the quality and estimated the precision for HPF.ICF. To estimate the recall, we further execute the ICF for a random set of 145 websites chosen from the initial $1,000$ websites that are discarded by the HPF. For these websites the ICF returns only 20 websites and manually checking the quality only 5 were relevant. To complete the evaluation, we manually checked the quality of 50 websites from the ones discarded by HPF and ICF, and for those accepted by HPC and discarded by ICF (Table 3). From Table 3 we can estimate the overall P and R of the filter algorithms over a random set of 1000 websites in Table 4.

We observe from Table 4 that a combination of the two filters HPF and ICF leads to a reasonable solution with $P = 0.87$. The loss in recall is related to multiple factors: non-english websites, index pages with a small list of target pages, dynamic navigation inside the website, non-representative anchor text for links that lead to target pages or to the category entry page. Notice that the obtained results are from a randomly selected 1000 websites, thus the ratio of relevant websites to non-relevant websites is not even. Hence, in the next evaluation we consider the effect of the filters on the top K websites returned by the ranking strategies.

		HPF	
		yes	no
ICF	yes	35/40	5/20
	no	12.6/105	0/125

Table 3: # Relevant websites / # non-relevant websites, for the HPF and ICF.

	P	R
HPF	0.33	0.62
ICF	0.41	0.84
HPF.ICF	0.87	0.45

Table 4: Estimated P and R of HPF and ICF.

In Figures 7(b, c, e, f), we show the impact of our filtering techniques on the ranking strategies, when $I = 1$. In Figures 7(b) and 7(e) we can observe that all the ranking strategies are positively affected by the HPF. Overall we have a gain of 0.2 in precision. The strategy that achieves the best boost is Backlinks Only with a gain of 0.3 in precision, when we have a seed set between $70 - 90$ sites. This result is confirmed in Figures 7(c) and 7(f) where the HPF is combined with the ICF. The precision of Backlinks Only is higher than 0.9 with seed set higher than 60 sites, whereas Search Only achieves only 0.82 in precision. A plausible explanation is that if we search for new websites using product ids, we are likely to find websites that provide pages that publish some information about the products with these ids. But it is not guaranteed that these new websites also publish specifications, e.g., price comparator and review websites. Overall the HPF+ICF combined with the Intersection ranking achieves the best scores, obtaining a precision of 0.95 for K ranging from 10 to 90, and for a seed set greater than 20.

Iteration Results.

Figures 8 and 9 show results of our ranking algorithms running our pipeline for multiple iterations, with initial seed set to $|S| = 20$ and $K = 10$. One may expect that when running multiple iterations the quality of the obtained sites will drop, even as the number of obtained sites increases. However, in our evaluation (Figure 8), we can observe that after a slight initial drop in the first 15 iterations, the precision of all the algorithms is almost stable, between iterations 15 and 50. The precision of Intersection is around 0.95 while Search Only has the worst precision, around 0.92. This result supports our statement that a complete iterative pipeline with multiple filtering steps can be adopted to harvest product specifications from the web. If we consider the absolute number of relevant sites obtained from the ranking algorithms, Figure 9(a) shows that Intersection and Search Only are the best approaches, discovering overall 160 new websites after 50 iterations. Whereas, in Figure 9(a),

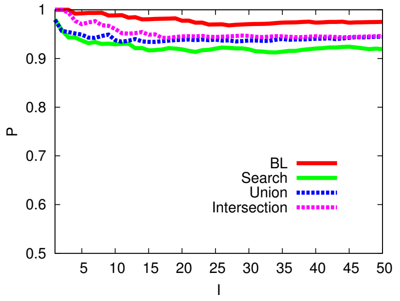
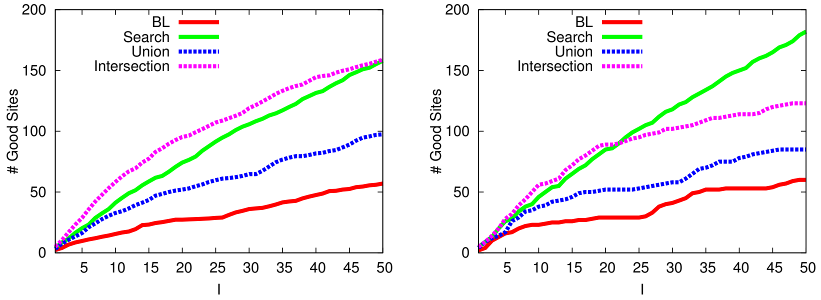


Figure 8: Precision with an increasing I, $S = 20$ and $K = 10$.



(a) All categories

(b) Notebook

Figure 9: Number of relevant websites with an increasing I, $S = 20$ and $K = 10$: (a) average on all domains (b) notebook.

we provide an average value across different categories, we observe that the growth in number of sites related to the ranking algorithm is strongly related to the searched category. Figure 9(a) shows also that on average category Intersection is better than Search Only for all the 50 iterations, while in Figure 9(b) for notebook category after 20 iterations Search Only obtains more relevant websites. This is due to: the quality of the hubs and the presence of a clear id attribute for the considered category.

To crawl the dataset, we considered as seed set well-known product websites. The rationale is that we expect that these websites are likely to be pointed by more pages on the web (more hubs) and to contain more product specification pages (more ids to search). This leads to more websites reachable from the initial seed set. We expect that our domain is characterized by a highly connected graph, as described by Dalvi et al. [13], thus almost all of the graph can be discovered with our iterative approach.

5.2 Specification Detection and Extraction

Data and setup. To evaluate our specification detection (SD) and specification extraction (SE) steps, we consider a subset of 37 websites from G (30 random websites and 7 well-known shopping websites). For each website, we manually crafted wrappers to extract the specifications and their attribute name/value pairs. To train the SD, we take at most 50 pages from each website and from each category (some websites have fewer pages). The positive examples are the specifications extracted by the wrapper and the negative examples are the tables/lists not considered relevant by the site wrapper. Since in this context, the number of negative examples is overwhelmingly higher than the positive ones, and this can affect the classification performance [29], we restricted the number of positive and negative examples to be the same.

We evaluate the SD and SE in two scenarios: 1) across sites and same category and 2) across sites and across categories. For the testing we adopt a *leave-one-outside* approach. More specifically: (1) we train a classifier for each website considering all the features from the other websites of the same category; and (2) we train the classifier considering also features from different categories.

For Specification Extraction, we consider two alternative solutions: one [12] based on wrapper inference from noisy annotations (WI) and one that follows the table structure heuristic (SE—SD).

Features	P	R	P (table)	R (table)
Our	0.84	0.90	0.88	0.92
Wang-Hu [28]	0.66	0.78	0.79	0.94
Combined	0.87	0.91	0.92	0.94

Table 5: Precision and recall for the Specification Detection.

Results. Table 5 compares precision and recall of a classifier trained with our features and the features defined in [6, 28] for specification detection. It also shows the average precision and recall obtained on our dataset. Our features are more robust to non-table specifications and have a better precision with a small loss in recall for only table specifications. Combining all the features from both approaches increases slightly the average precision by 0.03 and recall by 0.01, the increase is mostly for tables where the combination of our features achieves a 0.92 in precision and 0.94 in recall.

Table 6 shows the average precision and recall obtained by all approaches on the 37 sites and, for the sake of space, we present the individual results of only 10 of them, focusing our discussion on those cases where our approach (SE—SD) achieves the worst results. Table 6 compares our wrapper conditioned on a perfect SD's output (SE$|$SD*) i.e., we calculated the performance of the wrapper in isolation, the baseline WI, the quality of the annotations (An.), the wrapper inference conditioned on a perfect SD's output (WI—SD*) and a hybrid approach (WI+(SE—SD)) that chooses between WI and SE—SD when a quality check is passed.

Overall, our approach (SE—SD) obtained very high values of recall and precision over most of the sites (average precision equals to 0.80 and recall 0.90) and the results are comparable with WI with a loss in precision but a higher recall. The only exception was pcrichard, in which WI obtained a perfect score while our wrapper was not able to extract the correct results. We observed that pcrichard does not provide a table-like structure (it is the only site with a *dl* structure) leading to mistakes for SD and SE. For precision, SE—SD obtained poor results in some websites: newegg, alibaba and abt. The reason is that these websites provide in their specification pages other types of information that have similar structure to specifications or might also be considered as part of them. For instance, some of the lists misclassified by SD on newegg contained information of some product features, which were not presented in the specification of the golden set for this product. Regarding recall, the loss occurs in those websites with specifications consisting of small tables, as in Figure 1(b).

In addition, when one compares SE—SD vs SE$|$SD*, it is clear that SD is the main reason for the limitations of SE—SD. The number shows that SE performs an almost perfect job: average precision and recall equals to 0.95. The loss in quality is related to sites such as pcrichard, where no table structure is present, shop.lenovo and netplus, where the specifications' table rows sometimes consist of three columns, two dedicated to labels and one to the value.

In WI, the extraction performance is determined by the quality of the set of annotations. The annotator generally performs poorly with an average precision of 0.38 and an average recall of 0.39. For some websites WI achieves perfect scores. The loss in preci-

	SE—SD		SE|SD*		WI		An.		WI—SD*		An.—SD*		WI+SE—SD	
website	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec
shop.lenovo	0.81	0.87	0.87	0.87	1.00	0.87	0.24	0.22	1.00	0.87	0.24	0.22	0.81	0.87
uk.hardware	0.83	0.76	0.83	1.00	0.96	0.82	0.46	0.33	0.95	0.79	0.68	0.23	0.96	0.82
abt	0.46	1.00	1.00	1.00	0.99	1.00	0.30	0.57	1.00	1.00	1.00	0.48	0.99	1.00
alibaba	0.35	0.75	0.90	0.90	0.07	0.07	0.10	0.05	0.99	1.00	0.48	0.05	0.35	0.75
bhphotovideo	0.96	0.78	1.00	1.00	0.97	1.00	0.49	0.43	0.93	1.00	0.99	0.64	0.97	1.00
buzzillions	0.73	1.00	0.95	1.00	0.86	1.00	0.24	0.42	0.92	1.00	0.82	0.32	0.86	1.00
cyberguys	0.66	0.99	1.00	0.99	0.99	0.96	0.49	0.15	0.99	0.96	0.89	0.07	0.99	0.96
netplus	0.60	0.96	0.80	0.96	0.99	1.00	0.44	0.48	0.98	1.00	0.87	0.45	0.99	1.00
newegg	0.92	0.54	1.00	1.00	0.92	1.00	0.29	0.41	0.92	1.00	0.74	0.38	0.92	1.00
pcrichard	0.00	0.00	0.00	0.00	1.00	1.00	0.17	0.14	1.00	1.00	1.00	0.06	1.00	1.00
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
Average	**0.80**	**0.90**	**0.95**	**0.95**	**0.85**	**0.85**	**0.38**	**0.39**	**0.92**	**0.86**	**0.88**	**0.32**	**0.92**	**0.95**

Table 6: Results for our wrapper and the baselines on 10 websites (the most erroneous among the 37 sites).

	SD (websites)		SD (categories)	
	Prec	Rec	Prec	Rec
camera	0.88	0.92	0.94	0.96
cutlery	0.87	0.85	0.96	0.97
headphone	0.86	0.89	0.89	0.97
monitor	0.83	0.87	0.89	0.98
notebook	0.81	0.88	0.98	0.98
software	0.92	0.86	0.96	0.97
sunglass	0.67	0.56	0.97	1.00
toilets	0.43	0.80	0.93	0.98
tv	0.83	0.91	0.96	0.98

Table 7: Results for the SD, per site and per category.

sion and recall is strongly related to the quality of the annotators [12]. Another aspect that strongly affects the extraction error rate is the dependency of the annotations' mistakes. In fact, in [12] the authors adopted a random distribution to control and define an annotator of desired quality, in our setting we observed that there is a strong dependency on mistakes made by our annotator.

We observe that inferring the specifications from the right portion of the HTML page boosts both the annotator's precision and the extraction quality. WI—SD* obtains a precision of 0.92 and a recall of 0.86 compared to the previous 0.85 and 0.85 of WI. The most common mistakes are (1) the presence of short feature lists that are not specifications but with specification values and (2) the presence of specification values that are used as attribute names in other websites.

The first issue is addressed by WI—SD*, so that the annotation process is applied only on the portion of the HTML document with a specification. The second issue is addressed by taking into account both specifications values and attribute names during training: matches on values are positive annotations while matches on attribute names are negative annotations.

In WI+(SE—SD), we considered a hybrid approach. We observe that often WI and SE—SD make mistakes for different reasons and that it is possible to define a criterion to choose one approach over another. The intuition is that we can check the quality of the WI by comparing the attribute names and the values. As for WI, the system learns an extraction rule to extract the values of the specifications by using an annotator that matches the values in the test website with the values in the training websites. We adopt the same technique to learn an extraction rule to extract the attribute names of the test website, and compare the two extraction rules, the one for the values and the one for the attribute names. We observe that rules that extract values and attribute names for the specifications are likely to extract paired nodes. When it successfully learns two paired rules, it uses WI, or uses SE—SD if no paired rules are found. WI+(SE—SD) achieves really high precision and recall. The loss in quality is related to few websites, where WI fails and the heuristic for SE—SD is not perfect, e.g., alibaba.

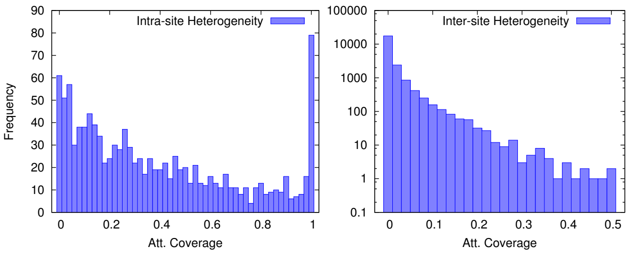
We observe that the diversity across websites is the main issue that affects the SD quality. This observation is confirmed by Table 7 where we compared the classification quality with a per categories basis. The SD (websites) has the same configuration as Table 6 but with the average score for each category. Here we observe the classification quality is not drastically affected by the variability across categories. The next question is: if a classifier is trained to recognize a set of categories, can it be used on other categories? The answer is in Table 7, in SD (categories) for each category, the training uses features from other categories and the testing on websites that contain pages related to the considered category and at least another category, i.e., the classifier has been trained to recognize another category for the same website. The quality is much higher: for notebook and tv, we have almost perfect precision and recall with a small drop only for headphone and monitor precision. This motivates our consideration that after discovering the specifications for some categories of a website, we can use the same classifier to recognize the specifications for new categories in the website.

5.3 Comparison and Statistics

Comparison with Product Aggregators and Common Crawl. We compare the dataset collected by DEXTER (D) with product datasets of two product aggregator sites: Google Products (GP) and Semantics3 (S3). These websites provide APIs that, given an id, return a product, and eventually its specification. In Table 8, we show a comparison: we randomly picked a set of 1000 ids across several categories and we searched for these ids in Google Product and Semantics3. We show the number of products of our dataset available in those sites and the average number of attributes per product. Google Products only covered 61% of DEXTER's products and Semantics3 only 41%. With regards to the average number of attributes per product, Google Products has the highest value (48), followed by Semantics3 (36) and Dexter (40). It is important to point out that DEXTER does not do any linkage across sites, which could considerably increase this number for DEXTER. Even without linkage, DEXTER has comparable performance with these two aggregators. This observation is confirmed in Table 8 where we compare the aggregators with DEXTER for three different categories. For Notebook and Headphone the number of attributes is comparable while DEXTER contains many products not present

Cat.	Sol.	% Prod.	# Attr.
all	S3	41%	35.9
	GP	61%	48.4
	D	100%	40.6
notebook	S3	44%	48.0
	GP	49%	53.4
	D	100%	53.2
headphone	S3	14%	18.0
	GP	43%	30.4
	D	100%	24.7
camera	S3	50%	47.2
	GP	61%	66.2
	D	100%	45.0

Table 8: Coverage comparison with aggregators.



(a) Intra-site (b) Inter-site

Figure 10: Statistical distribution of attributes.

in Google Products and Semantics3. Semantics3 contains fewer headphones and the number of attributes is sensibly lower. DEXTER extracts fewer attributes for Camera, but it still contains 39% of products not present in Google Products. DEXTER has a better coverage because it successfully collects specifications from non-mainstream products and sites. In fact, we expect aggregators to have a good coverage of well-known products while their coverage drops when we consider the long tail.

We also compare our dataset with pages crawled by Common Crawl: 68% of the sources discovered by DEXTER are not present in Common Crawl, only in 20% of the websites DEXTER discovered fewer pages, but product specification pages are just a small fraction of all the discovered pages. In fact, on a sample set of 12 websites where Common Crawl indexed more pages, 99.2% of their pages were non-specification pages.

Collection Statistics. To provide statistics of the collected attributes, we applied SD and SE to $700k$ pages from all the $3,005$ websites. We do not adopt any integration technique, thus we consider two attributes to be the same only if they have the same normalized string. We normalize attributes by converting the string to lowercase and removing all the non alpha-numeric characters. To reduce the presence of potential errors generated during the extraction process, we only consider attributes that occur in at least two product specifications from the same website and category, and published in at least two different websites.

DEXTER discovered on average $2.2k$ unique attributes per category; Camera has the highest number of attributes (5.5k) and Toilets has the smallest one (97). Figure 10 shows the distribution of the attributes with two histograms, one with the coverage of the attributes over the product specifications inside a website (intra-site heterogeneity) and the second across multiple websites (inter-site heterogeneity).

The histogram in Figure 10(a) shows the number of site-category pairs that has a given average attribute coverage. The coverage of an attribute is the fraction of the product specifications in the site-category pair that contain that attribute; the average coverage is computed over all the attributes discovered in the site-category pair. These numbers show that many websites are characterized by a high heterogeneity (coverage ¡ 20%), i.e. most attributes for a category in a website are present in few product specifications. However, many site-category pairs tend to have attributes that are present in many product specifications.

The histogram in Figure 10(b) shows the number of attribute-category pairs that have a given inter-site coverage. The inter-site coverage of an attribute for a category is the fraction of websites of that category that contain that attribute. The numbers (note the log scale on the Y axis) show that very few attributes are published by more than 20% of the websites, while the vast majority are specific to few websites. The highest coverage is only 50%. The reason for this high degree of heterogeneity is the presence of synonyms, i.e. different labels are used by different websites to describe the same attribute name. These results show an interesting starting point for integration efforts that should match synonymous labels and reduce the total number of attribute names.

5.4 Summary

Our results show that, with a limited human effort, DEXTER:

- Efficiently **discovered** and **crawled** 1.46M product specification pages from 3,005 websites for nine different product categories. Tables 1 and 2 present the collected dataset and Figures 7, 8 and 9 show that the *vote-filter-iterate* principles applied to our setting can accurately discover websites with product specifications with a high precision.

- Accurately **detected** product specifications. In Tables 5 and 7 our specification detector achieves on average $F = 0.87$ for specifications in unknown websites and known categories and F that goes from 0.94 to 0.98 for known websites and unknown categories.

- Accurately **extracted** attribute name/value pairs. Table 6 shows that a hybrid approach that combines a domain independent with a domain dependent approach achieves a 0.92 in precision and 0.95 in recall, close to settings where the detection is given as perfect.

6. RELATED WORK

Webtable. Many previous approaches try to explore the web to obtain structured data [6, 19, 28]. The WebTable project [6] extracts HTML tables which contain relational data, similar to [28], and applies techniques to search and explore these tables. Similarly, Gupta and Sarawagi [19] propose a system that extracts and integrates tuples from HTML lists. Both approaches target at constructing a corpus of high-quality data, where, to *recover* the semantics of the data content, a huge amount of post-processing effort such as entity resolution and schema matching [5, 15] is needed. In contrast, our approach automatically categorizes the obtained product specifications, without doing integration across sites.

Wrappers. Along with these systems, various techniques/tools have been proposed to extract structured data from web pages. Strategies exploit the opportunity of web page similarity in both HTML structure and natural language. Usually, a pattern (aka. wrapper) exploring the underlying similarity is obtained and will later be applied to other pages for further extraction. Much work [8] studies how to develop wrappers automatically but the quality of the output, in many cases, is low and not controllable. To control the automatic generation of wrappers, several techniques adopt: *domain knowledge* [18], but a knowledge base has to be crafted for each category; *redundancy* [3, 7], but products are characterized by many "rare" attributes that are present only in few sources; *annotators* [12], but it is hard to define a priori a set of annotators that can annotate all the present attributes.

Source Discovery. An analysis of structured data on the web has been described by [13]. The authors adopted a search paradigm to discover all the websites related to several domains and extracted some attributes from them. The authors found thousands of websites for several domains, but their evaluation was limited to few id attributes, making the extraction step simpler. Many other works adopted the search paradigm [2, 16], but the number of visited websites is one order of magnitude lower than our approach, thus the task of getting high efficiency and quality was simpler.

Products. Another topic related to our work is product integration and categorization. Existing work follows a supervised approach, which starts from an already well-established product database, and accomplishes integration for new product instances. Nguyen et al. [25] propose a scalable approach to synthesize product offers from thousand of merchants into their centralized schema. Kannan et al. [21] build a system to perform matching from unstructured product offers to structured specifications. Das et al. [14] describe a technique to infer tags of known products based on the attributes published in the specifications. However, none of them has focused on collecting high-quality product specifications aforehand.

Crawling. Techniques to automatically crawl target pages have been studied [20, 24, 30]. All these techniques guide the crawler by adopting some kind of knowledge: in [24] authors guide the focused crawler considering the semantic annotations on target pages; in [20], authors exploit the expected structure of a forum to efficiently crawl generic forums; in [30], the system infers relationships among instances present in a database from parallel navigation paths. In our approach the concept of target pages and the domain differ from previous approaches making these previous approaches not directly applicable.

7. CONCLUSIONS

This paper presents DEXTER, an end-to-end solution for the task of building big collections of product specifications from web pages. For that, we propose techniques to discover, crawl, detect and extract product specifications. To efficiently discover product websites DEXTER explores different techniques that rely on existing search APIs, for keywords search and navigating backlinks. To collect product pages DEXTER crawls product websites. To detect specifications, the Specification Detector identifies the tables and lists that contain product specifications. Finally, to extract the attribute-value pairs from the detected specification fragments, DEXTER adopts two wrapper generation techniques, a domain independent and a domain dependent approach.

A future direction is to use the collection of specifications obtained using our technique to perform entity linkage and schema alignment in order to build a universal product database. Another interesting direction is the automatic discovery of new categories based on the navigation structure of the product websites.

8. REFERENCES

[1] L. Barbosa, S. Bangalore, and V. K. R. Sridhar. Crawling back and forth: Using back and out links to locate bilingual sites. In *IJCNLP*, pages 429–437, 2011.

[2] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti. Supporting the automatic construction of entity aware search engines. In *WIDM*, pages 149-156, 2008.

[3] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. Extraction and integration of partially overlapping web sources. *PVLDB*, 6(10):805–816, 2013.

[4] M. Cafarella, A. Halevy, and J. Madhavan. Structured data on the web. *Communications of the ACM*, 54(2):72–79, 2011.

[5] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[6] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[7] S.-L. Chuang, K. C.-C. Chang, and C. Zhai. Context-aware wrapping: synchronized data extraction. In *VLDB*, pages 699–710. VLDB Endowment, 2007.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[9] V. Crescenzi, P. Merialdo, and P. Missier. Clustering web pages based on their structure. *DKE*, 54(3):279–299, 2005.

[10] E. Crestan and P. Pantel. Web-scale table census and classification. In *WSDM*, pages 545–554, 2011.

[11] N. Dalvi, P. Bohannon, and F. Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *SIGMOD*, pages 335–348, 2009.

[12] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, Jan. 2011.

[13] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *PVLDB*, 5(7):680–691, 2012.

[14] M. Das, G. Das, and V. Hristidis. Leveraging collaborative tagging for web item design. In *KDD*, pages 538–546, 2011.

[15] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817-828, 2012.

[16] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *WWW*, pages 100–110, 2004.

[17] E. Ferrara, P. D. Meo, G. Fiumara, and R. Baumgartner. Web data extraction, applications and techniques: A survey. *CoRR*, abs/1207.0246, 2012.

[18] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. Diadem: Thousands of websites to a single database. *PVLDB*, 7(14), 2014.

[19] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *PVLDB*, 2(1):289–300, 2009.

[20] J. Jiang, X. Song, N. Yu, and C.-Y. Lin. Focus: learning to crawl web forums. *IEEE TKDE*, 25(6):1293–1306, 2013.

[21] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman. Matching unstructured product offers to structured product specifications. In *KDD*, pages 404-412, 2011.

[22] N. Kushmerick. Wrapper induction: efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, Apr. 2000.

[23] X. Li, X. L. Dong, K. Lyon, W. Meng, and D. Srivastava. Truth finding on deep web: Is the problem solved. *PVLDB*, 6(2):97–102, 2013.

[24] R. Meusel, P. Mika, and R. Blanco. Focused crawling for structured data. In *CIKM*, pages 1039-1048, 2014.

[25] H. Nguyen, A. Fuxman, S. Paparizos, J. Freire, and R. Agrawal. Synthesizing products for online catalogs. *PVLDB*, 4(7):409–418, 2011.

[26] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *VLDB*, pages 738-741, 1999.

[27] S. Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, 1999.

[28] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *WWW*, pages 242–250, 2002.

[29] G. Weiss and F. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Intell. Res. (JAIR)*, 19:315–354, 2003.

[30] T. Weninger, T. J. Johnston, and J. Han. The parallel path framework for entity discovery on the web. *ACM Trans. Web*, 7(3):16:1–16:29, 2013.