Redundancy-Driven Vertical Domain Explorer

Celine Badr
Dipartimento di Ingegneria
Università Roma Tre
Rome - Italy
badr@dia.uniroma3.it

***Abstract*—Entities, generally, represent real-world concepts, such as a person (writer, singer, etc.), a product (book, camera, etc.), a business, etc. In large data-intensive websites, sections related to an entity in a given vertical domain consist of a thousands of data-rich pages, each displaying attribute values for one instance of the given entity. Ideally, to build a rich repository of entity instances that serves the unlimited search needs of Web users, data aggregators aim to collect all the possible instances available for that given entity and apply data extraction for its attributes. A manual approach would be costly in time and effort. In this work, we propose a system that automatically discovers new large websites publishing pages about a conceptual entity, by exploiting the large amount of overlap on the Web among sources in the same vertical domain. Starting with information from one training site, specific queries are generated and results returned by search engines are analyzed and filtered. The sources retained from these search results undergo then a semantic, syntactic, and structural evaluation to detect data-intensive pages for the domain entity. Semi-structured attributes location is also identified on the discovered entity pages. Our approach can thus be exploited by vertical search engines in pre-processing to enhance web page crawling, as well as in data extraction.**

Keywords–entity discovery; vertical domain; search; keywords.

I. INTRODUCTION

Large data-intensive websites are composed of categories, each listing thousands of pages related to real-world conceptual entities. We refer to this set of pages, generally sharing a common structure or template, as *entity pages*. In Web data extraction, inferred wrappers extract selected attribute values on a subset/all of the site's entity pages. However, the extracted attribute values remain limited to the data available from the site's repository. Ideally, to build a rich vertical warehouse of instances of a given entity, data aggregators aim to collect all the possible instances available for that entity and extract its attributes on a large web scale. A manual approach would be costly in time and effort. Figure 1 shows, for example, a page representing an instance of the `Book` entity offered on a book selling website. Common attributes for the `Book` entity, like title, price, ISBN, publisher, etc., are listed on each such page. The actual values displayed on the page pertain to one `Book` instance and originate from one record in the underlying database. We use *instance page* to refer to one individual example of the entity pages.

In order to complement, enrich, or validate data gathered from a training site, it is useful to find instance data available on other sites that offer similar information on the same type of entities. This requires performing two main operations:

1) Find other large data-intensive websites offering pages on the given entity,
2) Download entity pages to extract their instance data.

Given one large website in a vertical domain, we presented in [2] a synergic method to locate its entity pages and extract instance data on them, with our CoDEC system that implemented it. The approach was facilitated by exploiting redundancies in the site's HTML structure, navigation paths, and content tokens. To extend it to other websites, in this work we address the problem of automatically locating large data-intensive sources in a given vertical domain. For that, we propose an approach that can be exploited by vertical search engines to enhance web page crawling and filtering by using domain knowledge in a pre-processing phase. Our solution uses as input the information collected during inference on a training site to find other large websites containing instance pages about the entity of interest. This is based on the fact that there is a large amount of overlap on the Web among sources in the same vertical domain [1], so information we have from one site, or possibly more, can lead to overlapping information in sources not yet discovered. For example, if our training website in the `Book` domain contains distinct instances of `Jane Eyre` book, `Oliver Twist`, and `Madame Bovary`, and we can find another website that also lists instance pages for these 3 books, it is very likely that this new website is a data-intensive source in the `Book` domain.

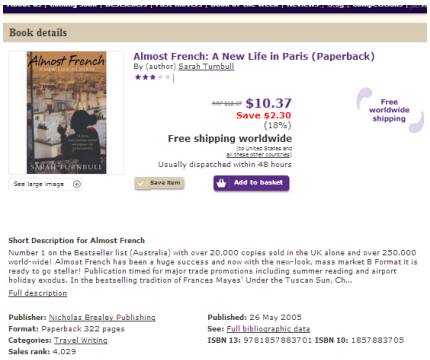

Figure 1: Data-Rich Instance Page of the Book Entity

The search is made possible by extracting from the collected information repository, keywords that represent the domain, the entity of interest, and a few selected instances, in order to *run queries* through a search engine. Many of the returned results are not data-intensive entity pages (e.g., blogs, news, reviews), which are of no relevance to our targeted approach. Thus we need to apply a two-level *filtering mechanism* to confirm or discard a result page based on its relevance: First, at the level of the returned URLs to determine the subsets of search results to be further examined; second, the pages pointed to by these URLs need be checked for

 ISBN: 978-1-61208-361-2

semi-structured data formatting to be considered candidates for eventual data extraction tasks.

We aim to perform these tasks in a fully automated way, independently of the training website. Our system, then, outputs a set of similar instance pages found on newly discovered websites and points out content nodes on them.

II. Proposed System Model

Entity instance pages are spread out on the Web, populating many large data-intensive websites. Thus finding and collecting these pages in a repository for a pre-determined domain consists mainly of a targeted web search activity, followed by an appropriate result filtering process. To conduct the above activities, we propose a system that exploits the data and domain knowledge collected on a training website. The gathered information is then used to facilitate discovering other potential large websites in the same vertical domain and eventually extract instance data from their entity pages.

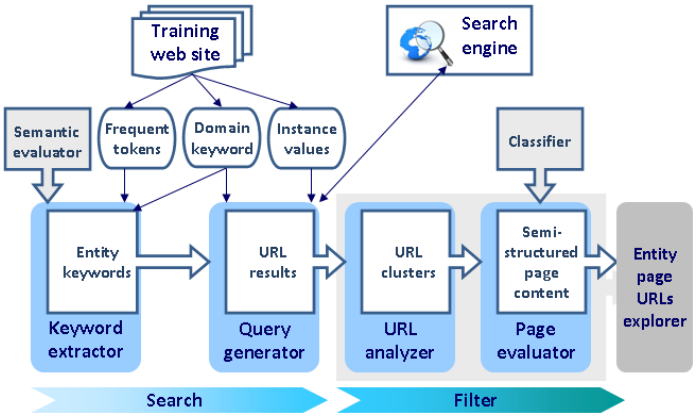

Figure 2: System components

The system is conceived of 4 main components illustrated in Figure 2. The components' goal is to direct and enhance the entity page search and filtering process as follows:

- The input to the first component is the information obtained from the training website, in particular, the template tokens from the sample pages. This component then analyzes these tokens in combination with the domain identifying terms and with the help of a semantic evaluator to output keywords for the entity of interest (Section III).
- Keywords are then passed to the query generator component that builds a series of boolean queries and sends them to a web search engine (Section IV).
- The URL results from the respective queries are passed as input to the following component, the URL analyzer. The latter applies a clustering algorithm on the set of URL results and keeps only relevant clusters for further analysis (Section V).
- These are then passed to the page evaluator component that downloads the pages from the Web and identifies entity-related semi-structured content (Section VI).

When various instance pages are identified on a website, a customized crawler can find the page that links to them on that site and download all the other instances that it contains. In addition, the system keeps record of the location (containers) of instance data on the pages, facilitating further data extraction efforts. The following sections describe in more detail each system component and the functionality that it implements.

III. Finding Entity Keywords

In general, any effort to find content on the Web passes through a search engine by providing keywords. Search engines today use implicit techniques (contextualization, approximation, search history, PageRank, etc.) to offer useful results in a ranking order estimated relevant to the information need. However, keywords remain essential to state the initial intent of the search, and choosing "good" search terms minimizes the distance between what's sought and what's found in the search engine's indexed content repository. This section presents the keyword extractor component for selecting query terms likely to generate relevant results. Results are relevant if they lead the system to discover new data-intensive large websites for the given domain entity.

In our case, we would like to use search engines to find semi-structured instance pages for a given entity. For an efficient web search, we need to have well formulated queries and a subsequent mechanism to confirm or discard a returned result page based on its relevance. For this operation, a user's manual intervention would not scale. Consequently, in our automated approach, it is the system's responsibility to formulate the queries by choosing a combination of "useful" keywords likely to lead to new web pages for the entity of interest. In later sections, we also explain how the system automatically filters returned search results. Our result acceptance criteria combines both the page content and the format in which it is presented.

To automatically select *entity keywords* for our search queries, that is, keywords tightly associated with the entity of interest, we rely on the information derived from the training site. We aim to find terms that are both quantitatively and qualitatively related to that entity. Common tf-idf measures are counter-intuitive here as we want template words occurring frequently on all sample pages, and not the opposite. When estimating fixed template tokens during the inference process [2], the system kept text nodes appearing frequently at the same location in the set of sample pages as potential template tokens, while tokens that were particular to one or few pages were considered to be variable content. Many may be labels typical of the domain, while others may be more general. Some may be stop words or extra information present on the page but not related to the domain entity. The recurring text fields collected constitute then a good starting point to find candidate domain keywords related to the entities. We propose to narrow down on entity vocabulary with the help of a semantic evaluator. The semantic evaluator takes two words or sets of words and returns a value that represents their semantic distance. The closer they are semantically, the higher the score returned by the evaluator. Since the domain identifier token is already given to the system as input, the system is able to rate, for each of the words collected, its relationship to the domain identifier expression. The words that score highest are kept. After also cross-evaluating these top-scoring words, the final keyword candidates set consists of terms that

- Appear frequently on instance pages of the vertical domain,
- Are closely related to the domain identifier string,

 ISBN: 978-1-61208-361-2

- And, additionally, are closely related among each other from a semantical aspect.

This automatically selected collection of terms replaces the need to manually select entity attribute labels for each domain without knowing the relevance or frequency of these labels when applying the search to the remaining pages on the Web.

Book	NBA Player	Restaurant	University
paperback	player	hotel	college
publish	soccer	shop	campus
author	game	pub	faculty
bible	tennis	cuisine	school
write	sports	store	graduate
chapter	golf	food	student
stock	team	establishment	education
item	stadium	nightlife	teaching
note	season	drink	institutional
	jersey	club	institute
	complete	city	sciences
			undergraduate

Figure 3: Entity keywords

Examples of resulting word sets for 4 domains are shown in Figure 3. Two entity keywords (highlighted) are derived from the related words pool, such that, with the domain keyword, they have pairwise a high semantic correlation.

IV. Constructing Queries

Since we are interested in finding large websites containing entity instances of one vertical domain, we build on the observation in [1] that there is a significant amount of connectivity and redundancy in content among data sources within the same domain on the Web. The existence of overlapping entities among different sources permits the discovery of new websites starting with entities already discovered, operating with a set-expansion approach. Based on this fact, we propose to start with the entities gathered from our inference website and conduct a search for overlapping entities on the Web.

The domain knowledge acquired on the training website consists of the domain identifier terms, the entity keywords described in Section III, and all the values of the entity attributes extracted on the instance pages by the inferred wrappers. With this knowledge, we propose to build search queries for different instances that are likely to find other pages on the Web describing these respective instances. When some discovered instances are determined to belong to a new data-intensive website, they can constitute the seeds for a crawler tailored to find the rest of its instances.

Each query is composed of the domain identifier terms, the entity keywords extracted for that domain, and one record from the instance attributes stored in the information repository. Various queries, each for a specific instance in the repository, run in parallel. For each instance query, pages returned by a search engine are considered relevant if they are semi-structured pages about the same or a similar entity instance. To restrain the search scope, we require that attribute values used in the query be matched on the result pages as they are part of the instance-related content. The identifying attribute is required, while other attributes can be optionally added to the query terms. The domain and entity identifier terms are rather descriptive and not necessarily expected to be in the content of an instance page. Therefore, in each query composition, attribute instances are combined with logic AND, while the domain and entity terms are joined with the OR conjunction.

For example, in the book domain, the domain identifier term is `book`. The entity keywords derived from our inference website are `publish` and `write`. A random instance extracted on the inference site has the attribute values `Great Expectations` for title and `Charles Dickens` for author. The title attribute is the instance identifier. The query composition for this example is then:

```
``Great Expectations'' AND ``Charles Dickens'' ``book'' ``publish'' ``write''
```

The `OR` operator is implied by default. Queries are constructed for a number of different instances and each is sent to the search engine. The search results URLs are collected for each query, but results pages are not yet downloaded.

We note that some attribute values are likely to yield less matching search results than others, due to differences in dates or measures format, for example, or the presence of abbreviations and spelling variations in the values of search phrases. This can be mitigated by diversifying instance values and attributes selection for a wider search coverage.

V. Filtering URL Results

In this section, we explain how the system goes about the numerous query results returned by the search engine, and how the first step of the automatic filtering is performed. The main idea is to select only a useful subset of the URLs collected from different instance queries for further processing, instead of downloading all the web pages listed in the results.

When a query is sent out to a search engine, the latter returns a very large number of results in response. Because of the entity redundancy principle discussed in section IV, the more frequently a website appears in the result sets, the higher the probability of it being a large website with content redundancy responding to our search needs. Equally, the more overlap there is, the more frequently that website will show up in the result sets. We aim to exploit similarities among URLs to group result pages into clusters, such that pages in a cluster respond to distinct instance queries, but belong to a single website and have little dissimilarity in their URL patterns. Each qualifying cluster is then analyzed to see if it contains candidate pages belonging to a large website.

http://www.rakuten.com/prod/the-cold-war-a-new-history/31198770.html
http://www.abebooks.com/book-search/kw/fact%F3tum-charles-bukowski/page-1/
http://www.alibris.com/Power-Multi-Level-Marketing-Mark-Yarnell/book/5267499
http://amblingbooks.com/books/view/emotional_alchemy_2
http://www.shelfari.com/books/8515328/The-Landscape-of-History
http://www.rakuten.com/prod/your-first-year-in-network-marketing/30327356.html
http://www.alibris.com/African-Cry-Jean-Marc-Ela/book/158642
http://productsearch.barnesandnoble.com/search/results.aspx?ATH=Mark+Yarnell
http://www.shelfari.com/books/373097/Stone-Rain
http://www.abebooks.com/book-search/title/sharpes-prey/page-1/

Figure 4: Sample query result URLs

There are some existing works that propose algorithms to cluster large websites [3]. However, these approaches assume that the website is already given and they try to discover its structure. Inspired by Blanco et al. [4] that combine URL analysis with some simple content features, we use URLs clustering as a starting point for a further website exploration.

 ISBN: 978-1-61208-361-2

By looking at a reduced sample set of URLs gathered from a results pool for queries in the book domain (Figure 4), we can already spot some recurrences. To operate on a large scale, the system has to automatically determine which URLs have similar patterns and group them together. Thus, we opt for a hierarchical agglomerative clustering (HAC) algorithm to process the result URLs collected from all the instance queries. HAC is a simple "bottom-up" technique that fits our data set, where the percentage of results to be merged into clusters is small with respect to the entire set of URLs returned by the search engine for the instance queries. We need to specify for our problem a metric defining a distance measure for two URLs. A URL can be broken into different parts (protocol, host, port number, path, query string, etc.), some of which are optional. We define the distance, or dissimilarity, between two URLs based on the parts they have in common and those that are different, as shown in Figure 5. For successive iterations,

Algorithm 1: Measuring Dissimilarity of Two URLs

Input : URLs $\{u_1, u_2\}$
Output: The distance D between u_1 and u_2

```
if u1.authority == u2.authority then
    Let D be the number of different parts between
    u1.path and u2.path;
    if u1.protocol != u2.protocol then
        D++;
    if u1.query != u2.query then
        D++;
    if u1.ref != u2.ref then
        D++;
else
    D = INFINITY;
return D;
```

Figure 5: Measuring Dissimilarity of Two URLs

a linkage criterion needs to be defined for the algorithm to compute the distance between two sets of elements as a function of the distance of the elements these sets contain. A possible function is the minimum distance between elements of each cluster, referred to as single-linkage clustering. For clusters $C1$ and $C2$, their distance is:

$$\forall x \in C1, \forall y \in C2 : d(C1, C2) = min(d(x, y)) \quad (1)$$

where x and y are URLs. Hence, at a given iteration, the two clusters separated by the shortest distance are merged. This implies that for a subsequent iteration, the minimum distance between clusters is larger than that at the previous step. A stopping condition for the algorithm can be either a threshold that says clusters have become too distant to be merged, or, when applicable, a predefined number of clusters to reach. Based on the URL distance metric, we set our stopping condition as a small integer, between 1 and 3.

From the clusters generated by HAC, we can consider as valid candidates for instance pages those URLs that:

- Occur in multiple distinct instance search results,
- Originate from the same website,
- Share a pattern with low dissimilarity value.

URLs satisfying these properties make it through the first filtering step of the system and are then further examined to determine if they constitute semi-structured instance pages from a data-intensive website. URLs occurring only occasionally or not matching any cluster (other than their own singleton) are not considered of interest and their pages are not downloaded.

VI. Page Evaluation

In this section, we describe how pages at the resulting URLs undergo the second stage of filtering through our system. We adopt as a reasonable pre-condition the fact that pages belonging to large data-intensive websites are generated by regular templates with some level of structure. The responsibility of the page evaluator is then to determine whether the pages at the given URL addresses contain semi-structured data sections that can be of interest for the data extraction task. Pages not contaning any data sections with structure can be discarded for our purposes. For each URL in the clusters returned by the previous system component, the corresponding web page is downloaded to be analyzed. To separate interesting from non-interesting pages, we proceed in three steps:

- Locate relevant fragment(s) on the page,
- Extract features from page fragment(s),
- Classify the page based on extracted features.

Given a downloaded page, the first task in this component's process consists in identifying in its content the fragments to evaluate with regards to the originating search purposes. In large data-intensive websites, instance pages have at least a section that displays the entity attributes in semi-structured format. Since our data extraction goal is to retrieve semi-structured attributes values where available, we need first to locate the HTML part where they are displayed on the page. Some works propose vision-based analysis and DOM tree alignment, but we opt for a less complex approach. Instead, we take the query that generated this result page and we search for the least common ancestor HTML container of the attribute values in that query on the given result page.

Depending on the occurrence of the attribute values, the following are the possible scenarios that can be encountered:

- Attribute values are located in one page fragment: the corresponding HTML container is returned for analysis.
- Attribute values are found in several page fragments: a list of HTML containers is returned and each will be analyzed separately.
- Attribute values are not found on the page: the given result page is discarded.

Given an HTML fragment where these attribute values appear, the automatic page filtering sequence proceeds to analyze it with respect to some structural and content features that are commonly observed on data-intensive web pages. Occurrences in text sections are not of any interest for our data extraction purposes. Recurrent characteristics have been identified and used to train a classifier in order to automatically distinguish structured from non-structured content. Namely, we look into the text length, recurrence of characters such as the column, usage of lists or table cells, and other HTML formatting aspects. A result page is boosted as potential instance page if at least one of its extracted fragments is classified as semi-structured. Otherwise, the page is not considered a valid candidate and is discarded.

 ISBN: 978-1-61208-361-2

The output from the page evaluator component consists then of search result pages already retained in a URL cluster, where the respective query attribute terms occur in a semi-structured layout. These are the final candidate instance pages of the system. Clusters with more classified candidate pages will have higher priority to be processed by the site explorer for crawling more instances from the discovered website to populate the vertical domain repository.

VII. EXPERIMENTS

We describe here our Vertical Domain Explorer system *(VerDE)* implementation, the experiments we conducted, and the results obtained, with some analysis and comments.

VerDE system is implemented in Java as a set of packages. For the semantic evaluator, we use the semantic similarity service provided by the University of Maryland, Baltimore County, which combines Latent Semantic Analysis (LSA) and knowledge extracted from WordNet to evaluate word similiarity. For the clustering module in the URL analyzer, we opt for the HAC algorithm with single linkage, and base our implementation on the Java library provided by the *Sape* research group at the University of Lugano. Support scores are computed for each URL results cluster to reflect their coverage of the different instance queries. Clusters with a score below a set threshold are not processed any further. Finally, the page evaluation component requires a classifier to assess the structure of the page content where the instance attributes are located. Because of the binary nature of the expected output (semi-structured content or not), the classifier is implemented as a logistic regression. Eight features related to text length, punctuation, and HTML formatting are used. We report the accuracy scores listed in Table I for the performance of our classifier model on positive and negative HTML content collected on various pages on the Web. All the implemented VerDE components are integrated to smoothly deliver the functionalities of the automated search and filter approach that the entire system builds on.

TABLE I: CLASSIFIER ACCURACY

Training set accuracy	81.82%
Cross-validation set accuracy	93.75%
Test set accuracy	100.00%

We evaluate the results obtained from VerDE by running experiments in 4 vertical domains: Restaurant, University, Book, and NBA Player. Once retrieved from the inference site tokens, entity keywords are stored for future use to enhance performance. For attribute values, random records are selected from the database to formulate queries during the system execution. A match for these values is then sought in the search results. For evaluation, we include in query constructions an entity identifier attribute and a variable second attribute and we run experiments with 5 and 50 random instances for each domain. The instance number and attribute selection of each run are specified in the experiment configuration.

In total, 10104 URLs were collected during the experiments, while only about 15% of them became part of clusters with URL redundancies and 11.73% were finally classified as semi-structured. This translates into a considerable effort saved from downloading unuseful pages. From the pages where the attribute values were matched in the first phase of automatic filtering, 79.6% were classified positive in the second filtering phase, highlighting the benefit of the URL pre-processing step.

Table II lists the percentage of examined pages with respect to the total URLs collected from search engine results, precision of the semi-structured classification, and number of distinct new discovered sources. The results reflect a clear

TABLE II: EXPERIMENT RESULTS

Domain	% Examined	Precision	# Discovered
RESTAURANT	3.32	0.83	54
UNIVERSITY	25.89	0.91	342
BOOK	8.03	0.83	245
NBA PLAYER	26.32	0.84	469

optimization in the automated effort of exploring websites for crawling and data extraction, allowing a site explorer to focus the processing on likely candidates of large data-intensive websites. Due to the huge amount of results returned by the search engine, recall is hard to evaluate, but an estimate of 73.4% was calculated by manual verification on a sample URL subset. We note that we observed poor accuracy on numerical attributes, e.g., height, phone numbers, etc., mostly due to wide variations in formatting on the Web and our heuristics being based on exact matching. However, diversifying the attribute selection in queries for a given entity can compensate any loss in results coverage. For example, queries with books title and publisher would find matching sources that queries with title and ISBN numeric values did not find. Another option would be to relax the boolean search with approximation or regular expressions when matching values on pages.

VIII. RELATED WORK

The VerDE system we presented touches on many subjects in information retrieval. The research work in [5][6][7][8] presents topical crawlers that filter the portions of the Web to be crawled. However, topical crawlers do not necessarily find large data-intensive websites. Challenges are also highlighted in selecting non-biased crawl seeds and the ability of the crawler to distinguish relevant from non-relevant documents. Our approach relies on filtering the URLs to be downloaded by identifying redundancies. It automatically selects candidate URL seeds likely to yield entity pages matching the crawling objectives. It also allows to limit the crawl to sections of the website that satisfy constraints both on content and format.

In recent years, the need for vertical search engines dedicated to topical services like news, finance, shopping, etc., has motivated efforts for highly accurate information retrieval techniques. Nie et al. [9] build object search engines in the academic and product search vertical domains. They statistically estimate that 12.6% of randomly crawled pages are product pages. This echoes the estimation range we also report in our result findings. Similarly, Nguyen et al. [10] consider the issues of data extraction, schema reconciliation, and data fusion, in building a system that synthesizes products for shopping verticals or product search engines. Hao et al. [11] start with one labeled example site from a given vertical and trains a system to extract data from unseen sites in that same vertical, independently of the domain. Also, Song et al. [12] exploit entity redundancy in different websites to learn entity attributes from inner- and cross-site features. In all these works, input pages are provided either by topical crawlers or by manually specifying the web source to analyze. No

 ISBN: 978-1-61208-361-2

optimization is performed at the search and download stages to only target data-intensive and semi-structured entity pages, which would reduce considerably the amount of later page processing. On the other hand, our approach discovers new data-intensive sites automatically, thus reducing human effort and also avoiding unnecessary page downloads. Moreover, VerDE individuates the HTML containers where the semi-structured data are displayed. This can resolve some extraction obstacles the other systems face where an attribute like book title appears several times on the page with different values, as various recommended instances are listed on the same page with the entity instance data.

One approach similar to ours is by Blanco et al. [13]. While both our work and theirs address the domain-independent page gathering task, the two approaches differ in several aspects:

- They analyze several similar websites to perform quantitative keyword extraction and bootstrap the system. Our approach uses data from one training site with a quantitative and qualitative semantic evaluation to find meaningful keywords for the vertical domain.
- Their system determines instance pages based on entity keyword appearance and hyperlinks location. This may lead to false positives in websites that do not offer semi-structured data-intensive pages, while our page filtering is based on a trained classifier that evaluates sections of interest considering content, structure, and formatting, which also reduces noise.
- For each URL result returned by the search engine, they run a full website evaluation, which is not a trivial task, especially when the URL belongs to a large website composed of thousands of pages. The website exploration is even repeated if a derived template evaluates as a potential instance page. In contrast, we minimize the number of candidate URLs before they are processed any further. The second step of page classification avoids exploring a website if no semi-structured content is detected.

Another related work is by Weikum and Theobald [14]. They present a knowledge harvesting technique to construct a comprehensive knowledge base of facts by extracting semantic classes, mutual relations, and temporal contexts of named entities. In this context, the semi-structured nature of data-intensive pages cannot be exploited in a pattern-based facts extraction without substantial postprocessing of the output.

Some recent data extraction approaches address wrapper generation using visual content features from the web sources [15][16][17]. Such approaches examine the resemblance of data records to build a block tree, then proceed to data record extraction and data item extraction, assuming the relevant information block is centered in one main region on the page. In contrast, our page evaluator component detects and classifies the block containing relevant attribute values.

IX. Conclusion and Future Work

In this work, we presented an approach to automatically and efficiently locate large data-intensive web sources in a given vertical domain, by starting with knowledge gathered from a training site and exploiting redundancies in entity occurrences on the Web. The system prototype implemented is composed of 4 logical components: a keyword extractor and an automatic query generator for the search tasks, then a URL analyzer and a page evaluator for the filter step. Our experiment results show a great advantage in using the automatic 2-stage filtering before exploring websites returned by search engines, and a high level of precision of our classifier. Future work can address the implementation of the website crawler that exploits output page clusters, in addition to application of data extraction techniques on the semi-structured containers identified on the pages.

References

[1] N. Dalvi, A. Machanavajjhala, and B. Pang, "An analysis of structured data on the web," Proc. of the VLDB Endowment, vol. 5, no. 7, 2012, pp. 680–691.

[2] C. Badr, P. Merialdo, and V. Crescenzi, "Synergic data extraction and crawling for large web sites," in ICIW 2013, The 8th International Conference on Internet and Web Applications and Services, 2013, pp. 200–205.

[3] I. Hernandez, C. R. Rivero, D. Ruiz, and R. Corchuelo, "A tool for link-based web page classification," in Advances in Artificial Intelligence. Springer, 2011, pp. 443–452.

[4] L. Blanco, N. Dalvi, and A. Machanavajjhala, "Highly efficient algorithms for structural clustering of large websites," in Proc. of the 20th international conference on World wide web. ACM, 2011, pp. 437–446.

[5] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," Computer Networks, vol. 31, no. 11, 1999, pp. 1623–1640.

[6] M. Chau, "Spidering and filtering web pages for vertical search engines," in Proc. of the Americas Conference on Information Systems, AMCIS, 2002.

[7] S. Sizov et al., "The bingo! system for information portal generation and expert web search." in CIDR, 2003.

[8] A. Patel and N. Schmidt, "Application of structured document parsing to focused web crawling," Computer Standards & Interfaces, vol. 33, no. 3, 2011, pp. 325–331.

[9] Z. Nie, J.-R. Wen, and W.-Y. Ma, "Object-level vertical search." in CIDR, 2007, pp. 235–246.

[10] H. Nguyen, A. Fuxman, S. Paparizos, J. Freire, and R. Agrawal, "Synthesizing products for online catalogs," Proc. of the VLDB Endowment, vol. 4, no. 7, 2011, pp. 409–418.

[11] Q. Hao, R. Cai, Y. Pang, and L. Zhang, "From one tree to a forest: a unified solution for structured web data extraction," in Proc. of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. ACM, 2011, pp. 775–784.

[12] D. Song, Y. Wu, L. Liao, L. Li, and F. Sun, "A dynamic learning framework to thoroughly extract structured data from web pages without human efforts," in Proc. of the ACM SIGKDD Workshop on Mining Data Semantics. ACM, 2012, p. 9.

[13] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti, "Supporting the automatic construction of entity aware search engines," in Proc. of the 10th ACM workshop on Web information and data management. ACM, 2008, pp. 149–156.

[14] G. Weikum and M. Theobald, "From information to knowledge: harvesting entities and relationships from web sources," in Proc. of the 29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2010, pp. 65–76.

[15] P. L. Goh, J. L. Hong, E. X. Tan, and W. W. Goh, "Region based data extraction," in Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on. IEEE, 2012, pp. 1196–1200.

[16] K. Simon and G. Lausen, "Viper: augmenting automatic information extraction with visual perceptions," in Proc. of the 14th ACM international conference on Information and knowledge management. ACM, 2005, pp. 381–388.

[17] L. Li, Y. Liu, and A. Obregon, "Visual segmentation-based data record extraction from web documents," in Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on. IEEE, 2007, pp. 502–507.

 ISBN: 978-1-61208-361-2