UNIVERSITÀ DEGLI STUDI DI ROMA TRE
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 84 – 00146 Roma, Italy.

Automatically Generating Reports from Large Web Sites

LORENZO BLANCO†, VALTER CRESCENZI†, PAOLO MERIALDO†

RT-DIA-90-2004 **July 2004**

† Dipartimento di Informatica e Automazione
Università di Roma Tre
00146 Roma – Italy

http://www.dia.uniroma3.it

ABSTRACT

Many large web sites contain highly valuable information. Their pages are dynamically generated by scripts which retrieve data from a back-end database and embed them in HTML templates. Based on this observation several techniques have been developed to automatically extract data from a set of structurally homogeneous pages. These tools represent a step towards the automatic extraction of data from large web sites, but currently their input sample pages have to be manually collected. To scale the data extraction process this task should be automated, as well. We present techniques to automatically gathering structurally similar pages from large web sites. We have developed an algorithm that takes as input one sample page, and crawls the site to find pages similar in structure to the given page. The algorithm minimizes the number of pages downloaded by building a local model of the site topology, and then focusing the navigation onto paths that most likely lead to the target set of pages. The pages collected by the crawler can feed an automatic wrapper generator to extract data and generate reports. We have conducted experiments over several web sites, obtaining encouraging results.

Contents

1 Introduction **4**

2 Gathering web pages **4**
2.1 Preliminaries 5
2.2 Crawling for Structure: Intuition 6
2.3 Crawling algorithms 6
2.3.1 The TRIVIALSEARCH algorithm 6
2.3.2 INDESIT: Learning Promising Paths 7

3 Experimental Setting **9**
3.1 Sites and sample selection 9
3.2 Metrics 10

4 Experimental Results **10**

5 Related work **11**

6 Conclusions **12**

List of Figures

1 DOM trees 5
2 The TRIVIALSEARCH Algorithm 7
3 Pages and clusters 8
4 Summary of the INDESIT algorithm 9
5 Sample pages and sites 10
6 Experimental results 10
7 Searching unique pages 11

1 Introduction

Many web sites represent authoritative information sources for several domains. Unfortunately, the precious information provided by these sites is spread across thousands of unstructured HTML pages, connected by a tricky network of hypertext links. Such an interface is suitable for a human consumer, but makes the published data not easily accessible to electronic applications. However this problem can be significantly alleviated for a relevant class of web sites. Consider that the bigger the sites are, the more probably pages are automatically generated by scripts. These are simple programs that retrieve data from a back-end repository and serialize them into HTML pages. The automatic nature of the page generation process confers a certain degree of regularity to both the internal structure of the pages and the topological structure of the site graph. Pages generated by the same script share a common template, and the graph site, though large and complex, is organized according to a regular network of links.

Recently, several researchers have proposed [3, 8, 26] techniques that exploit the similarities of pages generated by the same script in order to automatically generate a web wrapper, i.e. a program that extracts data from the HTML source code. Based on these techniques they have developed tools that, given a set of pages sharing the same template, infer a wrapper that can be used in order to extract the data from all pages conforming to that template.

Although these tools represent a step towards the automatic extraction of data from large web sites, several issues remain unsolved. To give an example, consider a web site providing detailed and reliable information about the stock market, such as `http://www.axl.co.uk`. The site contains more than 20,000 pages. Among these, about 3,750 pages contain a careful and fresh description of the UK companies (one page per company): the data provided by these pages can be an important information source for many applications. It could be interesting to produce a document, say an XML document, reporting data about *all* the companies described in the site. This can be accomplished by fetching all the company pages, and then generating a wrapper for them. However, the dimensions of the web site, and the complexity of its graph, make the manual collection of the company pages neither feasible nor scalable.

This paper presents techniques to automatically crawling a web site in order to reach pages containing data of interest. The only input we require is a web page: the system picks out from the site all the pages that have the same structure of the input page. The output set of pages, as uniform in structure, can be done as input to a wrapper generator system (such as [3] or [8]).[1]

Since we aim at obtaining the largest coverage while minimizing the number of pages downloaded, a simple local model of the site structure is incrementally built during the crawling. This model is used to focus the navigation towards the paths that more quickly lead to the target pages.

Contributions

The main contribution of this paper is an efficient method for retrieving from a large web site useful pages for data extraction purposes. Our method is based on (i) a simple yet effective model that abstracts the structural features of web pages, and (ii) an efficient algorithm that navigates the site to fetch pages similar in structure to one input sample page. Combined with an automatic wrapper generation system, our method provides infrastructure for efficient and scalable data extraction from the web.

Paper Outline

The paper is organized as follows. Section 2 presents our crawling method in detail; Section 3 describes the general experimental setting, including the evaluation methodology, metrics, and the sites we have adopted to experiment our proposal; Section 4 reports the results of the experimental evaluation of our technique on real-life web sites. Section 5 discusses related works, and Section 6 concludes the paper.

2 Gathering web pages

In this section we present an automatic method to retrieve pages similar in structure to one given sample page. We first present a model to abstract the structure of web pages, and to describe at intensional level the topology of a web site (Section 2.1). Based on the model, we have developed techniques for crawling the site focusing the exploration towards pages structurally similar to the input one. We first give an intuition of the approach (Section 2.2), then we describe in detail two crawling algorithms that we have developed (Section 2.3).

[1] We use our ROADRUNNER system, whose source code has been recently released under GPL.

2.1 Preliminaries

To abstract the main structural properties of a HTML page, we refer to features associated to its DOM tree [1]. In particular, we model a web page by considering only the links it contains. The reason of our choice is twofold: on the one hand links are first class citizens for our purposes, as they are the means to support the site crawling. On the other hand, the organization of links within one page is strictly dependant of the structure of the page itself, as pages with similar structure contain links organized according to the same layout and presentation properties.

Thus, we elect links as an effective means to represent the structure of a page: in our framework, a web page is then described by considering only the paths of the corresponding DOM tree that start from the root and terminate into a node associated with a HTML anchor element. We call *link collection* one of such DOM paths together with all the `<url, anchor>` pairs that share that path. Also, we call *tag-list* the root-to-anchor path of each link collection. Finally, the *page-schema* of a page p, denoted $\Delta(p)$, is the set of tag-lists in p. [2]

To give an example consider Figure 1: it shows the DOM trees of three pages p_1, p_2 and p_3. According to our definition, the three pages are modelled by link collections, as follows:

p_1 : `HTML-TABLE-TR-TD` $\{\langle url_1, "X"\rangle, \langle url_2, "Y"\rangle\}$
`HTML-UL-LI` $\{\langle url_{10}, "E"\rangle, \langle url_{11}, "F"\rangle\}$
p_2 : `HTML-TABLE-TR-TD` $\{\langle url_3, "X"\rangle, \langle url_4, "Y"\rangle\}$
`HTML-UL-LI` $\{\langle url_{13}, "G"\rangle\}$
p_3 : `HTML-TABLE-TR-TD` $\{\langle url_5, "H"\rangle, \langle url_6, "K"\rangle,$
$\langle url_7, "J"\rangle\}$
`HTML-P-B` $\{\langle url_{20}, "L"\rangle\}$

Moreover, p_1 and p_2 have identical page-schemas: $\Delta(p_1) = \Delta(p_2)$ ={`HTML-TABLE-TR-TD, HTML-UL-LI`}; the page-schema of p_3 is $\Delta(p_3)$={`HTML-TABLE-TR-TD, HTML-P-B`}.

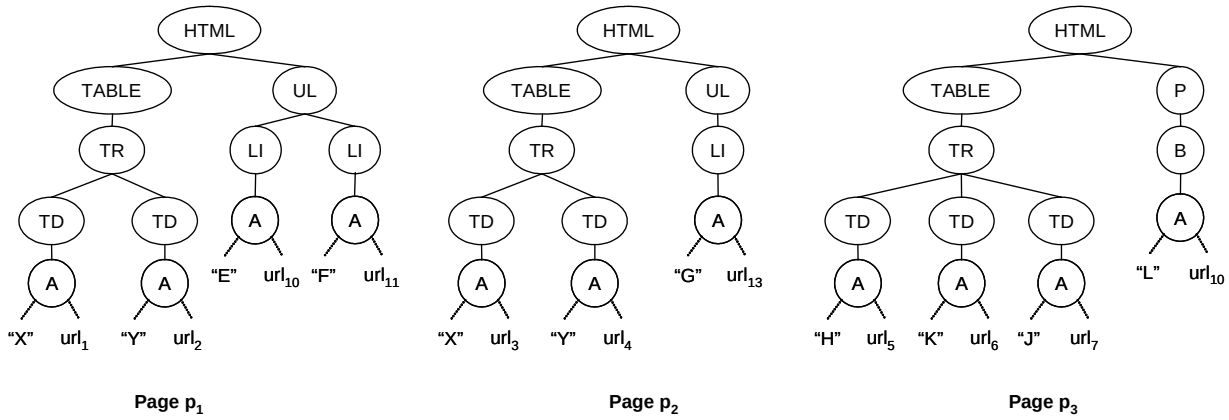

Figure 1: DOM trees

Pages can be grouped by their schema into clusters. A *cluster of pages* is a (possibly singleton) set of pages with identical schema. Consider again the three pages in Figure 1: pages p_1 and p_2 form a cluster, as they have identical schemas. Also, page p_3 gives raise to a singleton cluster.

We introduce the concept of *class-link* to model at the intensional level a set of link collections sharing the same root-to-anchor path.

The concepts of link collection and class-link capture another fundamental feature about the intrinsic structures manifested by large, automatically built, web sites: links belonging to the same link collection are likely to point to pages similar in structure.

With these ideas in mind, we can now introduce the algorithms to crawl the web focusing on the structure of one sample page.

[2] Actually, we also consider the name and value of HTML attributes associated to the nodes of the path. However, for the sake of simplicity, we present examples including only HTML elements.

2.2 Crawling for Structure: Intuition

Given one sample page containing data of interest, our goal is to pick out from the site the largest number of pages similar in structure to the input page. Our approach is based on simple observations about the organization of links and pages in large automatically built web sites. In the following we introduce the main intuitions behind our approach by means of an informal and simple example; then, we describe two algorithms that have been developed and experimented.

A first naive approach is to fetch pages that are reachable from the sample page, and to select those that fall in the cluster which would include the sample page.[3] As pages of large web site are widely connected one each other, it is likely that one page contains links leading to pages sharing the same structure (i.e., offering, at the intensional level, the same information content). For example, consider the pages describing products in any e-commerce web site: they usually contain links to pages of related items as well. This technique is very simple, and it usually gathers a small number of pages, so it is not effective for our purposes. [4]

However, we observe that some of the pages that we have reached, though not directly target pages, may result useful because they contain links that point backward to the target page; it is likely that these pages also list links to other pages belonging to our target set. Continuing our e-commerce web site example, a page describing one product may contain a link to a page listing links to products of the same category. We may say that, these pages offer indices to the target pages we are looking for. So, it could be convenient to concentrate the crawling towards the outbound links of these pages. However, observe that one page can offer many links, and just a subset of them could be promising for our purposes. To define such a subset we still rely on our model: namely, we only consider links belonging to the link collections that contain some pointer to pages in the target cluster.

To discover other pages the approach can be iterated: this can be done following different strategies. For example, we can look for pages similar to those containing indices. This corresponds to trigger recursive search targeted onto the index pages.

We now describe an algorithm, called TRIVIALSEARCH that is built over the above strategy. Then we discuss the limitations of this approach, and introduce a more sophisticated algorithm, called INDESIT.

2.3 Crawling algorithms

2.3.1 The TRIVIALSEARCH algorithm

TRIVIALSEARCH starts by fetching all the pages reachable by following the links in the input sample page p_0. These pages are then grouped according to their schema into clusters. Let C_0 be the cluster containing the sample page p_0 (in the fetched pages there might exist some page having the same schema as p_0). Consider Figure 3: from the input page p_0, seven pages organized into four clusters have been reached.

The link collections of the fetched pages are then evaluated; in particular those that include some link leading back to pages in P_0 are selected. The idea is that these link collections can work as indices to the target pages. Therefore all the pages pointed by these link collections are fetched. Still from Figure 3 link collections with tag-list tl_{11}, tl_{12}, and tl_{31} have links that point back to C_0, therefore they would be selected.

At this point, the algorithm searches for other pages like those that contains such link collections, i.e. it looks for other indices to our target pages. Continuing the example in Figure 3, pages similar to p_{11}, p_{12}, p_{31}, p_{32} will be searched.

The search of these index pages is performed recursively; the link collections of the returned set of pages are then evaluated, and those containing links to pages in C_0 are used to fetch new pages.

It should be clear now that the $depth$ parameter of the algorithm in Figure 2 indicates how many levels of indices should be searched. As it is a common practice that every page should be reachable from the home by following at most three links, we set this parameter to 2.

Another remarkable point is that the recursive function returns the subset of the fetched pages that are *similar* to p_0. To measure the similarity between two pages we consider the distance between their schemas. Namely, we consider the symmetric set difference between the two schemas: let $\Delta(p_i)$ and $\Delta(p_j)$ be the schemas of pages p_i and p_j, respectively; then:

$$dist(\Delta(p_i), \Delta(p_j)) = |(\Delta((p_i) - \Delta(p_j)) \cup (\Delta(p_j) - \Delta(p_i))|$$

[3] Actually we can relax the approach and take into account also clusters with schema "similar" to that of the sample page.

[4] It is worth noting that, as automatic wrapper generators need at least two input pages, this technique is effective whenever the goal is to wrap just the input sample page.

```
Algorithm TRIVIALSEARCH
Parameter: depth max distance of indices from target page;
Input: a sample page p0;
Output: a set of pages similar to p0;
begin
  return Search(p0, depth);
end

Function Search(page p0, integer depth): Set of Pages
begin
  if (depth=0) return {p0};
  Let D be all pages reached by following links from p0;
  Let C0 be the subset of D composed of pages with
                the same schema as p0;
  Let L be the set of link collections from D
                containing a link to pages of C0
  for each link collection lc ∈ L departing from a page p_index do
      add to D all pages reachable by following lc;
      Let P' be Search(p_index, depth-1);
      Let L' be the set of link collections departing from pages
                of P' and associated with the same tag list of lc;
      add to D all the pages reachable by
                following link collections in L';
  end
  return the subset of D which are similar to p0;
end
```

Figure 2: The TRIVIALSEARCH Algorithm

Note that if $\Delta(p_i) = \Delta(p_j)$ (identical schemas), then $dist(\Delta(p_i), \Delta(p_j))$ equals 0. We say that two pages p_1 and p_2 are *similar* if $dist(\Delta(p_1), \Delta(p_2)) < ST$, where ST is a threshold value determined empirically.[5] Also, it is worth observing that other definitions of the similarity function are possible, and could be immediately plugged into the algorithms we propose.

The pseudo-code of the TRIVIALSEARCH algorithm is shown in Figure 2.

The strategy of TRIVIALSEARCH is rather naive, as it focuses the crawling onto *any* page that can serve as an index to our target. Some of the fetched index pages could be very effective, as they can provide access to a large number of new pages; others could simply lead to pages that have been already reached, adding the cost of searching such index pages without any benefits. It is worth observing that the problem is also related to the depth at which we seek for index pages. As we do not know anything about the structure of the site, we set the depth value so that a large enough portion is visited.

2.3.2 INDESIT: Learning Promising Paths

INDESIT has been designed in order to follow only paths to effective index pages, i.e. pages containing links that lead to a large number of new target pages. The underlying idea of INDESIT is that while crawling a site, it is possible to acquire knowledge about the navigational paths it provides.

The INDESIT algorithm performs four main steps. In order to clarify its recursive nature, it is more convenient to restate the problem as if we aimed at obtaining all the pages of clusters similar to a given cluster (namely, that of the sample page). So, the algorithm starts with the (singleton) cluster containing the sample page. In the case of Figure 3, the starting cluster is C_0.

In the first step, *Neighborhood cluster discovery*, the link collections of the input cluster are used to discovered as many neighborhood clusters as possible. In order to minimize the number of downloads, whenever three or more

[5] In our experiments we have set $ST = 10$.

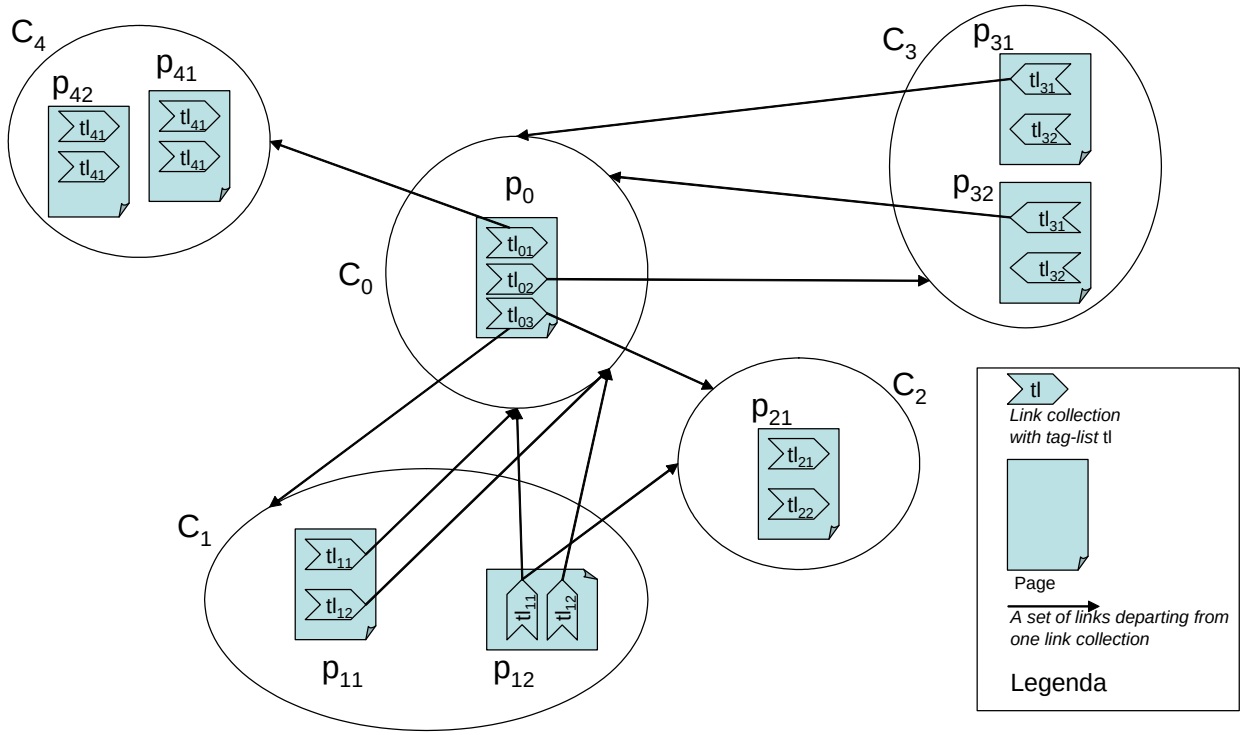

Figure 3: Pages and clusters

links of the same collection lead to pages of the same cluster, the remaining links of the collection are not followed. The rationale is that for large link collections, it is likely that the skipped links will lead to clusters already met at the beginning of the collection. In Figure 3, from the starting cluster C_0, four neighborhood clusters (C_1, C_2, C_3, and C_4) have been discovered.

In the second step, *Neighborhood clusters evaluation* step, the neighborhood clusters are analyzed to select the most promising directions. As done for the TRIVIALSEARCH algorithm, we can look for navigational paths pointing back to pages of the target cluster. At the intensional level of clusters, paths are represented by class-links, i.e. sets of link collections with the same tag-list. So we now search for class-links leading back to the target cluster. We could consider all the class-links made by link collections with at least one link pointing back to pages of the target cluster. However, a simple optimization consists in anticipating the navigation of class-links that provide access to many *new* target pages. Therefore for every class-link we compute a score indicating a measure of earn, in terms of new target pages, the class-link carries on.

The score of a class-link is computed by navigating its link collections, and then counting how many *new* pages with schema similar to that of p_0 are reached. A page is considered as new if it was not already present in C_0. Observe that during the evaluation of one link collection, it may happen that a page has been already downloaded to evaluate another link collection; but, if such a page was not in C_0 at the beginning of the evaluation, it will be considered as new. Note that in this way the evaluation order does not affect the scores. The scores of class-links are finally computed as the average scores of its link collections.

Actually, most collections can be evaluated by following only a small fraction of its links: as soon as a link lead to a page whose schema is not similar to the schema of C_0, the collection and its class-link are immediately scored 0 without downloading any other page. Consider again our example in Figure 3 and suppose that C_2 is not similar to C_0: the class-link with tag-list tl_{11} will be quickly scored 0 because its link collection in page p_{12} includes some link to pages of the cluster C_2.

At the end of this step we have a score associated with every class-link. Only class-link leading to fresh target pages will have positive scores; we call them *back class-links* (in Figure 3 only class-links of tag-lists tl_{12} and tl_{21} may be back class-links). The overall score of each neighborhood cluster is finally computed as the sum of the scores of its back class-links. A cluster with a high score corresponds to an effective index for our target pages.

In the third step, which is called *Recursion to catch index pages*, similarly to the TRIVIALSEARCH algorithm, a new instance of our problem is recursively triggered. The target of the new search is the neighborhood cluster with

Algorithm INDESIT
Input: a cluster containing page p_0
Output: a set of pages with the same structure as p_0
1. Neighborhood clusters discovery links of the initial cluster are followed to gain knowledge of as many neighborhood clusters as possible
2. Neighborhood clusters evaluation neighborhood clusters are evaluated and ordered: the best cluster concentrates in a small number of pages the largest number of links leading to new target pages
3. Recursion to catch index pages a new instance of our initial problem is triggered. Pages similar to the best neighborhood cluster are the target of the new problem. They will serve as indices over target pages of the original problem.
4. Iteration discovered index pages are used to fetch as many target pages as possible. Other neighborhood clusters are eventually considered.

Figure 4: Summary of the INDESIT algorithm

the highest score, which we call *index cluster*. The recursive invocation will return a set of *index pages* (possibly of different clusters) all similar to the pages of the index cluster. The link collections of these index pages with the same tag-list as the back class-links in the index cluster are followed to gather up as many target pages as possible.

Once these pages have been collected, the last step, called *Iteration*, considers the remaining neighborhood clusters. As after the last evaluation several pages have been downloaded, the scores can significantly decrease. Usually, in few iterations all the scores equals zero. Otherwise a new index cluster is chosen and the algorithm goes on along the same way.

Figure 4 describes a summary of the main steps performed by the INDESIT algorithm.

As a final remark about the INDESIT algorithm compared to TRIVIALSEARCH observe that INDESIT does not depend on the parameter $depth$ which, on the contrary, greatly affect TRIVIALSEARCH's performances. In some sense, INDESIT automatically recognize how far from target pages it needs to crawl.

3 Experimental Setting

We have implemented both the TRIVIALSEARCH and the INDESIT algorithms and have used them in order to conduct experiments on real-life web sites. The goal of the experiments was to evaluate the performances, in term of effectiveness and efficiency, of the algorithm. We now present the sites and the input sample pages that we have used in our experiments, and the metrics we use to evaluate the performance of our algorithms against this settings.

3.1 Sites and sample selection

As there are no analogous experiences in the literature, we could not rely on any reference test bed. We have therefore selected a bunch of sites, and for each of them we have identified one or two pages to be used as input samples. As our studies concentrate on large automatically built web sites, we have chosen sites for which there is evidence that their pages are generated by scripts. We selected sites from different domains and of different sizes (ranging from 5,000 to more than 30,000 pages). For all the sites we have built a local mirror. Since we aim at measuring the coverage of our output, we restricted our experiments to sites for which we were able to determine the size of the classes of searched pages. For some of the sites it is possible to derive this information as classes of pages describe real-world entities whose number is known; for example, we have used the official web sites of some relevant sport events for which the number of athletes and teams is given. For the other sites, as pages of these sites are generated by scripts, the urls of their pages contains the name of the script itself; therefore, we have counted on the file system storing the local mirror the number of files corresponding to each script.

For every site we have considered some sample page for our algorithms. We have chosen pages belonging to classes with different cardinalities. As we describe in Section 5, we have also run the algorithm in order to find pages of singleton classes. The description of the sites is given in Figure 5. Note that, for the Uefa2004 web site we have considered also the WAI (disable accessible) version. The peculiarity of this version is that it offers the same contents and the same site topology as the standard version, but the HTML code of its pages is extremely simple.

Site	Class	|Class|	Sample
fifaworldcup.yahoo.com	players	736	"Ronaldo"
	teams	32	"Turkey"
www.supercarsite.net	cars	294	"Ferrari 360 Modena"
www.sendit.com	dvds	1,011	"Mystic River"
www.olympic.org/uk/	sports	36	"Tennis"
	athletes	292	"Stefania Belmondo"
www.guitaretab.com	guitare tabs	29,032	"Smells like teen spirit tab"
	groups pages	3608	"Led Zeppelin tabs"
www.axl.co.uk	companies	3,744	"Imperial Chem. Industries"
www.nba.com	players	648	"Marc Jackson"
access.euro2004.com	teams	16	"Portugal - WAI"
www.euro2004.com	teams	16	"Portugal"

Figure 5: Sample pages and sites

Sample	TRIVIALSEARCH				INDESIT			
	R	P	#dwnl	#evl	R	P	#dwnl	#evl
"Ronaldo"	100%	100%	1,979	136,570	100%	100%	1,046	30,558
"Turkey"	100%	100%	2,255	88,021	100%	100%	249	2,299
"Ferrari 360 Modena"	89.46%	100%	1,292	4,857,198	100%	99.32%	351	1,816
"Mystic River"	90.9%	100%	3,403	57,868,141	85.95%	100%	3,377	1,274,934
"Tennis"	100%	100%	204	1,963	100%	100%	125	770
"Stefania Belmondo"	98.97%	100%	554	2,888,285	100%	100%	489	8,731
"Smells like teen spirit tab"	82%	100%	28,547	10,004,523	87.29%	100%	29,099	80,151
"Led Zeppelin tabs"	100%	10.16%	23,703	75,530,234	100%	95.98%	3,809	343,852
"Imperial Chem. Industries"	99.89%	96.77%	3,869	43,112,394	99.89%	98.09%	3,835	3,117,193
"Marc Jackson"	65.90%	24.04%	3,935	470,291	76.23%	100%	936	5,682
"Portugal - WAI"	100%	0.02%	811	27,207,334	100%	0.02%	755	820,468
"Portugal"	100%	100%	192	5,720	100%	100%	163	1,581

Figure 6: Experimental results

3.2 Metrics

The effectiveness of our approach can be measured in terms of precision and recall. Let C_i be the set of pages to be retrieved, and C_j the set of retrieved pages. Precision P, and recall R are defined as follows:

$$P = \frac{|C_i \cap C_j|}{|C_j|} \qquad R = \frac{|C_i \cap C_j|}{|C_i|}$$

As our algorithm aims at minimizing the pages to fetch, a straightforward measure of the efficiency is given by the total number of downloads. In addition, we observe that the algorithms can require to visit and compute over the same page more than once during the computation. Even if the page have been downloaded this influences the CPU workload. Therefore, we also report the number of visits ran over the downloaded pages.

4 Experimental Results

The experimental results of our evaluation are summarized in Figure 6, that illustrates the performances of both the TRIVIALSEARCH and the INDESIT algorithms. For each sample page, we report recall (R), precision (P), number of downloaded pages (#dwnl), and total number of visits performed over the downloaded pages (#evl).

As we can see, recall values are rather high with both the algorithms. This means that we are able to retrieve most of the pages with the same structure as the sample page. Also, the recall performed by the INDESIT algorithm usually outperforms that obtained by TRIVIALSEARCH (this is not true only for the "Mystic River"). The values of precision are more interesting. The precision is related to the number of false positive results produced by the search. In this

Site	Sample	P	#dwnl	#evl
FIFA02	homepage	100%	168	534
FIFA02	teams index	100%	120	260
SuperCarSite	homepage index	100%	47	92

Figure 7: Searching unique pages

case, we observe that the differences between the performances of the two algorithms are more accentuated. The most remarkable result is that of the "Led Zeppelin tabs" sample (10.16 vs. 95.98). It is worth noting that this is the sample for which we obtain also the best improvements in the number of downloaded pages (23,703 vs. 3,809). We comment on this saying that as the TRIVIALSEARCH algorithm visits a large portion of the site, it is more dependant on the threshold that we use to define the similarity among pages.

Let us now concentrate on the efficiency. A first observation is that the INDESIT requires the download of a small number of pages. This is particulary evident for samples belonging to small classes, such as, for example, "Turkey" whose class has 32 members (in a web site offering more than 20,000 pages). Whenever the performances of the two algorithms are close, such as for the "Imperial Chem. Industries" sample, it is due to the presence of a clear hierarchical organization of indices leading to the target pages. However, it is important to observe that the total number of visits performed over the downloaded pages is dramatically higher for the TRIVIALSEARCH algorithm. As in modern web sites pages are trickly connected one each other, the TRIVIALSEARCH performs a large number of traversals, passing more several times, through different paths, on the same pages. Conversely, the INDESIT algorithm is able to quickly learn the most promising navigation paths.

Figure 6 shows that the worst results are those obtained in the WAI version of the UEFA2004 web site. This is not surprising: as the HTML code of these page is extremely simple and uniform the algorithms are confused, as pages cannot be distinguished. The extremely low value of precision (0.02%) indicates that almost all the downloaded pages have been considered member of the target class. On the contrary, observe that in the standard version, INDESIT produces sharp results, with nice performances.

As a final experiment we have run the algorithms with an input sample page that is unique in the site. In particular we have used pages belonging to singleton classes, and pointed by all the pages of a site. The latter feature should push the algorithms to consider every page as a potential index. Figure 7 reports the results we have obtained. We only show the results of INDESIT, as TRIVIALSEARCH was stopped after hours of computations. These results confirm the ability of INDESIT in learning dead paths.

5 Related work

The issue of extracting information from the web has been the subject of several studies. Most of the work concentrates on the generation of wrappers (for a survey see, e.g. [18]). Early approaches was based on semi-automatic techniques [13, 12, 24, 17, 16, 14, 20]: taking as input a set of training pages and a set of labels marking data to extract, they generate a data extraction program. To alleviate the efforts of producing the training set, subsequent researches have focused on the development of sophisticated user interfaces to assist the programmer in building the training set for the target pages [4].

More recently, studies have focused on the automatic generation of wrappers. The ROADRUNNER system is based on a grammar inference approach [8, 9, 7]. Based on different techniques, systems for the same goal have been developed also by Arasu and Garcia-Molina [3] and by Wang and Lochovsky [26]. All these systems are based on the observation that data published in the pages of large sites usually come from a back-end database and are embedded within a common HTML template. Therefore the wrapper generation process consists of inferring a description of the common template.

With respect to the work presented in this paper, it is worth saying that both automatic and semi-automatic approaches assume that the pages containing the data have been collected by some external module (possibly they have been manually chosen).

In the information extraction field, another related work is that by Agichtein and Gravano, who have developed a method to identify from a large database documents that are promising for the extraction of a relation from text [2]. Their method requires a bunch of tuples representing samples of the relation to extract; these tuples are used by an information extraction system to retrieve a sample of documents from the database. Machine learning techniques are then applied over these documents to learn queries that will tend to match additional useful documents. Though this

approach is applicable on web documents as well, it does not exploit the access structures offered by web documents.

Our work is related to recent research for discovering the structure of web sites. Liu et al. model a web site as a hierarchical structure, whose nodes are either *content pages*, i.e. pages providing information contents, or *navigation pages*, i.e. pages containing links to content pages [19]. A combination of several domain independent heuristics is used to identify the most important set of links within each page. Based on these heuristics they have developed an algorithm to infer the hierarchical organization of navigational and content pages. Kao *et al.* [15] have studied methods to address a similar problem; they focus on news web sites with the objective of distinguishing pages containing indexes to news, and pages containing news. They propose a method, based on an analysis of the entropy of the hyperlink structure, to select the most informative links; a companion technique eliminates redundant information, such as navigational panels and advertisements. Compared to our approach, these proposals aims at finding paths to all generic content pages, while we aim at finding all pages of a specific class. In addition, the approach by Kao et al. is tailored for web sites of a specific domain (news).

In [10], a preliminary version of our page model has been adopted in order to infer an intentional description of a web site. In that work, a crawler navigates the web site starting from the home page and an agglomerative clustering algorithm groups pages into classes. The goal is to identify all the classes of pages offered by a site.

The notion of link collection is close to that of "pagelet", which was first introduced by Chackrabati et al. to denote a contiguous set of links [5]. In particular, Chackrabati et al. make the hypothesis that links from the same pagelet more tightly focus on a single topic than links from the entire page can do. This hypothesis has been exploited in subsequent works [6], with the goal of driving a crawler towards pages of a given topic. Compared to their work we may say that, in some sense, our crawler is specialized in recognizing structures, rather than topic-relevant pages. The notion of pagelet has been formalized by Bar-Yossef and Rajagopalan, who introduce a semantic and a syntactic definition, and an algorithm to extract pagelets from a web page, as well [27]. According to their syntactic definition, a pagelet is a DOM subtree such that none of its children contains more than k links, and none of its ancestor elements is a pagelet. We argue that such a definition is weak for our purposes, as it depends on a parameter (k) and limits the number of link that can be contained in a pagelet.

Our assumption that links from the same link collection point to related pages is reminiscent of the co-citation principle, first described in [23], which claims that the relationship between two document can be rated by the frequency at which they appear together in citation lists. In the context of the Web this principle has been exploited by several authors (e.g. [25, 11]): the overall idea is that two pages are likely to be related if they are linked by the same page. In the context of large web sites, we claim that, as sites are built automatically, this principle can be restated by saying that pages pointed by the same link collection are likely to be structurally homogeneous.

6 Conclusions

In this paper, we developed an automatic technique for retrieving web pages similar in structure to one only sample page. The output set of pages, as structurally homogeneous, can be done as input to a wrapper generator system. We have experimented our method over several real life web sites obtaining interesting results. The method has been designed in order to collect pages for a automatic web wrapper generator systems. We are currently integrating the presented techniques with ROADRUNNER, our web wrapper generation system. We believe that the presented techniques can allow for significant improvements in data extraction from the web. In fact, as completely automatic, the approach is extremely scalable.

Finally, it is worth saying that our approach cannot crawl the so called hidden web, i.e. pages from behind forms. These pages represent a large majority of the web, and they are suitable for wrapping, as they are usually generated automatically. In order to extract data also from these important sources, our system could cooperate with specific techniques, such as, for example, those proposed in [22, 21].

References

[1] Document Object Model (DOM) Level 1 specification. W3C Recommendation, October 1998. `http://www.w3.org/TR/REC-DOM-level-1`.

[2] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *19th International Conference on Data Engineering (ICDE03)*, 2003.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'2003), San Diego, California*, 2003.

[4] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *International Conf. on Very Large Data Bases (VLDB 2001), Roma, Italy, September 11-14*, pages 119–128, 2001.

[5] S. Chakrabarti, B. DOM, S. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Mining the web's link structure. *Computer*, 32(8):60–67, August 1999.

[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands)*, 31(11–16):1623–1640, 1999.

[7] V. Crescenzi and G. Mecca. Automatic information extraction from large web sites. *Journal of the ACM*, 2004. Accepted for publication.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. ROADRUNNER: Towards automatic data extraction from large Web sites. In *International Conf. on Very Large Data Bases (VLDB 2001), Roma, Italy, September 11-14*, 2001.

[9] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Automatic data extraction from data-intensive web sites. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'2002), Madison, Wisconsin*, 2002.

[10] V. Crescenzi, P. Merialdo, and P. Missier. Fine-grain web site structure discovery. In *Proceedings of the fifth ACM international workshop on Web information and data management*, pages 15–22. ACM Press, 2003.

[11] J. Dean and M. R. Henzinger. Finding related pages in the world wide web. *Computer Networks*, 31:1467–1479, 1999.

[12] D. Freitag. Information extraction from html: Application of a general learning approach. In *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, pages 517–523, 1998.

[13] T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[14] C. Hsu and M. Dung. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[15] H. Kao, S. Lin, J. Ho, and C. M.-S. Mining web informative structures and contents based on entropy analysis. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):41–44, January 2004.

[16] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.

[17] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1997.

[18] A. Laender, B. Ribeiro-Neto, A. Da Silva, and T. J. A brief survey of web data extraction tools. *ACM SIGMOD Record*, 31(2), June 2002.

[19] Z. Liu, W. K. Ng, and E.-P. Lim. An automated algorithm for extracting website skeleton. In *Proceedings of DASFAA 2004*, pages 799–811, 2004.

[20] I. Muslea, S. Minton, and C. A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 190–197, 1999.

[21] J. Palmieri, A. da Silva, P. Golgher, and A. Laender. Collecting hidden web pages for data extraction. In *ACM WIDM 2002*, 2002.

[22] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *International Conf. on Very Large Data Bases (VLDB 2001), Roma, Italy, September 11-14*, 2001.

[23] H. Small. Co-citation in the scientific literature: a new measure on the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):28–31, July-August 1973.

[24] S. Soderland. Learning information extraction rules for semistructured and free text. *Machine Learning*, 34(1–3):233–272, 1999.

[25] E. Spertus. Mining structural information on the web. In *Proceeding on 6th International World Wide Web Conference*, pages 587–595, 1997.

[26] J. Wang and F. Lochovsky. Data-rich section extraction from html pages. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering (WISE 2002), 12-14 December 2002, Singapore, Proceedings*, pages 313–322. IEEE Computer Society, 2002.

[27] Z. Ziv Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the eleventh international conference on World Wide Web*, pages 580–591. ACM Press, 2002.