Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 84 – 00146 Roma, Italy.

Discovering the structure of large web sites

Valter Crescenzi†, Paolo Merialdo†, Paolo Missier†

RT-DIA-89-2004 **February 2004**

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

http://www.dia.uniroma3.it

ABSTRACT

Several techniques have been recently proposed to automatically generate web wrappers, i.e., programs that extract data from HTML pages, and transform them into a more structured format, typically in XML. These techniques automatically induce a wrapper from a set of sample pages that share a common HTML template. An open issue, however, is how to collect suitable classes of sample pages to feed the wrapper inducer. Presently, the pages are chosen manually.

In this paper, we tackle the problem of automatically discovering the main classes of pages offered by a site by exploring only a small yet representative portion of it. We propose a model to describe the structure of a web site as a graph: nodes are classes of pages that share a common structure, edges represent links among instances of the page classes. Based on this model, we have developed an algorithm that accepts the url of an entry point to a target web site, visits a limited number of pages, and produces an accurate model of the site structure. We also report on experiments performed on actual web sites.

Contents

List of Figures

1 Introduction

A large number of web sites contain highly structured regions. The pages contained in these regions are generated automatically, either statically or dynamically, by programs that extract the data from a back-end database and embed them into an HTML template. As a consequence, pages generated by the same program exhibit common structure and layout, while differ in contents.

Based on this observation, several researchers have recently proposed techniques that leverage the structural similarities of pages from large web sites to automatically derive Web wrappers [8, 29, 3, 9, 30], i.e., programs that extract data from HTML pages, and transform them into a machine processable format, typically in XML. These techniques take as input a small set of sample pages exhibiting a common template; they generate as output a wrapper that can be used to extract the data from any page that share the same structure of the input samples.

Applying automatically generated wrappers on a large scale, i.e. to the structured portion of the Web, could anticipate some of the benefits advocated by the Semantic Web envision, because large amounts of data exposed throughout HTML web sites could become available to applications. For example, the financial data published by several specialized web sites only in HTML, could be constantly extracted and processed for mining purposes; data delivered on the web by thematic communities could be extracted and integrated. Also, some recent studies illustrate that the structured views over large web sites can improve the effectiveness and the efficiency of search engines [19].

However, automatically building wrappers for a large number of web site poses several issues. A first problem which significantly affects the scalability of the approaches based on wrappers is how to collect the sample pages to feed the wrapper generation system; this corresponds to identify clusters of structurally homogeneous sample pages; presently, sample pages are chosen manually. Also, once a library of wrappers for a web site have been generated, we have to choose which wrapper to apply over a target page. Finally, we need a mechanism to navigate the site in order to reach target classes of pages; therefore, we need a model to describe the navigational paths among the identified classes.

This paper addresses all these issues; we present a system that automatically discovers the main classes of pages offered by a site by exploring a small yet representative portion. The system produces a *model* of the site consisting of classes that are suitable for wrapping, as they exhibit the required structural uniformity. Also, the model includes features to describe the navigational relationships among classes of pages: given a link within a page that belongs to a certain class, the model specifies which class of pages the link points to.

This work is done in the framework of the RoadRunner project, which aims at studying methods and techniques for the extraction of data from the Web. However the approach we propose can be applied even with other automatic web wrapper generators, such as [3, 8, 29, 30].

1.1 Overview

We illustrate our approach by means of an example. Consider the official FIFA 2002 world cup web site,[1] whose roughly 20,000 pages contain information about teams, players, matches, and news. The site content is organized in a regular way; for example we find one page for each player, one page for each team, and so on. These pages are themselves well-structured, for instance all the player pages share the same structure and, at the intensional level, they present similar information (the name of the player, his current club, a short biography, etc.). Similarly, all team pages share a common structure and a common intensional information, which are different from those of the players. Also, pages contain links to one another, in order to provide effective navigation paths

[1] `http://fifaworldcup.yahoo.com`

that reflect semantic relationships; for example, every team page contains links to the pages of its players.

A key observation in our approach is that links reflect the regularity of the structure. To illustrate this property consider the web page of Figure 1, which is taken from the FIFA web site and presents information about a national team. Observe that links are grouped in collections with uniform layout and presentation properties; we call these groups *link collections*.[2] Usually, links in the same collection lead to similar pages. For example, the large table on the right side of the page contains links to player pages; the list located in the central part of the page has links to news pages. Also, we observe that these link collections are present in every team page. Analogous properties hold for the pages in Figure 2, which offer statistics about teams. Every stats page has a collection of links organized in the left most column of a large table; all these links point to player pages.

Based on these observations, we argue that:

- it is reasonable to assume that links that share layout and presentation properties usually point to pages that are structurally similar. In our example, in a team page, the large table on the right contains links all pointing to player pages, while links in the central list actually lead to news pages;

- the set of layout and presentation properties associated with the links of a page can be used to characterize the structure of the page itself. In other words, whenever two (or more) pages contain links that share the same layout and presentation properties, then it is likely that the two pages share the same structure. In the FIFA example, if two pages contain a link collection inside a large table on the right, and a collection of links inside a central list, then we may assume that the two pages are similar in structure (they look like the team page in Figure 1).

Our approach for discovering the structure of a web site relies on the above observations. We have designed and implemented an algorithm that builds the structure of a web site incrementally. The structure of a web site is modelled as a directed graph: nodes are classes of pages homogeneous in structure; arcs represent link collections connecting classes; every link collection is then characterized by a source class, a destination class, and the layout and presentation properties of the actual links it describes. Pages are described in terms of the link collections they offer, and the similarity of a group of pages is measured with respect to these features.

The algorithm starts from an entry point, such as the home page, whose (singleton) page class represents the initial model of the site. It then refine the model by iteratively exploring its boundaries. The model outbound links are followed, trying to exploit the properties of link collections. Pages reached from the same collection are assumed to form a page class; at the same time suitable techniques are applied for handling the situations where this assumption is violated.

1.2 Paper Outline

The remainder of the paper is organized as follows. Section 2 illustrates in detail our model. Section 3 describes the algorithm for exploring the site and building a site description. Section 4 reports the results of some experiments we have conducted over some real-life web sites. Section 5 discusses related works, and Section 6 concludes the paper.

[2]Link collections might be singleton.

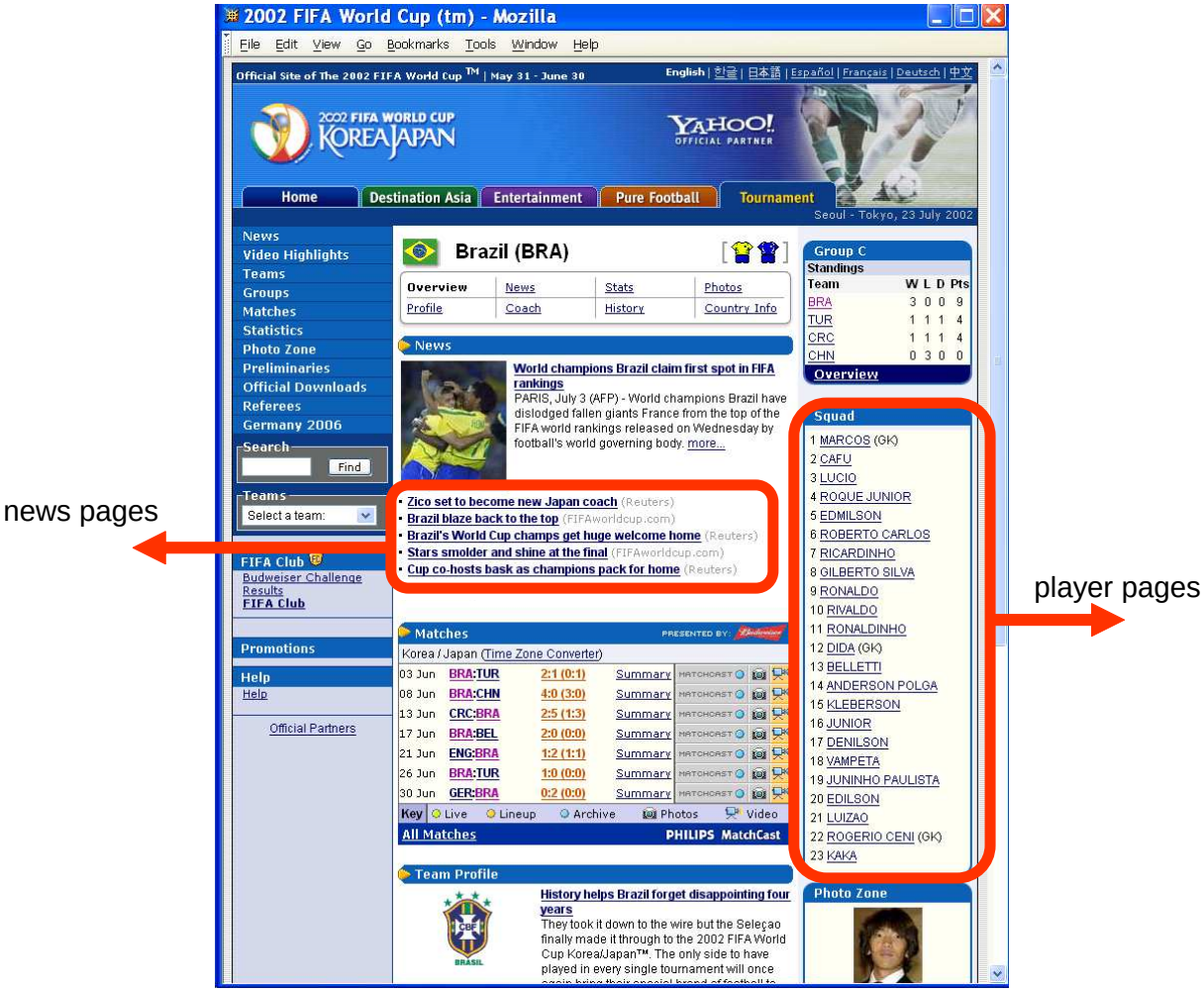

Figure 1: Example of HTML page: a team pages

2 Web site structure model

In this section we present our proposed model to abstract the structure of a web site, based on the main idea that layout and presentation properties associated with links can characterize the structure of a page.

For our purposes, a web page is represented by a subset of the root-to-link paths in the corresponding DOM tree representation [1], along with the referenced URLs themselves.[3] Figure 3 shows the portion of the DOM tree which would be considered for two sample pages. We call *link collection* one of such DOM root-to-link paths together with all the URLs that share that path. In our model a page can be described exclusively through its set of link collections. According to our definition, the page p_1 of Figure 3 has two link collections: `HTML-TABLE-TR-TD` $\{url_{B1}, url_{B2}\}$, `HTML-UL-LI` $\{url_{C1}, url_{C2}\}$; and the page p_2 has three link collections: `HTML-TABLE-TR-TD` $\{url_{B3}, url_{B4}, url_{B2}\}$, `HTML-B` $\{url_{B1}\}$, `HTML-UL-LI` $\{url_{C2}\}$.

We abstract the single pages by defining a *page schema* as a set of DOM root-to-link paths. Given a web page, its page schema can be trivially obtained by the corresponding DOM tree considering only the set of paths that, starting from the root, lead to links. Therefore, the schemas of the

[3] Actually, to exploit the richness of the presentation of modern web sites, we also consider the name and value of tag attributes. However, for the sake of simplicity, we will present examples including only elements.

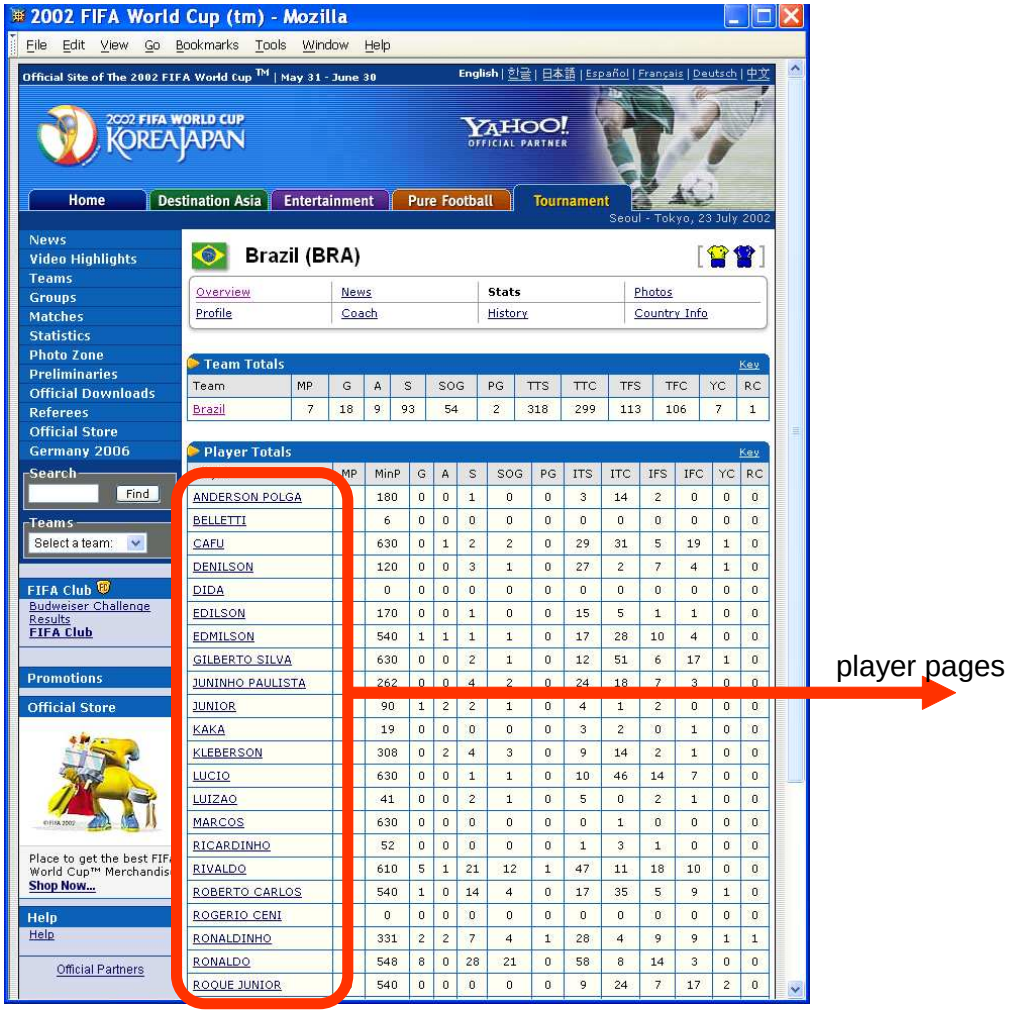

Figure 2: Examples of HTML pages: a stats pages

two sample pages in Figure 3 are: {`HTML-TABLE-TR-TD, HTML-UL-LI`} for p_1, and { `HTML-TABLE-TR-TD, HTML-B, HTML-UL-LI`} for p_2.

Based on the notion of page schemas we define a *page class* as a collection of pages; the schema of a page class is given by the union of the individual page schemas, i.e. by the union of the root-to-link paths of the pages participating the collection. The page class that includes the two sample pages of Figure 3, would then have the following schema: {`HTML-TABLE-TR-TD, HTML-UL-LI, HTML-B`}.

Intuitively, we expect that the page schemas for a set of structurally similar pages overlap largely, and thus the resulting class schema is not going to be much larger than the individual page schemas.

So far we have discussed the modelling of pages. Now we extend the model considering links. Given a page class C_1 and one of its DOM root-to-link path *path*, consider the link collections of the web pages in C_1 associated with *path*; we say that there exists a *class link* L between C_1 and the page class C_2 if there are links in the link collections associated with *path* that point to pages in C_2. Also, we say that *path* is the path of L.

A *site model* is a graph whose nodes are page classes, and whose directed arcs are class links.

The notion of class link is illustrated in Figure 4. Pages p_1 and p_2 are grouped together into a page class C_A, and pages p_3 and p_4 are members of a different page class C_B. Since the link

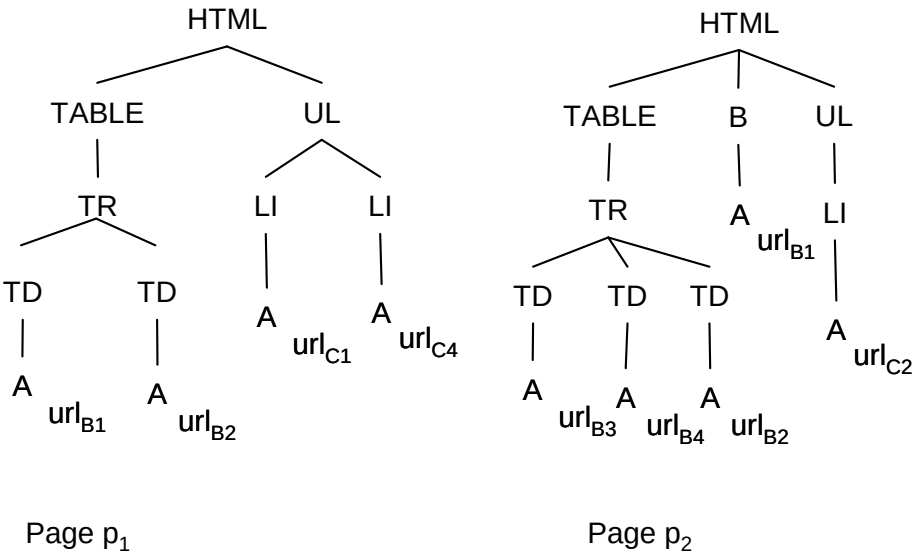

Figure 3: Sample pages

collections in C_A with path HTML-TABLE-TR-TD contain links that refer to pages in C_B, there exists a class link from C_A to C_B, and HTML-TABLE-TR-TD is its path.

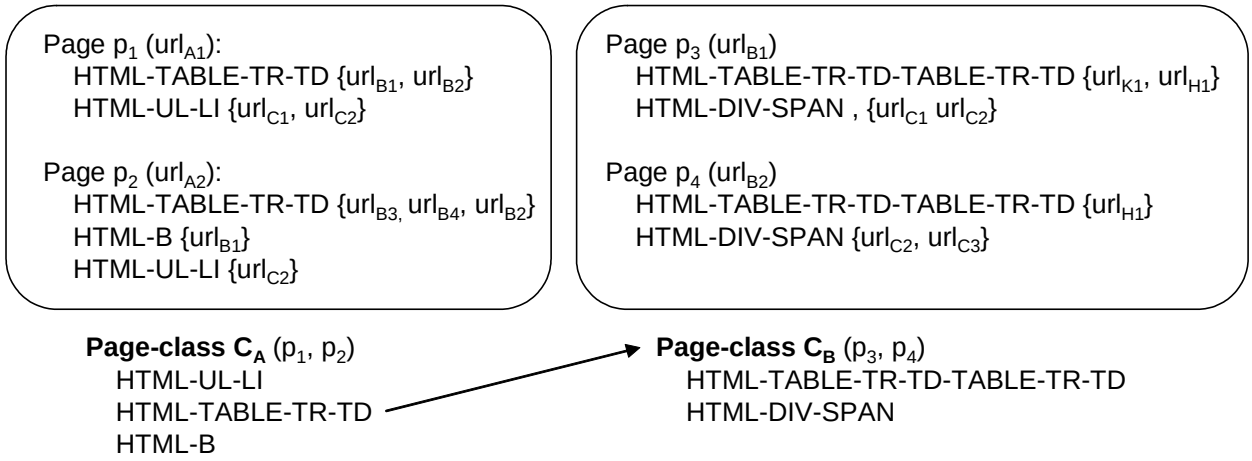

Figure 4: A class link

It is worth noting that a site model can actually describe the navigation among the classes. To illustrate this, consider the model in Figure 4: it says that for every page belonging to class C_A, the links having `HTML-TABLE-TR-TD` as path lead to pages in C_B.

Intuitively, a desirable model of the site structure is one in which classes represent a useful partition of the pages, such that pages in the same class are structurally homogeneous. In the next section, we address the issue of building a site-model by exploring only a fraction of the web site.

3 Site-Model Generation Algorithm

We have designed an algorithm that builds the site model incrementally, while crawling the site. The quality of the site model is evaluated with an information-theoretical approach based on the

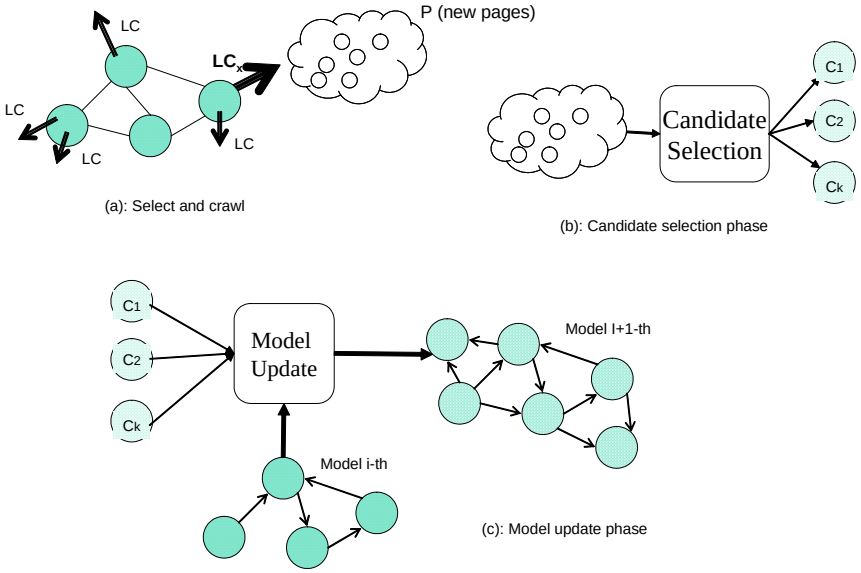

Figure 5: Main steps in the model-building algorithm

Minimum Description Length Principle (MDL) [26, 16].

The entry point to the site is a given seed page, which becomes the first member of the first class in the model. Its link collections are extracted, and pushed into a priority queue. At each iteration, the algorithm selects a link collection from the queue, and fetches a subset of the pages its links point to (Figure 5(a)). We follow only a subset of the potentially large set of links is sufficient to determine the homogeneity properties of the entire collection. The subset has a given minimal cardinality, which is a tunable parameter for the system.

The new *wave* of pages is then processed in two phases, as depicted in Figures 5(b) and 5(c).

- In the first phase, called *candidate selection*, pages are grouped according to their schemas. Groups are then analyzed and clustered into candidate classes; every candidate contains groups of pages having "similar" schemas. The goal of the analysis is twofold: on the one hand, we aim at grouping together pages with identical or slightly different schemas. On the other hand, we want to reveal heterogeneous link collections, such as those serving as menus, noting that these collections are normally characterized by links that lead to a large number of small (possibly singleton) candidate classes, whose schemas differ visibly from each other.

- In the second phase, which we call *model update*, the candidate classes are used to update the model. The goal of this phase is to determine whether each candidate class corresponds to a new class to be added to the model, or it should be merged with an already existing class.

At the end of the second phase, the algorithm has created a refined version of the model. Finally, the new collections of outgoing links from the wave of pages just classified are added to the priority queue and the next collection to visit is popped from the queue. The algorithm terminates when the queue is empty.

In the remainder of this section, we describe the two main phases of the algorithm in detail, and the heuristics to set the priorities of the collections in the queue.

3.1 Candidate Selection Phase

The input of the candidate selection phase is a set of pages obtained by following the links of a collection extracted from the queue. This phase partitions the input pages into a set of candidate

classes for the site model.

The pages are initially grouped according to their schema, obtaining a set of groups of pages with identical schema. Note that a totally homogeneous set of pages would yield a single group, and conversely, links collections to heterogeneous pages such as those contained in a menu, would result in several singleton groups. In general, we aim at obtaining a small number of candidate classes, each containing pages with identical or slightly different schemas. On account of these differences, we must determine when two groups should be merged into one single candidate class.

The idea is to define a similarity measure for pairs of groups, and to repeatedly collapse group pairs whose similarity is higher than a given threshold. Note that this is similar to clustering a set of points according to their relative distance, and indeed the technique we use is a variant of the well known *K-means* algorithm [12, 20].

To define similarity, we proceed as follows. Consider a reference schema S consisting of an ordered (for example, lexicographically) set of root-to-link paths $path_1, path_2, \ldots, path_n$. To each page p, we associate a bit vector $v(p)$ of length n, where $v(p)[i] = 1$ if $path_i \in S$, and 0 otherwise.

These bit vectors are used to represent the "projection" of a page schema onto a reference schema. Consider for instance a reference schema consisting of 16 paths, and the following vectors for two pages: `[0011000111111111]`, and `[1111111000000000]`. We see that the first page contains all paths in the schema but number 0,1,4,5, and 6 whereas the second page contains only the first seven paths.

Note that given a reference schema, a bit vector can be uniquely associated with each group. We define the similarity between two groups G_i and G_j as the ratio $sim(G_i, G_j) = h(G_i, G_j)/l$ where l is the number of paths of the schema of $G_i \cup G_j$ and $h(G_i, G_j)$ is the Hamming distance [17]. If the similarity $sim(G_i, G_j)$ is higher than a given *similarity threshold st*, then G_j is *collapsed* into G_i and it is removed from set of groups.

The idea is to let smaller groups collapse into larger groups, which are the "attractors", on the rationale that groups with many members tend to be authoritative compared to groups with only a few members. The algorithm produces a few large classes rather than a large number of smaller classes. This is accomplished by sorting the groups by size, and performing multiple collapsing steps between the larger and smaller group pairs.

To illustrate, consider a set of 20 pages, grouped into G_1, G_2 and G_3, with cardinalities 10, 5 and again 5, respectively. Also, assume that the bit vectors associated with the three groups are v_1[`1111111000000000`], v_2[`0011000111111110`], and v_3[`0011000111111111`]. Initially, groups with vectors v_1 and v_2 are compared: they differ in 13 bits, with distance 0.8. At threshold 0.5, the groups are not collapsed. Then, the pair of groups with vectors v_1 and v_3 are compared, again finding a distance 0.9 which is greater than the threshold. This means that the larger class does not attract any of the smaller classes. However, the two smaller groups are then compared with each other and collapsed into a single group, because their vectors, v_2[`0011000111111111`] and v_3[`0011000111111110`] differ by only 1 bit, with distance 0.06.

The two resulting groups are represented by the two vectors [1111111000000000] (the original vector), with cardinality 10, and the new group described by vector [0011000111111111], also with cardinality 10. In this example, which is taken from an actual run of the system, the merging strategy proves effective: the two original smaller groups consist of pages containing sport news, differing for only one link collection, and hence for a single bit in their vectors.[4] This accidental difference is detected and taken into account by the collapsing step, correctly producing a single group of news pages.

It is worth noting that the algorithm aims at collapsing large dominant groups. Whenever this does not occur, like in the case of menus (which are characterized by small groups with different

[4]News from *Reuters* have an additional link to the web site of the agency.

schemas), groups are kept well separated.

3.2 Model Update Phase

The candidate classes produced in the previous step must now be combined with the existing site model, to yield a refined model. The main issue is whether the candidate classes already appear in the model, or they do represent genuinely new classes. This decision process is modelled by generating a set of candidate models and then choosing the best among them. This decision is taken by following an information-theoretical approach based on the Minimum Description Length (MDL) principle [26, 16].

Roughly speaking, the MDL principle states that the best model to describe a set of data is the one which minimizes the sum of: (A) the length of a string of bits which encodes the model, and (B) the length of a string of bits which encodes the data with the help of the model. The shorter the description the better the model.

The two parts (A) and (B), which compose the MDL cost, represent model conciseness and precision, respectively. Part (A) is the number of bits required to describe the model and is thus related to its conciseness. Part (B) of the MDL cost captures a model's precision: the more precise the model, the more accurately it captures the common structure of the data it describes, with the consequence that fewer bits are required to encode the data according to the model. Therefore, the MDL principle implies a tradeoff between the conciseness and the precision of the model.

In our context, the data is the set of the visited pages, and the MDL principle can be used as an effective mechanism for quantifying how well a set of web pages fits in a site model. In order to apply the MDL principle, we have to define an encoding function specifically tailored to our framework.

Let us consider first part (A) of the encoding, which represents the cost of the model. In our context the model is a set of page class schemas, therefore they can be rewritten as the concatenation of the encoding of its members. In turn, since the schema of a page class is a set of DOM root-to-link paths, the encoding of a class schema is obtained by concatenating the encoding of its paths.

Consider for example a model consisting of two page classes $M = \{C_A, C_B\}$, and assume that C_A is associated with a schema having the three paths $\{path_1, path_2, path_3\}$, and that three pages are in $D = \{p_1, p_2, p_3\}$ such that $C_A = \{p_1\}$ and $C_B = \{p_2, p_3\}$. Then, $enc(M) = enc(C_A) \cdot enc(C_B)$.[5] In turn, the encoding of C_A can be expressed as $enc(C_A) = enc(path_1) \cdot enc(path_2) \cdot enc(path_3)$. Similarly for C_B.

Part (B) of the MDL cost represents the encoding of a set of instances D, with the help of the model M, and it is denoted $enc(D|M)$. In our framework, the instances are the fetched pages; as discussed in Section 2, we abstract a web page as a set of DOM root-to-link paths, together with the urls associated with each path. Therefore we rewrite $enc(D|M)$ in terms of the encoding of every page p in D with the help of the class C it belongs to: $enc(D|M) = enc(p_1|C_A) \cdot enc(p_2|C_B) \cdot enc(p_3|C_B)$. These terms can in turn be encoded as follows. For every path in the schema of the instance p of C, we encode either its index if that path is in C's schema, or directly the path if it is not. For any path, the urls are explicitly encoded. Also, we consider a special encoding, denoted $enc(\{\})$, to take into account paths that are present in the class schema but are missing in the page schema.

Considering again the previous example, suppose we have a page p structured as follows :
$p = \{path_1(url_{11}, url_{12}), path_3(url_{31}, url_{32}, url_{33}), path_4(url_{41}, url_{43})\}$.
Therefore, the encoding of p with the help of C_A, is:

$$\begin{aligned} enc(p|C_A) = \quad & enc(1) \cdot enc(\{url_{11}, url_{12}\}) \cdot enc(2) \cdot enc(\{\}) \cdot enc(3) \cdot \\ & enc(\{url_{31}, url_{32}, url_{33}\}) \cdot enc(path_4) \cdot enc(\{url_{41}, url_{43}\}) \end{aligned}$$

[5] $a \cdot b$ indicates the concatenation of strings a and b.

In order to compute the cost for this encoding, we assign weights to the different type of schema components, namely urls (c_u), paths (c_p), and indices (c_i), along with the cost of encoding a link whose path is not in the schema (c_{miss}). The cost of $enc(p|C)$ is the weighted sum of each component in the page.

Following our previous example, the cost of the schema of C_A is $1 \cdot c_p$. The cost of encoding p according to the class C_A is $3 \cdot c_i + 7 \cdot c_u + 1 \cdot c_p + 1 \cdot c_{miss}$.

With this MDL cost formulation, we may now describe the model updating algorithm in detail. For each candidate class C, and for each class C_i in the site model several alternative models are evaluated; namely, all those obtained by merging C with C_i plus a model in which C compare as novel class. The model with the lowest MDL cost is chosen.

3.3 Navigation Heuristics

Link collections are assigned a priority based on heuristics designed to generate a high-quality model by visiting the fewest possible pages. Heuristics may be used to pursue two contrasting goals. The first goal is to drive the crawler towards unexplored regions in the site, and seek the relatively few pages that are potentially representative of entire classes of pages. The resulting model is rich in nodes and arcs, but enjoys low *support* because it contains few instances. The second goal is to discover a portion of the model and then accumulate support for only that portion, by visiting pages that belong to the same classes with high probability.

For the same number of waves (or pages) visited, the two goals differ in terms of completeness of the model vs. the probability of finding new pages that do not fit with the model. In the first case, the model is more complete but the probability of non-fitting pages is higher than in the second case, where the crawling effort is spent on accumulating confidence in the model fragment.

In our algorithm, we have experimented with, and compared results for heuristics that cover both goals. The first, called *densest-first*, favors support-building by assigning higher priority to link collections that have many instances relative to the total number of outgoing links for a cluster. The reason is that long lists in a page are likely to point to pages with similar content (perhaps because they are generated by a program), and therefore the next wave will provide high support to their common class. Symmetrically, the *sparsest-first* strategy assigns high priority to the thinner link collections, in the hope that they may lead to new regions in the site. An extreme case is that of menu items, which usually belong to short collections. Regardless of the strategy chosen, collections in singleton classes are always assigned top priority.

4 Experiments

The algorithm has been implemented in a working prototype, which has been used to conduct several experiments. To tune the algorithm parameters we have run the system against a hand-crafted test sites with known structural properties. We then moved to two production Web sites, namely the FIFA 2002 World Cup web site, and the Davis Cup web site.[6] These sites have a complex hypertext structure; they contain 20,000 and 30,000 pages, respectively, organized in a small number of classes.

For each of the test sites, we have built a local mirror, then we have manually classified all the pages.[7] This hand-built classification was then used as *reference model* for all the experiments described in this section.

[6] http://fifaworldcup.yahoo.com, and http://www.daviscup.com, respectively

[7] The test sites were also selected with the criteria that manual classification was driven by regularities among the pages URLs.

We have conducted experiments in two main directions. First, we are interested in evaluating the quality of the final computed model. Second, we aim at tracking the quality of a model during its generation. Experiment settings, metrics and results are discussed in the remainder of this section.

4.1 Model Quality

We have prepared two experiments that make use of the computed model. In the first experiment we use the model to classify pages from the target web site; in the second experiment, we use the model to classify a navigation, i.e. a sequence of linked pages, over the site. The results of these experiments are evaluated by means of standard metrics.

Page Class Quality The goal of this experiment is to assess the quality of the classes of the computed model. We have therefore randomly chosen N pages from the target web site, and we have classified them by the help of the computed model. Classification is performed as follows: given a page p, the best class C for p is the one minimizing $enc(p|C)$.

Navigational Quality Although our site model is a graph, the previous experiment only focuses on the quality of page classes, i.e. graph nodes. In this second experiment, we assess the topology of the model, considering class links as well. The idea is to navigate the site starting from a known initial page p_0, and to map the path $p_0 \rightarrow p_1 \ldots \rightarrow p_n$ on the site, onto a path $C_0 \rightarrow \ldots \rightarrow C_m$ on the model. Recall that links in web pages are described according to their DOM root-to-link path. Assuming that the class C_0 of p_0 is known (e.g. p_0 is the home page), the traversal $p_0 \rightarrow p_1$ corresponds to identify all the class links starting from C_0 having the same DOM root-to-link path of the link collection leading to p_1. Consider the set of classes S_1 reached by these class links:[8] according to the model one of these is p_1's class. This process can be iterated taking the link collection from p_i to p_{i+1}, for $i = 1 \ldots n$. Among the set of classes S_n, we choose the best-fitting class $\hat{C}$ for the final page. We expect that $\hat{C}$ is the class for p_n.

4.1.1 Evaluation Metrics

We evaluate the results of our experiments by adopting the F-measure, which is computed in terms of precision and recall. Let $C_1, C_2, \ldots C_n$ be the set of classes from the reference model, and $\hat{C_1}, \hat{C_2}, \ldots, \hat{C_m}$ the set of classes from the computed model. Precision and recall for a class $\hat{C_j}$ with respect to a reference class C_i are defined as follows:

$$P(C_i, \hat{C_j}) = \frac{|C_i \cap \hat{C_j}|}{|\hat{C_j}|} \qquad R(C_i, \hat{C_j}) = \frac{|C_i \cap \hat{C_j}|}{|C_i|}$$

F-measure[9] is the harmonic mean of precision and recall:

$$F(C_i, \hat{C_j}) = max(\frac{2P(C_i, \hat{C_j})R(C_i, \hat{C_j})}{P(C_i, \hat{C_j}) + R(C_i, \hat{C_j})})$$

To compute these terms, for each of the above experiments, we organize our results in a *confusion matrix* as usual: the element e_{ij} contains the count of pages from class C_i that have been assigned

[8] Consider that several class links may share the same source class and the same path; this is what happens, for example, for class links that derived from a menu.

[9] The F-measure score is in the range [0,1]. A higher F-measure score implies a better performance. For a more in deep introduction and motivation of the F-measure, see e.g. [24, 27, 28].

to $\hat{C}_j$. From this matrix, precision and recall metrics of a computed class $\hat{C}_i$ with respect to a reference class C_j can be computed as follows:

$$P(C_i, \hat{C}_j) = \frac{e_{ij}}{\sum_{k=1}^{m} e_{ik}},\ R(C_i, \hat{C}_j) = \frac{e_{ij}}{\sum_{k=1}^{n} e_{kj}}$$

Finally, for each correct class C_i, the F-measure of that class is computed:

$$F_i = max_{j=1..m} \frac{2P(C_i, \hat{C}_j)R(C_i, \hat{C}_j)}{P(C_i, \hat{C}_j) + R(C_i, \hat{C}_j)}$$

We also compute the F-measure of the model, denoted F^*, to obtain a synthetic measure:

$$F^* = \sum_{i=1}^{n} F_i \cdot \frac{|C_i|}{|\cup_{k=1}^{n} C_k|}$$

4.1.2 Model Quality: Experimental results

We have run our prototype over both the Davis Cup and the FIFA World Cup web sites. Models for the two sites have been obtained by visiting only about 1,000 pages (as we will discuss in the remainder of this section this number of pages suffices).

For the page class quality experiment, we have tested the computed models against a set of 2,000 pages for each site. The test sets were generated by random sampling from the reference classification.

The test bed for the second experiment consists of 100 randomly generated navigational paths including 8 linked pages. Each path has been generated by randomly picking a link within a randomly chosen link collection of the home page; this process is then iterated from the page reached by such a link until a path of 8 pages is built.

Figures 7 and 8 report the main results. For each site we present a table summarizing the two experiments, and the values of the F^*. Each table contains: the name of the class, its support, i.e. the number of pages of that class in the web site, and the computed F-measures.

First, we observe that the two values of F^* are very close. Overall the results indicate that in both the experiments the most relevant classes (according to the support) are built correctly. The results obtained for the Davis Cup web site indicate a more accurate model. We comment on this, observing that the structure of the Davis Cup web site is simpler than that of the FIFA World Cup web site, since it contains a larger number of pages, but a smaller number of classes.

Let us comment the main errors. In the FIFA web site, the worst values of F correspond to a limited number of classes, namely: 2006contact, friendlies, coach, teamCountryInfo, teamHistory, teamProfile. The first two classes have a very small support (the first one is even a singleton). The other three classes are more interesting and the errors can be interpreted looking at the details of the pages. In fact, the pages of classes teamCountryInfo, teamHistory, teamProfile are quite similar according to our model; in particular they do not contain links except those common to every page in the site. A similar observation occurs also for the coach class, whose pages are very similar to those of players (the class of players performs better because of their larger support). Similar argumentation can be observed for classes in the Davis site.

In the results of the navigation quality evaluation it is interesting to observe that several singleton classes are involved. This is due to the fact that building our path we randomly choose links; then links that appear in all the pages of the site have a higher probability to be followed; usually these links lead to singleton classes, such as for example, the home page class and the copyright class. Clearly these page are reached following different paths; nevertheless, we do not observe any remarkable error among these classes.

4.2 Tracking Model Quality in Incremental Generation

In this experiment we aim at tracking the quality of the model during its generation, and how quickly, in terms of number of fetched pages, the algorithm converges towards a final result.

Since during its generation the model is incomplete, and the class supports are not reliable, standard quality measures do not apply suitably. Therefore we base the quality evaluation on the similarity between the schema of a reference model and that of the on going computed model.

In our context schemas are sets (of paths). In order to evaluate how well the schemas of the classes in the current computed model match with those in the reference model we adapt the F-measure, applying it directly on the schemas.

Let σ_p, σ_C be the schemas for page p and optimal class C, respectively. We measure the extent of overlap between σ_p and σ_C in terms of precision and recall, as $R(\sigma_p, \sigma_C) = \mid \sigma_p \cap \sigma_C \mid / \mid \sigma_C \mid$ and $P(\sigma_p, \sigma_C) = \mid \sigma_p \cap \sigma_C \mid / \mid \sigma_p \mid$. Then:

$$F(\sigma_p, \sigma_C) = 2\frac{P(\sigma_p, \sigma_C)\ R(\sigma_p, \sigma_C)}{P(\sigma_p, \sigma_C) + R(\sigma_p, \sigma_C)}$$

We use $F(\sigma_p, \sigma_C)$ as an elementary measure of similarity between a single page and a class. By extension, we measure the similarity between a computed class $\hat{C}$ containing pages $p_1, \ldots, p_k$, and the schema of a reference class C as the average similarity over each $p \in \hat{C}$:

$$sim(C, \hat{C}) = \frac{\sum_{p \in \hat{C}} F(\sigma_p, \sigma_C)}{k}$$

The overall similarity between the classes $\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_m$ of the current computed model, and the classes $C_1, C_2, \ldots, C_n$ of the reference model, is computed as:

$$F = \frac{\sum_{i=1}^{n} max_{\hat{C}_j}\{sim(C_i, \hat{C}_j)\}}{n}$$

However, observe that by proceeding from the computed classes to reference classes in the similarity matching, the "asymmetric" use of F alone would tolerate the introduction into the model of spurious classes that are not similar to any reference class. Also, a single comprehensive class with a schema that trivially includes all the paths in the reference classes would receive a high score.

A complementary function, called $RevF$, is introduced to remedy these shortcomings. For each computed class, $RevF$ finds its most similar reference class:

$$RevF = \frac{\sum_{j=1}^{m} max_{C_i}\{sim(C_i, \hat{C}_j)\}}{m}$$

In order to achieve a balance that rewards computed classes that are both relevant and at the same level of granularity as the reference classes, we combine F with $RevF$ into the harmonic average Q:

$$Q = 2\frac{F\ \ RevF}{F + RevF}$$

We use Q for tracking the quality of the model during its generation.

4.3 Incremental Generation: Experimental Results

The main results of our experiments over the FIFA web site are summarized in Figure 9, where the quality Q of the computed model is plotted against the number of visited pages during a run. The main interesting result is that both densest-first and sparsest-first strategies produce a satisfactory model after visiting only a small fraction of the site, with the former strategy performing slightly better. At each iteration, we identify a discovery phase during which the queue fills up with a new wave of links to follow, up to a maximum that corresponds to the model reaching some stability. Beyond this point, the algorithm is not discovering new structure, and the time is spent to increase the support for the existing model.

These results[10] suggest that the termination condition used in our algorithm (empty queue) may actually be too pessimistic. A more aggressive condition would look for the max in queue size, although we have not shown that this is a sufficient condition in the general case.

5 Related work

Web site modeling for data extraction purposes

The issue of modeling the logical structure of web sites for extraction purposes has been studied in several research projects. A pioneering approach is that proposed in the ARANEUS project [5, 6], where a web page is considered as an object with an identifier (the URL) and a set of attributes. The notion of *page scheme* is then introduced to model sets of homogeneous pages. Attributes of a page may have simple or complex type. Simple attributes correspond essentially to text, images or links to other pages. Complex attributes model possibly nested collections (lists) of objects. Also, the model includes a heterogeneous union type, which can be used to describe links pointing to a disjunction of page-schemes. To extract data from web pages, a wrapper is associated to each page scheme. Wrappers in ARANEUS are built either by means of a procedural language, called *Cut and Paste* [25], or adopting *Minerva* [25], a formalism and a tool for rapidly writing wrappers in a more declarative fashion.

In *WebOQL* [4], Arocena and Mendelzon describe web data according to an object data model, and thus propose a language (*WebOQL*) for extracting and querying the web. The language allows users to pose queries in a SQL-like fashion, and includes features to specify a mapping between data from the actual pages and the logical constructs of the underlying data model.

A more recent contribution is that of the *Wiccap* project [21]. In *Wiccap*, web data are mapped onto a hierarchical logical structure. The focus is on the usability of the model: the goal is to map information from a web site into a logical organization of concepts, as they would be perceived by ordinary users. Nodes of the target structure can then contain data extracted from several pages, integrated to compose a uniform concept. Nodes are associated with mapping rules, i.e. primitives that extract data from the physical structure and map them onto the target model. To ease the burden of creating the hierarchical view over a target web site, a suite of visual tools, called *Mapping Wizard*, has been developed [22].

Classification of web documents

The issue of classifying HTML web pages according to their structure has been recently addressed in [10, 13]. Both the approaches developed in [10] and [13] take as input a large set of HTML (or XML) pages, and create clusters of pages based on properties related to the frequency and the distribution of tags.

[10] As reported in [11], a similar situation is obtained running the system over a different web site.

Bertino et al. have proposed an approach to classify XML documents. They develop a matching algorithm for measuring the structural similarity between an XML document and a given DTD. The algorithm compares the structure on the document against those on the DTD. Commonalities and differences are then evaluated to compute a numerical rank that represents the structural similarity. The matching algorithm is then exploited for the classification of XML documents against a set of DTDs.

The use of the MDL principle [26, 16] for model selection in the web context is not new. In [14], Garofalakis et al. address the problem of inferring a reasonable DTD from a collection of XML documents; the MDL principle is used to choose a high quality DTD from a set of candidates inferred DTD. A similar technique has been proposed by Hong and Clark in order to infer wrappers for web sources [18].

Web site structure inference

In [23], Liu et al. have developed an algorithm, called *SEW*, for discovering the *skeleton* of a target web site. The skeleton here refers to the hypertextual structure throughout the delivered contents are organized. It is assumed that a skeleton is a hierarchical structure whose nodes are either *content pages* or *navigation pages*. The former are those pages providing information contents; the latter are pages containing links to content pages. The *SEW* algorithm aims at automatically discovering the hierarchical organization of navigational and content pages. It relies on a combination of several domain independent heuristics to identify the most important set of links within each page. Compared to our approach, *SEW* only distinguishes two predefined classes of pages, navigational vs. content pages, whereas we aim at classifying pages according to their structure, without any a priori assumption about the number of classes and the exposed features.

A similar problem has been studied also by Kao et al. [19], who have developed a technique for analyzing news web sites. The goal of their proposal is to identify, within a news web site, pages of indexes to news, and pages containing news. An entropy based analysis is performed to eliminate the redundancy of the hyperlinked structure, thus distilling its complexity. A companion technique, also based on entropy analysis, eliminates redundant information, such as navigational panels and advertisements. As the authors argue, the most interesting field of application for this technique is that of search engines. Since their system is able to polish the link structure and the contents of news web sites, their approach reaches higher performances than conventional methods.

Another field that is related to our research is that of focused-crawling [7]. The goal of a focused crawler is to selectively seek out pages that are relevant to a pre-defined set of topics possibly from many different web sites. However, our goals are different: first, in some sense, our crawler is focused on recognizing structure rather than looking for topic-relevant pages, and second, we aim at discovering the complete structure of one site by visiting only a small yet representative portion of its pages.

6 Conclusions and further work

In this paper we present an algorithm to infer a model of the structure of a web site, by visiting a small yet representative number of pages. The output model describes the structure of the site in terms of classes of pages and links among them. The algorithm aims at producing classes whose pages are structurally homogeneous. The structural similarity among pages is defined with respect to their DOM trees.

The model we produce can be used for several purposes. First, for each class of pages in the model we can generate a wrapper: the visited pages which have been grouped into one class can be used as input samples for automatic wrapper generator systems. Once a wrapper for each class has

been built, the model can be used to determine which wrapper has to be applied against a given page. Also, the model can be navigated; for example, we can target the extraction of data for all the pages that belong to a given class: the model describes the navigational paths that lead to the pages of the target class.

We are currently applying this approach in the framework of the RoadRunner project. Combining the capabilities of our RoadRunner wrapper generator [9] with the presented approach, we have developed a web data grabber [2], that is, a system that automatically explores and extracts informative contents from data intensive web sites.

An interesting idea we are currently exploring is that of associating semantics with the discovered classes. So far, we produce anonymous classes. Pages in each class are related because of their common structure. We have observed that in many cases, within a web site, each class is associated with specific concepts. Techniques that aim at classifying pages by analyzing their contents like those proposed, for example, in [15] could be complemented with our approach.

References

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

[2] L. Arlotta and V. Crescenzi and G. Mecca and P. Merialdo and P. Missier The RoadRunner Data Grabber for Web Sources. Submitted for publication.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD 2003*, 2003.

[4] G. O. Arocena and A. O. Mendelzon. Weboql: Restructuring documents, databases, and webs. *TAPOS - Theory and Practice of Object Systems*, 5(3):127–141, 1999.

[5] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *International Conf. on Very Large Data Bases (VLDB'97), Athens, Greece, August 26-29*, pages 206–215, 1997.

[6] P. Atzeni, G. Mecca, and Merialdo P. Managing web-based data: Database models and transformations. *IEEE Internet Computing*, 6(4):33–37, 2002.

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

[8] Chia-Hui Chang. Iepad: Information extraction based on pattern discovery. In *Proceedings of the tenth international conference on World Wide Web*, 2001.

[9] 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.

[10] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Wrapping-oriented classification of web pages. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 1108–1112. ACM Press, 2002.

[11] Valter Crescenzi, Paolo Merialdo, and Paolo 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.

[12] A.K. Dubes, R.C.and Jain. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[13] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between xml documents. In *Proceedings of the Int. Workshop on The Web and Databases (WebDB 2002)*, pages 55–60, 2002.

[14] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from xml documents. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'2000), Dallas, Texas*, 2000.

[15] E.J. Glover, K. Tsioutsiouliklis, S. Lawrence, D.M. Pennock, and G.W. Flake. Using Web structure for classifying and describing Web pages. In *Proceedings of WWW-02, International Conference on the World Wide Web*, 2002.

[16] P. Grunwald. *The Minimum Description Length Principle and Reasoning under Uncertainty*. PhD thesis, ILLC 1998-2003, University of Amsterdam, 1998.

[17] Hamming R. *Coding and Information Theory*. Prentice-Hall, 1980.

[18] T. W. Hong and K. L. Clark. Using grammatical inference to automate information extraction from the Web. *Lecture Notes in Computer Science*, 2168:216+, 2001.

[19] H.-Y. Kao, S.-H Lin, J.-M. Ho, and Chen 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.

[20] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, 1999.

[21] Zehua Liu, Feifei Li, and Wee Keong Ng. Wiccap Data Model: Mapping Physical Websites to Logical Views. In *Proceedings of the 21st International Conference on Conceptual Modelling (ER2002)*, Tempere, Finland, October 7-11 2002.

[22] Zehua Liu, Wee Keong Ng, Feifei Li, and Ee-Peng Lim. A visual tool for building logical data models of websites. In *Proceedings of the fourth international workshop on Web information and data management*, pages 92–95. ACM Press, 2002.

[23] Zehua Liu, Wee Keong Ng, and Ee-Peng Lim. An automated algorithm for extracting website skeleton. In *Proceedings of DASFAA 2004*, pages 799–811, 2004.

[24] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction, 1999.

[25] G. Mecca and P. Atzeni. Cut and Paste. *Journal of Computing and System Sciences*, 58, 99.

[26] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[27] M. Steinbach, G. Karypis, and Kumar V. A comparison of document clustering techniques. In *Proceeding on KDD Workshop on Text Mining*, 2000.

[28] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[29] J Wang and F.H. 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.

[30] J. Wang and F. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003.*, 2003.

Algorithm ComputeModel
Parameter: n maximum number of pages to download from one link collection
Parameter: st similarity threshold
Input: l_0, a seed link to start off the site navigation;
Output: the site model M as a set of page classes C;
begin
  Let $\mathcal{M} = \emptyset$; // set of page classes
  Let $Q = \{l_0\}$; // priority queue of link collections
  while (Q is not empty) {
    extract the highest priority link collection lc from Q;
    fetch all but no more than n pages pointed by lc into a set w;

    //Candidate Class Selection
    Let $H = (G_1, \ldots, G_k)$ **be** the pages in w grouped by schema and ordered by cardinality;
    while ($\exists i, j$ such that $i \neq j$ and $sim(G_i, G_j) < st$) **do**
      choose the highest index h and the lowest index k such that $h \neq k$ and $sim(G_h, G_k) < st$;
      collapse G_h and G_k into a new group G;
      replace G_h, G_k with G in H;
    end
    Let $S = (C_1, \ldots, C_g)$ **be** the resulting candidate classes in H ordered by cardinality;

    //Model Update
    for each $C \in S$ **do**
      consider all $M' = (M - \{C'\}) \cup \{C \cup C'\}$ for any $C' \in M$;
      Let M_{merge} **be** M' such that MDL(M') is minimized;
      $M_{new} = M \cup \{C\}$;
      if (MDL(M') $<$ MDL(M'')) $M = M_{merge}$;*//merge class*
      else $M = M_{new}$; *//create new class*
    done;
    add to Q the new links collections from w;
  done;
  return M;
end

Function MDL(Model M) { return the mdl cost of M as model of all pages visited; }

Figure 6: The Compute-Model algorithm

Page Class Quality

Class Name	|C|	F
headtohead	8421	0.99
headtohead-info	458	1
interviewaudio	77	0.8
matchReport	87	0.5
mediaGallery	653	0.98
newsArticle	96	0.57
newsIndex	17	1
photoGallery	927	1
player	8173	1.00
playerWinloss	152	1
team	156	0.87
teamWinloss	152	1
tie	5233	0.99
tieResult	5164	0.97
tvSchedule	22	1

F*=0.984

Navigation Quality

Class Name	|C|	F
feedback	4	1
headtohead	8421	1
homepage	1	1
matchReport	87	0.22
newsArticle	96	0.93
photoGallery	927	0.9
player	8173	0.95
registration	3	1
team	156	1
tie	5233	1
tieResult	5164	1
about	1	0.64
galleryArchiveIndex	1	1
contact	1	0.66
faq	1	1
legal	1	0.66
terms	1	0.64

F*=0.979

Figure 7: Results for `www.daviscup.com`

Page Class Quality

Class Name	|C|	F
2002preliminaries	39	1
2006contact	1	0.2
calendar	1	1
classicGame	10	0.67
classicPlayer	16	0.67
coach	32	0.08
mainSponsorRanking	8	1
destinationAsiaCityOverview	20	1
destinationAsiaCityPhotoGallery	40	0.67
destinationAsiaDayInTheLife	28	1
destinationAsiaDayInTheLifeIndex	3	1
destinationAsiaVenue360	20	1
variousDescription	11	0.67
france98	361	0.95
friendlies	4	0.00
groupNews	8	1
match	64	1
matchGallery	218	0.98
matchNews	63	1
matchReport	580	1
mediaCenter	1	1
news	4333	0.99
newsPf	4320	0.79
newsPhoto	8068	0.81
photoGallery	14	1
player	736	0.89
previousWorldCupPhotoGallery	15	0.86
pureFootballGamesInex	1	1
referee	73	1
stats	57	0.86
team	32	1
teamCountryInfo	32	0.06
teamHistory	32	0.11
teamNews	32	1
teamPhotoGallery	326	1
teamProfile	32	0.08
teamsIndex	1	1
teamStats	32	1

F*=0.863

Navigation Quality

Class Name	|C|	F
2002photogallery	1	1
2006City	12	1
2006description	12	1
classicGame	10	1
destinationAsia	2	0.66
destinationAsiaCityOverview	20	0.85
destinationAsiaIndex	1	1
destinationAsiaWelcome	3	0.66
downloads	1	1
goalOfCentury	1	0.5
group	8	0.4
groupIndex	1	0.66
home	1	1
match	64	1
matchIndex	1	1
matchReport	580	1
misc	1	0.47
news	4333	1
newsPhoto	8068	0.8
player	736	0.57
previousWorldCupIndex	1	0.57
privacy	1	0.63
pureFootballIndex	1	0.75
rimetCup	1	0.26
sponsor	1	1
team	32	0.92
teamAward	1	1
teamHistory	32	0.28
teamPhotoGallery	326	1
terms	1	0.37
tournament	1	1
variousDescription	11	0.88

F*=0.861

Figure 8: Results for `fifaworldcup.yahoo.com`

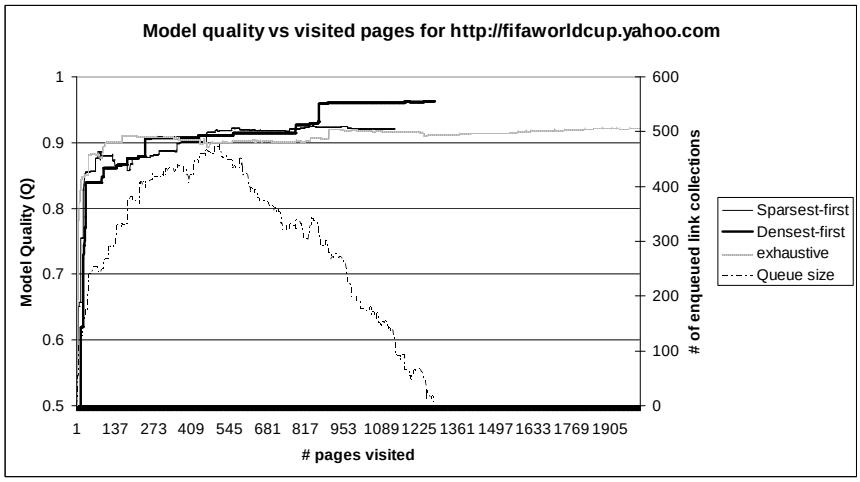

Figure 9: Quality evaluation for FIFA