

Top-down Extraction of Semi-Structured Data

Berthier Ribeiro-Neto Alberto H. F. Laender Altigran S. da Silva*

Department of Computer Science
Federal University of Minas Gerais
31270-970 Belo Horizonte MG Brazil
{berthier,laender,alti}@dcc.ufmg.br

Abstract

In this paper, we propose an innovative approach to extracting semi-structured data from Web sources. The idea is to collect a couple of example objects from the user and to use this information to extract new objects from new pages or texts. We propose a top-down strategy that extracts complex objects decomposing them in objects less complex, until atomic objects have been extracted. Through experimentation, we demonstrate that with a small number of given examples our strategy is able to extract most of the objects present in a Web source given as input.

1 Introduction

As the data on the Web grows at explosive rates, a tremendous research effort has been initiated to make such data available, usually in some structured form such as a table, for querying and further manipulation. The main motivation is that structured data allow asking queries which cannot be asked with data in text form. A typical example of such a query is: retrieve all hotels in downtown New York with daily rate smaller than US\$ 120.00.

In this work, we discuss an innovative example-based approach to extracting data from a set of Web pages for populating a database. We consider that the Web pages present some inherent structure which can be readily recognized. Since this structure appears implicitly on the pages and might vary from one page to another, we say that the data is *semi-structured* [1].

To extract the data, we need some description of what to extract. A common approach to providing such description is to build a specific grammar which details the surroundings of each piece of data to extract [5]. In this work, however, we consider a new approach in which the description

of what to extract is fully based on a small set of examples provided by the user. Based on this idea, we propose a top-down strategy that extracts complex objects decomposing them in objects less complex, until atomic objects have been extracted. Through experimentation, we demonstrate that just a couple of examples are sufficient for extracting hundreds of objects from new Web pages. Our approach is simple, intuitively appealing, quite effective, and does not suffer from the main drawbacks of alternative approaches in the literature, as discussed below.

A number of different approaches have been proposed to extracting data from Web sources [2, 4, 5, 10, 11, 15, 17]. The most common of such approaches involves the use of wrappers [2, 4, 5, 11]. Although wrappers provide an effective approach to data extraction from Web sources, they have two major drawbacks. Firstly, they require a previous knowledge of the structure of the data source. Secondly, additional work might be required to adapt the wrapper when the source changes.

An alternative approach to extracting data from Web sources is based on natural language processing (NLP). In such an approach, NLP techniques are used to find relevant fragments that can be extracted from a source document [8]. The system described in [17] is an example of a tool that uses such techniques. Extraction tools based on NLP techniques are effective, but they are very specific and usually demand a large number of training examples.

An ontology-based approach to extracting data from Web sources is presented in [9, 10]. This approach uses a semantic data model to provide an ontology that describes the data of interest and its location on the source pages, including relationships, lexical appearances, and context keywords. Although this approach can be quite effective with the so-called data rich and narrow in ontological breadth Web sources [10], it suffers from problems similar to those faced by wrapper-based approaches in the sense that it requires a previous knowledge of the data source.

Finally, a more specific but very interesting approach

*On leave from University of Amazonas, Brazil.

to extracting tabular data from Web sources is provided by TINTIN (Table INformation-based Text INquiry) [15]. This tool extracts tabular data from unstructured documents based on a purely structural analysis of such documents. Heuristics are used to recognize the headings and the values that compose the different columns of a table.

The paper is organized as follows. In Section 2, we introduce the problem of data extraction in more detail. In section 3, we discuss how to generate automatically a text context for data selected by the user. Section 4 presents our example-based extraction strategy. Section 5 discusses our experimental results. Our conclusions follow.

2 The Problem

Consider a portion of a Web page obtained from the DB&LP site [14] as illustrated in Figure 1. We notice that there is an inherent structure to the text on the page. For instance, we are able to distinguish four *objects* and, for each object, we identify attributes such as authors and title. Such structure has not been declared anywhere but is clearly identifiable. Texts or pages which present such type of inherent

Volume 20, Number 3, September 1995

- Weidong Chen: *Query Evaluation in Deductive Databases with Alternating Fixpoint Semantics*. 239-297, *Electronic Edition* (link)
- Yannis E. Ioannidis, Raghuram Ramakrishnan: *Containment of Conjunctive Queries: Beyond Relations as Sets*. 288-324, *Electronic Edition* (link)
- Dennis Shasha, François Liribat, Eric Simon, Patrick Valduriez: *Transaction Chopping: Algorithms and Performance Studies*. 325-363, *Electronic Edition* (link)
- Stefano Ceri, Piero Fraternali, Stefano Paraboschi, Letizia Tanca: *Addendum to "Automatic Generation of Production Rules for Integrity Maintenance"*. 364, *Electronic Edition* (link), see *TODS* 19(3): 367-422 (1994)

Figure 1. Web page from the DB&LP site illustrating a data rich example (ACM TODS).

structure are said to be *data rich* and *narrow in ontological breadth* [10]. Such pages constitute the target of our study and are referred to simply as data rich pages.

Given a set of data rich pages, we investigate how to extract (from them) objects and their attributes such that they can be inserted into (nested) tables for later querying. If properly done, this would allow retrieving information which cannot be obtained with standard text searching techniques. For instance, in Figure 1, one might be interested in all the titles which have been published by a given author alone and which are longer than 20 pages. Notice that this information is present on the page but cannot be obtained using standard text retrieval techniques.

To be able to extract information from a set of data rich pages, we need some type of description of what to ex-

tract. For instance, we could assume the existence of a grammar detailing how to parse and recognize tokens for insertion on a table. In the example of Figure 1, such grammar could specify that names of authors appear after a black dot and are separated by commas. Further, the grammar could state that the title appears in the line immediately after the line containing names of authors. The main weakness of grammar-based approaches is that they are too rigid for processing typical text which appear in practical situations (particularly, in the Web). For instance, an entry might be missing the black dot which identifies the object, might be missing information on authors, or might misplace the information on the title such that it appears prior to the information on the authors. To deal with such situations, the designer of the grammar would have to anticipate which exceptions could occur in practice and adapt the grammar accordingly. Clearly, such task might be quite hard in loose domains such as the Web.

In this work, we take a different approach and assume that an well informed user simply specifies examples of objects to extract. Using these examples, we study how to extract the data from new pages (with similar structure) in the presence of mismatches, variation on the ordering of attributes, and an inconsistent (implicit) structure. The user provides such examples by cutting pieces of data from the page and assembling an example object. Using such example object, we investigate how to process new pages, recognize objects in them, extract this objects, and insert them on a table for later querying. We expect that a couple of example objects should suffice to allow processing hundreds of new (but similar in structure) pages.

We consider that the example object the user provides is a complex object with a hierarchical structure. For instance, for the Web page in Figure 1, the user could specify a 2-level hierarchical example as illustrated in Figure 2. In this case,

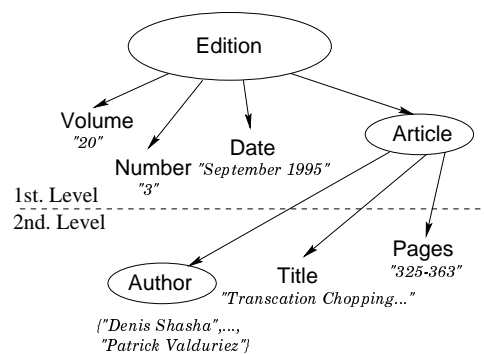


Figure 2. Object provided as example for the Web page in Figure 1

the user specified a single example of a paper with 4 au-

Edition															
Volume	Number	Date	Article												
20	3	September 1995	<table border="1"> <thead> <tr> <th>Author</th> <th>Title</th> <th>Pages</th> </tr> </thead> <tbody> <tr> <td>{Denis Shasha, . . ., Patrick Valduriez}</td> <td>Transaction. . .</td> <td>325-363</td> </tr> <tr> <td>{Widong Chen}</td> <td>Query Evaluation. . .</td> <td>239-297</td> </tr> <tr> <td>. . .</td> <td>. . .</td> <td>. . .</td> </tr> </tbody> </table>	Author	Title	Pages	{Denis Shasha, . . ., Patrick Valduriez}	Transaction. . .	325-363	{Widong Chen}	Query Evaluation. . .	239-297
			Author	Title	Pages										
			{Denis Shasha, . . ., Patrick Valduriez}	Transaction. . .	325-363										
{Widong Chen}	Query Evaluation. . .	239-297													
.													
20	4	December 1995	<table border="1"> <thead> <tr> <th>Author</th> <th>Title</th> <th>Pages</th> </tr> </thead> <tbody> <tr> <td>{Min A. Chen, . . ., Denis McLeod}</td> <td>An Execution Model. . .</td> <td>365-413</td> </tr> <tr> <td>{Piero Fraternali, Letizia Tanca}</td> <td>A Structured. . .</td> <td>414-471</td> </tr> <tr> <td>. . .</td> <td>. . .</td> <td>. . .</td> </tr> </tbody> </table>	Author	Title	Pages	{Min A. Chen, . . ., Denis McLeod}	An Execution Model. . .	365-413	{Piero Fraternali, Letizia Tanca}	A Structured. . .	414-471
			Author	Title	Pages										
			{Min A. Chen, . . ., Denis McLeod}	An Execution Model. . .	365-413										
{Piero Fraternali, Letizia Tanca}	A Structured. . .	414-471													
.													
.												

Figure 3. Nested table containing extracted objects

thors, 39 pages, and which appeared in September of 1995. Once an object is properly structured, it can be directly inserted into a nested table for later querying, as illustrated in Figure 3. Further, this nested table can be flattened for querying as a standard relational table. For each piece of data in the example object in Figure 2, we assume that we know the position in the original page where it came from. For instance, the author name ‘Dennis Shasha’ initiates at the second non-blank character of the ninth non-blank line in Figure 1. If we assume the existence of a graphical tool which allows the user to mark pieces of data on a page and drag them, this positional information is trivially generated by storing the positions (in the text) of the pieces marked by the expert. Such a tool is actually implemented and we refer the reader to [13, 16] for further details on it. Positional information is easy to obtain but represents a crucial piece of evidence for assisting with the data extraction process as we later demonstrate.

Given a very small set of example objects, we conceive a strategy for extracting data from new pages with similar structure. This strategy is based roughly on the algorithm described in Figure 4 (consider for a moment that the examples provided by the user are flat i.e., they have a single level hierarchical structure).

This algorithm works by assembling a context for an object and using this context description to identify new objects in new pages. The context describes the surroundings of the object provided as example by the expert. Despite its simplicity, this approach, which we call *top-down*, works well with data rich pages presenting some variations in their structure. This strategy will be further discussed in more details in Section 4.

```

foreach example object  $O_e$  do
begin
  foreach attribute  $A$  of  $O_e$  do
  begin
    determine a local text context for the
    piece of data associated with  $A$ ;
  end;
  combine all the local contexts and generate
  a context for the object  $O_e$ ;
  use the context description for  $O_e$  to recog-
  nize and extract new objects in other pages;
end

```

Figure 4. Sketch of the top-down extraction algorithm.

3 Attribute-Value Pair (AVP) Patterns

To specify an example, the user selects pieces of data and uses them as atomic components of an example object. Each piece of data selected is called a *value* while each atomic component is referred to as an *attribute*. Thus, we use the terminology *attribute-value pair* (or, AVP) to refer to an attribute and one of its values. A same attribute might have multiple values, which form a list. In this case we say that the attribute is itself a complex component.

The recognition and extraction of (implicit) objects present on a page is based on the notion of a local context for each attribute-value pair (AVP). Local contexts are derived from the text in which the AVP occurs as follows. Consider the position in the text (or Web page) of an AVP value selected by the user. The terms surrounding the AVP value constitute a passage (or window) [6, 12] which can be

used as a local context. For instance, Figure 5a illustrates the AVP value “Eric Simon” as seen and selected by the user from a page presented by a Web browser. Figure 5b illustrates a passage which can be used as the local context for the value selected as it occurs in the HTML source of the page. Using this context information we build a *pattern* (also called *AVP pattern*) which can be later used to identify an author name such as “Eric Simon”.

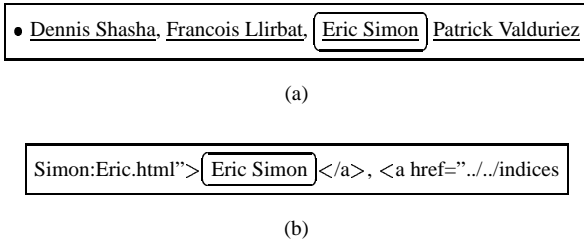


Figure 5. AVP value *Eric Simon* and its local context (or passage).

To illustrate, the AVP pattern corresponding to the AVP value “Eric Simon” in Figure 5 is shown in Figure 6. In this

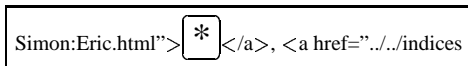


Figure 6. AVP pattern corresponding to the passage in Figure 5.

case, the symbol $*$ represents a string pattern that matches a sequence of characters of any length. This AVP pattern can be used to match any author name which appears in a context similar to the one presented in Figure 5b. However, most likely, this AVP pattern will not retrieve any other author name in any other page. Thus, to be able to effectively use this AVP pattern for extracting new author names, we must adopt a more flexible pattern recognition strategy. This can be accomplished as follows.

Given an AVP selected by the user, we determine a passage surrounding this AVP value in the text. In this work, we adopt symmetric passages composed of W terms to the right and W terms to the left of the AVP value. The following two restrictions apply: (1) a passage does not go beyond a (textual) newline mark and (2) a new passage must not overlap with passages corresponding to previous AVP values selected by the user. Due to these restrictions, an AVP pattern might be asymmetric.

The width W of an AVP pattern is determined empirically for each type of AVP as follows. We start with a small pattern represented by the symbol $*$, a term to its right and a term to its left. These terms can be words or special symbols. For instance, in Figure 6 the initial AVP pattern would

be composed of the symbol $*$ surrounded by the prefix “>” and the suffix “<”. Notice that the prefix and the suffix are present in the page, and can be recognized automatically once the user marks “Eric Simon” as a value of his interest. We then parse the excerpt of a page from which the user selects the AVP value, compare it with the AVP pattern just defined, and count the number of matches. Following, we ask the user for the number of string values for authors’ names he can see and compare this with the number of matches we just counted. If the number of matches exceeds the number specified by the user we add additional terms to the pattern, increasing its width W and the amount of contextual information attached to it. This process is repeated automatically until we have a good definition for the local context of the AVP in consideration (we stop when the number of matches is smaller than the number of values identified by the user). Notice that all the user has to provide is a single number which indicates the number of occurrences of values for authors’ names in the excerpt of the Web page he is working with. Such an information is quite simple to provide and presents no inconvenience to the user.

The AVP patterns generated are then used as the building blocks of our extraction strategy as we now discuss.

4 Top-Down Extraction

We discuss here our *top-down* strategy to extract complex objects from data rich Web sources. Before discussing the details of the proposed strategy, we define some basic concepts and terminology required to deal with the hierarchical structure of complex objects.

Since the structure of Web page objects is not always flat, it is necessary to introduce the concept of a complex object with a hierarchical structure as we now do. Instead of formally defining a complex object, we introduce the terminology through the discussion of an example. The example we adopt is the complex object labeled *Edition* in Figure 2.

The object *Edition* in Figure 2 is composed of four other objects which are called component objects or simply *components*. The first three component objects (i.e., *Volume*, *Number*, and *Date*) are atomic and are referred to as attribute objects or simply *attributes*. The fourth component (i.e., *Article*) is a list object or simply a *list*. This list is formed by several *identical* complex objects. Each of these objects is composed of a list (i.e., *Author*) and two attributes (i.e., *Title* and *Pages*). The list *Author* is itself formed by atomic elements.

Throughout our discussion below, we use the following notation:

- A: an attribute or atomic component;
- C: an object component (atomic or complex);

```

function pattern(A) {
/* returns an AVP pattern for the attribute A */
  This function works as discussed in Section 3 and is not
}   detailed here

function o-pattern(Oe) {
/* returns a pattern for the object Oe */
op = nil; /* initialize an object pattern string */
Traverse the structure of Oe in pre-order and visit all its components;
foreach visited component C do
begin
  case type of C is:
    'atomic': op = op . * . pattern(C);
    'complex': op = op . * . o-pattern(C);
end
'list': op = op . * . "(" . o-pattern(C) . "*"";
return(op);}

function top-down-extraction(G,Oe) {
/* extracts from set G all the objects similar to Oe */
R = ∅; /* a set variable for holding the extracted objects */
foreach page g, g ∈ G, do
begin
while not end of page g do
begin
  o-string=string-match(g,o-pattern(Oe));
  R = R + object-from-string(o-string);
end
end
return(R);}

```

Figure 7. Detailed top-down extraction algorithm.

O_e : an example object specified by the user;
 G : a set of sample pages for extracting data from;
 g : a single page from the set G .

Our top-down extraction strategy follows the main steps presented in Section 2. The central idea is to find new objects which have a structure *identical* to the structure of an object O_e provided as example. We consider that each attribute of O_e has been replaced by its respective AVP pattern. Thus, the example object O_e combines information on the object structure with information on the AVP patterns associated with its attributes. This is a crucial aspect because allows that all the extraction procedure be based on the example object O_e only.

A detailed view of our top-down extraction algorithm is presented in Figure 7.

The top-down-extraction function applies the (string) pattern for the whole object O_e (generated by the function o-pattern) to each page g in the set of pages G . The strings in g that match the O_e pattern are retrieved. Each of these strings corresponds to a new complex object whose components are readily determined (through a top-down decomposition operation performed by the function object-from-

string). Each new complex object is then stored in the result variable R .

The function o-pattern is at the core of our top-down extraction strategy. It works by traversing the structure of the object O_e in pre-order, visiting all its components, and concatenating the respective AVP patterns in a resultant object pattern op . Concatenation is an operation on strings indicated as a dot. An atomic component leads to the direct concatenation of its AVP pattern (intermediated by a wild character “*”) to the resultant pattern op . A complex (component) object that is not a list requires first deriving its pattern (through a recursive call to the function o-pattern) prior to the concatenation. A list object requires deriving its pattern and surrounding it with the strings “(” and “)” to indicate that repetition is allowed (because it is a list).

We say that this strategy is top-down because each new object is recognized and extracted in its entirety (using string pattern matching) prior to the identification of its component objects (which is done through a top-down decomposition operation). Figure 8 illustrates graphically the behavior of our extraction strategy.

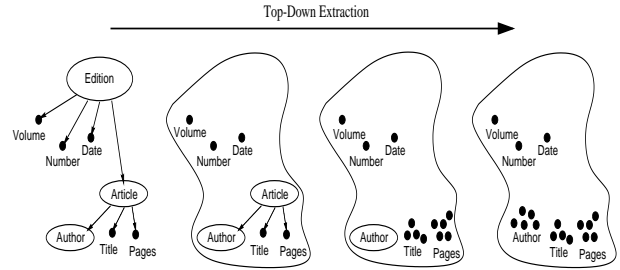


Figure 8. Behavior of our extraction strategy.

5 Results

In this section we discuss the application of our extraction strategy to HTML pages collected from popular Web sources. The results are presented in two parts. In the first part, we use Web sources whose (implicit) objects are structured in one-level. In the second part, we use a Web source whose (implicit) objects are structured in two-level hierarchies.

For the first part of our experiments, we use three Web sources: (a) CDNow [7], (b) Travelocity [18], and (c) Amazon [3]. Figure 9 illustrates excerpts of pages from these three sources. We notice that the (implicit) objects in these sources are all flat (i.e., their structure is given by a one-level hierarchy).

We collected sample pages from our three sources and applied the top-down extraction algorithm to them. A single example object is provided for each source. The results are illustrated in Table 1. The sample pages from CDNow

Placebo <i>Without You I'm Nothing</i>	\$16.97	\$11.88
Portishead <i>Pnyc</i>	\$16.97	\$11.88
Louis Prima <i>Collectors Series</i>	\$11.97	\$8.38
Queen <i>Greatest Hits I & II</i>	\$29.97	\$20.98
R.E.M. <i>Up</i>	\$16.97	\$11.88

(a) CDNow

Sam Lord's Castle - Barbados \$333- (5 nts.) Stay at this legendary landmark castle on one of the world's most beautiful beaches! Dive into 3 pools, soak in the whirlpool, play tennis on lighted courts, sail, snorkel, waterski, and work out in the fitness room. All on 72 landscaped acres.
3-Night Southern Caribbean Cruise \$349- Now you can explore the faraway islands of the Southern Caribbean on a three-night cruise -- and save a minimum of \$140 off the brochure rate! Aboard the <i>Nordic Empress</i> , you will depart San Juan and visit St. Thomas and St. Maarten.
Wyndham Aruba Beach Resort & Casino \$366- (5 nts.) Couples, singles and families alike will love this resort located on Aruba's most exclusive beach. Enjoy swimming, water sports, tennis and nearby championship golf. And the Kids Klub is fun for all ages.

(b) Travelocity

Internet Publishing Kit Hardcover / Published 1995 Our Price: \$149.95 (<i>Special Order</i>)
Internet Publishing with Microsoft Word 7.0; With CDROM With CDROM Connie Dunn / Paperback / Published 1998 Our Price: \$29.95 (<i>Not Yet Published</i>) Read more about this title...
Internet Quick Reference Guide Glenn Davis / Paperback / Published 1997 Our Price: \$9.60 ~ You Save: \$2.40 (20%) (Back Ordered) Read more about this title...

(c) Amazon

Figure 9. Excerpts from three Web sources used in our experiments.

contain 219 retrievable objects which were all recognized and extracted. The sample pages from Travelocity contain 162 retrievable objects. The algorithm was able to recognize 79% of them (i.e., 57 or 21% of the objects are not retrieved). The reason is that the implicit object structure, while present in all pages, is now more difficult to be identified. The sample pages from Amazon contain 178 recognizable objects. For this source, the top-down algorithm presents a poor retrieval performance and is able to recognize only 30% of the objects. The main reason is that the objects might now have components which are out of order or missing. However, we can improve the performance of the top-down algorithm by increasing the number of examples given, as discussed below.

Consider again the sample pages from Amazon and assume that the (implicit) objects on those pages are ranked

Source	Total	Retrieved
CDNow	219	219 (100%)
Travelocity	162	129 (79%)
Amazon	178	54 (30%)

Table 1. Number of objects retrieved by our top-down algorithm for three Web sources.

according to their order of appearance. Given this unusual ranking of the objects to be retrieved, we can plot curves of precision and recall (in 11-standard recall levels) for the results of the top-down algorithm, as illustrated in Figure 10. Precision is measured as the ratio between the number of objects retrieved and the total number of objects that could have been retrieved at any point in the ranking. Recall is measured in conventional form.

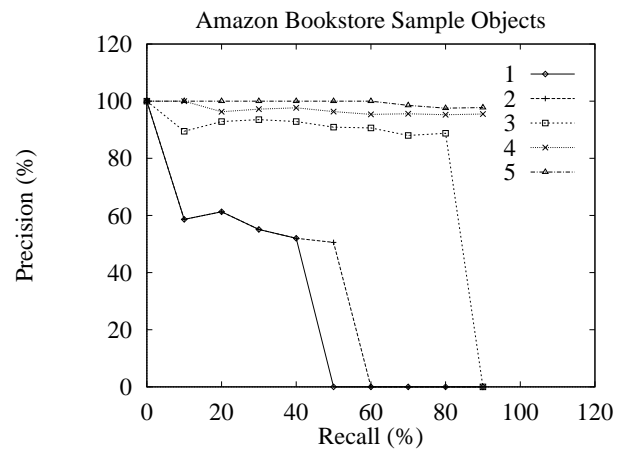


Figure 10. Precision and recall curves relative to sample pages from Amazon.

Each curve in the graph of Figure 10 refers to an extraction using increasing numbers of examples (e.g., 1, 2, 3, 4 and 5). We first notice that the retrieval performance of the extraction using a single example deteriorates as it proceeds. This is also true for the extraction using 2 examples. This effect indicates that the top-down algorithm, with such a few number of examples, fails to match objects early on and never recovers. When 3 examples are given, the algorithm is able to maintain high precision for levels of recall up to 90%. Then, its precision suddenly drops to zero. This indicates that the objects in the final sample pages have a distinct structure. We then built an additional example object (derived from one of the final sample pages), and rerun our algorithm using now 4 example objects (the three previously used and the new one just built). As a result, the corresponding precision/recall curve (labeled 4) indicates a

very nice improvement. In fact, the levels of precision are now very close to 100% for the various recall numbers. As the graph shows, there is also a little improvement when we give an additional example, what corresponds to the curve labeled 5.

For the second part of our experiments, we use Web pages from DB&LP ACM TODS [14]. The objects in these pages have a two-level hierarchical structure as indicated in Figures 1 and 2. To demonstrate the efficiency of the top-down algorithm in dealing with multi-level hierarchies, we use it to extract complex objects representing journal volumes and papers from these pages. The results are summarized in Table 2. The algorithm was able to identify and

1st. Level		
Object	Total	Retrieved
Volume Number	20	20 (100%)
Date	20	20 (100%)
Edition	20	20 (100%)
2nd. Level		
Title	76	76 (100%)
Author	188	187 (99.5%)
Page	66	66 (100%)
Article	76	76 (100%)

Table 2. Number of objects retrieved by our top-down algorithm for the DB&LP Web source

to extract most of the objects present in the pages. The 20 first-level Edition objects were extracted. Then, these objects were decomposed and the 76 Article objects composing them were also extracted. We notice that for this experiment only one object of each type was given as example. This retrieval performance was obtained because DB&LP pages present, in general, a very regular (implicit) structure. Indeed, the top-down algorithm presents very nice extraction capability for Web sources like this.

6 Conclusions

We have studied the problem of extracting semi-structured data from Web pages. Our approach is innovative because it is based solely on a couple of examples provided by the user.

We showed an example-based data extraction top-down strategy, discussed it in detail and analyzed its behavior. Through experimentation, we demonstrated that the strategy works well when either the target Web source (implicit) objects present little variations in their structures or when enough examples are given to capture the variety of possible structures.

The top-down strategy recognizes objects in their entirety. Thus, recognition of partial objects (i.e., objects

which are missing a component) is not done. To address this, we are currently working on a new improved extraction strategy that is adaptive and that can retrieve more objects with fewer examples provided.

Acknowledgments

This work is partially funded by the SIAM (MCT/FINEP/PRONEX grant 76.97.1016.00) and the CYTED (VII.13 AMYRI) projects. Financial support from CNPq and CAPES is also gratefully acknowledged.

References

- [1] S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of International Conference on Database Theory*, pages 1–18, Delphi, Greece, 1997.
- [2] B. Adelberg. NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 283–294, Seattle, Washington, 1998.
- [3] Amazon.com, Inc. Amazon.com Bookstore Web Site. <http://www.amazon.com>.
- [4] N. Ashish and C. Knoblock. Wrapper Generation for Semi-structured Internet Sources. *ACM SIGMOD Record*, 26(4):8–15, 1997.
- [5] P. Atzeni and G. Mecca. Cut & Paste. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 144–153, Tucson, Arizona, 1997.
- [6] J. P. Callan. Passage-Level Evidence in Document Retrieval. In *Proceedings of the ACM SIGIR Conference on Information Retrieval*, pages 302–309, Dublin, Ireland, 1994.
- [7] CDnow, Inc. CDnow Web Site. <http://www.cdnw.com>.
- [8] J. Cowie and W. Lehnert. Information Extraction. *Communications of the ACM*, 39(1):80–91, 1996.
- [9] D. W. Embley, D. M. Campbell, Y. Jiang, Y. Ng, and R. D. Smith. A Conceptual-Modeling Approach to Extracting Data from the Web. In T. W. Li, S. Ram, and M. Lee, editors, *Conceptual Modeling - ER'98*, Springer Verlag, pages 78–91, Berlin, 1998.
- [10] D. W. Embley, D. M. Campbell, S. W. Liddle, and R. D. Smith. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 52–59, Bethesda, Maryland, 1998.
- [11] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting Semistructured Information from the Web. In *Proceeding of the Workshop on Management of Semistructured Data*, Tucson, Arizona, 1997.
- [12] M. Kaszkiel and J. Zobel. Passage Retrieval Revisited. In *Proceedings of the ACM SIGIR Conference on Information Retrieval*, pages 178–185, Philadelphia, USA, 1997.
- [13] A. H. F. Laender, E. S. Silva, and A. S. da Silva. DEByE - A Tool for Extracting Semi-Structured Data. In *Proceedings of the XIV Brazilian Symposium on Databases - SBBD'99*, Florianópolis, Brazil, 1999. In portuguese.

- [14] M. Ley. DB&LP Computer Science Bibliography Site. <http://www.informatik.uni-trier.de/~ley/db/>.
- [15] P. Pyreddy and W. B. Croft. TINTIN: A System for Retrieval in Text Tables. In *Proceedings of the Second ACM International Conference on Digital Libraries*, pages 193–200, 1997.
- [16] E. S. Silva. Extracting Semi-Structured Data Through Examples. Master's Thesis, Department of Computer Science, Federal University of Minas Gerais, 1999. In portuguese.
- [17] S. Soderland. Learning to Extract Text-based Information from the World Wide Web. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining - KDD-97*, pages 251–254, Newport Beach, California, 1997.
- [18] The SABRE Group, Inc. Travelocity Vacations Web Page. <http://www2.travelocity.com/vacations/>.