AUTOBIB: Automatic Extraction of Bibliographic Information on the Web*

Junfei Geng Jun Yang
Department of Computer Science, Duke University, Durham, NC 27708, USA
{geng,junyang}@cs.duke.edu

Abstract

The Web has greatly facilitated access to information. However, information presented in HTML is mainly intended to be browsed by humans, and the problem of automatically extracting such information remains an important and challenging task. In this work, we focus on building a system called AUTOBIB *to automate extraction of bibliographic information on the Web. We use a combination of bootstrapping, statistical, and heuristic methods to achieve a high degree of automation. To set up extraction from a new site, we only need to provide a few lines of code specifying how to download pages containing bibliographic information. We do not need to be concerned with each site's presentation format, and the system can cope with changes in the presentation format without human intervention.*

AUTOBIB *bootstraps itself with a small seed database of structured bibliographic records. For each bibliographic Web site, we identify segments within its pages that represent bibliographic records, using state-of-the-art record-boundary discovery techniques. Next, we find matches for some of these "raw records" in the seed database using a set of heuristics. These matches serve as a training set for a parser based on the Hidden Markov Model (HMM), which is then used to parse the rest of the raw records into structured records. We have found an effective HMM structure with special states that correspond to delimiters and HTML tags in raw records. Experiments demonstrate that for our application, this HMM structure achieves high success rates without the complexity of previously proposed structures.*

1. Introduction

The amount of information on the Web has reached an astonishing level. However, most of the information is presented in the HTML format, intended to be browsed by humans rather than to be manipulated by computers. Automatically extracting such information from the Web remains an active and challenging research problem. The conventional approach to Web information extraction requires constructing a wrapper for each Web site. Traditionally, wrapper construction relies on a significant amount of human input of site-specific knowledge, in the form of either programming or hand-labeled training examples. Such construction is labor-intensive and does not scale when many sites need to be wrapped. Wrapper maintenance also becomes a problem when sites change their presentation formats. The "holy grail" of Web information extraction has always been an automated extraction engine that requires no human input. With the recent advances in information extraction research, how close are we to achieving this goal?

In an attempt to answer this question, we set out to build a system called AUTOBIB for extracting computer science bibliographic information on the Web. There are several excellent computer science bibliographic sources (DBLP [3], CCSB [4], CSWD [2], etc.), although none of them is comprehensive on its own. From a practical standpoint, it would be useful to provide a uniform and integrated view of the bibliographic information from these sources. From a research standpoint, extraction of bibliographic information presents a challenging problem, since there is considerable variation in both the structures of bibliographic records (e.g., articles versus books) and the ways they are presented on different sites (e.g., DBLP versus CCSB).

Throughout the project, we consciously avoid using site-specific human inputs. Instead, we use a host of powerful techniques to automate AUTOBIB. We use state-of-the-art record-boundary discovery techniques to extract "raw records," e.g., segments of the Web pages that represent bibliographic records. We use a parser based on the Hidden Markov Model (HMM) to parse these raw records into structured bibliographic records. To generate the training data for the HMM parser automatically, we take advantage of the fact that there is considerable amount of content overlap among the sources. We develop heuristics to match raw records with known structured records, and annotate the matched raw records for use as training data. After experimenting with different methods to structure the

* This work was supported in part by a National Science Foundation CAREER Award under grant IIS-0238386.

HMM, we have found an effective structure with special states that corresponds to delimiters and HTML tags in raw records. Experiments demonstrate that this structure can effectively exploit the structural information encoded by delimiters and HTML tags in raw records, allowing the HMM to achieve high success rates without the complexity of previously proposed structures.

We have achieved a high degree of automation. In order to set up AUTOBIB to extract from a new bibliographic Web site, we only need to supply a few lines of code that specifies how to download the pages containing publications of a particular author. The rest of the process is completely automated. We do not need to be concerned with how each site formats its bibliographic information, and the system can cope with changes in the presentation format without human intervention.

2. Related Work

An excellent survey of Web data extraction tools can be found in [25]. The extraction process is usually performed by software components called *wrappers*. Wrappers are often generated manually or semi-automatically [18, 5]. For example, TSIMMIS [18] allows users to generate wrappers according to declarative specifications. A specification states where the data of interest is located on the HTML pages, and how the data should be "packaged" into objects for integration. This approach of manual specification takes advantage of human input, and thus generically performs well for any specific site. However, the manual approach by nature is labor-intensive and difficult to maintain.

A number of semi-automatic approaches [22, 27, 17, 23, 24] to wrapper generation use the idea of learning by examples. The user first labels a number of examples of extracted data, and the software then generates extraction rules based on these examples. For instance, the approach used by Kushmerick et al. [23, 24] is able to learn from labeled examples any wrapper that uses the HLRT (head-left-right-tail) parsing strategy. Since this approach still requires manual construction of training data for each site, it inherits some of the same problems as the manual approach.

Several systems [12, 6] deserve special discussion because they support fully automatic generation of wrappers. These systems examine the structures of sample Web pages and automatically generate a template for the data contained in these pages. However, these systems are based purely on syntax and do not take advantage of the semantics of the specific domain (e.g., bibliographic data). Consequently, these systems do not semantically label their extracted data (e.g., using bibliographic field names such as author, year, etc.). In addition to this fundamental difference from our work, RoadRunner [12] assumes that the template generating the pages contains no union/disjunction. Therefore, RoadRunner cannot handle a page that contains bibliographic records of different types (e.g., journal articles and conference papers), which are formatted differently in HTML. Arasu and Garcia-Molina [6] support templates with optionals and disjunctions. However, their approach still cannot ensure that the template generated corresponds to the natural structure of bibliographic records.

Techniques for record-boundary discovery in Web pages are proposed in [13, 9, 10, 19]. These techniques work by analyzing the statistical properties of the HTML tag tree structure of Web pages. Heuristics are used to identify the HTML tags that separate individual records. Our system uses the techniques in [13] to extract segments within the Web pages that correspond to bibliographic records. More details are given in Section 6.

Recently, a number of projects have demonstrated the effectiveness of Hidden Markov Models for information extraction. Bikel et al. [7] use HMM's to extract named entities such as "price" from free-text document. Freitag et al. [14] extract relevant phrases from documents containing much irrelevant text. Leek [26] uses HMM's to extract information about gene names and locations from scientific abstracts. Borkar et al.'s work [8] relates most closely to our project. They have developed a tool called DATAMOLD to segment records into elements. DATAMOLD uses nested HMM's, where the outer HMM captures the sequencing relationship among elements, and the inner HMM's learn finer structures within individual elements. These authors use a taxonomy on the symbols for hierarchical feature selection. In addition, they propose augmenting the HMM's with an external database of semantic relationships, and have developed a modified version of the Viterbi Algorithm (although the modified version does not guarantee the optimality of the solution). Experimental results indicate that DATAMOLD outperforms the deterministic rule-learner system Rapier [11]. In building AUTOBIB, we have adopted many ideas from DATAMOLD (e.g., the use of HMM's and hierarchical feature selection). However, for our target application, we show that with a proper choice of structure, a standard HMM can also achieve very high accuracy, but is more efficient computationally and easier to implement than nested HMM's.

The idea of bootstrapping is not new in machine learning, and has been applied to text learning [28] and example-based Web data extraction [16]. However, these approaches differ significantly from ours in how bootstrapping is used.

Finally, it is also important to mention CiteSeer [1] since it also deals extensively with bibliographic information on the Web. CiteSeer works by crawling the Web and downloading documents that are possible publications. The documents (often PostScript files) are then converted into plain text, and bibliographic citations in the text are extracted and parsed. However, CiteSeer can still be improved in many ways. For example, as discussed in [8], the algorithm used by CiteSeer for citation matching is an approximate record-level matching algo-

A raw record:

```
<td ...>121</td><td ...><a href=...>EE</a></td>
<td>Pankaj K. Agarwal,
<a href=...>Binay K. Bhattacharya</a>,
<a href=...>Sandeep Sen</a>:
Improved Algorithms for Uniform Partitions of Points.
<a href=...>Algorithmica 32</a>(4): 521-539 (2002)</td>
```

A structured record:

author =	“Pankaj K. Agarwal and Binay K. Bhattacharya and Sandeep Sen”,
title =	“Improved Algorithms for Uniform Partitions of Points”,
journal =	“Algorithmica”,
volume =	“32”,
number =	“4”,
pages =	“521–539”,
year =	“2002”.

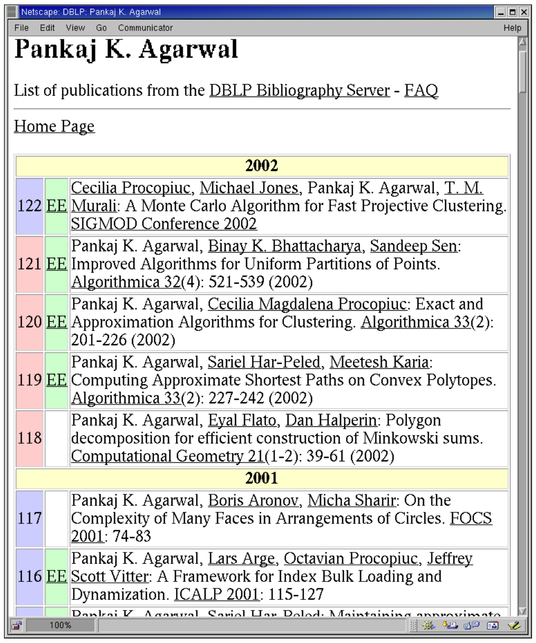
Figure 1. Raw and structured records.

rithm; a field-level matching on title, year, and author would be more accurate. Thus, an HMM-based parser that is capable of identifying fields within the record can be used to improve the accuracy and efficiency of citation matching.

3. Problem Formulation

Before we proceed, we introduce some terminology and assumptions that will be used later.

- A *structured record* refers to a bibliographic record which has been properly parsed into labeled *fields*, e.g., *author*, *title*, *year*, etc., following the popular BibTeX format. An example of a structured record is shown in Figure 1. Different records do not necessarily contain the same set of fields.
- A *raw record* refers to a string representing a bibliographic record formatted in HTML. Fields inside the record have not been identified. An example of a raw record (from the Web page in Figure 2) is shown in Figure 1. One of our goals is to parse raw records into structured records.
- The *structured record database* is a database consisting of structured records. We start initially with a *seed database* populated with a small set of semi-manually constructed structured records. Over time, more structured records are automatically obtained from the Web and added to the database.
- A *source* is a Web site that provides bibliographic information in HTML. In general, different sources may present bibliographic information in very different ways, and the same source may change its presentation format over time. We assume that each source has some query interface that accepts an author name and returns Web pages containing the author’s publications, which is a common functionality provided by many bibliographic Web sites. An example Web page is shown in Figure 2. Each result Web page contains a list of raw records to be extracted. The boundaries between raw records are not known a priori, and need to be discovered.


Netscape: DBLP: Pankaj K. Agarwal
File Edit View Go Communicator Help
100%

Pankaj K. Agarwal

List of publications from the DBLP Bibliography Server - FAQ

Home Page

		2002
122	EE	Cecilia Procopiuc, Michael Jones, Pankaj K. Agarwal, T. M. Murali: A Monte Carlo Algorithm for Fast Projective Clustering. SIGMOD Conference 2002
121	EE	Pankaj K. Agarwal, Binay K. Bhattacharya, Sandeep Sen: Improved Algorithms for Uniform Partitions of Points. Algorithmica 32(4): 521-539 (2002)
120	EE	Pankaj K. Agarwal, Cecilia Magdalena Procopiuc: Exact and Approximation Algorithms for Clustering. Algorithmica 33(2): 201-226 (2002)
119	EE	Pankaj K. Agarwal, Sariel Har-Peled, Meetesh Karia: Computing Approximate Shortest Paths on Convex Polytopes. Algorithmica 33(2): 227-242 (2002)
118		Pankaj K. Agarwal, Eyal Flato, Dan Halperin: Polygon decomposition for efficient construction of Minkowski sums. Computational Geometry 21(1-2): 39-61 (2002)
		2001
117		Pankaj K. Agarwal, Boris Aronov, Micha Sharir: On the Complexity of Many Faces in Arrangements of Circles. FOCS 2001: 74-83
116	EE	Pankaj K. Agarwal, Lars Arge, Octavian Procopiuc, Jeffrey Scott Vitter: A Framework for Index Bulk Loading and Dynamization. ICALP 2001: 115-127
		Pankaj K. Agarwal, Sariel Har-Peled: Maintaining approximate

Figure 2. A sample DBLP page (June 2002).

- We assume there is “enough” overlap between a source and the structured record database. More specifically, a subset of the raw records obtained from the source must match some structured records in the structured record database. This common subset should provide good coverage of different publication types, e.g., journal articles, conference papers, technical reports, etc., which might be formatted differently by the source. Note that the seed database does not have to be large or overlap directly with all sources; instead, it only needs to be connected to every source either directly or indirectly via a chain of overlapped sources. This assumption is sufficient to ensure that we can gradually “build up” a structured record database to encompass all sources.

We are now ready to state our problem more precisely: Given a number of sources, whose contents and presentation formats are different and may change over time, we wish to build a structured record database by automatically extracting, parsing, and merging raw records from these sources. For our deployment, we focus on the publications of computer science faculty members at Duke University.

4. System Overview

The architecture of AutoBib is shown in Figure 3 with major components shown in ovals. We start with a seed database of structured records generated from a collection of BibTeX entries by the *cleaner*. For each source, we download relevant Web pages containing bibliographic information and extract raw records from them using the raw record *extractor*. Next, the *matcher* attempts to match these raw records with

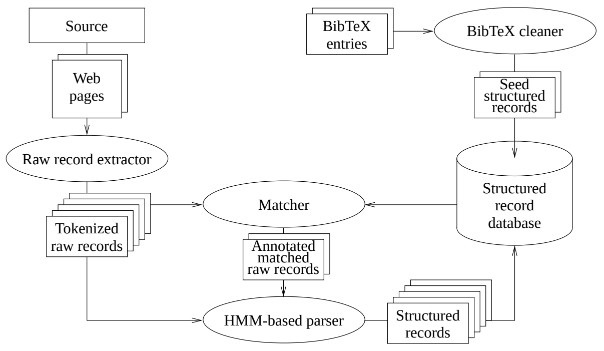


Figure 3. Architecture of AUTOBIB.

the structured record database to generate a training set. The training set is then used to train an HMM-based *parser* which parses the unmatched raw records into structured records. Finally, a *merger* incorporates the newly parsed structured records into the structured records. With a bigger structured record database, we move on to the next source.

We repeat this process periodically (e.g., every week). Only during the very first run, we use the cleaner to generate the seed database. Any subsequent run starts with the structured record database produced by the previous run. Since bibliographies are mostly append-only and have a relatively low rate of content change, we expect there to be significant overlap between the current contents of the sources and the structured records produced by the previous run. It is possible that the sources have changed their presentation formats since the previous run. Our system handles such format changes seamlessly by learning the new presentation formats from overlapping contents; in fact, a new visit to an existing source is handled in the exact same way as the first visit to a new source.

In the following sections, we describe the components of AUTOBIB in more detail. We will focus more on two important subproblems: generating training data for HMM's automatically (Section 7) and using HMM's to parse raw records (Section 8), for which we have developed new techniques or improved over existing ones.

5. Generating the Seed Database

When AUTOBIB runs for the very first time, a small collection of BibTeX entries are fed into the cleaner to generate a small seed database. These BibTeX entries are obtained from CCSB. The cleaner performs a series of cleaning and normalization steps on the BibTeX entries, including the removal of LaTeX formatting directives, expansion of standard abbreviations, e.g., "`J.`" $\longrightarrow$ "`Journal`", "`Proc.`" $\longrightarrow$ "`Proceedings`", "`Jan.`" $\longrightarrow$ "`January`", etc. The cleaner discards any BibTeX entry deemed either incomplete (e.g., missing *title* or *year* field) or poorly formatted (e.g., *booktitle* contains also year and pages). Discarding these BibTeX entries is not a problem because the seed database does not need to be complete, as discussed in Section 3. Detailed description of the cleaner can be found in the full version of this paper [15]. Our experience indicates that although our cleaner is relatively simple and unsophisticated, it is still able to produce enough structured records to ensure overlap with a source and coverage of all types of publications. Finally, we emphasize again that this step of generating the seed database is performed only once.

6. Extracting Raw Records

For each source, AUTOBIB downloads all relevant Web pages containing bibliographic information. In our deployment of AUTOBIB, we query the source for publications of all members of the computer science faculty at Duke. The user needs to supply the list of authors and specify how to query each source for a particular author and how to follow "next page" links when multiple pages are returned. This information is the only piece of source-specific information that needs to be specified manually. For each Web page, the AUTOBIB raw record extractor extracts all raw records on the page using record-boundary discovery techniques from [13]. Because of space limit, we only briefly describe the procedure below; details are available in [15].

Identifying the subtree of interest. The raw record extractor makes one pass over the page to construct a tag tree, whose nodes correspond to the HTML tags in the page. At the end of this pass, the node with the highest fan-out is identified. The subtree rooted at this node is chosen to be the subtree of interest as in [13]. Buttler et al. [9] suggest using the content size of a subtree as an additional metric, because many Web pages contain lots of advertising, which makes the highest-fan-out heuristic ineffective. In our context, however, considering fan-out suffices because the amount of advertising and other types of "noises" is generally small to none on bibliographic sites.

Identifying record separators. Within the subtree of interest, we identify a list of frequently occurring HTML tags (those that appear more than 10% of the time) as candidate record separators. Next, we rank these candidates using the following heuristics (again, from [13]):

- *Highest-count heuristic.* Tags with more occurrences are ranked higher, based on the observation that a separator tag occurs about as many times as the number of records on the page, which is usually high.
- *Identifiable separator heuristic.* Tags are ranked based on how frequently they are used as records separator on the Web as a whole. We use the ranking {`<tr>`, `<table>`, `<p>`, ...} from Buttler et al. [9], computed from over 2000 Web pages from 50 sites.
- *Standard deviation heuristic.* This heuristic is based on the observation that records within a page have about the same

size. We calculate the standard deviation in the amount of text between identical tags. Tags with lower standard deviation are ranked higher.

To make a final decision combining all three heuristics, we follow the approach of [13, 9] to calculate the *compound certainty factor* for each candidate tag from the certainty factors of individual heuristics, assuming that their decisions are independent. We use the certainty factors of individual heuristics from [13, 9], which are measured empirically from samples of real Web pages.

Computing the optimal record separator tag requires only two traversals of the subtree of interest, which in our case always fits in main memory. As an optimization, we only need to perform this computation on a small number of Web pages, enough to determine a "consensus" tag that can be used on all other Web pages from the same source.

Generating tokenized raw records. Once the separator tag is chosen, we scan the subtree of interest to extract HTML segments between separator tags and convert these segments into raw records. To guard against possible extraneous segments, we use the following two heuristics. (1) Any segment whose size is much less than the mean (we use 20% of the mean as threshold) is discarded. (2) Any segment whose text does not contain the last name of the author being searched for is discarded (recall that we obtain Web pages by querying particular authors). For example, for the Web page shown in Figure 2, we find `<tr>` to be the record separator tag. A raw record found between two `<tr>` tags is shown in Figure 1. On the other hand, headings "`2002`" and "`2001`", which are also found between `<tr>` tags, are discarded by the two heuristics.

The raw record extractor also tokenizes the raw records in preparation for the ensuing operations. A raw record is split into tokens according to delimiters, HTML tags, and spaces. Delimiters include characters such as "`() [] , . ; : / \ -`". Spaces include normal whitespace characters (space, tab, newlines, etc.) as well as the special HTML element `&nbsp;`. We categorize tokens into four types: words, numbers, delimiters, and tags. A word is defined as a sequence of alphanumerical characters, including hyphens. A number is a sequence of purely numerical characters. Note that hyphen is treated as a delimiter when it is located between two numbers (e.g., "`10-12`"); if it is between two words (e.g., "`R-trees`") it is treated as part of a longer word. We do not distinguish different kinds of HTML tags; they are all represented by a special token "`^`". We only consider the top-level HTML tags within a raw record and ignore those nested deeper inside; we have found this level of detail sufficient for later operations. Intuitively, the reason for keeping delimiters and tags is to capture more structural information inside raw records, which can be exploited by ensuing operations. In Sections 7 and 8 we describe in detail how delimiters and tags help improve the accuracy of both training data generation and HMM

```
^ 121 ^ EE ^ Pankaj K . Agarwal ,
^ Binay K . Bhattacharya ^ , ^ Sandeep Sen ^ :
Improved Algorithms for Uniform Partitions of Points .
^ Algorithmica 32 ^ ( 4 ) : 521 - 539 ( 2002 ) ^
```

Figure 4. A tokenized raw record.

parsing.

For example, Figure 4 shows the result of tokenizing the raw record in Figure 1. Word and number tokens are underlined. Spaces are not represented as tokens.

7. Generating Training Data

Once the set of tokenized raw records have been extracted from a source, the AutoBib matcher attempts to match them with the structured record database in order to generate training data for the HMM-based parser. In the following, we first describe the matching algorithm, and then show how to annotate the matched raw records so they can be used to train the HMM in Section 8.

7.1. Matching Raw and Structured Records

To see if a tokenized raw record R matches a structured record S, we apply the following tests:

1. Suppose that R was obtained by searching for author a. Then, one of the authors of S must be a.
2. S.*year* appears as a token in R.
3. If S.*pages* exists, the start page number must appear as a token in R.
4. S.*title* is "approximately" contained in R. The measure of approximateness we choose is the Levenshtein edit distance, defined as the minimum number of edits (insertions, deletions, and substitutions of characters) required to transform one string to another. We use the approximate substring matching algorithm from [29]. We set the acceptable level of approximateness to be 10%, i.e., for every 10 characters of S.*title* there may be an error. For the purpose of matching, we convert both S.*title* and the tokenized raw records into lower-case words separated by single spaces.

For each raw record, we search the structured records database for those records that pass test 1 using an index on *author*. Then, we apply tests 2, 3, and then finally test 4, which is the most expensive one to evaluate.

The design of these heuristic tests strikes a balance between performance, ease of implementation, and the requirements of a good training set. For example, one of the requirements is that the training set should contain as few false positives as possible. We notice that it is possible for two papers of the same title written by the same authors to appear in two different publication venues even in the same year (typically one paper is a conference version and the other is a

	# structured records (CCSB)	# raw records	# matches identified	# false positives	# false negatives
DBLP	138	121	81	0	3
CSWD	213	88	51	0	8

Table 1. Effectiveness of matching.

journal version). Test 3 is a very effective way of weeding out these false positives, and it is simpler and more efficient than a test based on matching publication venues. On the other hand, test 3 does seem to be a bit too restrictive because it implies that a raw record missing the page information would not match any structured record. Fortunately, our goal here is to produce a good enough training set, so it is acceptable that the matching algorithm generates some false negatives (i.e., misses some matches). Hence, in the classic trade-off between precision and recall, we favor precision in this case.

Furthermore, note that test 3 is only applied if S.*pages* actually exists. Without this qualification, we would never be able to match those types of records that typically contain no page information (e.g., books and theses). Thus, this qualification is necessary because a good training set requires complete coverage of all types of publications.

To measure the effectiveness of our matching algorithm, we apply the algorithm to match raw records from DBLP and CSWD with a collection of structured records obtained from CCSB. We also manually match the records in order to report the numbers of false positives and negatives. The results are reported in Table 1. For each target source (DBLP or CSWD), we use raw and structured records obtained by searching for the same author. According to Table 1, our matching algorithm produces no false positives (i.e., all matches found are correct matches), leading to a high-quality training set. For CSWD, the number of false negatives (i.e., matches that are not identified by our algorithm) is higher in comparison, but still well within acceptable limits. Again, the primary goal of the algorithm is to identify enough number of matches to generate a sizeable training set, which is still achieved here. Furthermore, the identified matches cover all types of bibliographic records.

We find that the false negatives produced in the above experiments come in two types. (1) The raw record has missing page information. (2) There is some small discrepancy between the raw and structured records (e.g., their publication years differ by one). The second type is rare. By dropping test 3, we can decrease the number of false negatives to 2 for DBLP and 0 for CSWD, but the number of false positives would increase to 1 for both DBLP and CSWD. We choose to keep test 3 because it improves precision while still keeping recall at an acceptable level.

7.2. Generating Annotated Raw Records

For each pair of raw and structured records identified as a match, we annotate the tokens in the raw record with field names, so they can be used to train the HMM-based parser in Section 8. The basic annotation algorithm works by comparing each word or number token in the raw record against all fields in the structured record. If the token is contained in a certain field, we annotate the token with the name of that field (e.g., token "`Agarwal`" in Figure 4 is annotated with *author* when compared with the structured record in Figure 1). If the token is not contained in any field, we annotate it with a special field name *unknown* (e.g., tokens "`121`" and "`EE`" in Figure 4).

This basic algorithm has a number of problems. First, some common word tokens, e.g., "`of`" and "`the`", may be found in multiple fields of the structured record, and the basic algorithm is often unable to assign the correct field name. Another problem is that bibliographic records often abbreviate names of authors and publication venues, e.g., "`VLDB`" for "`Very Large Data Bases`". If one of the two records uses abbreviation while the other one does not, the basic algorithm will usually fail to find any field name to annotate some parts of the raw record.

To combat these deficiencies, we have developed the following two improvements over the basic algorithm.

- *Grouping heuristic*. The idea behind this heuristic is to use the context in which a token appears to help annotate the token. For example, consider a raw record containing a sequences of word tokens "`depth of two polytopes in 3D`". The basic algorithm might incorrectly annotate the token "`of`" with field name *journal*, since the article happens to appear in "`Nordic Journal of Computing`". However, since "`of`" appears among a group of *title* tokens, we can infer that it should also be annotated with *title*. The modified algorithm works as follows:
  - ◇ Use the basic algorithm to pick a set of candidate field names for each token. Those with no candidate field names receive *unknown* as the candidate. Each candidate is *weighted* by $1/(\text{total of candidates for this token})$.
  - ◇ Identify all *groups* of tokens in the raw record. A group is defined as a sequence of word/number tokens between neighboring delimiter or tag tokens. For each group, perform the following two steps.
  - ◇ Find the candidate field name f with the highest total weight computed over all tokens in the group. If f is *unknown*, pick f to be the candidate with the second highest total weight (if it exists).
  - ◇ Use f to annotate all tokens in the group, except any number token whose only candidate field name identified by the basic algorithm is *year* or *pages*.

  Intuitively, the improved annotation algorithm relies on delimiter and tag tokens to partition the raw record into groups whose tokens have a high probability of belonging to the same field. This algorithm is one of the many in AutoBib that take advantage of the structural information encoded by delimiters and tags. Without such structural information, a simpler alternative to grouping would be "sequencing," in which we use the annotations of previous tokens in the sequence to help annotate the current token. Although sequencing is easier to implement, we find it to be less robust because many fields start with tokens with ambiguous annotations (e.g., *title* often starts with "`the`").

- *Acronym expansion.* To improve the accuracy of annotation further, we use a small database of acronyms of publications venues obtained from the ACM Digital Library. Currently, the database contains 163 entries. An example of such an entry is "`SODA = Symposium on Discrete Algorithms`." Using this database, we expand all acronyms in raw and structured records before applying the annotation algorithm. A generalization of this approach would consider other types of abbreviations, e.g., author names, country and state names, etc.

In Table 2 we compare the baseline performance of the basic annotation algorithm with those of the two improvements in two experiments. The first experiment uses 50 pairs of matched raw and structured records from DBLP and CCSB, respectively. The second experiment uses 50 matched pairs from CSWD and CCSB. The total number of word and number tokens in the raw records is 1159 for DBLP and 1645 for CSWD. The "id rate" (identification rate) is the percentage of the tokens in the raw records that are annotated with field names other than *unknown*. However, some *unknown* annotations may indeed be correct, e.g., the extraneous tokens "`121`" and "`EE`" in Figure 4. Therefore, the "actual id rate" (counting correct *unknown* annotations) are higher. The third metric, "correctness rate," is the percentage of the correctly annotated tags among those not annotated with *unknown*. Finally, the last metric, "overall rate," is the percentage of all tags that are correctly annotated. All four metrics exclude delimiter and tag tokens because they do not need to be annotated.

According to Table 2, the grouping heuristic improves over the basic algorithm in all metrics for both DBLP and CSWD. Actual identification rates benefit the most, rising from below 90% to about 94%. Correctness rate increases to 100% for DBLP, and 96.5% for CSWD. Acronym expansion (in addition to the grouping heuristic) further improves the actual identification rate for DBLP by a small margin. Overall, 95.8% of all tokens for DBLP and 90.8% of all tokens for CSWD are annotated correctly. We attribute the success of these simple heuristics in part to the matching algorithm in Section 7.1, which preconditions the input to the annotation algorithm so that the latter only needs to deal with those records that do match.

	algorithm	id rate	actual id rate	correctness rate	overall rate
DBLP	basic	83.1%	88.8%	97.3%	86.6%
	+ grouping	88.3%	94.3%	100%	94.3%
	+ acronym	89.6%	95.8%	100%	95.8%
CSWD	basic	84.1%	89.1%	95.7%	85.5%
	+ grouping	88.7%	93.9%	96.5%	90.8%
	+ acronym	88.7%	93.9%	96.5%	90.8%

Table 2. Effectiveness of annotation.

In practice, because of significant content overlap among the sources, there are plenty of matches between raw and structured records, much more than required by a training set. Thus, we can afford to discard those matched raw records with many *unknown* annotations. Using this strategy, the accuracy of the training data can exceed the overall rate and approach the correctness rate.

8. Parsing Raw Records Using HMM's

As discussed in previous sections, the extractor extracts a set of tokenized raw records from a source, and the matcher annotates a subset of them that match with known structured records. In this section, we show how to use these annotated raw records to train an HMM-based parser to convert the rest of the raw records into structured records. We briefly introduce HMM and explain how the parser works in Section 8.1. Then, in Section 8.2, we turn to the problem of training the HMM. The most interesting aspect of training is choosing a right structure for the HMM, which is discussed separately in Section 8.3.

8.1. Parsing Raw Records Using an HMM

A *Hidden Markov Model* (HMM) is a probabilistic finite-state automaton characterized by the following:

- A set of n *states* $q_1, q_2, \ldots, q_n$.
- A set of m possible *output symbols* $v_1, v_2, \ldots, v_m$.
- A $n \times n$ *transition matrix* $A = \{a_{ij}\}$ where a_{ij} is the probability of making a transition from state q_i to state q_j. The sum of probabilities of transitioning out from any state should be 1, i.e., $\sum_j a_{ij} = 1$ for all i's.
- A $n \times m$ *emission matrix* $B = \{b_{ik}\}$, where b_{ik} is the probability that symbol v_k is observed in state q_i.
- An *initial probability vector* $\{\pi_i\}$ of size n, where π_i is the probability of starting in state i.

Roughly speaking, in the context of parsing bibliographic records, each token in a raw record may be regarded as an output symbol emitted by a state corresponding to a field of the bibliographic record. Figure 5 shows an example HMM for parsing bibliographic records. States are labeled by field

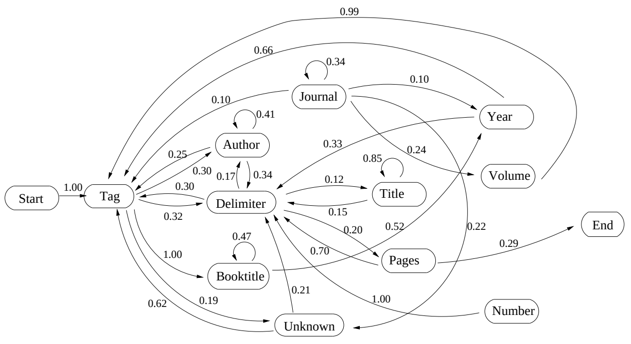


Figure 5. An example HMM.

names and edges connecting states are labeled with transition probabilities. For example, after seeing a *journal* token, the probability of seeing a *year* token next is 0.1, while the probability of seeing another *journal* token is 0.34. For compactness, some states and edges are omitted, and no emission probabilities are shown.

Note that an HMM may contain special states that do not correspond to any fields of a bibliographic record, e.g., *start*, *end*, *delimiter*, and *tag*. *Start* and *end* states denote the start and end of a raw record. The choice of special states greatly impacts the effectiveness of the HMM parser, and this point will be further explored in Section 8.3. Figure 5 illustrates only one of the possible methods (and in fact *not* the best method) for structuring the HMM.

Given a properly trained HMM with states labeled by field names, the problem of parsing a raw record reduces to the standard problem of choosing an optimal *state sequence* for a given *observation sequence*. Here, the observation sequence is the sequence of tokens in the raw record, and the state sequence would assign each taken to a specific field. The optimal state sequence in this case is the most probable state sequence, i.e., one that generates the observation sequence with the highest probability.

Given n states and an observation sequence of length l, there exists n^l distinct state sequences that could possibly generate the observation sequence. Thus, a brute-force approach that enumerates all possible state sequences has a cost of $O(n^l)$. This cost can be significantly reduced to $O(l \cdot n^2)$ by the well-known Viterbi algorithm [20] based on dynamic programming. We use a standard implementation by Kanungo [21]. Detailed description of the algorithm can be found in the full version of this paper [15].

8.2. Training the HMM

We train the HMM parser using the annotated raw records generated by the matcher (Section 7.2). First, we need to choose the set of normal and special states in the HMM. We defer the discussion of this problem to Section 8.3. In the following, we briefly discuss what constitutes a unique output symbol in the HMM, and how we learn the transition matrix (A), the emission matrix (B), and the initial probability vector (π).

Feature selection. It is possible to treat each distinct token in the raw record as a unique output symbol, but this approach discards the common features among tokens of certain types. For example, although "`1975`" and "`2003`" are distinct tokens, they obviously share the common feature of being 4-digit numbers (and hence possibly years). Since the number of distinct tokens is huge and the training cannot possibly cover them all, it is important to leverage the common features among tokens.

We use the standard technique of hierarchical feature selection. Our feature taxonomy is adapted from [8]. All 4-digits number tokens are treated as one special output symbol, and all other number tokens as another output symbol. Distinct word tokens are treated as distinct output symbols. Finally, all delimiter (or tag) tokens are treated as a single delimiter (or tag) output symbol. The original taxonomy in [8] was shown to provide best performance in segmenting texts containing address information; it did not distinguish 4-digit numbers from other numbers. For bibliographic data, we find that making this distinction yields better performance, because years occur frequently in bibliographic records.

Learning A, B and π. The training data consists of a set of annotated raw records, each of which corresponds to a sequence of symbol-state pairs. We make one pass over the training data and calculate the transition matrix as follows:

$$a_{ij} = \frac{\text{total number of transitions from } q_i \text{ to } q_j}{\text{total number of transitions out from } q_i}.$$

The emission matrix is calculated by:

$$b_{jk} = \frac{\text{total number of times that } q_j \text{ emits } v_k}{\text{total number of times that } q_j \text{ emits any symbol}}.$$

The above formula for b_{jk} needs to be smoothed because the training data does not cover all possible distinct output symbols. If the parser encounters an output symbol that has not been seen during training, the above formula will assign a probability of zero to this unseen symbol, making the entire observation sequence improbable. Following [8], we use *absolute discounting* to redistribute some probability to unseen symbols. The absolute discounting method captures the intuition that the probability of an unseen symbol in a state q increases with the total number of distinct symbols emitted by q during training.

To simplify the learning of π, we use two special states *start* and *end*. Every observation sequence starts in *start* and ends in *end*. Therefore, the initial probability is 1 for *start*, and 0 for all other states.

8.3. Choosing a Structure for the HMM

The choice of states in an HMM plays a crucial role in the performance of our parser. We have evaluated four methods of structuring an HMM. They all use special states *start* and *end* in addition to *normal states* that correspond to fields in bibliographic records, e.g., *author*, *title*, *year*, etc. Also, they all use a special state *unknown* to capture unknown tokens in raw records (e.g., tokens "`121`" and "`EE`" in Figure 4 would be emitted by *unknown*). However, these methods differ in the extent to which they take advantage of the structural information encoded by delimiters and tags. We first describe these methods, starting from the simplest. We conclude this section with experiment results.

Method I: no delimiters or tags. With this method, all delimiter and tag symbols are removed from observation sequences, and the HMM only includes *start*, *end*, *unknown*, and normal states. This simple approach serves as a basis for comparison with other approaches. Without any delimiters and tags, the HMM has trouble detecting boundaries between fields. The presence of unseen symbols greatly exacerbates the problem.

To illustrate the problem, consider a simple HMM with only two normal states *author* and *title* shown in Figure 6. Suppose that all training records have the form $(\textit{author}^3, \textit{title}^3)$.[1] Hence, there is clear boundary between *author* and *title*. The first three tokens always belong to *author* and the last three always belong to *title*. The result transition probabilities are shown on the edges in the graph. Unfortunately, given an observation sequence of six unseen symbols, the HMM cannot correctly determine the boundary between *author* and *title*. Specifically, assuming that the emission probability of an unseen symbol (after applying the smoothing method) in both states is 0.1, the probability for the state sequence $(\textit{author}^3, \textit{title}^3)$ to generate the given observation sequence is:

$$\Big(1 \times (2/3)^2 \times (1/3) \times (2/3)^2 \times (1/3)\Big) \times (0.1)^6.$$

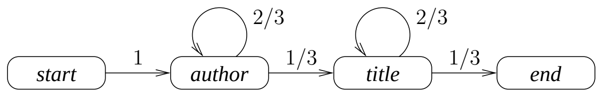
On the other hand, the state sequence $(\textit{author}, \textit{title}^5)$ has the exact same probability:

$$\Big(1 \times (1/3) \times (2/3)^4 \times (1/3)\Big) \times (0.1)^6.$$

Thus, the HMM is unable to determine the boundary between *author* and *title*, even though the HMM appears to be able to capture the expected length of each field (which can calculated from its self-loop probability).

What if all records are of the form $(\textit{author}^4, \textit{title}^2)$, i.e., *author* always contains four tokens and *title* has two? The self-loop probabilities on *author* and *title* become 0.75 and 0.5, respectively. Here, the most probable state sequence is unique,

1 We use q^k as a shorthand for a sequence of k q's.



$$\begin{tikzcd} \textit{start} \arrow[r, "1"] & \textit{author} \arrow[loop above, "2/3"] \arrow[r, "1/3"] & \textit{title} \arrow[loop above, "2/3"] \arrow[r, "1/3"] & \textit{end} \end{tikzcd}$$

Figure 6. No delimiter/tag states.

but it turns out to be $(\textit{author}^5, \textit{title})$, again incorrect because the HMM becomes overly biased towards *author*.

Method I fares better if the states have low self-loop probabilities, i.e., each state tends to emit just one token before transitioning to the next state in a sequence. However, many fields in bibliographic records can contain multiple tokens, e.g., *title*, *journal*, *booktitle*, etc., so we expect Method I to perform poorly in these cases.

Method II: delimiters and tags emitted by normal states. This method preserves all delimiter and tag symbols in observation sequences, and considers them to be emitted by normal states. In the training data, each delimiter/tag token in a matched raw record is annotated with the field name of the preceding token. By taking advantage of delimiters and tags in input, this method outperforms Method I. Even for an observation sequence consisting entirely of unseen word symbols, delimiter and tag symbols can still serve as good field separators.

However, there is another problem. Consider again the same simple example of *author* and *title* parsing. Again, suppose that *author* and *title* each contains exactly three word tokens, with a single delimiter token separating them. During training, Method II treats the delimiter token as belonging to *author*. Therefore, during parsing, a delimiter token will likely be picked up by *author*, allowing the HMM to assign the three preceding tokens correctly to *author*. Unfortunately, nothing prevents *author* to pick up succeeding tokens. In fact, the HMM is now more inclined to do so since the self-loop probability of *author* exceeds that of *title* when the delimiter is included in *author*. Intuitively, we know that a delimiter should be the *last* symbol emitted by *author*, so succeeding tokens should belong to *title*. However, the HMM is unable to capture the sequential ordering of emitted symbols when in a self-looping state.

Method II has also been considered by Borkar et al.and is called a "naive HMM" in their work [8]. To overcome the limitation of a naive HMM, they propose the use of a "nested HMM." In a nested HMM, each outer HMM state contains an inner HMM to capture the sequencing of tokens within fields. An inner HMM uses a parallel path structure to handle a field with variable number of tokens. The accuracy of nested HMM's comes at the cost of increased complexity. We find that our Method IV, discussed in detail later in this section, can achieve comparable accuracy without the complexity of nested HMM's.

Method III: single delimiter and tag states. In addition to *start*, *end*, *unknown*, and normal states, Method III introduces two more special states *delimiter* and *tag*. The HMM

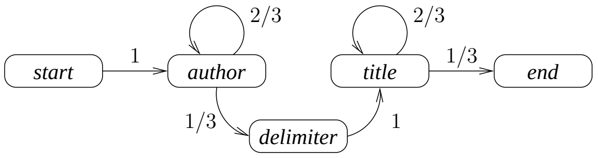


Figure 7. With delimiter/tag states.

shown in Figure 5 is in fact trained using this method. In the training data, delimiter and tag tokens in matched raw records are annotated with *delimiter* and *tag*, respectively.

Method III avoids the problem of Methods I and II by encouraging a state transition whenever a delimiter/tag symbol is encountered. Once again, consider the simple example of parsing *author* and *title*. A new HMM structured using Method III is shown in Figure 7. All training records now have the form $(\textit{author}^3, \textit{delimiter}, \textit{title}^3)$. The result transition probabilities are shown on the edges in the graph. Also, assume that the emission probability of an unseen symbol is 0.1 for *author*, *delimiter*, and *title*. Consider again the observation sequence consisting of three unseen symbols followed by a delimiter and then three more unseen symbols. In this new HMM, the probability for the state sequence $(\textit{author}^3, \textit{delimiter}, \textit{title}^3)$ is:

$$\begin{array}{ll} & \Big(1 \times (2/3)^2 \times (1/3) \times 1 \times (2/3)^2 \times (1/3)\Big) \\ \times & \Big((0.1)^3 \times 0.9 \times (0.1)^3\Big). \end{array}$$

For $(\textit{author}, \textit{delimiter}, \textit{title}^5)$, the probability is:

$$\Big(1 \times (1/3) \times 1 \times (2/3)^4 \times (1/3)\Big) \times (0.1)^7.$$

The probability of the correct state sequence is 9 times that of the incorrect sequence, making it easier for the HMM to pick the correct one.

However, the use of single delimiter and tag states have an unintended consequence. Although they help in detecting boundaries between normal states with significant self-loop probabilities, they destroy the HMM's ability to capture sequential ordering of normal states. We illustrate this problem through the following simple example. Consider a sequence of fields A, B, C, and D, each of which contains a single token, with delimiter tokens in between them. An HMM structured using Method I or II, shown in Figure 8, would perform well since all states have low self-loop probabilities. On the other hand, if we introduce a single delimiter state, we would obtain the HMM structure shown in Figure 9. Now, the ordering information—that B always precedes C—is lost because A, B, and C all transition into the common delimiter state. As a result, the HMM might have trouble distinguishing B and C tokens.

To alleviate this problem to some extent, we may use the following heuristic. In the training data, we annotate delimiter (or tag) tokens following multi-token fields with *delimiter* (or *tag*, respectively), but we ignore delimiter and tag tokens following single-token fields. This heuristic allows the HMM to benefit from the separate delimiter and tag states while avoiding their adverse effects on sequences of normal states with low self-loop probabilities. However, this heuristic still fails to address the loss of ordering information in general. To solve this problem completely, we need the method described next.

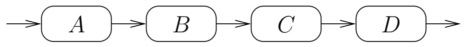


Figure 8. A sequence of HMM states with no delimiter/tag states.

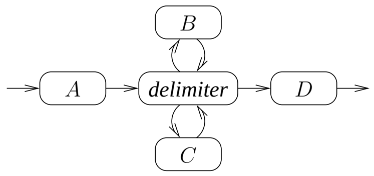


Figure 9. A single delimiter/tag state.

Method IV: multiple delimiter and tag states. This method introduces separate delimiter and tag states for each normal state, e.g., *author-delimiter* and *author-tag* for *author*, *title-delimiter* and *title-tag* for *title*, etc. In the training data, if a delimiter (or tag) token follows a token that has been annotated with f, we annotate the delimiter (or tag) token with f*-delimiter* (or f*-tag*, respectively).

Method IV has the same advantage as Method III in detecting field boundaries, but it does not have the problem of Method III in preserving ordering information among fields. For the previous example of parsing a sequence of fields A, B, C, and D, an HMM obtained using Method IV would have the structure shown in Figure 10. A disadvantage of Method IV is that it does increase the number of states in the HMM by a factor of 3. On the other hand, the HMM has a simpler structure than one that would be produced by Method III, because only state f can transition into f*-delimiter* and f*-tag*. Furthermore, in the context of parsing bibliographic data, the number of normal states, which corresponds to the number of fields in bibliographic records, is very small to begin with (on the order of 20). Therefore, the constant factor increase in size is manageable. Compared with nested HMM's [8], HMM's produced by Method IV still have much simpler and more compact structures.

Experiment results. To evaluate the four methods described above, we conduct experiments on both DBLP and CSWD. In these experiments, we train HMM's using 115 and 117 annotated raw records (generated automatically by the matcher as discussed in Section 7) from DBLP and CSWD, respectively. Using these HMM's, we parse 50 unmatched raw records from each of the respective sources. The total number of word and number tokens in these raw records is 1213 for DBLP and 1522 for CSWD. In order to measure the perfor-

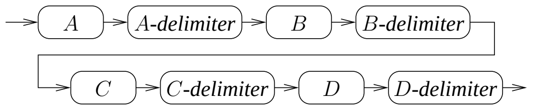


Figure 10. Multiple delimiter/tag states.

mance of the HMM's, we manually parse the raw records and compare the results with those produced by the HMM's. The success rate is defined as:

$$\frac{\text{number of word/number tokens labeled correctly by the HMM}}{\text{total number of word/number tokens}}.$$

Note that delimiter and tag tokens are excluded by this metric. They are generally identified by the HMM's perfectly, so including them would only inflate the success rate.

The success rates of HMM's produced using the four methods are compared in Table 3. Method I, which completely ignores delimiters and tags, has success rates of 89.1% (DBLP) and 87.2% (CSWD). From Method I to Method IV, success rates steadily improve, except between Methods II and III for CSWD. Method IV, which is now powering AUTOBIB, achieves good success rates of 98.9% (DBLP) and 93.4% (CSWD). Overall, the results confirm the benefits of using delimiters and HTML tags in parsing, and validate our analysis of the four HMM-structuring methods. The scale of the current experiments is not as large as we really want because of the significant amount of manual work involved in measuring success rates. While we still need experiments with a larger number of raw records from more sources, we do not expect the new results to change our conclusions significantly. After all, our current experiments, albeit not large, are based on arbitrarily chosen bibliographic records and sources.

9. Conclusion and Future Work

To conclude, we have implemented a framework for extracting bibliographic information from multiple sources on the Web. Using a combination of bootstrapping, statistical, and heuristic methods, we are able to achieve a high degree of automation and accuracy for the domain we consider. The only source-specific information that needs to be supplied manually is how to query each source for a particular author and how to follow "next page" links when multiple pages are returned. As of February 2003, AUTOBIB is fully functional and accessible at `http://www.cs.duke.edu/dbgroup/autobib/`. In the following, we highlight some of the lessons we have learned from our experience of building AUTOBIB:

- Content overlap among sources can be a blessing for information extraction (although it also can be a curse for information integration). In AUTOBIB, it allows automatic generation of training examples.
- HMM's are very effective in parsing raw records into structured records. They are also robust in dealing with variations in the record structure such as optional fields and different field orderings.
- Delimiters and HTML tags are extremely useful in extraction because of the structural information they encode. This observation is not new, but it is still surprising to see how effective delimiters and tags can be in every part of the system, including structuring the HMM.
- While truly automatic extraction may unattainable in general, we may be able to approach it at least for specific domains (such as bibliographic information). A fair amount of domain-specific knowledge goes into building AUTOBIB, but almost no source-specific knowledge needs to be supplied manually.

We hope that the above observations, as well as how we have put together the collection of new and existing techniques into one system, can be useful to other researchers and practitioners of automatic Web information extraction.

There are many possible directions for future work:

- We need to investigate how the size of the training set affects the performance of an HMM-based parser. Currently, we use all matches between raw and structured records to generate the training set, but a much smaller subset could be used instead without affecting accuracy.
- In this paper, we have not discussed how to merge structured records from different sources. Currently, AUTOBIB uses the simple matching algorithm in Section 7.1 to detect duplicates; conflicting information on the field level is resolved arbitrarily. We need a better approach to handling overlapping and conflicting information.
- No matter how good a computer algorithm is, there are always special cases that require human input. For such cases, the system should provide enough information to help the user in making a manual decision. We plan to augment AUTOBIB to track the "lineage" of each record (including its raw and structured forms and a pointer to the original Web page), allowing the user to drill down to different levels to locate the root of the problem.
- We plan to extend our work to other application domains such as comparison shopping, where there is also significant amount of content overlap among sources.

	DBLP	CSWD
Method I	89.1%	87.2%
Method II	91.6%	89.5%
Method III	94.1%	89.1%
Method IV	98.9%	93.4%

Table 3. Success rates of four HMM-structuring methods.

Acknowledgment. We thank Vinayak R. Borkar and Sunita Sarawagi for their insightful discussions of HMM-based parsers and critique of this paper. We also thank Ronald E. Parr and Yuguo Chen for their comments on the full version of this paper.

References

[1] CiteSeer: Scientific Literature Digital Library. `http://citeseer.nj.nec.com/`.

[2] CompuScience WWW-Database. `http://turing.zblmath.fiz-karlsruhe.de/cs/`.

[3] DBLP Computer Science Bibliography. `http://dblp.uni-trier.de/`.

[4] The Collection of Computer Science Bibliographies. `http://liinwww.ira.uka.de/bibliography/`.

[5] B. Adelberg. NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 283–294, Seattle, Washington, June 1998.

[6] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, June 2003.

[7] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: A high-performance learning name-finder. In *Proc. of the 5th Conf. on Applied Natural Language Processing*, pages 194–201, Washington, D.C., 1997.

[8] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, pages 175–186, Santa Barbara, California, May 2001.

[9] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the World Wide Web. In *Proc. of the 2001 Intl. Conf. on Distributed Computing Systems*, pages 361–370, April 2001.

[10] D. Buttler, L. Liu, C. Pu, H. Paques, W. Han, and W. Tang. OminiSearch: A method for searching dynamic content on the Web. In *Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of Data*, Santa Barbara, California, May 2001.

[11] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proc. of the 1999 National Conf. on Artificial Intelligence*, pages 328–334, Orlando, Florida, July 1999.

[12] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proc. of the 2001 Intl. Conf. on Very Large Data Bases*, pages 109–118, Roma, Italy, September 2001.

[13] D. Embley, S. Jiang, and Y. Ng. Record-boundary discovery in Web documents. In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, pages 467–478, Philadephia, Pennsylvania, June 1999.

[14] D. Freitag and A. K. McCallum. Information extraction with HMMs and shrinkage. In *Proc. of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[15] J. Geng. Automatic extraction and integration of bibliographic information on the web using hidden Markov models. Master's thesis, Duke University, 2003.

[16] P. B. Golgher, A. S. da Silva, A. H. F. Laender, and B. A. Ribeiro-Neto. Bootstrapping for example-based data extraction. In *Proc. of the 2001 Intl. Conf. on Information and Knowledge Management*, pages 371–378, Atlanta, Georgia, November 2001.

[17] P. B. Golgher, A. H. F. Laender, A. S. da Silva, and B. A. Ribeiro-Neto. An example-based environment for wrapper generation. In *Proc. of the 2nd Intl. Workshop on the World Wide Web and Conceptual Modeling*, pages 152–164, Salt Lake City, Utah, 2000.

[18] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the Web. In *Proc. of the Workshop on Management of Semistructured Data*, pages 18–25, Tucson, Arizona, May 1997.

[19] W. Han, D. Buttler, and C. Pu. Wrapping Web data into XML. *SIGMOD Record*, 30(3):33–38, 2001.

[20] D. Forney Jr. The Viterbi algorithm. *Proc. of the IEEE*, 61(3), March 1973.

[21] T. Kanungo. The UMD HMM package. `http://www.cfar.umd.edu/~kanungo/software/software.html`.

[22] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the Web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000.

[23] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proc. of the 1997 Intl. Joint Conf. on Artificial Intelligence*, pages 729–737, Nagoya, Japan, 1997.

[24] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68, 2000.

[25] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of Web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[26] T. R. Leek. Information extraction using hidden Markov models. Master's thesis, University of California at San Diego, 1997.

[27] B. A. Ribeiro-Neto, A. H. F. Laender, and A. S. da Silva. Top-down extraction of semi-structured data. In *Proc. of the 6th Symp. on String Processing and Information Retrieval*, pages 176–183, Cancun, Mexico, September 1999.

[28] E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proc. of the 1999 National Conf. on Artificial Intelligence*, pages 474–479, Orlando, Florida, July 1999.

[29] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, October 1992.