# Extracting Relational Data from HTML Repositories

Ruth Yuee Zhang
Dept. of Statistics
Univ. of British Columbia
Vancouver, BC Canada
ruth@stat.ubc.ca

Laks V.S. Lakshmanan*
Dept. of Computer Science
Univ. of British Columbia
Vancouver, BC Canada
laks@cs.ubc.ca

Ruben H. Zamar
Dept. of Statistics
Univ. of British Columbia
Vancouver, BC Canada
ruben@stat.ubc.ca

## ABSTRACT

There is a vast amount of valuable information in HTML documents, widely distributed across the World Wide Web and across corporate intranets. Unfortunately, HTML is mainly presentation oriented and hard to query. In this paper, we develop a system to extract desired information (records) from thousands of HTML documents, starting from a small set of examples. Duplicates in the result are automatically detected and eliminated. We propose a novel method to estimate the current coverage of results by the system, based on capture-recapture models with unequal capture probabilities. We also propose techniques for estimating the error rate of the extracted information and an interactive technique for enhancing information quality. To evaluate the method and ideas proposed in this paper, we conducted an extensive set of experiments. Our experimental results validate the effectiveness and utility of our system, and demonstrate interesting tradeoffs between running time of information extraction and coverage of results.

## Keywords

Information Extraction, Pattern, Coverage Estimation, Duplication

## 1. INTRODUCTION

The majority of the data on the web is encoded in HTML. Unfortunately, HTML documents are mainly presentation oriented, usually not well-structured and hard to query and analyze. Searching (e.g. using keyword search on search engines) and browsing offer but a limited approach for extracting information from HTML documents. Thus, converting information extracted from HTML repositories into structured form is valuable. The extracted information, e.g., records, can be marked up into XML in a subsequent step for consumption by applications.

For example, we could extract up to date data about job postings including `title, company`, `qualification`, `location` and `salary` from a large HTML repository such as (any fraction of) the World Wide Web. Or we could generate structured catalogues consisting of `models, brands` and `prices` of cars, computers and cameras available from a variety of online stores or dealerships. Then queries and analyses could be performed on the data so extracted and represented. The vision behind our project is to develop a general purpose system for methodically and systematically converting large HTML collections into XML, and provide some assurances about the coverage and quality of the results. This conversion entails (i) the extraction of records from the HTML repository and (ii) the subsequent marking up of the extracted data into XML. In this paper, we address record extraction and computing coverage and quality assurances.

Our system consists of two phases. Phase I includes extraction of records and their translation to structured form. Duplication of information is detected automatically and eliminated. In addition, the *coverage* of the result set with respect to the source HTML data set is estimated. Phase II is aimed at assessing and enhancing the quality of the extracted information. This phase allows the user to provide feedback to the system by giving examples of what she considers erroneous information from random samples presented to her. Based on the feedback, the system automatically "cleans" up previously extracted information. While the user is not forced to interact, the value-added service for user interaction that is offered by the system is *enhanced information quality*.

Records extraction in Phase I uses the framework of the method DIPRE (Dual Iterative Pattern Relation Expansion) pioneered by Brin [5]. One attractive feature of DIPRE is that records are extracted from thousands of web pages which are usually structured very differently, as opposed to one web page or even several similarly structured web pages. We implemented this framework, extend it to support multivariate records involving more than two attributes, and improved it in several ways. Figure 1 shows the structure of this method. Starting from a small seed set of examples (records), the program finds the occurrences of those seeds on a web page repository. Occurrences are grouped and patterns are recognized. Next the program searches the repository to find occurrences that match at least one of the patterns. New records are extracted from these occurrences according to the content of the patterns. We take this set as the new seed set and repeat this process until some termination criterion is met.

Because the repository is usually very large, it is not possible to run our procedure until full convergence, i.e., until no new records are found. To remedy this problem we introduced the concept of *coverage*, defined as the percentage of records currently recovered in relation to the records that can be recovered after full convergence is achieved. The user

*This author's work was supported by grants from NSERC and NCE/IRIS.

can set a desired coverage and the procedure will estimate the coverage after each iteration and continue until the desired coverage is achieved. Coverage estimation is further discussed in Section 3.1.

Owing to the large scale of heterogeneous information resources, it is inevitable that there will be erroneous records contained in the results. In phase II we assess the quality of results by estimating the error rate. Users are presented a random sample of records and asked to indicate which record is an error. Error estimation and quality enhancement are further discussed in Section 3.2.

Our paper makes the following contributions:

- Building on the DIPRE method, we develop techniques for extracting records from thousands of HTML documents. Occurrences and patterns are defined in a more flexible way compared to DIPRE, without increasing the error rate in the results. At the end of each iteration, we select the most reliable records to start the next iteration. Experiments show that these techniques can improve the efficiency, produce high coverage and low error rates.
- We propose a methodology to evaluate the coverage achieved at the end of each iteration. Statistical methods based on capture-recapture models with unequal capture probabilities are applied to estimate the coverage rate of the extracted results. This can be used to determine when to stop the iterations, based on a user specified coverage threshold.
- We propose a method for estimating the error rate of the extracted results, by having the user to identify correct and erroneous records in a random sample. The size of the sample is automatically determined based on a specified level of confidence in the estimated error rate.
- We propose an interactive method to enhance the quality of the results. Specifically, our method is able to track down the patterns that led to errors, thus flagging problem patterns. Using interactive feedback from the user, we can estimate the error rate in the data discovered using a problem pattern. We propose useful techniques for cleaning up erroneous results that came from "problem patterns".

The rest of the paper is structured as follows. Section 1.1 discusses related work. In Section 2, we describe the procedure for extracting records from HTML files. In Section 3, we discuss the coverage estimation and data cleaning in details. Section 4 gives the experimental results. Finally Section 5 discusses future work.

## 1.1 Related Work

There has been much work on information extraction from unstructured and semistructured data sources. Laender et al. [12] is an excellent survey on recent web information extraction tools which compares them on several qualitative aspects such as degree of automation, support for complex structure, etc.

The traditional approach for extracting web information is to write specialized programs, called wrappers, that identify and extract data of interest. TSIMMIS [11] is a major example of declarative language based tools for wrapper generation. It generates wrappers through specification files,

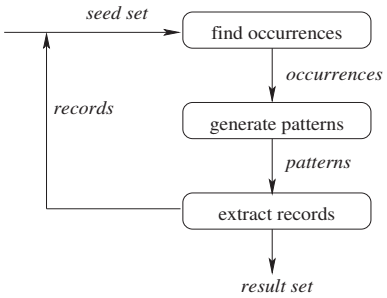


Figure 1: The General Structure of DIPRE

written by the user, which describe the structure of HTML pages.

RoadRunner [8] and EXALG [1] generates wrappers automatically to extract data from template-generated web pages. A visual web information extraction tool *Lixto* is proposed in [2; 3; 4; 10]. *Lixto* generates wrappers semi-automatically by providing the visual interface and browser-displayed example pages that allow a user to specify the desired extraction patterns. [10] discusses the application of monadic datalog, a declarative query language for information extraction. Freitag [9] makes use of machine learning techniques for information extraction in specific domains. Recently, an elegant algorithm called MDR was proposed [14] for extracting data records marked up using table and form related HTML tags. The algorithm is based on the premise that data in a web page is contained in one or more data regions. Each data region contains several data records, all of which are marked up using similar tags. Data regions in a page need not be contiguous. The algorithm works by identifying data regions first and then extracting data records in them.

To the best of our knowledge, our system is the first to provide a detailed analysis of coverage of results extracted and quality of extracted information as well as an interactive approach for quality enhancement. It is worth mentioning another evaluation metrics, recall, which is used by the Message Understanding Conferences (MUC) to evaluate information extraction systems. Recall is defined as the number of correctly extracted records over the number of records in answer key. Recall is used to experimentally evaluate various information extraction systems. The information sources are restricted to a fixed range and all the useful records (answer key) are known, usually gathered manually. The notion of coverage has advantages comparing to recall. Given the large scale of the Web and other HTML repositories, we cannot know exactly how many records are included in these sources. For the same reason, the information extraction system is computationally expensive and usually takes a huge amount of time to reach convergence. Coverage can estimate how much information could be extracted when convergence could be reached, and what proportion of all findable information has already been extracted.

# 2. RECORD EXTRACTION

Starting from several examples of target records, our goal is to extract records from a large collection of web pages. A record (e.g. a book) usually has several attributes (e.g. author, title, publisher, year and price). Here we discuss the

```
url: www.1-electronics-store.com/product+Canon
-i70-Color-B00008CMU9.html
attribute1: Canon i70 Color Bubble Jet Printer
attribute2: Canon
attribute3: $217.34
order: 0
prefix: <td><a><b>
separator1: </b></a><br><a>
separator2: </a><br>Price:
suffix: </a></td></tr>
```

Figure 3: Example of Finding Printers: An Occurrence

four steps in an iteration of our extracting procedure.

## 2.1 Finding Occurrences

This step is to find occurrences of the records in the seed set. An occurrence of a record is the context surrounding attributes of this record in a web page (an HTML file). It is defined as a tuple over url, order, prefix, separators, suffix and values of attributes of the record, where:

*url* is the URL of the web page that the record appears in.

*order* is a number indicating the order that the attributes show up in. For example, there are six possible orders for records with three attributes.

*prefix* is the names of tags and contexts preceding the attribute appearing first.

*separators* are the names of tags and context between attributes.

*suffix* is the names of tags and contexts following the last attribute.

For each record, regular expressions are constructed by combining its attributes in different orders. Then the program looks through all HTML files to search texts that match these regular expressions.

From each of the matching text, we get an occurrence by extracting the prefix, separators and suffix as we described and set the proper order. We also record the URL of the web page for this matching.

## 2.2 Generating Patterns

A pattern is used to summarize a group of occurrences. It is defined as a tuple over (*urlprefix, order, prefix, separators, suffix*). The occurrences are first grouped by order and separators. Patterns are generated for each of these groups. The *order* and *separators* of a pattern are the same as those of the occurrences in the group. For each group, the longest matching prefix of all the url's is the *urlprefix* of the pattern, the longest matching suffix of all the prefixes is the *prefix* of the pattern, and the longest matching prefix of all the suffixes is the *suffix* of the pattern.

Figures 2, 3 and 4 are examples of HTML scripts, an occurrence and a pattern of printers (model, manufacturer, price).

The basic algorithm for generating patterns is shown in Figure 5. The input is all occurrences, which is denoted by $O$, found in the previous step. First, these occurrences are partitioned into groups $O_1, \ldots, O_k$ according to the order and separators. The occurrences in each group have the same order and separators. Then for each of those groups that have more than one occurrence, GenOnePattern() is called to generate a pattern $p$.

```
urlprefix: www.1-electronics-store.com
order: 0
prefix: <td><a><b>
separator1: </b></a><br><a>
separator2: </a><br>Price:
suffix: </a></td></tr>
```

Figure 4: Example of Finding Printers: A Pattern

1. $(O_1, \ldots, O_k)$ = GroupOccurrences($O$)
2. for $i = 1$ *to* $k$
   if $|O_i| == 1$ then next;
   else GenOnePattern($O_i$)
   (a) $p.urlprefix$ = FindUrlprefix(all urls of $O_i$)
   (b) $p.prefix$ = FindLongestSuffix(all prefixes of $O_i$)
   (c) $p.suffix$ = FindLongestPrefix(all suffixes of $O_i$)
   (d) if $p.urlprefix$==NULL
   SubGroupbyUrl($O_i$), goto step 2
   elseif $p.prefix$==NULL
   SubGroupbyPrefix($O_i$), goto step 2
   elseif $p.suffix$==NULL
   SubGroupbySuffix($O_i$), goto step 2
   else
   $p.basedoccur$=$|O_i|$, push($P$, $p$)

Figure 5: The Algorithm for Generating Patterns

The procedure GenOnePattern() first try to generate the *urlprefix*, *prefix* and *suffix* of a pattern according to their definitions. If the urlprefix is null, no pattern is generated. This group of occurrences has to be subgrouped by prefixes of the urls, that is, in each subgroup either occurrences have a common prefix of urls or there is only one occurrence in that subgroup. GenOnePattern() is called again for these subgroups. If the prefix is null, this group is subgrouped by suffixes of the prefixes, i.e., occurrences in each subgroup have a common suffix of prefixes. The same action is taken if the suffix is null.

If none of them is null, a pattern $p$ is generated. The number of occurrences that support this pattern is assigned to $p.basedoccur$. The output is a set of patterns $P$.

Here is an example of subgroup of occurrences by suffixes of prefixes. The prefixes of occurrences in a group are `<b></b><i>`, `<a></b><i>`, `<b></b><br>`. The first two occurrences have the common suffixes of prefixes, so they are going to be in a subgroup after the SubGroupbyPrefix procedure applied. The third occurrence falls in another subgroup.

## 2.3 Extracting Records

In this step, the program searches through all the HTML files to find new occurrences that match one of the patterns obtained in previous step. An occurrence is said to match a pattern if there is a piece of text that appears in a web page whose url matches the `urlprefix*`, and matches the regular expression `*prefix *separators* suffix*` where '`*`' represents any string. A new record can be extracted from an occurrence. Values of attributes are extracted and assigned according to the order of the pattern.

The index of the pattern that a record matches is saved with this record. *It plays a key role in data quality assessment and enhancement.*

Going over all the HTML files and patterns, the system produces a set of records as well as indexes of patterns they match.

```
www.1-electronics-store.com/product+Canon-i70-Color-B00008CMU9.html
  ...<tr><td align="center">...</td><td><a href="/product+Canon-i70-Color-B00008CMU9.html"><b>
  Canon i70 Color Bubble Jet Printer</b></a><br>Manufactured by <a href="/man+Canon.html">
  Canon</a><br>Price: $217.34</a></td></tr>...

www.1-electronics-store.com/product+HP-LaserJet-1012-B0000C1XHY.html
  ...<tr><td align="center">...</td><td><a href="/product+HP-LaserJet-1012-B0000C1XHY.html"><b>
  HP LaserJet 1012 Printer</b></a><br>Manufactured by <a href="/man+Hewlett-Packard.html">
  Hewlett Packard</a><br>Price: $149.99</a></td></tr>...
```

Figure 2: Example of Finding Printers: HTML Scripts from two HTML files

## 2.4 Eliminating Duplicates

It happens very often that some records are found more than once. Usually duplication can not be identified by merely comparing the values of attributes directly. For example, the author of a book may appear in different formats, such as "A. E. van Vogt" and "Van Vogt,A. E.". We apply an approximate string matching method to detect those kinds of duplicates.

After the four steps of an iteration, a set of records is obtained. To track the relationships between occurrences and patterns, for each record, we store a list of indices of patterns that this record matches.

Intuitively, the records that have more occurrences are more likely to be correct. The records that have more than one occurrence can be selected as the seed set of the next iteration. One of the advantages of this method is that, the next iteration starts from more reliable records and less noise will be added to the pattern generation process. This will also reduce the number of erroneous records in the resulting database. Another advantage is that the next iteration starts from a smaller number of records therefore saves running time. Because records with more occurrences are selected, patterns are more likely to be generated from these records. In our experiments, we noticed that when we use a set of records that have more than one occurrence as seed into the next iteration, it runs faster and returns almost the same number of new records as if we had used all records as seed, but with better quality.

# 3. COVERAGE AND ERROR ESTIMATION

One of our goals is to capture a high percentage of all the records available in an HTML repository. To that effect, we implemented an automatic procedure to estimate, after each iteration, the percentage of all the records available in the repository that are already included in our result (result = set of recovered records). Another goal is to minimize the number of errors in our result. To that effect, we implemented an interactive (semi-automatic) procedure to estimate and reduce the percentage of erroneous records.

In what follows, we give some details of the statistical procedures employed in our system.

## 3.1 Coverage Estimation

Given the large scale of the web or other HTML repository, searching for records is computationally expensive. Therefore we wish to minimize the number of iterations, especially when using large seed sets. Instead of running our system until full convergence (i.e., until no new record is found) we stop when a given coverage target has been met.

We define the "population size" $N$ as the number of records our system could possibly find, i.e., the size of the result set after full convergence. Let $N_i$ be the size of the result set after the $i$th iteration. The current coverage is defined as

$$C_i = \frac{N_i \times 100}{N}. \quad (1)$$

Obviously, the denominator in (1) is unknown and must be estimated in order to estimate $C_i$.

In the next subsection, we discuss the adaptation of "capture-recapture" models and techniques – widely used in Biology to estimate the size of wildlife populations – to estimate the size of (the no less wild) population of "findable" records.

### 3.1.1 Capture-Recapture Models

The capture-recapture models were originally used by biologists to estimate the size of wildlife populations. The basic idea is to set traps to capture some animals and release them after they have been marked. A second trapping is conducted after the animals have had enough time to return and mix back with their population. The number of recaptured animals can then be used to estimate the size of the population. Capture-recapture models and techniques have been used in other areas such as estimating the size of the indexable Web and the coverage of search engines [13]. In our case, instead of animals we "capture" records and each trapping occasion corresponds to an iteration of our extraction system. Starting from a randomly selected seed set of records (a trap), we run the next iteration of the extraction program to get a larger set of records (captured objects).

We now briefly introduce the basic ideas underlying capture-recapture procedures. In the first trapping occasion of the capture-recapture experiment, a number $n_1$ of individuals or records are captured, marked and released. At a later time, the second trapping occasion, a number $n_2$ of individuals or records are captured, of which, say $m_2$ have been marked. The Petersen estimator for the population size is based on the observation that the proportion of marked individuals in the second sample (recaptured sample) should be close to the same proportion of marked individuals in the total population, i.e.

$$\frac{m_2}{n_2} = \frac{n_1}{N}$$

Thus the estimator for population size is

$$\hat{N} = \frac{n_1 \times n_2}{m_2}$$

The general assumptions for this basic capture-recapture model are: (i) the population is closed; (ii) all marks are correctly noted and recorded; (iii) marks are not lost; (iv) each individual has a constant and equal capture probability on each trapping occasion.

The first assumption means that the size of the population is constant over the period of experiment. This is a strong assumption and usually not true in the case of biological populations. Nevertheless, this assumption is valid when we deal with a static repository (e.g., a fragment of the web crawled and stored). Assumptions (ii) and (iii) are also always true in our setting. However, the last assumption is strongly violated here. Different records usually have different probabilities of being found (captured). We applied two methods to solve capture-recapture models with unequal capture probabilities. We first introduce the post-stratification method, which is transparent and easy to understand. A more complicated method, the generalized jackknife, proposed by Burnham [6], is discussed later.

### 3.1.2 Post-stratification Method

When a web page contains a large number of records, it is more likely that records from this page follow (a small number of) patterns, because the contexts of records in the same web page are usually similar. Consequently the records in this page are more likely to be found (by matching these patterns). Therefore, depending on the patterns that a record matches, it becomes easier or harder to be found. For a pattern, we define its score as the number of records matching this pattern. Patterns with higher scores are associated with higher capture probabilities.

For each trapping occasion, we obtain a set of patterns and a set of records. With each record, the program keeps a set of indexes of patterns which are matched by this record. The score of a pattern can be obtained by scanning through the set of records and counting how many times a pattern has been matched.

Based on the scores of patterns, we set up a score for each record. The score of a record is the maximum of scores of all the patterns which are matched by this record. The higher the score of a record, the larger the capture probabilities of this record.

The post-stratification analysis is based on two trapping occasions, each of which produces a set of patterns, and a set of records. From the discussion above, the score can be obtained for each pattern, as well as for each record. Then the two sets of records are combined by the following strategy. For each record: (i) if it is in both sets, it is marked 3 and the score of this record is the larger of its two scores; (ii) if it is only in set 1, it is marked 1 and its score is the same as in set 1; (iii) if it is only in set 2, it is marked 2 and its score is the same as in set 2.

Based on the compound set, the records are stratified by their overall scores. For example, records can be stratified to three strata according to their scores, greater than 10000, 1000–10000, and less than 1000. Capture probabilities of records in the same strata are assumed to be the same. Thus the basic capture-recapture model can be applied in each stratum to estimate the population size of this stratum. The total estimated population size then will be the sum of the population sizes of the strata.

The algorithm for estimating the population size based on two trapping occasions is shown in Figure 6. Variables manipulated by the algorithm are as follows.

- $x_{i1}, x_{i2}, x_{i3}$ denote the number of records in $i$th ($i = 1, 2, 3$) stratum, which are only in set 1, only in set 2 and in both sets, respectively.
- $n_i$ is the number of records in the $i$th stratum captured in set 1.

1. Calculate ScoresofPatterns($R_1$), ScoresofPatterns($R_2$)
2. Calculate ScoresofRecords($R_1$), ScoresofRecords($R_2$)
3. $R$ = Combine($R_1$, $R_2$), for each $r$ in $R$
   If $r$ only in $R_1$, $r.mark = 1$
   If $r$ only in $R_2$, $r.mark = 2$
   If $r$ in both $R_1$ and $R_2$
   $r.mark = 3$
   $r.score = \max(r.score$ in $R_1$, $r.score$ in $R_2)$
4. $(G_1, G_2, G_3)$ = StratifyByScore($R$)
5. For each $G_i$, $i = 1, 2, 3$
   $x_{ij}$ = the number of $r$ that $(r.mark = j)$, $j = 1, 2, 3$
   $n_i = x_{i1} + x_{i3}$, $m_i = x_{i2} + x_{i3}$, $x_i = x_{i3}$
   $\hat{N}_i = n_i \times m_i / x_i$
6. $\hat{N} = \hat{N}_1 + \hat{N}_2 + \hat{N}_3$

Figure 6: Algorithm of Post-Stratification

- $m_i$ is the number of records in the $i$th stratum captured in set 2.
- $x_i$ is the number of recaptured records.
- $\hat{N}_i$ is the estimated population size of $i$th stratum.
- $\hat{N}$ is the estimated population size.

### 3.1.3 Generalized Jackknife Estimate

There are detailed discussions on the capture-recapture models on closed animal populations with unequal capture probabilities in [15]. One of three types of models with unequal capture probabilities for closed populations is based on the following assumption:

*$M_h$: Each individual has its own capture probability independent of all other individuals. This probability remains the same at each trapping occasion.*

The capture probabilities for different records are usually different. For each record the capture probability is constant at each trapping occasion. So the model $M_h$ is suitable here. Suppose we do the trapping $t$ times, the basic data are the trapping histories of records, denoted by $x_{ij}, i = 1, \ldots, N, j = 1, \ldots, t$, where $N$ is the population size,

$$x_{ij} = \begin{cases} 1 & \text{if the } i\text{th record is captured on the } j\text{th trapping occasion,} \\ 0 & \text{otherwise.} \end{cases}$$

The unknown parameters are population size $N$ and the capture probabilities $p_1, \ldots, p_N$. We assume that $p_1, \ldots, p_N$ are a random sample from an unknown distribution and all $x_{ij}$'s are independent.

We define the capture frequencies as $f_0, f_1, f_2, \ldots, f_t$, where for $j = 1, \ldots, t$, $f_j$ is the number of records that have been caught exactly $j$ times in all the $t$ times trapping, and $f_0$ is the number of records never captured. The capture frequencies have multinomial distribution. The number of all different records captured in the $t$ times trapping occasions is $S = \sum_{j=1}^{t} f_j$. It could be proved that the set of capture frequencies $f_j$ is a sufficient statistic for the data $x_{ij}$.

Burnham [6] gives the generalized jackknife estimator for population size $N$. The $k$th ($k < t$) order jackknife estimator is

$$\hat{N}_k = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \left( \begin{array}{c} k \\ i \end{array} \right) (n - i)^k \hat{N}_{(t-i)} \qquad (2)$$

where the $\hat{N}_{(t-i)}$ is an unbiased estimator of population size based on all combinations of $t-i$ trapping occasions. The quantity $\hat{N}_{(t-i)}$ is calculated by

$$\hat{N}_{(t-i)} = S - \left( \begin{array}{c} t \\ i \end{array} \right)^{-1} \sum_{r=1}^{i} \left( \begin{array}{c} t-r \\ i-r \end{array} \right) f_r. \quad (3)$$

From (2) and (3) we can see that the $\hat{N}_k$ is a linear combination of the capture frequencies,

$$\hat{N}_k = \sum_{i=1}^{t} a_{ik} f_i. \quad (4)$$

The coefficients $a_{ik}$ can be obtained from (2) and (3 ).
The estimated variance of this estimator is

$$\widehat{\text{var}}(\hat{N}_k) = \sum_{i=1}^{t} a_{ik}^2 f_i - \hat{N}_k$$

The variance of $\hat{N}_k$ increases as $k$ increases. Therefore if two estimators are not significantly different, we take the one with the smaller order. We test the hypothesis that there is no significant difference between $i$th and $(i+1)$th order jackknife estimates, that is

$$H_0 : E(\hat{N}_i - \hat{N}_{i+1}) = 0, H_1 : E(\hat{N}_i - \hat{N}_{i+1}) \neq 0$$

The test statistic is given in [6]. Under the null hypothesis the test statistics has approximately a standard normal distribution. If we can not reject the null hypothesis, we take $\hat{N}_i$ as the final estimator of population size. If all of these estimators are significantly different, we take the $(t-1)$th order jackknife estimator as the final estimator.

### 3.1.4 Coverage Estimation Procedure

The extraction and coverage estimation procedures are integrated in the following way. First, starting from several popular examples of records, the extraction procedure runs for a given number of iterations (e.g. 2 or 3 iterations) or until the number of records exceeds a given threshold (e.g. 5000 records).
Then we start the coverage estimation procedure. Random samples are selected from the result of current extraction iteration. Trapping occasions are carried out by taking these random samples as seed sets. The population size is estimated based on these trapping occasions. After that, coverage of the result of the current iteration can be estimated. If the desired coverage criteria has been met, the extraction procedure stops. Otherwise the extraction procedure goes to next iteration. As we mentioned before, we could choose a subset of the result of the current iteration as the seed set of the next iteration. Estimation procedure is applied again at the end of the iteration.

## 3.2 Quality Assessment

The error rate for a set of records is defined as the number of erroneous records over the total number of records in the set,

$$r = \frac{\text{\# of erroneous records}}{\text{\# of records}}.$$

Unfortunately, the occurrence of errors in the resulting database of records is unavoidable. On the other hand, a high error rate can greatly diminish the usefulness of the database. Therefore, it is important to consider steps to estimate and reduce $r$.

**1 system:** Generate a random sample $R$ of size $n$ from the result $W$.
**2 user:** Identify true records $T$ and erroneous records $E$, using user feedback.
**3 system:** Estimate the error rate of $W$.
**4 user:** If estimated error rate is acceptable, stop else continue.
**5 system:** $PP$ = FindProblemPatterns($E$).
For each $p$ in $PP$
- **(1) system:** $B$ = ProblemSet($p$), $m = |B|$
- **(2) user:** Determine to discard or keep $B$. If we decide to keep $B$,
- **(3) system:** $R(p)$=RandomSample($B$)
- **(4) user:** Identify true records $T(p)$ and erroneous records $E(p)$.
- **(5) system:** $r(p)$=ErrorRate($p$)
- **(6) user:** Determine to discard, keep as it is or clean $B$. If we decide to clean, continue.
- **(7) system:** Clean(B)

Figure 7: Quality Assessment and Enhancement Algorithm

The first step to control $r$ is to obtain an estimate of its magnitude. We call this step "Quality Assessment". If, as a result of this step, $r$ is considered too large, then measures are implemented to reduce it. We call this second step "Quality Enhancement" and discuss it in the next section.
For interactive quality assessment and enhancement, the system initiates the dialogue described in Figure 7. The first three steps are for the quality assessment of the result. The following steps are to enhance the quality of the result. First the system generates a sample $R$ of $n$ records that are randomly selected from the resulting database $W$ of extracted records. The sample size is determined automatically by the system, based on the required level of confidence in the estimated error rate. Then the user examines $R$ manually to identify erroneous records. The error rate is estimated by the proportion of errors in $W$ and reported together with a 95% confidence interval for $r$. Based on the estimated error rate, the user decides whether just to accept $W$ as the final result or to do further quality enhancement steps. The estimation of $r$ is further discussed below. Quality enhancement is further discussed in the next section.
The estimate of the error rate, $\hat{r}$, can be obtained using simple random sampling. A relatively small number of records, $n$, are randomly selected from the database and submitted to the user for inspection. The user manually checks the records and identifies erroneous ones. The number of erroneous records, denoted by $n_e$, has hypergeometric distribution with parameters $N$, $n$ and $r$ and an unbiased estimate of $r$ is

$$\hat{r} = \frac{n_e}{n}.$$

The estimated standard error of this estimator is given in [16],

$$\widehat{\text{se}}(\hat{r}) = \sqrt{\frac{N-n}{N} \frac{\hat{r}(1-\hat{r})}{n-1}}.$$

The key issue is to choose the number of records needed to estimate the error rate with a desired precision (upper bound on the estimate standard error). If we want $\text{se}(\hat{r}) < \beta$, then

$V$=values of attribute $k$ of $(W - B)$.
For each $r$ in $B$

1. Spellingcheck($r$). If pass,
2. $v = r.attribute[k]$
3. if($v$ not in $V$) and (occ($r$)==1)
   $r$ is an error
   else $r$ is true

Figure 8: The Algorithm of Clean($B$)

we need

$$n > \frac{Nr(1-r)}{(N-1)\beta^2 + r(1-r)}, \quad (5)$$

where $N$ is the total number of records in the database.
To get $n$, we need an "initial estimate", $\hat{r}_0$, of $r$ and the desired precision, $\beta$. When there is no information about $r$, $\hat{r}_0 = 50\%$ can be used to obtain a conservative sample size $n$. However, the actual error rate of the extracted result should be much less than 50%. Otherwise we would need to reconsider the pattern generation rule. In our implementation, $\hat{r}_0 = 20\%$ , $\beta = 2\%$ are for default determination of $n$.
By central limit theory, the estimator $\hat{r}$ is approximately normally distributed with mean $r$ and variance var($\hat{r}$). A 95% confidence interval for the error rate $r$ is

$$\hat{r} \pm 1.96 \times \hat{\text{se}}(\hat{r}).$$

## 3.3 Quality Enhancement

The quality enhancement procedure first finds the problem patterns, that is, patterns that are matched by erroneous records. Recall that each record in the result has a set of indexes of patterns that it matches. For each problem pattern $p$, all records that match this pattern are called problem set $B$. The system will report features of $B$ such as the number of records and the percentage significance of this set compared to $W$. Based on this report, the user either decides to accept the result or to perform quality enhancement. For example, if there are only few records in $B$, or the proportion of a problem set $B$ is only 0.1%, the user may decide to discard the entire problem set from the result. On the other hand, if the size or relative size of $B$ is large, the user may instruct the system to improve the quality of this problem set $B$.
The error rate for the problem set $B$ can be obtained using the same random sampling method described before. Again, the user must decide here the future course of action. If the error rate is very high, this problem set should probably be discarded. If the error rate is very low, we may keep this set as it is. Otherwise, when the error rate of a problem set is moderate and the size of the set is not trivial compared to the whole set, the user can let the system move to the "cleaning" step.
Spell-checking is a simple but very powerful tool to filter out erroneous records containing alphanumeric values. The user can specify the spelling constraints on attributes of records. For example, names of people or products usually start from capital letters. Prices usually begin with currency signs and then digits.
We propose another effective method of finding erroneous records in a problem set $B$. The basic algorithm is shown in Figure 8. The method is first to verify the validity of the value of a specific attribute of a record. Based on the fact that the number of records in the result is substantial, some attributes of records are very likely to have repeated values in the result. For instance, an author usually appears more than once in a large set of books. A manufacturer or brand usually appears more than once in a large set of printers. The value ($v$) of such an attribute of a record in the problem set is compared to the values ($V$) of this attribute from all patterns other than the current problem pattern. If we cannot find $v$ in $V$, this record is likely to be an error. In addition, we check whether this record comes from one pattern or more than one pattern. If it matches only this problem pattern. It is very likely that this record is an error. Notice that it is possible that some correct records may be wrongly treated as errors. The larger the number of records in the result $W$, the smaller this probability. Experiments show the effectiveness of this method.

# 4. EXPERIMENTAL RESULTS

We successfully applied our prototype to several domains such as printers (model, manufacturer, price), papers (author, year, title, journal) and books (author, title, price). Comprehensive experiments were conducted for the simple case of two attributes: authors and titles of books. The results of these experiments are presented in this section.

## 4.1 Searching for Books

The experiments ran on a collection of 16128 HTML files used as a testbed. These files were obtained by searching the Web using Google. For each of 370 randomly selected books, we run the Google search automatically, setting the author and title of a book as the keywords. For each book the first 50 web pages of the searching results were selected. We took the union of the 370 sets of urls and successfully downloaded 16128 HTML files.
We started the experiment with 3 popular books.

| Author | Title |
|---|---|
| Isaac Asimov | The Robots of Dawn |
| David Brin | Startide Rising |
| Charles Dickens | Great Expectations |

The first iteration produced 266 occurrences and 19 patterns. Matching these patterns over all the HTML files produced 1267 books.
The key issue is how to choose the seed sets for the following iterations. Three methods of choosing seed sets were used in our experiments.
**Blind**: This method takes all of the results from the previous iteration as the seed set for the next iteration. Because the number of seeds was large, only the second iteration was performed. Starting from the 1267 books, 16525 occurrences were found, 880 patterns were generated and 45034 books were extracted.
**GoodSeed**: This method chooses books with more than one occurrence from the results of previous iteration as the seed set. For the same reason as above, only the second iteration was performed. There were 454 books that have more than one occurrence out of the 1267 books. Starting from these 454 books, we found 9200 occurrences, 588 patterns and 40715 books.
**TopSeed**: This method chooses some number of books that have the most number of occurrences from the results of pre-

Table 1: Searching Results by Choosing Seeds with Most Number of Occurrences

| iteration | books | run time(h) | # results |
|---|---|---|---|
| 1 | 3 | 0.55 | 1267 |
| 2 | 53 | 3.62 | 29471 |
| 3 | 103 | 6.5 | 33167 |
| 4 | 153 | 9.08 | 36985 |
| 5 | 203 | 13.75 | 39412 |

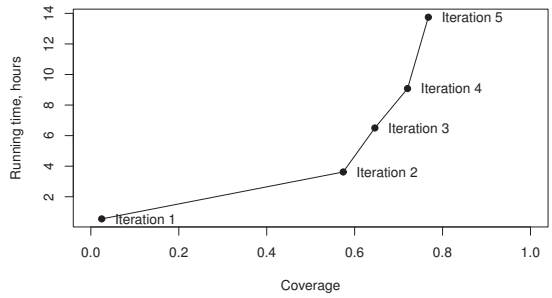


Figure 9: The Coverage Versus Running Time

vious iteration. To show the relationship between the running time and the coverage of the results, the numbers of seeds are set to be 50 more than the number of seeds of previous iteration. That is, the number of seeds for the second iteration is 53, for the third iteration is 103, ..., four more iterations were performed. The number of books obtained is 39412. The results are shown in Table 1.

By doing the above experiments, we can compare three quantities (the coverage, error rate and running time) of these three methods. The results are shown in Table 4. The `Blind` has a little higher coverage but its error rate is the largest and the running time is much longer than the other two methods. The `GoodSeed` has the shortest running time. What is very impressive is that, the estimated error rate for the `TopSeed` is very low (zero). Because in `TopSeed` method, the seeds for each iteration are selected with high positive confidence, the results are highly reliable. The running time of `TopSeed` is less than half of that of `Blind`. In general, the last two methods are better than the `Blind`. `TopSeed` took more than than `GoodSeed` because we ran more iterations, but it offers the least error (zero in this case). The coverage of `GoodSeed` and `TopSeed` are about the same.

We plot the coverage versus running time for the `TopSeed` in Figure 9. It shows that once the coverage becomes large, it takes much more time to improve it.

## 4.2 Coverage Estimation

The analyses in this section are based on the 10 trapping occasions. In this experiment, 10 seed sets are randomly selected from the result of `Blind` which contains 45034 books. All seed sets have the same number of books $n = 200$. For each seed set, we run one iteration of the extracting program to get a set of books. The number of occurrences, patterns and books in the results are shown in Table 2. As we described in Section 3.1, these experiments are 10 trapping occasions of capture-recapture study. The coverage estimation will be based on these experiment results.

Table 2: Results of 10 Trapping Occasions

| NO. | # occurs | # patterns | # results |
|---|---|---|---|
| 1 | 388 | 39 | 31482 |
| 2 | 421 | 38 | 31053 |
| 3 | 368 | 39 | 31609 |
| 4 | 424 | 34 | 31199 |
| 5 | 449 | 44 | 32143 |
| 6 | 716 | 79 | 32736 |
| 7 | 1016 | 100 | 34694 |
| 8 | 458 | 48 | 32388 |
| 9 | 408 | 52 | 31704 |
| 10 | 499 | 53 | 31992 |

Table 3: Post-Stratification Result for Two Sets of Books

| | only in set1 | only in set2 | both | total | estimation |
|---|---|---|---|---|---|
| 1st stratum | 39 | 1128 | 27427 | 28594 | 28596 |
| 2nd stratum | 2398 | 1676 | 945 | 5019 | 9272 |
| 3rd stratum | 744 | 911 | 56 | 1711 | 13814 |
| total | 3181 | 3715 | 28428 | 35324 | 51682 |

### 4.2.1 *Post-stratification*

The post-stratification methods are applied on each pair of 10 trapping occasions, $\begin{pmatrix} 10 \\ 2 \end{pmatrix} = 45$ pairs in total. The estimated population size is the average of these 45 estimations. We find empirically that books with scores more than 600 are very likely to be captured in each trapping occasion, therefore those books should belong to one stratum. Books are stratified into three strata according to their scores, greater than 600, between 300 and 600, and less than 300. For example, the post-stratification result for set 1 and set 2 is shown in Table 3. This gives the following estimation from set 1 and set 2,

$$\hat{N}_1 = 28596, \hat{N}_2 = 9272, \hat{N}_3 = 13814, \hat{N} = 51682.$$

As the final result, we get the estimated population size $\hat{N} = 52448$.

### 4.2.2 *Generalized Jackknife*

For these $t = 10$ trapping occasions, the capture frequencies $f_1, \cdots, f_{10}$ are 3320, 2155, 2203, 1241, 921, 1152, 1421, 1393, 2081 and 25046.

The generalized jackknife estimates from 1st to 9th order can be calculated by equation (2) and (3). They are 43921, 45045, 45904, 46891, 48027, 49184, 50202, 50950 and 51342. The test statistics to test whether there is significant difference between two consecutive order jackknife estimates are calculated. The results show that there are significant differences between these estimates. We take the 9th jackknife estimates as the estimation of the population size, $\hat{N} = 51342$. The estimated standard error $\hat{se}(\hat{N}) = 634$.

The post-stratification method is simple to implement and understand. However, it assumes that records in each stratum have the same capture probabilities. When this assumption is highly violated, this method usually underestimates the population size. In the current system, the generalized jackknife estimator is used to estimate the coverage. For example, the estimated coverage of the result of `Blind` is 45034/51342=87.7%.

Table 4: Performance of the Three Methods. (C.I. is the 95% confidence interval of the error rate).

| sets | # results | coverage | C.I. | time(h) |
|---|---|---|---|---|
| `Blind` | 45034 | 87.7% | [0, 0.02] | 59.04 |
| `GoodSeed` | 40715 | 79.3% | [0, 0.01] | 17.25 |
| `TopSeed` | 39412 | 76.8% | 0 | 33.5 |
| Brin's set | 15257 | unknown | [0,0.03] | unknown |

## 4.3 Quality Assessment and Enhancement

First we estimate the error rates of the results of the three methods: `Blind`, `GoodSeed`, `TopSeed` and the book set obtained by Sergey Brin, which is posted on the web. Because all of these sets have very good quality, the error rates are much less than 10%. By choosing $r = 10\%$ and $\beta = 2\%$ in Equation (5), we get the number of records needed to estimate the error rates $n = 225$. All these samples are manually checked. The results are shown in Table 4. All of the error rates are very low. In practical situation the results may satisfy the user's requirement. However, to apply and evaluate our quality enhancement methods, we look through the result of `Blind` and find several erroneous books. From each of these errors, more erroneous books are detected using the method described in Section 3.3.

For example, there is an erroneous book with author "Away Melody" and title "A Far". Following the quality enhancement procedure, the problem pattern for this error is the pattern with index 151. The problem set $B$ has 421 books in total, which is about 1% percent of the result of `Blind`. So we decide to keep this set and go to further steps. Manually checking a sample of 100 randomly selected books, 28 of them are errors. The reported confidence interval of the error rate is [0.2, 0.36]. The procedure Clean($B$) is applied and 190 out of 421 books (45%) are marked as erroneous books. As we mentioned before, some of these 190 books are correct books and discarded wrongly as an error. The other 231 books are all correct books. So after the quality enhancement procedure, the error rate of the problem set becomes 0.

# 5. FUTURE WORK

The current experiments are based on a local web page repository. In future work, our system will be based on a large scale and up-to-date web page repository. Consequently, we need to improve the performance of our program. Our long-term goal is to extract a large set of records automatically, based on several example records given by the user. Users will be allowed to specify the structure of the output XML documents. We also plan to allow extraction of data with more structure as opposed to just flat records.

It is interesting to ask whether analysis similar to what reported here can be developed for calibrating other promising algorithms for data extraction. E.g., the MDR algorithm [14] is an interesting candidate, especially given that it has been empirically shown to outperform its predecessors. A fundamental difference of MDR compared with DIPRE is that MDR extracts data records from individual web pages as opposed to identifying common patterns across web pages for data extraction, as DIPRE does. The underlying technique is thus fundamentally different from that of DIPRE, which may warrant different analytical tools.

Finally, it is interesting to ask whether we can scale up such techniques to a level where information extracted is in response to questions posed by the user. Our ongoing research addresses some of these questions.

# 6. REFERENCES


[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD* Conference, pp. 337–348, 2003.

[2] R. Baumgartner et al. Declarative information extraction, Web crawling, and recursive wrapping with lixto. *Lecture Notes in Computer Science*, 2173:21, 2001.

[3] R. Baumgartner et al. Supervised wrapper generation with lixto. In *The VLDB Journal*, pp. 715–716, 2001.

[4] R. Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

[5] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop*, pp.172–183. Springer-Verlag, 1999.

[6] K. P. Burnham and W. S. Overton. Estimation of the size of a closed population when capture probabilities vary among animals. *Biometrika*, 65(3):625–633, December 1978.

[7] K. P. Burnham and W. S. Overton. Robust estimation of population size when capture probabilities vary among animals. *Ecology*, 60(5):927–936, October 1979.

[8] V. Crescenzi et al. RoadRunner: Towards automatic data extraction from large Web site. VLDB Conference 2001, pp. 109–118.

[9] D. Freitag. Machine Learning for Information Extraction in Informal Domains. *Mach. Learning*, 39 (2-3), 2000, pp. 169–202.

[10] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. In *ACM PODS* Conference, pp. 17–28, 2002.

[11] J. Hammer et al. Extracting semistructured information from the web. In *Proceedings of the Workshop on Management of Semistructured Data*, pp. 18–25, May 1997.

[12] A. H. F. Laender et al. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[13] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science 280*, 98–100, 1998.

[14] B. Liu, R. Grossman, and Y. Zhai. Mining Data Records in Web Pages. *KDD-03*, 2003.

[15] D. L. Otis et al. *Statistical inference from capture data on closed animal populations*. Number 62 in Wildlife Monographs. Wildlife Society, October 1978.

[16] R. L. Scheaffer et al. *Elementary Survey Sampling*. Duxbury Press, Boston, 1986.