Semi-automatic Extraction of Nested Data from Web Pages

Hima Somarapu
Department of Computer Science
University of Colorado
1420 Austin Bluffs Parkway
Colorado Springs CO 80918

1. INTRODUCTION

There is a tremendous amount of information available on the web. This includes information pertaining to goods, which can be bought and sold. The web provides an alternate form of shopping for a very large number of customers. Every customer doing such on-line shopping would want the best deal available. To get the best deal, he has to spend a significant amount of time, trying to compare prices of similar products available for sale on the web. To help the customer in such situations, several sites called, “Shopping Agents” do the exact same task of obtaining the prices of similar goods being sold by different on-line stores, and provide them to the customer. This makes it very easy for the customer to select the best deal suitable for him/her.

Such shopping agents however, should have relevant data extracted from semi-structured sources such as the web pages in some structured form, such as a table. Once stored in tables, data can be queried or manipulated in any required manner. Structured data allows asking queries, which cannot be asked with data in text form. Considerable research has been done in this area to extract relevant information from similar web pages that assist shopping agents to help the end customers to do comparison-shopping. Basic idea in all the approaches that have been proposed is to discover the structural information from implicit objects found in data rich pages. Such objects found on web pages can have a simple flat structure or a complex nested
structure. We have provided support to extract nested structures from web pages, to a system that extracts flat structures [6].

1. Related Work

A number of different approaches have been proposed to extract hierarchical information.

The most common of these approaches is the use of wrappers. A wrapper parses pages from specific web sources based on some kind of grammar and maps these data into a pre-specified format. To determine hierarchical structure, the system proposed by Naveen Ashish and Knoblock [1] uses two heuristics- Font Size and Indentation. WIEN [5] uses wrapper classes for extraction, but assumes that there is exactly one attribute permutation for a given web site. It cannot handle sites with missing attributes, multiple attribute values and variant attribute permutations.

Stalker [2] is an automated wrapper construction approach that can extract data using an Embedded Catalog (EC) description and a set of extraction rules. The EC description of a page is a tree-like structure in which the leaves are the item of interest for the user. User marks up relevant data in sample pages, to learn extraction rules. Extraction rules are based on “landmarks”, a group of consecutive tokens, which enable a wrapper to locate an item of interest. For each list node, a list iteration rule that decomposes the list into individual tuples is used. Stalker can also deal with unrestricted field order, missing fields but we need to provide a description of the structure of the document.

Wai-Yip Lin and Wai Lam proposed a method called HISER (Hierarchical Record Structure and Extraction Rule learning) [3]. It employs a two stage learning task, namely, hierarchical record structure learning in which it automatically generates a
representation of hierarchical structure for the records in an information source and extraction rule learning in which extraction rules are induced for each node in the hierarchical record structure. HISER can handle missing items, multi-valued items, and in unrestricted order. It even incorporates both syntactic and semantic generalization in the learning process to enrich the expressiveness of the extraction rules. HISER tries to infer the structure information automatically based upon the samples shown by the human trainer. They use several techniques to induce the structure. They use HTML tags like UL, OL, TR, TD, TABLE, etc, which give implicit indication about the hierarchical structure to infer the nesting between the attributes. It also uses repetitive patterns to imply that those attributes can be grouped together to form list within the record structure. To form records it uses a bottom up extraction approach. Various fields are extracted independent of their records. The structure information is then used in the final step to assemble them into nested records.

Berthier Ribeiro-Neto, Alberto H.F Laender, Altigran S. da Silva proposed a system [4] in which the description of what to extract is fully based on examples provided by the user. Instead of trying to derive structural (semantic) information from the text format (syntax), it induces the user to inform the structure, as he perceives it. To extract data, the system uses a bottom up strategy, that given a set of Web pages as input, it recognizes objects matching the given examples and extracts them. It supports extraction of complex objects, which present a non-flat structure. In their approach they use markers to delineate the boundaries of nested structures. They first compose the simple objects and then use the boundary marker information to compose complex objects from those objects.

Paritosh Rohilla proposed a system [6] in which the user defines the structure of the records and provides sample records to the learner module. The rules learnt by the
learner module, are then used by the extractor to extract relevant information from the web pages.

2. Overview of the Paper

First, we describe the design and development of a GUI based system called SEFRA, which extracts product information from the web pages. The basic system successfully extracts flat structures from similar web pages. The basic approach of SEFRA is as follows: A structure suitable for relevant information on the web page is defined. The learning engine is provided with samples that fit the structure, which generates several extraction rules for them. Such extraction rules are then applied to similar web pages. All structures with rules similar to the learnt extraction rules are extracted. These structures are records with user-defined fields. The records are inserted into the database for use by a shopping agent. If the extracted records are not as expected, the learnt extraction rules are fine-tuned. The learning approach makes use of the inherent structure of tags and syntactic properties of plain text, to infer rules. The entire web page is converted into a document tree. The tree is made up of tags and plain text nodes. The plain texts of the web page end up as leaf nodes of the tree. The learner tries to identify a node of interest by exploiting some properties of this tree and the plain text nodes. The rules learnt are stored in a database to be used by the extractor.

Below is a list of the problems we will be dealing with and the approach taken, in brief. We will discuss in detail, the basic system and each of the enhancements in the rest of the.
i. **Support for Nested Structure:** The hierarchy between attributes of the nested records can be of two types. The first type is where a field of the record is a list of similar attributes or another sub-record. The second type is simpler where a field is just a sub-record. And in either case there can be several levels of nesting. And we can even have a combination of lists and sub records, in any order. We provide support to extract nested record structures where a field can be a sub-record, or a list of similar attributes, or a list of sub-records.

ii. **Support for Multi-Node Fields:** Information pertaining to a single field can sometimes span across multiple nodes. We provide additional support to be able to handle such fields.

iii. **Reducing fine tuning overhead:** Experimental results for the initial system showed that the length of the text for different fields of the record structure is adjusted very frequently, when trying to adjust values learnt by the user. It is inappropriate for our system to rely so heavily upon text lengths, as this feature is very unpredictable. We reduce this overhead.

iv. **Support for multiple fields combined together:** Sometimes information pertaining to multiple fields is combined together and is associated with a single node. We will provide support for such fields.

v. **Removal of irrelevant information associated with a field:** Sometimes the text associated with a node has some irrelevant information in-addition to what is required for a particular field. We will provide support to be able to remove any additional information.

vi. **Support for situations that result in the non-formation of any records:** There can be situations when the record extraction algorithm of the initial implementation fails to
form correct records. One such situation is when multiple fields of a record structure share the same extraction rules. Another situation is when the first node’s field type is not the same as the first field of the record structure. We will provide support for such situations.

2. WORKING OF SEFRA

This chapter is a brief overview of SEFRA. It describes the major steps involved in learning and extracting flat record structures.

Figure 1. Learning and Extraction Process

Figure 1 shows the overall process of SEFRA, the process of learning rules and extracting data from a web page.

Following are the steps involved in the working of SEFRA, to extract relevant information from web pages:
1. Structure definition

The first step involves defining a structure suitable for the relevant information on the web page. For the records shown in figure 2, the desired structure of each record is as shown in figure 3. As part of the record definition, the user provides the following information for each field in the record: the field name, whether a field is mandatory or not and the data type of the field. Following are the three basic data types that can be assigned to any field along with their meaning:

i. Alpha - Text of the target field will only contain alphabets.

ii. Numeric - Text of the target field will only contain integers and decimals.

iii. Both - Text of the target field will contain anything, including special characters.

The entire definition of the record structure, which includes field names, their data type and mandatory flag is stored in the database.

2. Providing training samples

The training samples that are provided to the learner are the records contained in the web pages. The learner has a GUI interface that allows the user to load sample pages one at a time. All the text that appears on the browser is shown in the display area of the GUI. The entire web page is first converted into a document tree. The plain texts, which appear in the display area, are indented according to the depth of the plain text node in the tree. Each text node in the document tree is given a number by traversing the tree in a depth first fashion. The GUI has a form area below the display area that is used to show where various fields of the record structure appear on the web
The process begins by the GUI prompting for the first field of the first record. The trainer enters the node number associated with the text of that field. After the first field, the GUI then prompts for the next one and so on for all the fields in the first record. The process is then repeated for the next record. The trainer can skip certain fields if the
sample record does not have the field defined in the record structure. Also, the trainer can go back to a previous field if he realizes that he has made a mistake. After the last field of a record, the entire record is committed to the database. The learner algorithm will use these records to infer rules. The prompting continues until the trainer is satisfied, that all possible variations of the record structure have been shown. If other sample records are on a different page, the page can be loaded and the process can be repeated for those records.

3. Generation of extraction rules

The record samples stored in the database are used to generate extraction rules, which are also stored in the database. A schema to store the various relationships between record structures, template and extraction rules for a web site has been developed (Figure 30). Properties of the document tree such as the repetitive pattern of the records, uniqueness of a node, leaf nodes as fields, are used to form the required rules.

Following is the information that is gathered for every field after showing each record.

i. Depth - The depth of the node in the document tree is gathered. In figure 5, the node containing the text “CGI Programming with Perl” is at depth 9. The root is at a depth 0.

ii. Tag Sequence - The tag sequence is a string containing a sequence of tags separated by ‘;’. For each field that is shown, its tag sequence is constructed by getting the sequence of all tags starting from the root node to the field’s node. For example the node containing the text “CGI Programming with Perl”, has the tag sequence “html;body;table;tbody;tr;td;font;a;b;”.
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>CGI Programming with Perl</strong></td>
<td>by Guelich, Scott / Gundavaram, Shishir</td>
</tr>
<tr>
<td></td>
<td>List Price: <strong>US$ 34.95</strong>, A1 Price: <strong>US$ 24.25</strong></td>
<td>Add to cart (Ships in 24 hours)</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Designing with JavaScript</strong></td>
<td>by Heinle, Nick / Webbi, Martin</td>
</tr>
<tr>
<td></td>
<td>List Price: <strong>US$ 34.95</strong>, A1 Price: <strong>US$ 24.25</strong></td>
<td>Add to cart (Ships in 24 hours)</td>
</tr>
</tbody>
</table>

Figure 4. A sample web document from www.A1Books.com
iii. Relative position - The relative position of a field is the difference between its node number and the node number of the first field in the record structure. The first field has a relative position of zero. The relative position of each field in the record structure is calculated and stored.

iv. Keywords - Any number of word(s) or characters that stay constant across all the records of a field are considered as keywords for that field. The trainer inputs appropriate keywords in the text box during the training process for each field. Keywords are helpful in resolving ambiguities.
v. Omit-words - Any number of word(s) or character(s) that should not be part of the plain text of the field are classified as omit-words for that field. These omit-words are used during extraction, and any plain text nodes matching the omit-words will be ignored. This feature is used when wrong nodes are associated to a field by the extractor.

vi. Value - The entire text associated with the field is its value. By examining the text of a field across all records, the average size of the text in the field is calculated.

All of the above information gathered for each field is stored in the database. The rule generation algorithm uses this information to formulate rules for each field of the record structure. Every time a new record is shown, it uses information from all the previous records and the new ones to re-generate the rules as follows:

For each field, its depth D, tag sequence T, relative position P, keywords K, omit-words O and the length of value V from all the records previously stored are fetched.

If the current D and T pair is different than the previous records, new D and T pair is added to the group of distinct pairs fetched from previous records.

The min and max value for P is calculated based on all the values of P.

If K is different than those of previous records, it is added to the group of distinct keywords fetched from previous records.

If O is different than those of previous records, it is added to the group of distinct omit-words fetched from previous records.
The length of V is calculated and also the min and max value for the length.

Finally all the rules, which includes D and T pairs, min and max value of P, all K values, all O values, and min and max values for length of V, for the field are stored in the database under a unique name.

4. Applying extraction rules

The extractor module handles the extraction and also provides the fine-tuning capability through a user friendly GUI.

The trainer first loads the web document in the extractor and then specifies the rule set that has to be applied to the document. The rule set is identified by specifying the name under which the rules for the current document are stored in the database.

The extraction process follows a bottom up approach to form records. This helps in dealing with incomplete records. Incomplete records are those, which do not have all the fields of the record structure. Every node that qualifies as a field is extracted from the page, irrespective of the record it belongs to. Extracted fields are then grouped together into records.

Following are the steps involved in the record extraction process:

1. A document tree T for the web document D is created and each node is assigned a node number N.

2. The record structure S for D is fetched from the database.

3. Following steps are executed for each field FN in the record structure S:

   a. depth D and tag sequence T pairs, data type DT, keywords K, omitwords O and min and max length values, minL, and maxL are fetched from the learned values stored for document D.
b. from T, plain text nodes with plain text PT at depth D, having corresponding tag sequence T, for every D & T pair are fetched.

c. For every node fetched, following steps are executed:

i. skip node if its PT is not of data type DT.
ii. skip node if its PT does not contain any keyword in K.
iii. skip node if its PT contains any omitword in O.
iv. skip node if length of PT > maxL or length of PT < minL.
v. If PT is not skipped then attach field name FN and its node number N to the PT.
vi. Put the passed node in the extracted nodes EN bucket.

4. sort the nodes in EN bucket by their ascending node number N.

5. make the first node’s field type FT in EN bucket as the record separator RS.

6. obtain all mandatory fields MF for this record structure.

7. for each node in sorted EN bucket following steps are executed:

i. if FT of the node is RS then start a new record and commit the previous one, if it has all the MFs.

ii. fetch relative position values, minP and maxP for this node’s FT from the learned values stored for D.

iii. associate the node with the new record, if its relative position >= minP and <=maxP.

After the individual records are extracted, mandatory field check is performed on them. Only if an extracted record has all the mandatory fields it is committed to the database.
The extractor module also makes use of templates. Templates help convert records from similar web sources but different record structure, into those with the same record structure. This makes comparison-shopping very easy. The extractor provides an easy way to define templates and associate similar record structure to one template. The records that are extracted by the Extractor, are converted to some standard template, and finally stored in the database. These stored records can then be easily queried using standard SQL, and can be used to perform comparison-shopping.

5. Rule refinement

The extractor GUI provides the facility to look at some of the rules and allows manual changes to the values. This facility is used when the Learner algorithms are unable to calculate proper values for various rules. The values that the trainer can modify are min and max relative positions, min and max string lengths, keywords and omit-words.

4. Overview of Approach taken

i. Support for Nested Structures

To be able to extract nested structures, some of the changes we made, in brief, are as follows:

The record definition module is enhanced to be able to accept nested structure definition from the trainer. New additional information for each field, as the learner provides the sample values for them, is gathered and new rules are formulated. Some of the new rules learnt for every field include, the boundary fields and relative position values for sub-records, for lists, for individual list items of a list, for list of sub-records, for the fields of a sub-record, for sub-records that belong to a list of sub-records, and such
similar structures. These new rules aid in the extraction and formation of nested records using a bottom-up approach. After extracting all the nodes that satisfy the rules learnt during the learning process, we first form any sub-records, lists or list of sub-records at each level. We then combine all these structures using the structure information, starting from the lower levels and moving towards the upper levels. We will then have the final complete record.

ii. Support for Multi-Node Fields

One of the main reasons for a field to span across multiple nodes is the presence of text formatting tags in the document tree. Hence as a solution to this problem we ignore all tags that are related to text formatting (<B>, <I>, <FONT> etc.,) while forming the document tree.

iii. Reducing fine tuning overhead

As an alternative to using the text length to eliminate invalid nodes, we use the fan-out information of sub-trees. When a sample value for a field of the record is provided, we determine the fan-out of the parent node, of the node shown as the sample value. Later at the extraction end we choose only those candidate sub-trees where the parent nodes of the leaf nodes have fan-out, similar to the learnt fan-out values. In this way we eliminate some of the invalid nodes.

iv. Support for multiple fields combined together

Most often, when multiple fields are combined together as a single node, the information pertaining to each field is either separated (preceded by, followed by) by some unique pattern from one another or it begins or ends with some unique pattern. In
all such cases, we let the trainer provide such information during the learning phase. Based on the information provided by the trainer for each field, new rules are formulated and used at the extraction end, to separate multiple fields combined together.

**v. Removal of irrelevant information associated with a field**

The approach used to separate multiple fields combined together is also used to remove any irrelevant information associated with a field.

**vi. Support for situations that result in the non-formation of any records**

We made changes to the record formation part of the record extraction algorithm, to support situations that can result in the non-formation of any records. In case of fields that share the same learning rules, instead of committing a record to the database whenever a record separator is found, we retain each record and postpone the commit to a later stage. Also, we check the relative position condition with each boundary field found in every previously formed record, instead of just the previous uncommitted record. And in case the first node’s field type is not the same as that of the first field of the record structure, we simply skip nodes till we reach the first node whose field type is same as that of the first field in the record structure.

**4. SUPPORT FOR SITUATIONS RESULTING IN NON-FORMATION OF RECORDS**

This section explains the situations that can result in the non-formation of correct records and the approach taken to solve the problem. This problem can occur irrespective of the record structure, either flat or non-flat. Hence, we discuss this problem and its solution before others.

**1. Situation 1**

When some of the fields in a record structure have the exact same extraction rules and they are all mandatory, no records are formed.
In the previous chapter, section “Application of extraction rules” explains the steps involved in the formation of records. Step 7 of the Record Extraction Algorithm is as follows:

7. for each node in sorted EN bucket, following steps are executed:

i. if FT of the node is RS then start a new record and commit the previous one, if it has all the MFs.

ii. Fetch relative position values, minP and maxP for this node’s FT from learned values stored for D.

iii. associate the node with the new record, if its relative position >= minP and <= maxP.

Let us look at an example to explain why the above sequence of steps involved in the formation of records will fail to form any records in situations where some of the fields in a record structure have the exact same extraction rules.

Consider a record structure that has the following two fields:

1. Restaurant Name
2. State

Let both the above fields be mandatory fields. Consider the case when both the fields have the exact same rules.

Step 3 b of the Record Extraction Algorithm is as follows:

3b. from T, plain text nodes with plain text PT at depth D, having corresponding tag sequence T, for every D & T pair are fetched.
The above step is executed for every field FN in the record structure S.

After step 3c is executed, the extracted nodes in the bucket EN would be as follows (I assigned certain node numbers for proper explanation):

Contents of EN bucket after step 3c:

<table>
<thead>
<tr>
<th>Field Name (FN)</th>
<th>Node Number(N)</th>
<th>Plain Text(PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant Name</td>
<td>45</td>
<td>beaverton</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>53</td>
<td>Oregon</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>100</td>
<td>USA</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>106</td>
<td>restaurant</td>
</tr>
<tr>
<td>State</td>
<td>45</td>
<td>beaverton</td>
</tr>
<tr>
<td>State</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>State</td>
<td>53</td>
<td>Oregon</td>
</tr>
<tr>
<td>State</td>
<td>100</td>
<td>USA</td>
</tr>
<tr>
<td>State</td>
<td>106</td>
<td>restaurant</td>
</tr>
</tbody>
</table>

The correct PTs that should have been associated with each of the FNs are as follows:

For the FN “Restaurant Name “, it is “Sharis”

For the FN “State”, it is “Oregon”.

After the execution of step 4, the contents of bucket EN would be as follows:

<table>
<thead>
<tr>
<th>Field Name (FN)</th>
<th>Node Number(N)</th>
<th>Plain Text(PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant Name</td>
<td>45</td>
<td>beaverton</td>
</tr>
<tr>
<td>State</td>
<td>45</td>
<td>beaverton</td>
</tr>
</tbody>
</table>
According to step 5, “Restaurant Name” is chosen as the record separator RS. Let minP and maxP values for FN “State” be 1 and 2 respectively and for FN “Restaurant Name” they are 0 and 0 respectively. Applying steps i, ii, iii of step 7,

- the first entry in bucket EN (Restaurant Name, 45, Beaverton) would become the first field of a new record.
- considering the next entry (State, 45, Beaverton), the relative position is 0. Condition in step (7 iii) fails.

The above same two sequences of steps are performed for the remaining entries in bucket EN and finally we do not have any records to commit into the database.

The correct record that should have been formed and committed into the database is:

<table>
<thead>
<tr>
<th>Field Name (FN)</th>
<th>Node Number(N)</th>
<th>Plain Text(PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant Name</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>State</td>
<td>53</td>
<td>Oregon</td>
</tr>
</tbody>
</table>
2. Solution for Situation 1

To overcome the above problem, step 7 of the Record Extraction Algorithm is modified as follows:

7. for each node N in sorted EN bucket following steps are executed:

   i. if FT of the node N is RS then start a new record and add the record to a record list RL. Records in RL are not yet committed.

   ii. Fetch minP and maxP for the node N's FT from the learned values stored for D.

   iii. for each record R in RL, do begin

       a. calculate relative position RP for the node N. RP is the difference in the node number of the node N and the node number of the first field in R.

       b. associate the node with record R, if its relative position >= minP and <= maxP.

       done.

   iv. for each record R in RL, do begin

       a. perform mandatory field check on all the fields of R.

       b. Commit record R to database if it passes the mandatory field check

Applying the above changed steps to the sorted nodes in bucket EN:

- the first entry in bucket EN (Restaurant Name, 45, Beaverton) would become the first field of a new record R1. R1 is added to record list RL.

- considering the next entry (State, 45, Beaverton), the relative position is 0. Condition in step (7 iii) fails.
After executing the steps i, ii, iii, on entries 3, 4,5 of EN, contents of RL would be as follows:

R1: (Restaurant Name, 45, Beaverton)
R2: (Restaurant Name, 51, Sharis)
R3: (Restaurant Name, 53, Oregon)

After executing the steps i, ii, iii, on entry 6 of EN, contents of RL would be as follows:

R1: (Restaurant Name, 45, Beaverton)
R2: (Restaurant Name, 51, Sharis)
(Restaurant Name, 53, Oregon)
R3: (Restaurant Name, 53, Oregon)

After executing the steps i, ii, iii, on the remaining entries of EN, we will have two more additional records:

R4: (Restaurant Name, 100, USA)
R5: (Restaurant Name, 106, Restaurant)

Executing step (7 iv) on RL, we will have the correct record as shown below:

<table>
<thead>
<tr>
<th>Field Name (FN)</th>
<th>Node Number(N)</th>
<th>Plain Text(PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant Name</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>State</td>
<td>53</td>
<td>Oregon</td>
</tr>
</tbody>
</table>
Notice that the problem associated with situation 1 will occur whenever any field of the record shares the extraction rules with the boundary field, which is the first field. In such a situation, if all the fields of the record structure are mandatory, then no records are formed. However, if not all fields are mandatory, then incomplete records may be formed.

3. Situation 2

When the first node’s field type is not the same as that of the first field of the record structure.

Step 5 of Record Extraction algorithm says:

5. make the first node’s field type FT in EN bucket as the record separator RS.

If the contents of bucket EN after step 3c are as follows:

<table>
<thead>
<tr>
<th>Field Name (FN)</th>
<th>Node Number(N)</th>
<th>Plain Text(PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>33</td>
<td>Colorado</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>45</td>
<td>beaverton</td>
</tr>
<tr>
<td>State</td>
<td>45</td>
<td>beaverton</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>State</td>
<td>51</td>
<td>Sharis</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>53</td>
<td>Oregon</td>
</tr>
<tr>
<td>State</td>
<td>53</td>
<td>Oregon</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>100</td>
<td>USA</td>
</tr>
<tr>
<td>State</td>
<td>100</td>
<td>USA</td>
</tr>
<tr>
<td>Restaurant Name</td>
<td>106</td>
<td>restaurant</td>
</tr>
<tr>
<td>State</td>
<td>106</td>
<td>restaurant</td>
</tr>
</tbody>
</table>
Then, according to step 5, “State” would be chosen as RS. This would create a problem and again result in the non-formation of correct records.

During extraction rules generation, the relative positions for each field in the record structure is calculated with respect to the first field defined in the record structure. The first field defined in S is “Restaurant Name”. While executing step 7 of Record Extraction Algorithm, the relative positions are calculated with respect to field “State”. This value is compared with minP and MaxP fetched from database. Hence, condition in step (7 iii) will fail repeatedly.

4. Solution for Situation 2

A simple fix to this problem is a change to step 5 of Record Extraction Algorithm. we simply skip nodes till we reach the first node whose field type is same as that of the first field in the record structure.

5. for each node N in bucket EN do begin
   i. if field type FT of N is same as field type FT of the first field in the record structure S, then make the field type FT of node N as the record separator RS
      else continue

   done

It is observed that most of the times invalid nodes come up before valid nodes, and hence ignoring such invalid nodes does not effect the formation of proper records. However, in case the very first record is an incomplete record with the first field of the
record missing, then in such a case we will lose that record. This is a general problem that will occur whenever the very first field is missing, as the relative position is calculated with respect to the first field.

5. NESTED STRUCTURES

This chapter explains in detail about the different kinds of nested structures—lists, sub-records, and list of sub-records. It also explains the changes made to SEFRA for each such type of nested structure. It explains the new extraction rules that need to be learnt and any changes that have to be made to each step of the learning and extraction process, to be able to extract nested structures.

1. Introduction

This section is a brief introduction to nested structures, the different kinds with examples.

The hierarchy between the attributes of the nested records can be of two types.

i. a field is another sub-record.

ii. a field of the record is a list of similar attributes referred to as a simple list or another sub-record referred to as a list of sub-records.

In either case there can be several levels of nesting.

Figure 6 shows a nested structure where a field is another sub-record:
Figure 6. Nested Structure where a field is another sub-record

In the above figure, field “Article” is a record structure with fields, i. Author, ii. Title, and iii. Pages. Since field “Article” is a field of the record structure Edition, we say “Article” is a sub-record. Here we have two levels of hierarchy. In a manner similar to above, we can have several levels of nesting.

Figure 7 shows a nested structure where a field is a list of similar attributes:

Figure 7. Nested Structure where a field is a list
When a field has more than one value associated with it, we call it a list. Each such value in the list is called a list item. Above figure shows the record structure of “Product”. Each product has a model number, description, price and a list of features. Hence, field “Feature” is defined as a list. In this case “Feature” is a list of similar attributes. Figure 8 shows a nested structure where a field is a list of sub-records

![Nested Structure where a field is a list of sub-records](image)

**Figure 8. Nested Structure where a field is a list of sub-records**

Instead of a simple case of list of similar attributes, we can have a list where each list item is another sub-record. Above figure shows the record structure of “Restaurant”. Field ”Address” is a list of sub-records. Each list item of the list of addresses is a sub-record with fields “Street”, “City”, “State”, “Zip” and “Phone”.

We can also have any combination of the above three cases, in which case the record structure becomes more complex. For example, a field in a sub-record of a list of such sub-records, can itself be another list of sub-records. We can find such situations in case of airline itineraries. Right now, we will discuss in detail about the three cases described above. In order to be able to extract records with the above three different types of hierarchies we will discuss the changes and the improvements that need to be done to the learner and extraction modules.
2. Sub-Records

This section explains in detail about sub-records and the changes and improvements made to the learning and extraction process of SEFRA, to be able to extract sub-records.

<table>
<thead>
<tr>
<th>Flight Details</th>
<th>Price</th>
<th>Choice and continue</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:20 AM Depart Portland (PDX) Thu 18-Apr Arrive San Jose (SJC) 8:05 AM 1hr 45mn</td>
<td>$154</td>
<td>Alaska Airlines 362 Nonstop flight</td>
</tr>
<tr>
<td>7:37 AM Depart Portland (PDX) Thu 18-Apr Arrive San Jose (SJC) 9:22 AM 1hr 45mn</td>
<td>$154</td>
<td>Alaska Airlines 596 Nonstop flight</td>
</tr>
<tr>
<td>9:40 AM Depart Portland (PDX) Thu 18-Apr Arrive San Jose (SJC) 11:21 AM 1hr 41mn</td>
<td>$154</td>
<td>Alaska Airlines 312 Nonstop flight</td>
</tr>
<tr>
<td>12:17 PM Depart Portland (PDX) Thu 18-Apr Arrive San Jose (SJC) 2:01 PM 1hr 44mn</td>
<td>$154</td>
<td>Alaska Airlines 332 Nonstop flight</td>
</tr>
</tbody>
</table>

Figure 9. Portion of a web page from www.Expedia.com
2.1. Example

Figure 9 shows a portion of a web page from www.Expedia.com. It shows different itineraries. They are the result of a search for some user provided conditions (originating city, destination city, specific travel dates and times). The record structure for each itinerary with only the required fields could be as shown in figure 10.

Clearly, the record structure is nested where some of the fields are sub-records. To be able to extract records with such a structure we need to make some changes and improvements to the existing system. Let's look at the changes and improvements that need to be made to each step of the learning and extraction process (explained in chapter 2).

2.2. Changes

2.2.1. Structure Definition

In this step, the trainer provides the record structure by specifying the field names, their field types and also indicates whether a field is mandatory or not.

In order to accept a nested structure definition from the trainer, the trainer should be able to indicate if a particular field of a record structure is a sub-record. Hence, in
addition to the information already provided, the trainer will also provide information indicating whether a field is a sub-record or not. If a field of a record structure is a sub-record, then the trainer will provide the sub-record structure definition in a manner similar to above. This process will continue till none of the fields of a record structure is a sub-record.

**Parent-Child Relationship:** If a field of a record structure is a sub-record, then that field is the parent field of all the fields in that sub-record. This also means that all the fields in that sub-record are the children of the original field. All the children of a parent share the same parent id. The parent id is the field id of the parent field. Fields that are not sub-records can be considered as parents with no children. Such children have a default parent id = 0.

### 2.2.2. Providing Training Samples

In this step the trainer provides sample records in a manner similar to as specified previously in chapter 2.

During the process of providing sample records, the GUI prompts for the first field of the first record. The trainer enters the node number associated with the text of that field. After the first field, the GUI then prompts for the next one and so on for all the fields in the first record. The process is then repeated for the next record.

However, in case any field is a sub-record, GUI will begin the process of accepting sample values for the fields in that sub-record. After finishing with all the fields in that sub-record, it will continue accepting sample values for the fields in the original record. We can have any levels of nested structure. The GUI will prompt for sample values for each sub-record at each level, starting from the upper levels in a manner described
above. This order of accepting sample values helps in determining boundary fields and calculating relative positions. We will discuss the significance of boundary fields and the relative positions values for sub-records in the next sub-section.

2.2.3. Generation of extraction rules

SEFRA follows a bottom-up approach to form records. Every node that qualifies as a field is extracted irrespective of the record it belongs to. Extracted fields are then grouped together into records.

In case of nested structures, this concept of bottom-up approach to form records is extended. We first form individual records at each level. Later we combine all of them together according to the record structure.

In figure 11, each of “DepartInfo”, “ArrivalInfo”, and “AirlinesInfo” is a sub-record. And “Price”, “Date”, “Duration” are fields. The sub-records are first formed using the relative position information, as explained in chapter 2. Later, all the fields and the sub-records should be combined together to form a complete record.

To form a complete record according to the record structure specified by the trainer (figure 10), we need to learn the relative position for each of the sub-records too. As specified before, relative position of a field is the difference in its node number and the node number of the first field in the record structure. Similar to this definition, we need to learn relative position for sub-records too.

Before we explain about the relative position for sub-records, lets first be clear with the concept of “lower level” and “higher level” in case of a nested record structure. “Highest-Level” would be level 0, the least level number. “Lowest-Level” would be the level with the highest-level number. In the figure 12, which shows the record structure of
"Itinerary" and the level numbers for each level of nested structure, the lowest level is level 2.

Figure 11. Fields and Sub-Records

Figure 12. Record structure of Itinerary and the level numbers

For sub-records, relative position is the difference in the node number of the first field in that sub-record and another field called the “Boundary Field”. Boundary field helps in combining together complex objects, using the bottom up approach. In several situations, the very first field in the sub-record will represent the whole sub-record.
Hence, boundary field of a sub-record refers to the boundary field of the first field in that sub-record. Boundary field of a sub-record is a field, which occurs immediately before it at level higher than itself. In case there is no such field, it takes the very first field of the record as its record boundary field. For all the fields other than the first field in each sub-record, relative position is the same as defined earlier. Any field that is a part of a sub-record, but is not the first field in that sub-record cannot be a boundary field of any other structure or field that might exist at the same or lower level. The order in which sample values are prompted by the GUI, helps in determining the boundary field for each field and calculating the relative positions.

Hence, In addition to depth, tag sequence, relative position, keywords, omit-words and value gathered for every field, boundary field for every field is also determined for each field after a record is shown. As done before, all of this information gathered for each field is stored in the database.

To summarize, boundary field for a field in a record structure is determined as below:

- For a field, which is the first field of a sub-record, boundary field is a field, which occurs immediately before it at level higher than itself. In case there is no such field, the very first field of the record structure is its record boundary field.

  For example, for the field ‘Depart_Time’, which is the first field of the sub-record ‘DepartInfo’, the boundary field is ‘Price’.

- For a field, which belongs to a sub-record and is not the first field in that sub-record, boundary field is the first field of that sub-record.
For example, for fields ‘Depart_City’ and ‘Depart_Airport_Code’, which belong to a sub-record and are not the first fields of the sub-record, their boundary field is ‘Depart_Time’.

- For a field, which does not belong to a sub-record and is the first field of the record structure, boundary field is itself.

  For example, for the field ‘Price’, which is the first field of the record structure and does not belong to a sub-record, its boundary field is itself, which is ‘Price’.

- For a field, which does not belong to a sub-record and is not the first field of the record structure, boundary field is the first field of the record structure.

  For example, for fields ‘Date’ and ‘Duration’, which are not the first fields of the record structure and do not belong to a sub-record, their boundary field is ‘Price’.

**2.2.4. Applying extraction rules**

We have already seen the record extraction process for flat record structures. It follows a bottom up approach to form records. In this section we will discuss the additional steps of the record extraction algorithm. These additional steps are part of the bottom up approach to first form sub-records at each level and later combine them together to form complete records. Sub-record formation begins at the lowest level and moves up towards the highest level. Each of the sub-records are formed in a manner similar to the process of forming regular records (updated record formation algorithm, explained in chapter 3).
Steps 1, 2 and 3 of record extraction algorithm for flat-records are still the same.

Changes from step 4, are as follows:

4. Fetch from the database all parents, ordered in reverse, based on their levels.

5. **for** each parent **P** **do begin**
   
i. Fetch the sub-record structure SBS of **P**.

   ii. **for** each field(child) **CF** in SBS **do begin**

       a. retrieve from **EN**, all the nodes extracted for **CF** and store in a hash **NH**. Key is the node number and value is a collection of nodes that share the same node number.

       **done**

   iii. Sort the contents of **NH** in ascending order of their key values(node numbers).

   iv. **for** each key of **NH** **do begin**

       a. get the value for that key, which is a collection of nodes **CN**.

       b. sort the contents of **CN** based on the ascending order of their field-Id. (Each field has a field id).

       c. **for** each node **ND** in **CN** **do begin**

           i. If **ND** is the very first field of the sub-record, mark it’s field type **FT** as the record separator **RS**.

           ii. if **FT** of the node is **RS** then start a new sub-record and add the sub-record to the sub-record list SRL.
iii. Fetch minP and maxP for this node’s FT from learned values stored for D.

iv. Fetch the boundary field’s id BD for this node’s FT from the learned values for D, if the FT of the node is not the RS. If the FT of the node is RS, temporarily make its BD as itself.

**iv. for each sub-record SR in SRL, do begin**

a. Get the BD’s node number and calculate relative position RP for the node. RP is the difference in the node number of the node and the node number of its BD.

b. associate the node with sub-record SR, if its relative position >= minP and <= maxP.

done

done

done

done

At the end of step 5, we will have several collections of sub-records. Each collection will have the same type of sub-records.

After step 5, we have to form complete records. After discussing the learning and extraction process for lists and list of sub-records, we will discuss the final steps involved in the complete record formation. This is because a nested record can have sub-records, lists or list of sub-records in any number and order and at different levels of hierarchy.
Figure 13 shows the bottom-up extraction approach graphically. EN has a collection of nodes extracted for the various fields in the record. These nodes are then grouped separately. Each group consists of the nodes belonging to a different sub-record. All the fields that do not belong to any sub-record are grouped separately. In the next step, each group is processed separately. The nodes within each group are ordered by applying the extraction rules to form proper sub-records.
Figure 13. Bottom UP Extraction approach
3. SIMPLE LISTS

This section explains in detail about lists and the changes made to the learning and extraction process of SEFRA, to be able to extract lists.

3.1. Example

Figure 14 shows a portion of a web page that displays alphabetically all the authors whose names begin with ‘A’, and some of their texts.

Figure 14. Athena, Authors and Texts
The web page displays for each author, a list of books written by the author. Hence the record structure for this web page called “Athena” is as shown in figure 15.

The record structure shown in figure 15 introduces a different type of hierarchy mentioned previously- a field of the record is a list of similar attributes. In this case the field is “Books”. Field “Book” is called a list and each of the values in the list are called “List Items”.

To be able to extract records with this type of nested hierarchy, we have to make changes to the existing system. As done before for sub-records, lets look at the changes that need to be made to each step of the learning and extraction process.

![Figure 15. Record Structure for Athena](image)

### 3.2. Changes

#### 3.2.1. Structure Definition

In this step, the trainer provides the record structure by specifying the field names, their field types and also indicates whether a field is mandatory or not. Also, as seen in the previous section on sub-records, the trainer also indicates whether a field is a sub-record or not.
Similar to the case of a field being a sub-record, when the trainer provides the record structure definition, he should be able to indicate whether a field is a list. The trainer provides the required information.

3.2.2. Providing Training Samples

In this step the trainer provides sample records in a manner as specified previously.

A list has one or more list items in it. The trainer should be able to supply sample values for one or more list items in the list.

The GUI has a form area below the display area that is used to show where various fields of the record structure appear on the page. Additional features have been added to this form area to allow the trainer to provide sample values for one or more list items of the same list/field.

During the process of providing sample records, whenever a field is a list, the GUI prompts for the first list item. The trainer will enter the node number associated with the text of that list item. After the first list item, the trainer can enter sample values for several more list items, in a similar manner. Each of the list items is referred to by the same field name. Only when the trainer indicates that he is done providing sample values for the list items in that list, the GUI will continue and prompt for the next field. The order in which the sample values for the list items are prompted, helps in determining the boundary fields and relative positions for them.

If a list has more than one list item, the trainer should provide at least the first two list items as sample values. We will discuss the reason for this in the next sub-section.
3.2.3. Generation of extraction rules

From the discussion so far about lists, it is clear that all the list items in a list share the same name. Also, the number of list items in a list is variable. This is in contrast to a sub-record. Each of the fields in a sub-record have a unique name and hence a unique field id. Also, the number of fields in a sub-record is fixed. We need to learn new extraction rules to be able to extract all the list items of list.

*List-Id:* To distinguish between each of the list items in a list that share the same name, each of the list item is associated with a unique list-id. The first list item has a list-id of 1, the second has a list-id of 2 and so on. A field, which is not a list, has a default list-id of 0.

*Relative Position:* We have previously defined relative position both in the context of a field and a sub-record. Let us now define relative position for lists.

For the first list item, relative position is the same as defined in the section on sub-records. It is the difference in the node number of the first list item and the boundary field of the first list item. Again, boundary field of the first list item in a list is determined as explained in the section on sub-records. Depending on whether the first list item is the first field of a sub-record or not or whether it is the first field of a regular record or not, its boundary field is determined as explained in the section on sub-records. For example, considering figure 15, for the field “Book” with list-id = 1, its boundary field is “Author”. For all the other list items, boundary field is itself, but one with a list-id less than itself by one, i.e., for the second list item, boundary field is the first list item, for the third list item, boundary field is the second list item and so on. For example, for the field “Book” with
list-id = 2, its boundary field is “Book” with list-id = 1. And for the field “Book” with list-id = 3, its boundary field is “Book” with list-id = 2, and so on.

Some of the lists have many list items in it, while some have very few. It may not be practical for the trainer to provide sample values for all the list items in a list. Keeping in mind this difficulty and observing the fact that the relative position between any two consecutive list items is the same, it suffices for the trainer to provide at-least the first and the second list items of a list as sample values.

If a list has more than one list items, the trainer should provide the first and the second list items as sample values. In addition he can provide any other list item as a sample value in order to make the extraction rules more robust.

Relative position and boundary field for all list items with list-id > 2 are assigned the same value of boundary field and relative position calculated for the list item with list-id = 2.

If a field is a list, it will have multiple entries in a single record, but with different list-ids .

**Rule Generation Algorithm:**

We have already seen the rule generation algorithm. Every time a new record is shown it uses information from all the previous records and the new ones to re-generate the rules for each field. In case a field is a list, separate rules for list items with list-id = 1 and for all the list items with list-id >= 2 are formulated.

If a field F is a list, first, for all the entries of F with list-id = 1, its depth D, tag sequence T, relative position P, keywords K, omit-words O and value V from all the records previously stored are fetched.
The rules are then formulated for $F$ as specified in the rule generation algorithm. However, the comparison is done with the current value of $F$ with list-id = 1. The rules are stored in the database with a list-id = 1.

Next, for all the entries of $F$ with list-id = 2, their depth $D$, tag sequence $T$, relative position $P$, keywords $K$, omit-words $O$ and value $V$ from all the records previously stored are fetched.

The rules are again formulated for $F$ as specified in the rule generation algorithm. In this case, the comparison is done for each of the current values of $F$ with list-id >=2. The rules are stored in the database with a list-id = 2.

3.2.4. Applying extraction rules

Lists are formed in a manner similar to regular records. From among the sorted list of items, each time a list item with list-id = 1 is encountered, a new list is created. For any other list item, relative position condition is checked with every last list item in every list. Only if the condition is satisfied, that list item is added to that list.

Following changes to steps 4 and above of the record extraction algorithm (update version explained in the section on sub-records) have to be made to extract lists.

4. Fetch from the database all parents, ordered in reverse based on their levels.

5. for each parent $P$ do begin
   i. Fetch the sub-record structure SBS of $P$.
   ii. for each field(child) CF in SBS do begin
a. retrieve from EN, all the nodes extracted for CF.

b. If a field is a list, then store the nodes in a hash LNH.

c. If a field is not a list, then store the nodes in another hash SNH.

Key for both the hashes is the node number and value is a collection of nodes that share the same node number.

done

iii. Sort the contents of LNH in ascending order of their key values (node numbers).

iv. For the contents of LNH, create a new hash with key being the field-id and value being a collection of nodes that share the same field-id.

v. for each key of LNH do begin

   a. retrieve the value for that key, which is a collection of nodes CN.

   b. for each node ND in CN do begin

      i. if the list-id of ND = 1,

      then create a new list by adding the node ND to it and add that list to a list of lists LL.

      else if list-id of ND != 1,

      then,

      a. Fetch minP and maxP for this node’s FT from learned values stored for D.
b. **for** each list L in LL **do begin**

i. get the last node’s node number LNDNBR from L. Calculate relative position RP for the node. RP is the difference in the node number of ND and LNDNBR. We perform the relative position calculation with the last node in the list, because for each list item except the first, the boundary field is the list item before it, the one that has a list-id lesser than itself by one.

ii. associate the node with list L, if its relative position >= minP and <=maxP.

**done**

**done**

**done**

vi. create a new hash AL with field-id as the key and value being the list of lists LL.

**done**

At the end of step 5, we will have several collections of lists. After step 5, we have to form complete records. We will discuss that later.
Figure 16 shows the bottom up approach to form lists. All the nodes that satisfy the extraction rules for such list items are extracted irrespective of the list they belong to. Using the boundary field and relative position information, the individual list items are combined together to form separate lists.

Nodes extracted for field “Book”

<table>
<thead>
<tr>
<th>List 1</th>
<th>List 2</th>
<th>List 3</th>
<th>List 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>li1</td>
<td>li1</td>
<td>li1</td>
<td>li1</td>
</tr>
<tr>
<td>li2</td>
<td>li2</td>
<td>li2</td>
<td>li2</td>
</tr>
<tr>
<td>li3</td>
<td>li3</td>
<td>li3</td>
<td>li3</td>
</tr>
<tr>
<td>li4</td>
<td>li4</td>
<td>li4</td>
<td>li4</td>
</tr>
<tr>
<td>li5</td>
<td>li5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>li6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figures 16. Bottom up approach for lists
4. List of Sub-Records

This section explains in detail about list of sub-records and the changes made to the learning and extraction process of SEFRA, to be able to extract list of sub-records.

4.1. Example

Figure 17 shows a portion of a web page from www.Zagat.com. It shows for each top cuisine, a list of restaurants and their ratings, for a particular city.

![Figure 17. Web Page from www.Zagat.com](image_url)

The record structure for the information displayed on the above web page is as shown in figure 18.
Each restaurant has a corresponding rating value associated with it. Hence, “Restaurant Name” and “Rating” together form a sub-record with those two fields. Also, each top cuisine is associated with a list of “Name&Rating” sub-records. Here we see a new type of hierarchy. A field of a record is a list of sub-records. We can also observe that fields “Restaurant” and “Rating” form lists in themselves.

![Diagram of record structure]

**Figure 18. Record Structure of Zagat**

We will look at the changes that have to be made to each of the steps of the learning and extraction process, in order to be able to extract a list of sub-records.

### 4.2. Changes

#### 4.2.1. Structure Definition

No new changes are needed. The trainer indicates that a field is a list of sub-records by indicating that a field is both a list and a sub-record. Also, each of the fields in such a sub-record should be marked as lists.
4.2.2. Providing Training Sample

During the process of providing sample records, when a field is both a list and a sub-record, the GUI will prompt first for sample values in the first list item. The list item is actually a sub-record. Hence the GUI will first prompt for the first field in that sub-record, followed by the second field and so on for all the fields in that sub-record. Each time the GUI prompts for a field, the trainer enters the node number associated with the text of the field. After providing sample values for the first list item, the trainer can provide sample values for many more list items, which are sub-records, in a similar manner. Since each of the fields in such sub-records is a list in itself, the trainer should provide at least two such sub-records(list items) as sample values. We have already seen the reason behind this in the section on lists. Only when the trainer indicates that he is done providing sample values for the sub-records(list items) in that list of sub-records, the GUI will continue and prompt for the next field in the original record. The order in which the sample values are prompted, helps in determining the boundary fields and relative positions for the various fields.

4.2.3. Generation of extraction rules

As already mentioned, when a field is a list of sub-records, each of the fields in that sub-record is a list in itself. Hence, we first try to form individual lists for each such fields and then later combine them together into individual sub-records. We have already seen the rules that have to be learnt to form lists. However, to combine separate lists into list of sub-records, we need to learn some additional extraction rules. New additional information that is gathered for every field is as follows:
List of Sub-Records Boundary Field (LSBFD):

For each field in the sub-record (which occurs as a list), we determine a new boundary field called “List of Sub-Records Boundary Field”. The meaning of this field is the same as that of the boundary field defined in the section on sub-records. The meaning of the original boundary field would be the same as for lists.

For example, for the record structure shown in figure 18, for the field “Rating”, with any list-id, its LSBFD is “Restaurant” and for the field “Restaurant” with any list-id, its LSBFD is itself, which is “Restaurant”. For the field “Rating” with list-id = 1, its boundary field is “Restaurant” and for the field “Rating” with list-id = 2, its boundary field is the field “Rating” with list-id= 1. For the field “Restaurant” with list-id = 1, its boundary field is “Cuisine” and for the field “Restaurant” with list-id = 2, its boundary field is the field “Restaurant” with list-id= 1.

List of Sub-Records Relative Position (LSRP):

For each field in the sub-record (which occurs as a list), we calculate new information called “List of Sub-Records Relative Position”. This is calculated in a manner similar to the relative position calculated for the fields in a regular sub-record. Regular relative position is calculated in a manner as explained in the section on lists.

4.2.4. Applying extraction rules

New steps are added to the record extraction algorithm explained in the section on lists. These new steps pertain to the formation of sub-records from the individual list items. The process is very much similar to the formation of regular sub-records. For every field in the sub-record we fetch the list formed earlier(list 1) and also the list corresponding to the field’s boundary field(list 2). For every list item in list 1, we check
the relative position condition with all the list items in list 2. Each time the condition is satisfied, we either create a new sub-record with the list item from list 1 and the list item from list 2 or add the list item from list 1 to a previously created sub-record (the one that has the boundary field list item).

These new steps are listed below:

6. For all the fields in the record structure $S$, fetch the field-ids and their corresponding LSBFD’s from the database.

7. Group the field-ids and their LSBFDs into different groups, one for each LSBFD, by creating a hash LSBFDH with LSBFD as the key.

8. for each key in LSBFDH do begin

   a. for each value of field-id and its LSBFD in the group corresponding to key do begin

      i. fetch the list of lists LLF corresponding to the field-id from the hash AL.

      ii. fetch the list of lists LLB corresponding to the LSBFD from the hash AL.

      iii. for each list item(node NF) in each list of LLF do begin

             a. fetch the list of sub-records relative position values minLP and maxLP for NF

             b. for each list item(node NB) in each list of LLB do begin

                    i. Calculate the LRP, difference in the node number of NF and the node number of NB.

                    ii. if LRP >= minLP and <= maxLP, then...
a. If a sub-record containing NB does not already exits then
   i. create a new sub-record with NB and NF as the fields and add it to a list of sub-records LSR. Add LSR to another list that contains all such LSRs, LLSR.

else
   ii. add node NF to the sub-record that has NB in it, to the proper LSR.
   iii. Create a hash LLSRH with the LSBFD as the key and LLSR as its value.

   done.
   done.

iv. for each key of LLSRH do begin
 a. for each list LSR, in lists of sub-records corresponding to the value of key do begin
    i. add all those NB’s that are not included in any sub-records of any list LSR.

    done

   done

   done

   done
At the end of step 8, we will have several lists of list of sub-records. Corresponding to each LSBFD, we will have a list of lists of sub-records.

Step (8 a iii b ii a i) indicates that a new sub-record is formed only when the specified condition is satisfied. The sub-record has nodes NF and NB. Only if node NF with the proper node number exists then NB is added to the sub-record. However, in case of in-complete sub-records, NB may not have corresponding NF. In such situations, there will be no sub-records with NB alone. This will result in the loss of some list of sub-records and will have an effect during complete record formation too. To avoid this situation, we add step (8 a iv).

After step 8, we have to form complete records, which we will discuss in the next sub-section.

Figure 19 shows the process of combining the lists extracted for fields “Restaurant” and “Rating”, into lists of sub-records. Each sub-record of a list of sub-records contains fields “Restaurant” and “Rating”.

5. Complete Record Formation

This section explains the steps involved in the formation of complete nested record structures.

We have seen in the last three sections, the steps involved in the formation of sub-records, lists and list of sub-records. The end result of the record extraction process in each of the three cases is a hash.
In case of sub-records we have a hash SRH. The keys are field-id’s of the boundary fields (first) in each sub record. Value is a collection of sub-records, all of which have the same field as their boundary field.

In case of lists, we have a hash AL. Keys for this hash are the field-ids of the list field. Value is a collection of lists for the key field.

In case of a list of sub-records, we have a hash LLSRH. Keys for this hash are the first fields in the sub records of a list of sub records. Value for each key is a collection of list of sub-records, all of which have the same key field as the first field in the sub-records of the list of sub-records.
We follow a bottom-up approach to combine all these three different types of structures into a single nested record, according to the structure specified by the user. The basic idea is to start the combining process beginning from the lower levels and to keep moving up one level at a time, till we reach the highest level. At each level, for a collection of sub-records or lists or list of sub-records, we map each item in the collection with its corresponding boundary field item that might exist in a different collection at the same or higher level. By mapping we mean that a relative position calculation is performed and if the required condition is satisfied, we either create a new final record with the contents of the actual item and its boundary field item, or just add the current item to a partial final record, which was created previously by a similar process. Creation of a new record is only done when a field’s boundary field is itself. We can have a situation where a field that exists at a lower level has a boundary field at a higher level (which is not the highest level) and that boundary field has its boundary field at another higher level, and so on. In such a situation we only add the current item to its boundary field’s contents and not create a new record.

Figure 20 shows a web page from www.Zagat.com, which shows the most popular restaurants in a city. Figure 21 shows the record structure for the records of interest. Notice that the record has a nested structure. Field ‘Address’ at level 1 is a sub-record with fields at level 2. Field ‘Phone’ at level 2 is again a sub-record, with fields at level 3. Field ‘Cuisine’ at level 1 is a list with list items at level 2.

As indicated before, the combining process in this case would begin from level 3. Since the boundary field of ‘Phone’ (‘Area_code’) is ‘Street’, and the boundary field of ‘Street’ is ‘Restaurant’, we add the contents of sub-record ‘Phone’ to that of ‘Street’.
Boundary field of ‘Restaurant’ is itself. Hence, later when the combining process reaches level 2, a new record is created with the contents of ‘Restaurant’ and ‘Street’. Figure 22 shows such a process. The final record will have fields “Restaurant” and “Neighborhood”, sub-records “Rating” and “Address” and list “Cuisine”. Sub-record “Address” will have the contents of the sub-record “Phone” in it.
5.1 Record Extraction Algorithm: New Steps

Below are the new steps added to the record extraction algorithm, to form complete records:

9. Get the key values of the hashes SRH, AL and LLSRH. (In case of the keys of AL, ignore those key values that belong to a list of sub-records, but are not the first fields of such sub-records).

10. For each key value (field-id), fetch its level-id and boundary field from the database. Create a hash FLBH with the field-id as the key and an array containing the level-id and boundary field, as its value.

11. Get an array(1) of field-ids which are ordered based on the following conditions:

   Any field which has a boundary field, that itself has another field as a boundary field is given the first priority.

   Remaining field-ids are arranged in the decreasing order of their level-ids.

   This is done as part of the bottom-up approach of record formation.
12. for each field-id FID in array1
   a. get its boundary field BFD.
      a. if FID exists in SRH then
         i. get the array (array2) of sub-records, corresponding to FID.
         ii. search if BFD is the key of SRH or AL or LLSRH.
            a. if BFD is a key of SRH, then
               get the array (array3) of sub-records corresponding to BFD.
            b. if BFD is a key of AL, then
               get the array (array3) of lists corresponding to BFD.
            c. if BFD is a key of LLSRH, then
               get the array (array3) of list of sub-records corresponding to BFD.
         iii. for each first field FF1 in each sub record of array2,
            a. get its node number and the relative position values minP and maxP.
            b. if (a) then
               i. for each first field FF2 in each sub-record of array3
                  a. get FF2’s node number.
                  b. calculate RP, the difference in the node number of FF2 and FF1.
                  c. if RP >= minP and <=maxP, then.
                     i. create a final record containing the current sub-record of array3 and array2. The final record is a hash FH with the node number
of FF2 as the key and the value is a record containing the sub-records. This record is created only if a hash with the node number of FF2 does not exist and when the boundary field of FF2 is itself.

ii. if a hash with the node number of FF2 already exists, then
    the contents of the current sub-record of array2 are added to that record.

iii. if the boundary field of FF2 is not itself, then,
    the contents of the current sub-record of array2 are added to the current sub-record of array3.

c. if (b) or (c), similar steps are executed.

done

b. if FID exists in AL or LLSRH then
    steps similar to above are executed.

The steps after the formation of final records are mandatory field check, displaying final records in the display area of the extractor GUI and inserting the records extracted into the database.
5.2 Mandatory Field Check

Hash FH has node numbers as keys and the value is a record that can contain a sub record, or a simple list or a list of sub records in any combination and order. To perform mandatory field check, we first get the field-ids of all the mandatory fields for the record structure and check if they exist in every final record. Only if they do, they are displayed and inserted into the database. The fields of the record are indented in the display area of the GUI, according to their level, in-order to make their structure more explicit.

Figure 22. Bottom up approach to form a complete record
CHAPTER 5

SUPPORT FOR MULTI-NODE FIELDS

This chapter describes multi-node fields and the changes made to SEFRA to solve the problem associated with extracting such fields.

1. Explanation of the Problem

Certain fields span across multiple nodes in the document tree. Text pertaining to a single field can span across multiple fields, and in all such cases, the current system will pick only the first node. Figure 23 shows an extract from a web page (Prentice Hall of India). It shows information pertaining to certain “Java” books, the author and the price. It is the result of a search for the keywords “books”, “java”, “price” and “author”, using a search engine.

Sometimes, the keywords on which the search is based are highlighted. By “highlighting” we mean that the text is displayed in either bold or different color or different font or some other means. In case of HTML, text-formatting tags are used to achieve such an effect. Some of these tags are <B>, <I>, <FONT>, etc. Some of the text formatting tags in HTML are shown in figure 24. The presence of such text formatting tags can cause the text pertaining to a single field to span across multiple nodes.
Figure 23. Web Page from Prentice Hall of India

List of Text Formatting Tags in HTML

<B>
<FONT>
<BIG>
<SMALL>
<BASEFONT>
<BLINK>
<CITE>
<CODE>
<EM>
<I>
<KBD>
<PLAINTEXT>
<S>
<STRIKE>
<STRONG>
.SUB>
<SUP>
<TT>
<U>
<VAR>

Figure 24. Some of the text formatting tags in HTML
Figure 25. Portion of web page from http://www.phindia.com/, as displayed in the GUI display area of the learner

Figure 25 shows a portion of the web page shown in figure 23, as displayed in the GUI display area of the learner.

From figure 25 we can see how each of the titles “E-COMMERCE APPLICATIONS USING ORACLE8I™ AND JAVA”, “JAVA PROGRAMMING FOR THE ABSOLUTE BEGINNER”, and “JAVA (TM) 2 FAST AND EASY® WEB DEVELOPMENT” are split across multiple nodes. A portion of the document tree for the web page shown in figure 25, is shown figure 26. When the trainer provides sample values for any field of the record structure, he enters the node number associated with the text of that field. However, in a situation as above, for the field “Title”, the trainer can enter only one node number, which is usually the first. The extractor will then pick up only the first node for each text associated with the “Title” field.
2. Solution

To avoid the problem mentioned above, we ignore all tags that are related to text formatting while forming the document tree. This is because, the text formatting tags, besides increasing the depth of the tree, do not provide any other valuable information. Since all these tags are only applied to the text in the tree, removing them will not affect the current process, which relies heavily upon tags to extract the field nodes.

We have already seen that an entire web page is first converted into a document tree. Only the plain texts appear in the display area of the learner GUI.

Figure 26. Portion of the document tree for the web page in figure 25

Steps involved in removing the text formatting tags from a document tree are as follows:

1. let T be the document tree.
2. let TFT denote the collection of text formatting tags.
3. for each tag TG in TFT do begin
   a. find in T all those nodes NS that have the tag TG.
b. add nodes NS to a collection of tag nodes TN.

**done**

4. **for** each node N in NS **do begin**
   a. get the node N’s parent P.
   b. replace node N in P’s list of children, with the children of N.
   c. combine together all consecutive texts of P.

**done**

Figure 27 shows the result of removing the text formatting tags from the document tree. We can notice that each of the titles “E-COMMERCE APPLICATIONS USING ORACLE8I™ AND JAVA”, “JAVA PROGRAMMING FOR THE ABSOLUTE BEGINNER”, and “JAVA (TM) 2 FAST AND EASY® WEB DEVELOPMENT” are no longer split across multiple nodes. Figure 28 shows the document tree after removing the text tags. Compare it with the document tree shown in figure 26.

---

**Figure 27. Portion of web page from** [http://www.phindia.com/](http://www.phindia.com/), **as displayed in the GUI display area of the learner**
6. REDUCING FINE TUNING OVERHEAD

In this section we discuss the approach taken to reduce the fine-tuning overhead. The approach is an alternative to learning the length of sample text nodes provided for the different fields of a record structure.

1. Explanation of the Problem

The fine-tuning feature of the extractor GUI is used to quickly adjust values learnt by the learner. Although this feature was supposed to be used only when absolutely necessary, previous experimental results on SEFRA showed that this feature had to be used for almost every web site. However, most of the fine-tuning was done to adjust the length of the text for different fields of the record structure. The reason behind incorporating the length information was that most of the fields would have a good amount of text and by learning the variations in the length of the text, invalid nodes could be ignored. This is because, most of the invalid nodes have few or no text and sometimes blank characters.

Figure 28. Portion of the document tree after removing the formatting tags for the web page shown in figure 23

```
E-COMMERCE APPLICATIONS USING……. FROM SCRATCH

<table>
<thead>
<tr>
<th>Author: THAKKAR</th>
<th>Price: Rs.150.00</th>
</tr>
</thead>
</table>
```

E-COMMERCE APPLICATIONS USING……. FROM SCRATCH
As it seems inappropriate for a system to rely so heavily on a feature, which is so unpredictable, a better way of reducing the possibility of getting invalid nodes is needed.

2. Solution

One way is to run some heuristics on the document tree. Embely et. al.[7] follow such an approach. They break the tag tree into sub-trees, and look only at those sub-trees that have a high fan-out. Their approach assumes that there will be several records in a web page. As an alternative to making such an assumption, we use a similar technique by storing fan-out information of the parent nodes of the fields in a sample record and then apply heuristics at the extraction end to choose candidate sub-trees where the parent nodes of the leaf nodes have fan-out's similar to the learnt fan-out values.

3. Changes

Steps to which changes have been made to incorporate above are:

3.1. Generation of extraction rules

Following new information is gathered for every field of each record.

*Fan-Out:*

“Fan-out” is the number of children of a parent node. For a particular field, the parent (ancestor) node whose fan-out value is learnt, is the one that has fan-out > 1.

*Level:*

The number of levels we move up the tree, starting from the parent at the lowest level, till we reach a parent (ancestor) node whose fan-out > 1, is the “Level”.
Instead of directly obtaining the fan-out value of the parent node of a particular field, we first check if the fan-out of the parent node is > 1, and only if the fan-out of the parent node > 1, we get that value to be stored in the database. Else, if the fan-out = 1, we move one level up, and re-check. This we do till we find a parent(ancestor) node that has fan-out > 1. We also keep track of the number of levels we move up the tree, till we reach the proper parent(ancestor) node. Since the parent nodes of most of the invalid nodes have a fan-out of 1, we use this approach of moving up the tree till we find the proper parent(ancestor) node in-order to avoid extracting invalid nodes, which is the purpose of this whole process.

For example, look at the document tree shown in the figure 28. For the field ‘Title’ with the text “E-COMMERCE APPLICATIONS USING ORACLE8I™ AND JAVA™ FROM SCRATCH”, the fan-out value would be 3 and the level value would be 3. For the field ‘Author’ with the text “THAKKAR”, the fan-out and level values would be 2 and 2 respectively. Also for the field ‘Price’ with the text “Rs.150.00”, the fan-out and level values would be 2 and 2 respectively.

The rule generation algorithm uses the above new information to formulate the below new rules for each field of the record structure.

- The min and max value for Fan-out F is calculated based on all the values of F.
- If Level L is different than those of previous records, it is added to the group of distinct levels fetched from previous records.

3.2. Applying extraction rules
New steps to the record extraction algorithm are added. These new steps from step 3 a are:

a. depth D and tag sequence T pairs, data type DT, keywords K, omitwords O and min and max fan-out values, minF, and maxF, and level values L are fetched from the learned values stored for document D.

a. from T, plain text nodes with plain text PT at depth D, having corresponding tag sequence T, for every D & T pair are fetched only if, at a depth (level) L from the lowest level of T, the node has a fan-out NF such that NF >= minF and NF =< maxF.

b. for every node fetched following steps are executed:

i. skip node if its PT is not of data type DT.

ii. skip node if its PT does not contain any keyword in K.

iii. skip node if its PT contains any omitword in O.

iv. If PT is not skipped then attach field name FN and its node number N to the PT.

v. put the passed node in the extracted nodes EN bucket.

No changes to the other steps of the record extraction algorithm are needed.

Step 3 b means that while traversing the tree T, when we reach a level L (L is calculated from the lowest level of the tree), we set a flag only when the current node has a fan-out NF such that NF >= minF and NF =< maxNF. Later, when at a depth D, the node is fetched only if the flag is set. This ensures that we only look at certain sub-trees that can have valid nodes and reject those sub-trees that have invalid nodes.
7 SUPPORT FOR MULTIPLE FIELDS COMBINED TOGETHER AND REMOVAL OF IRRELEVANT INFORMATION

In this section we discuss the approach taken to separate multiple fields combined together as a single node and also the approach taken to remove irrelevant information from extracted nodes. The solution to both the problems is same. Hence we discuss the solution in the same chapter.

1. Explanation of the Problems

We have seen previously that for each field FN in the record structure S, the record extraction algorithm fetches a plain text node with plain text PT, if it satisfies the extraction rules. PT is the entire text associated with the node and should contain information specific for the field, for which it has been fetched.

However, sometimes the information contained in PT may not entirely be the required information. There can be two situations:

- Additional, irrelevant information, other than what is required for a field is contained in PT.
- PT contains information associated with more than one field in it.

For a portion of a web page shown in figure 9, figure 29 shows a portion of that web page as displayed in the GUI display area of the learner.

```
1506 : from $154 :
1513 : Choose and continue:
1520 : 6:20 AM Depart Portland (PDX):
1522 : Arrive San Jose (SJC) 8:05 AM:
1525 : Thu 18-Apr:
1527 : 1hr 45mn :
1536 : Alaska Airlines 362 :
1538 : Nonstop flight:
```

Figure 29. Portion of a web page as displayed in the GUI display area
We have already seen the record structure (figure 10) for the information displayed in figure 29, in the section on sub-records (chapter 4).

When the record extraction algorithm fetches node 1506, as a valid node for the field “Price”, the entire text “from $154” is fetched. However, the required text would just be “$154”. Similarly, if we look at the text of node 1520, it has information associated with three different fields- “Departure Time”, “Departure City”, and “Departure City Airport Code”. According to the working of SEFRA, the trainer can only define one field for the entire information associated with that node, such as “Depart Info”. However, it would be preferable to separate the information into different fields.

Most of the previous work done in the field of information retrieval was to formulate rules based on the text surrounding the record attributes. Although the approach of SEFRA is a paradigm shift from this normal approach, we could incorporate a similar technique as a solution to the above, two mentioned problems. For this, we would require the trainer to provide new information for each field in the record structure. While providing sample values, if a same node is being provided as a sample value for more than one field, unless the trainer tells the learner, which part of the text associated with that node is the relevant portion for that field, the extractor will not be able to separate the information associated with that node into different fields.

2. Solution

Changes are made to the current system to solve the two problems mentioned previously.

2.1. Changes

2.1.1. Generation of extraction rules
New information is gathered for every field of a record structure:

i. Preceded By

Any number of word(s) or characters that precede the text that pertains to a particular field, in the text of a plain text node is referred to as “Preceded By”. After looking at the entire text of a plain text node, the trainer inputs appropriate words or characters that precede the text of interest in the text box provided, during the training process for each field. For example, from the figure 29, we can see that for the field ‘Price’, the trainer will provide the value for this information as “from”.

ii. Followed By

Any number of word(s) or characters that follow the text that pertains to a particular field, in the text of a plain text node is referred to as “Followed By”. After looking at the entire text of a plain text node, the trainer inputs appropriate words or characters that follow the text of interest in the text box provided, during the training process for each field. For example, from the figure 29, we can see that for the field ‘Depart_Time’, the trainer will provide the value for this information as “Depart”.

iii. Begins With

Any number of word(s) or characters with which the text that pertains to a particular field begins with, is referred to as “Begins With”. After looking at the entire text of a plain text node, the trainer inputs appropriate words or characters with which the text of interest begins with, in the text box provided, during the
training process for each field. For example, from the figure 29, we can see that for the field ‘Price’, the trainer will provide the value for this information as “$”.

iv. Ends With

Any number of word(s) or characters with which the text that pertains to a particular field ends with, is referred to as “Ends With”. After looking at the entire text of a plain text node, the trainer inputs appropriate words or characters with which the text of interest ends with, in the text box provided, during the training process for each field. For example, from the figure 29, we can see that for the field ‘Depart_Time’, the trainer will provide the value for this information as “AM”.

New features have been added to the GUI to accept the above information for each field.

*Rule Generation Algorithm:*

Following new extraction rules are formulated for each field in the record structure:

For each field, apart from the ones previously mentioned, preceded by PB, followed by FB, begins with BW and ends with EW from all the records previously stored are fetched.

If the current PB value is different than the previous records, new PB value is added to the group of distinct PB values fetched from previous records.

If the current FB value is different than the previous records, new FB value is added to the group of distinct FB values fetched from previous records.
If the current BW value is different than the previous records, new BW value is added to the group of distinct BW values fetched from previous records.

If the current EW value is different than the previous records, new EW value is added to the group of distinct EW values fetched from previous records.

2.1.2. Applying extraction rules

Following new steps are added to the modified step 3 (explained in the previous section). New steps are added after step (3 c iii). These steps are executed for each distinct value of PB, FB, BW and EW. These steps are:

3 c iv. if PT contains PB then
   a. remove all characters from the beginning of PT till PB.
   b. remove PB from PT.
   c. set pbflag true.

v. if PT contains FB then
   a. remove all characters from the end of PT till FB.
   b. remove FB from PT.
   c. set fbflag true.

vi. if PT begins with BW then
    set bwflag true.

vii. if PT ends with EW then
    set ewflag true.

viii. if pbflag and fbflag and bwflag and ewflag are true then
    accept node.

else
    skip node
Remaining steps are same as those specified in the previous section from step (3 c iv).
Notice from the steps specified above that the same PT is being processed at each step.
If condition (3 c iv) is satisfied, PT is modified by removing certain characters from it.
Then, that modified PT is tested in step 3 c v, and so on.

8. IMPLEMENTATION AND USER MANUAL

This chapter describes important implementation issues and also discusses all the steps involved in using the current system, including an explanation of the various screens/forms with accompanying screen shots.

1. Implementation

We have mentioned in the beginning that our goal is to provide additional support to SEFRA. To do that, changes as discussed in all the previous sections have been made to SEFRA. As done before, in this section we will discuss important implementation issues.

1.1. Software and hardware requirements

Software:
Programming Language – Perl
Perl Libraries – Perl/TK, Dbd-Mysql, HTML-Parser, HTML-TreeBuilder
Database – Mysql
Operating System – Linux, Win 98, Win NT, Win 2000, Win XP
PC with Pentium III processor.

Refer [6] for a description of useful PERL modules, steps involved in the GUI creation, important DBI API calls and the basic functionality of each of the important system modules.

1.2. Database Schema

Several changes have been made to the database schema.

The database interfaces between the Learner and the Extractor modules. It primarily stores the rules learnt for the Web documents.

Figure 30 shows the database schema. The arrows between the various tables indicate a foreign key constraint.

- Table **SITES** contains the name assigned to a document type that should be learnt, and also its unique site-id.

- Table **SITEFIELDS** contains the record structure associated with a site-id. Every field of a record structure has a unique field-id. All the remaining fields in this table provide additional information for each field in the record structure.

- Table **SAMPLEVALUES** holds all the sample records shown by the trainer. For every field of record structure, which is not a list, there will be one row per record. If a field is a list, there will be more than one row with the same field-id but with different list-id’s per record.

- Table **FINALVALUES** contains the extraction rules derived for each field in the record structure, per site-id. For a field, which is not a list, there will be one row per site-id. For a field, which is a list, there will be two rows,
one with a list-id =1 and another with list-id = 2. Extraction rules are derived from all the records stored in the SAMPLEVALUES table, for that site. These are the rules that are applied to extract records from web documents.

The remaining four tables contain information that assists in comparison-shopping. Consider any two web sites that sell books and are used by the current system to extract records representing information pertaining to books. Most likely the record structure of the records for these two sites will be different. To be able to perform comparison-shopping for these two sites, we have to convert such different record structures, but which represent similar information(books), into a common structure. Such a common structure is defined using a template. Having a common template makes it easy to perform comparison between those two sites using standard SQL.

- Table TEMPLATES contains the name given to a template and its unique id.
- Table TEMPLATEFIELDS contains the structure of a template.
- Table TEMPLATEASSOCIATION contains the mapping between the fields of a record structure and the fields of a template.
Figure 30. Database Schema
Table EXTRACTEDRECS contains the records that are extracted by the extractor corresponding to a template. This table serves as the base table for comparison-shopping. Sub-records can be identified by their parent ids. All the fields with the same parent id form a sub-record. Similarly, all the entries that have a non-zero list-id are list-items. Each list has list items with the same field id.

2. User Manual

In this section we will discuss all the steps involved in using this system, including an explanation of the various screens/forms with accompanying screen shots.

2.1. Learner

The Learner module facilitates the learning process. Executing the PERL program Learner.pl, will bring up the Learner GUI. A screen shot of the same is shown in figure 31.

The top most area of the GUI is the menu bar. Using options in the “File” menu, a web document, previously saved, is first selected to be opened. A submenu item “Open”, is used to browse the file system.

Once opened, the text of the selected web document appears in the text area, called the display area. Figure 31 shows this. The node number at the beginning of each line is the node number that is assigned to each node of the plain text in the document tree. The web document selected is the one shown in figure 9 (in section on sub-records).

To define a record structure for the opened web document, the trainer should click the button Define New Site. This will bring up a form, which is used to specify the
site name and the number of fields. This form is shown in figure 32. For the nested record structure shown in figure 10, which is defined for the records of interest displayed in the web page shown in figure 9, the trainer, after specifying the site name, has to first enter the number of fields at level 1. In this case it will be 6.

When the trainer clicks the button “OK” on the form shown in figure 32, a new form to accept individual field details is shown. This new form is shown in figure 33.
Figure 32. Record Definition

The ‘Type’ column indicates the data type. It can take three different values – A for only alphabets, N for only numbers and B for any character. Default value is B.

The ‘Mandatory’ column can take two values – N for a non-mandatory field and Y for a mandatory field. Default value is N.

The ‘List’ column can take two values – N for a non-list field and Y for a list field. Default value is N.

The ‘Sub Record’ column can take two values – N for a non-sub-record field and Y for a sub-record field. Default value is N.

Figure 33. Record Field Definition
Notice that for the fields “Depart_info”, “Arrival_Info” and “Airlines_Info”, the value in the column “Sub Record” is Y.

For all the fields, which are sub-records (those that have Y in the Sub-Records column), another form as shown in figure 34, to first accept the number of fields in that sub-record, is shown. After clicking the button “OK” in that form, another form, as shown in figure 35, to accept the field details of that sub-record is shown. This form is same as the one shown in figure 33.

Figure 34. Sub-Record Details

<table>
<thead>
<tr>
<th>Field Names</th>
<th>Type</th>
<th>Mandatory</th>
<th>List</th>
<th>Sub Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival_City</td>
<td>B</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Arrival_Airport_Code</td>
<td>B</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Arrival_Time</td>
<td>B</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 35. Sub-Record Field Details

Once the trainer clicks button “OK” of the form shown in figure 35, the above two forms are shown for all the remaining fields at level 1, that are sub-records. This process again continues for all sub-record fields at level 2, and so on, till none of the fields at the lowest level are sub-records. In the above case, this process stops after similar forms are
shown for fields “Depart_Info” and “Airlines_Info”, as none of the fields that belong to a sub-record at level 1 are sub-records.

The **Start Learning** button prompts the trainer to select the name of the document structure that the learner should begin to learn. The form is a drop down menu, with the names of all record structures in the database. The trainer selects the one, which reflects the document loaded in the display area. Figure 36 shows this form. Then he can start showing the sample records.

![Select the site](image-url)

**Figure 36. Record Selection**

The form below the display area is used to show sample records. When the trainer clicks the button “OK” of the form in figure 36, buttons **CAPTURE** and **NEXT FIELD** are enabled. The **Rec #** column will show the record number of the current sample record. Figure 37 shows that number as 1, meaning that no previous sample records were provided. Column ‘Field Name’, shows the first field of the record structure selected previously. In the area below the label ‘Node #’, the trainer enters the node number associated with the text node that he wishes to provide as a sample value. He then clicks the button **CAPTURE**. This will cause the text associated with the node number entered previously, to be displayed in the area below the label ‘Value’. In the area associated with the label ‘Keyword(s)’, the trainer can either enter any keyword associated with that field or leave it blank. Similar is the case for all the remaining text
areas associated with labels 'Omitword(s)', 'BeginsWith', 'EndsWith', 'PrecededBy' and 'FollowedBy'. The trainer can enter any Omitwords, BeginsWith, EndsWith, PrecededBy, or FollowedBy values associated with that field, in the areas provided for them or leave them blank. Figure 37 shows the values entered for the field “Price”.

After entering all the information possible for the field ‘Price’, the trainer clicks the button NEXT FIELD. This will cause the GUI to prompt for the next field in the record. If a field is a sub-record, then a similar process begins for the fields in the sub-record, starting with the first field of the sub-record. The label on the button NEXT FIELD changes to NEXT SUBREC FIELD. Figure 38 shows this. The field being shown is ‘Depart_Time’, the first field of the sub-record ‘Depart_Info’. A message at the bottom most part of the GUI indicates the current field name or the sub-record name, for which the details are being accepted. After accepting the details for all the fields in the sub-record “Depart_Info”, the GUI will prompt for the next field in the actual record structure and so on. After accepting details for the last field in the record, clicking the button NEXT FIELD / NEXT SUBREC FIELD will start a new record. As the details for each field are shown, the values are collected and stored in the table SAMPLE VALUES. At the end of each record, extraction rules are generated and stored in the table FINALVALUES. Whenever the next field that should be prompted for, is a list, a new form as shown in figure 39 is shown. It is a reminder to the trainer that if a list has more than one field, then both the first and the second list items should be provided as sample values. This form is shown only once for the entire session and not for every list item or list field.

When the trainer clicks the button “OK”, buttons NEXT LIST ITEM and DONE on the learner GUI, are enabled and the button NEXT FIELD is disabled. Figure 41 shows this. After entering the required values for the first list item, if the trainer wishes to provide more sample values for that field, he has to click the button NEXT LIST ITEM. This will
allow him to provide sample values for the next list item. The field name will remain the same. If he wishes to stop providing any more sample values for that list field, he has to click the button **DONE**. This will cause the GUI to prompt for the next field in the actual record. If, after providing only one list item, the trainer clicks the **DONE** button, a warning as shown in figure 40, will be displayed. Clicking the button **CANCEL** on that form will let the trainer to continue providing sample values for other list items. Figure 41, shows the learner GUI, while accepting sample values for a list field.
Figure 38. Learner GUI – Accepting details for a sub-record

Figure 39. Informational form
When the next field to be prompted for, is a list of sub-records, then all the buttons NEXT SUBREC FIELD, NEXT LIST ITEM and DONE are enabled. Figure 42 shows this. To move to the next field in the same sub-record, the trainer has to click NEXT SUBREC FIELD. To move to the first field of the next list item, which is a sub-record, he has to click the button NEXT LIST ITEM. To indicate that he is done providing sample values for all the list items, and hence move to the next field in the actual record, he has to click DONE. Figure 42 shows the learner GUI while accepting sample values for a list of sub-records.

If the trainer makes an error in a previous field of the same record, he can go back to that particular field by continuously clicking the PREV button. Each click will go back by one field or one list item. The scope of PREV is the current record only. To be able to safely move back by one step, the entire state of the system at each step is saved.

The Clear Display button clears the display area. The Exit button exists from the application.
Figure 41. Learn GUI – Accepting details for a list field
2.2. Extractor

The extractor module facilitates the extraction process. Executing the PERL program Extractor.pl will bring up the extractor GUI. A screen shot of the same is shown in figure 43.

The trainer has to first load a web document using the ‘Open ’ submenu item of the ‘File’ menu. He then has to click the button Start Extracting, which displays a form with a drop down menu. The drop down menu holds the names of all the record
structures that were defined using the Learner GUI. The Extractor will apply the extraction rules related to the chosen record structure on the web document selected previously. The form is similar to the one shown in figure 36. The selection of a record structure will trigger the extraction algorithm and all the records extracted will be shown in the display area. Figure 43 shows the records extracted for the web document shown in figure 9. The record structure for this web document is nested with sub-records at level 2. The display in the text area of the Extractor GUI, with indentations, shows this.

![Extractor GUI](image)

**Figure 43. Extractor GUI**

If the trainer is not satisfied with the result, he can change the values learnt for all the fields of the record structure. The trainer has to click the button **Fine Tune** to bring...
up a form as shown in figure 44. It shows some of the values learnt for all the fields of the record structure. To see the remaining values learnt for those fields, he has to click the button **View Others**. This will bring up another form as shown in figure 45. Using these forms, the trainer can change any of the learnt values and click the **Update** button to make changes in the database. Although, forms as these have been provided, they should be used to change the values manually, only if absolutely necessary. Most of the times, additional sample records can be provided to improve the extraction result. We have already discussed the significance of each of the learnt values shown in these two forms. RELPOADJ is the difference in the max and min values for the RELPOS field. The meaning of any other field ending in ‘ADJ’ is similar.

Except for ‘LISTID’ shown in the first form, all the other information is provided by the trainer himself, during the learning process. Hence, he can change these values, if the result is not satisfactory. All the values shown in the second fine tune form are the calculated extraction rules. It is strongly advisable to not change these values. The trainer may use this form to only view the values. However, if absolutely necessary, he can change them with caution.

![Figure 44. Fine turn form with some of the learnt values](image)
Figure 45. Fine tune form with the remaining learnt values

Figure 46 shows the extracted records for the web document in figure 14. The record structure for these records is nested with one of the fields being a list.

Figure 47 shows the extracted records for the web document in figure 16. The record structure for these records is nested with one of the fields being a list of sub-records.

Figure 48 shows the extracted records for the web document in figure 18. The record structure for these records is nested with sub-records at level 2 and level 3, and a list at level 2.

We have previously discussed about Templates. They make it possible to compare records from different web sites selling the same products. To define a template, the trainer has to click the button Define Template, to launch a form as shown in figure 49. The trainer can provide the template name and the number of fields using this form.
Figure 46. Extractor GUI-Records extracted for the web document in figure 14
Figure 47. Extractor GUI – Records Extracted for the web document shown in figure 16
Figure 48. Extractor GUI – Records Extracted for the web document shown in figure 18

Figure 49. Template Definition
Clicking the button **OK** of this form will launch another form as shown in figure 50. The trainer can enter the template field names using this form. The template name and the fields associated with it are stored in the database.

![Image of template field definition](Image)

**Figure 50. Template Field Definition**

Different record structures can be associated with any desired template. To do that, the trainer has to click the button **Associate Template**, which brings up a form as shown in figure 51. It consists of two drop down menus. The menu under SITE NAME header shows all the record structures and the menu under TEMPLATE header shows all the defined templates. In order to associate a template to a record structure, the trainer has to select the appropriate entries from the two drop-down menus and then click the button **OK**, which brings up another form as shown in figure 52.
Figure 51. Template Association

This form allows the trainer to map the fields of the template to the fields of the record structure. This form has two rows. The row to the left displays every field of the template. The row to the right consists of drop down menus. Each menu contains all the fields of the record structure. The field that has to be mapped with the field of the template should be selected in every drop down menu. Once done, the trainer clicks the button **OK**, which puts the association in the database. It is not necessary to match every field of the template to a field in the record structure.
This is the last step in the entire process. Extracted records are put in the EXTRACTEDRECS table.

10. EXPERIMENTAL RESULTS

In this chapter we have tabulated the experimental results.

We carried out the experiments on a number of web sites. For each web site, we performed a search specific to that web site, and obtained pages showing the results. These pages were then stored on our local hard drive. In this project, we have
addressed different problems. Hence, the web sites were chosen with a specific problem in mind. For example some of the web sites chosen had nested structure, some had multi-nodes, some had multiple fields combined together, and so on. Many web sites chosen had multiple problems. We show this information in the tables. Also, we show the record structure for each web site for better understanding.

The results are shown in a tabular form. The meaning of each of the columns in the tables is as follows:

1. List:
   Indicates if one or more fields of the record structure are lists.

2. Sub-Record:
   Indicates if one or more fields of the record structure are sub-records.

3. List of sub-records:
   Indicates if one or more fields of the record structure are list of sub-records.

4. Number of Levels:
   Indicates the number of levels in the record structure.

5. Multi-Nodes:
   Indicates if one or more fields of the record structure are split across multiple nodes.

6. Multiple-fields:
   Indicates if two or more fields of a record structure share the same node.

7. Additional Information:
Indicates if one or more fields of the record structure had some irrelevant text associated with their text nodes.

8. Total time:
Indicates the total time it took to load a web page in the learner GUI, define the record structure, provide sample records to the learner, load the web page in the extractor GUI and to extract the records. The time may vary from person to person. However the time shown is representative of the average time needed to do the above listed functions.

9. Total Samples:
Indicates the total number of sample records provided to the learner, to be able to extract the maximum number of correct records.

10. Records Expected:
Indicates the total number of correct records, in the web pages provided.

11. Records Extracted:
Indicates the total number of records extracted by the extractor.

12. Wrong Records:
Indicates the total number of wrong records among the total records extracted.

13. Incomplete Records:
Indicates the total number of records among the extracted records that had some fields missing.

14. Fine Tune:
Indicates whether any fine-tuning was used.

15. R% - Recall percentage.
Indicates the number of correct extracted records to the total number of records in the web page.

16. P% - Precision percentage

Indicates the number of correct extracted records to the total number of extracted records.

Following are the results from the experiments.

1. Expedia (http://www.expedia.com/)

   The result of a search for itineraries satisfying the given conditions (specific originating city, destination city, travel date, etc.) was a web page containing itineraries with 0 stops, between the originating and destination cities.

Record Structure:
Tabular results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multiple Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The result of a search for authors and texts written by them was a web page containing the author names and a list of texts written by each author.

Record Structure:

```
Athena
    Author
    Books(list)
        Book
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>77</td>
<td>77</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The web pages were the result of a search for a list of restaurants corresponding to each cuisine type, in a particular city.

Record Structure:

```
Zagat
    |
    |
  Cuisine     Name&Rating(list&Sub-Record)
                      |
  Restaurant(list)       Rating(list)
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>31</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The web pages were the result of a search for a list of popular restaurants in a particular city.

Record Structure:

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>3</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
5. Continental (http://www.continental.com/)

The result of a search for itineraries satisfying the given conditions (specific originating city, destination city, travel date, etc.,) was a web page containing itineraries with 1 stop, between the originating and destination city. Each itinerary had corresponding to the price, a list of sub-records. Each sub-record had information, as shown in the figure below. Connecting flight details were not listed as another sub-record of the list of sub-records, but as another field of the sub-record.

Record Structure:

![Record Structure Diagram]

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
6. A1books (http://www.a1books.com/)

The web pages were the result of a search for a list of books on web programming.

Record Structure:

```
Book
   Title
   Author
   Price
   List_Price
   A1_Price
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
7. Amazon (http://www.amazon.com/)

The web pages were the result of a search for a list of books on cooking.

Record Structure:

![Diagram of Record Structure]

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>23</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
8. Delta (http://www.delta.com/)

The result of a search for itineraries satisfying the given conditions (specific originating city, destination city, travel date, etc.,) was a web page containing itineraries with 1 stop, between the originating and final destination city. Each itinerary had corresponding to the price, a list of sub-records. Each sub-record had information as shown in the figure below. Connecting flight details were listed as another sub-record of the list of sub-records.

Record Structure:

- Itinerary
- Price
- Details(List of Sub-Records)
- Flight(list)
- Cabin&Class (list)
- Depart_City (list)
- Depart_Time (list)
- Depart_Date(list)
- Arrival_City (list)
- Arrival_Time (list)
- Arrival_Date(list)
- Stops (list)

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The web pages were the result of a search for the movies and timings in all the theaters of a particular locality of a particular city.

Record Structure:

```
Movie
/    
|    |
Theatre Movie&Timing(list&Sub-Record)
/    
|    |
Movie(list) Timing(list)
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
10. Dell (http://www.dell.com/)

The web pages were the result of a search for all notebook details sold by Dell.

Record Structure:

```
Dell_Computer
  |
  v
Brand  Product  Description  Pricing_Info  Dealer_Info
  |
  v
    Actual_Price  Shipping_Fee  Total_Price  Name  Contact_Info(list)  State
    |
    v
      Contact_Info
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time (Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
The result of a search for itineraries satisfying the given conditions (specific originating city, destination city, travel date, etc.,) was a web page containing itineraries with 2-3 stops, between the originating and final destination city. Each itinerary had corresponding to the price, a list of sub-records. Each sub-record had information as shown in the figure below. Connecting flight details were listed as another sub-record of the list of sub-records.

Record Structure:

![Record Structure Diagram]

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>14</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>Y</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

The web pages were the result of a search for a list of books on java.

Record Structure:

```
<table>
<thead>
<tr>
<th></th>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
```

Tabular Results:

```
<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>4</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```
13. Half Price (http://www.halfpricebooks.com/)

The web pages were the result of a search for a list of books on java.

Record Structure:

```
Book
   Title
   Author
   Publishing_Date
   Price
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
14. Barnes And Nobles (http://www.barnesandnoble.com/)

The web pages were the result of a search for a list of books on Computer Engineering.

Record Structure:

```
Book
  /       /
Title  Author
  |       |
  \
Publishing_Date  Pricing_Info
        |       |
       /     /
New_Price  Used_Price
```

Tabular Results:

<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>17</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
15. Borders (http://www.borders.com/)

The web pages were the result of a search for a list of best sellers.

Record Structure:

```
Book
  └── Title
  └── Book_Details
  │    └── Cover
  │    └── Edition
  └── Author
       └── Pricing_Info
            └── Border’s_Price
            └── Discount
```

Tabular Results:

```
<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Total Time (Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```
16. Specialty Rental (http://www.specialtyrentals.com/)

The web pages were the result of a search for a list of rental cars.

Record Structure:

```
RentalCar
  Model  Passengers  Doors  Rates(list)  Features  Mileage
                  Rate
```

Tabular Results:

```
<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>13</td>
<td>26</td>
<td>13</td>
<td>13</td>
<td>N</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>Y</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```
17. Orbitz (http://www.orbitz.com/)

The web pages were the result of a search for hotels in a particular city.

Record Structure:

```
Hotel
   /\   /
  Name Address Hotel_Description Room_Details(list of sub-records)
     \   /         
      Nightly_Rate Total Room_Description
```

Tabular Results:

```
<table>
<thead>
<tr>
<th>List</th>
<th>Sub-Record</th>
<th>List of Sub-Records</th>
<th>Number of Levels</th>
<th>Multi-Nodes</th>
<th>Multiple Fields</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Time(Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Incomplete Records</th>
<th>Fine Tune</th>
<th>R%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>
```

The web pages were the result of a search for a list of rental cars.

Record Structure:

```
RentalCar
   Company
   Location
   Price
      Economy
      Compact
      Mid_Size
```

Tabular Results:

```
List | Sub-Record | List of Sub-Records | Number of Levels | Multi-Nodes | Multiple Fields | Additional Information |
-----|-------------|---------------------|------------------|-------------|----------------------|------------------------|
N    | Y           | N                   | 2                | N           | N                    | N                      |
```

```
Total Time (Min) | Total Samples | Records Expected | Records Extracted | Wrong Records | Incomplete Records | Fine Tune | R% | P% |
----------------|---------------|------------------|-------------------|---------------|-------------------|-----------|----|----|
3               | 1             | 10               | 10                | 0             | 0                 | N         | 100| 100|
```
The results show that fine-tuning was required for only a couple of sites. The values of “Omit-words” were fine-tuned. Only for these two sites there was some ambiguity, which was resolved using the “Omit-words”. The results also show that invalid nodes were not extracted for most of the sites. All of the sample records provided were carefully chosen. Total time shown in above tables, does not include the time it took to check if every record was extracted, and if every record had all the fields with proper values. Time needed to do that will vary depending on the number of records in the web page.

The precision rate for all the sites shown above is 100%. In case of recall rate, for all the sites, except for one, the recall rate is 100%. We tried to consider different kinds of on-line stores. Shown above are the results for products such as books, itineraries, restaurants, movies, hotels, rental cars and computers, all of which require some kind of comparison shopping.
CHAPTER 10

CONCLUSION

1. Future Work

In this project we have addressed different problems. Initial experiments on various web sites with different kind of problems were encouraging. However, there are certain problems, which have not been addressed, and some of the addressed problems need improvement. Some of the areas that we would like to address in our future work are as follows:

1. Support for Nested lists

Lists can be nested. Each list item can be associated with another list. This kind of nesting can extend up to any number of levels. An example of a nested list is as follows:

U.S.A
- California
  - San Francisco
  - San Jose
- Oregon
  - Portland
  - Beaverton
  - Salem

India
- Andhra Pradesh
  - Hyderabad
  - Ramagundum
- Karnataka
Associated with each country is a list of states. Each state is associated with a list of cities. The current system cannot handle such nested lists. Support for such nested lists can be provided, as done for simple lists. Changes to all the steps we have seen before should be made. Record structure definition module should be enhanced to allow the trainer to indicate fields, which are nested lists. The trainer should be able to provide sample values for all the necessary list items and new information for each such list item should be gathered. The new information will include boundary fields and relative position values. New extraction rules should be formulated. At the extraction end, individual lists should be formed first, irrespective of the list item they are associated with. Then, all such lists should be combined together to form lists at higher levels using the bottom up approach.

2. Support for complex list of sub-records

The current system can extract simple list of sub-records, where each field of the sub-record is just a simple field. That is, such a field of the sub-record cannot be another list, or sub-record, or list of sub-records. The current system can however, extract simple sub-records that can exist at any level. (Experimental results show one at level 3. The same can be done at even higher levels). Some of the airline itineraries have such complex record structures. Figure 53 shows itineraries from www.travelocity.com. The record structure for the itinerary is shown in figure 54.

Notice that in figure 54, field “Flight_Details” is a list of sub-records. Fields “Depart_Info” and “Arrival_Info” at level 2 are sub-records. Field “Airlines_Info” at level 2
is a list of sub-records (field “Airlines_Number” at level 3 of this sub-record is a list). Field “Connection_City” at level 2 is a list.

Figure 53. Portion of a web page from www.Travelocity.com

Our current approach to form a list of sub-records is to first form individual lists for each of the fields in the sub-record and then combine them all together in to a list of sub-records. However, in case of complex nested structures as shown in figure 54, an alternate approach may be needed. The record structure definition should be modified to allow the trainer to indicate a field (such as field “Airlines_Number”), which is a list in itself, as a list. Our current system will not allow this (we define all the fields in a sub-record belonging to a list of sub-records, as lists). To avoid the formation of deeply
nested lists, we can use an alternate approach of forming sub-records first and then combine them into a list of sub-records. Also, we can still use our current approach, and the support for nested lists might be helpful. Bottom up approach should be still used to form complete records.

3. Support for lists that do not have constant relative position values for all the list items

Most of the web sites that we have worked with, that had a field as a list, had constant relative position values for all the list items. However, in case of a couple of itineraries (travelocity), we found that the difference in the node numbers of some consecutive list items is not same. We have already discussed the problem of learning relative position values for lists.

Since the number of such sites that have this problem is very low, one solution for this is for the trainer to provide all list items, till the one that has different relative
position value, as sample values. Changes should be made to gather the relative position values for all the list items provided as sample values, and compute the min and max relative position values and associate those values with the list item with list id = 2, in the FINALVALUES table.

For the current system, we can also manually fine-tune the relative position values, without making many changes. However, it might be tedious and time consuming to manually compute the relative position values for all the list fields, especially if there are several of them.

4. Support for list items that share the same node number

Sometimes, all the list items of a list share the same node number and are separated by some character (comma, slash, etc). We usually find this type of lists for the authors of a book.

One solution for this type of lists is to determine the character(s) that separate the individual list items. The trainer himself can provide this information, or we can enhance the learner to determine such characters by examining the text and applying certain heuristics. The trainer can also highlight consecutive list items, one pair for each different type of character separating them. Doing that would make it easy to infer the characters separating the list items.

Once the separating character(s) is found, it is very easy to separate them into different list items.

5. Support for multiple fields that share the same node, when they are not separated by some unique patterns
The current system can separate multiple fields combined together, only if they are all separated by some unique pattern. In case there are several common patterns separating one another, the text will have the same pattern occurring at different locations. However, since the extractor module will always look for the first occurrence of a given pattern, and take the necessary action, the result will not be as expected.

To avoid such a problem, one solution is to be able to even learn the number of the common pattern separating a field from the others. By “number of the common pattern” we mean the n'th occurrence of the common pattern in the text starting from the left. One way to do that is to let the trainer show which parts of a text should be associated with each field. The trainer can highlight portions of the text, while providing sample values. The learner should then be able to infer the patterns that precede and follow the highlighted text, the n'th occurrence of such patterns, and also can infer the patterns with which the highlighted text begins and ends. This approach has the added advantage of reducing the user input. That is the trainer does not need to enter the values of “PrecededBy”, “FollowedBy”, “BeginsWith” and “EndsWith”.

References:


[7] Record-Boundary discovery in web documents D.W Embley, Y. Jiang Y.K Ng. Department of computer science Brigham Young University

