Geotemporal Search : A Literature Survey

Abhishek Mallik

June 27, 2013

1 Introduction

Traditional information retrieval (IR) systems retrieve a set of documents as search results according to the relevance score of each document to the user query. Main limitation of these systems is that they do not take into account the implicit geotemporal context associated with the query and depend only on the keywords in the query to satisfy the user's information needs.

Geotemporal searches are getting very popular now a days, *e.g.*, Google inside search [1]. Now queries that request geotemporally relevant information is very common. So goal of this kind of search system is to satisfy information need of an user in a better way than the traditional information retrieval systems when, either explicitly or implicitly a geotemporal context is associated with the user. An user may search for information related to restaurants opened at her place at the time when she is querying. With such system an user can find an ATM in a given neighborhood. She can search for movies going in theaters of her area with timings matching with her current timing. Also an user can search this system to get weather related information or information related to events that are geotemporally relevant to her geotemporal context at the time of querying. We can give a simple example to clarify the utility of geotemporal search. For example, consider the query, "Indian Premier League", here the user might be interested to know specific information about the teams and players of either the most recent Indian Premier League or all that have occurred. For such a query, a traditional information retrieval system returns a list of documents as results based on query-document similarity by ignoring the implicit user intent behind the query. Moreover, as most of the times geographic and temporal information are available in unstructured text in a web document, the information retrieval task becomes a process that is not so straight-forward. So ideally a geotemporal search process has to accurately infer the implicit geotemporal context

[1] http://www.google.com/insidesearch/landing/startsearchingindia.html

from user's information needs and further promote documents that are more geotemporally relevant to the user's query. In this report we shall discuss several papers directly or indirectly related to geotemporal searches. A block diagram of geotemporal system is as shown below:

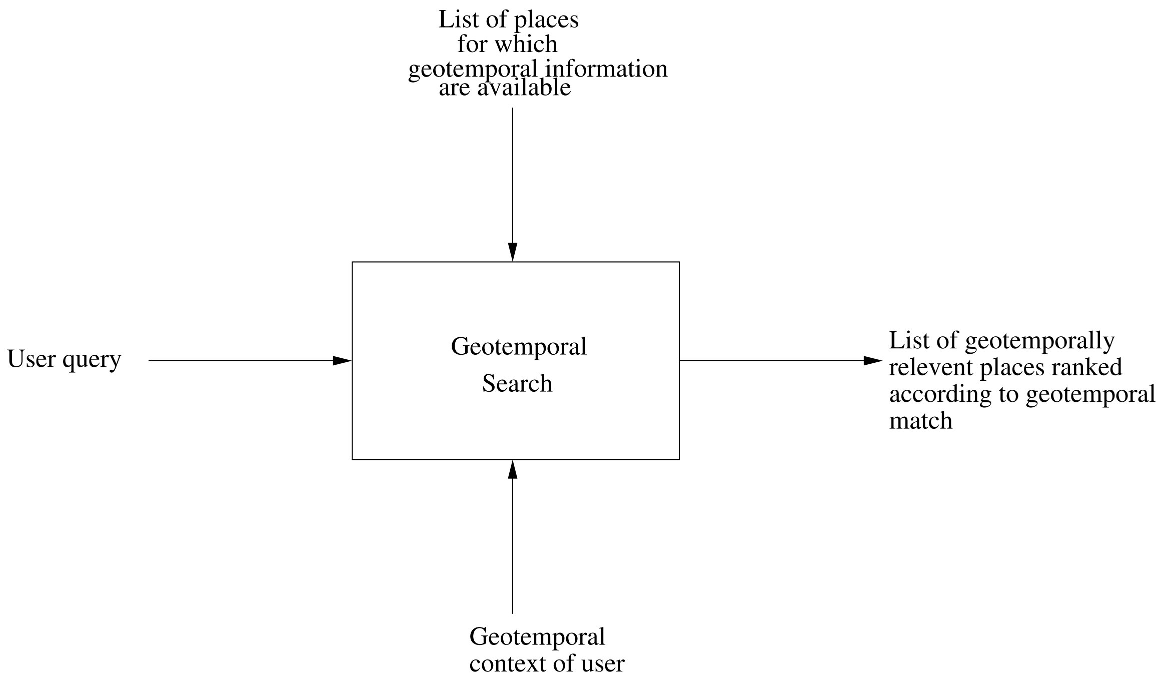

List of places for which geotemporal information are available
User query
Geotemporal Search
List of geotemporally relevent places ranked according to geotemporal match
Geotemporal context of user

Figure 1: Block Diagram of A Geotemporal Search System

It is clear from the discussion so far that, geotemporal context of the query has to be properly identified which is nothing but the information relevant to a particular time and location, user is looking for. Accurately and effectively detecting this geotemporal context where search queries are truly about has huge potential impact on increasing search relevance.

To utilize geotemporal context of an user we have to identify the geotemporal information in documents. Several challenges are involved in both identifying geotemporal context from user query and extracting geotemporal information in documents.

2 Variation of Query

Broadly, from the perspective of geotemporal search, queries can be classified depending on whether the geotemporal information that an user is interested in, is explicit or implicit in the query. Correctly identifying geotemporal context in query can help the search engines in retrieving better search results and thereby achieving search user satisfaction. Location name may or may not be present in a query. Even if it does, the location name present in the query may not mean the geographic location that users are interested in. On the other hand, there are queries that do not contain any location names but do have local context. So if we try to extract location in a query with the help of location dictionary then this look-up may result high false positives. This can occur for cases where there is no location context attached to the query or even if the query has a local context, the locality name that we have found above has nothing to do with associated context of user. Also for queries that do not contain location names, we falsely conclude that there is no location context associated with the query.

Wang et al. in their paper [12] have called the location context associated with the search query it Query Dominant Location (QDL). `Query Dominant Location` is nothing but the location associated with the query which most of the people who knows the answer agrees with. For example though the query `Oh! Calcutta` has a location name `Calcutta`, clearly it has no such associated dominant location. But on the other hand, for the query `Calcutta Sweets`, some may think it is name of a famous sweet shop, whereas many will think that search intent is for sweet shops in Calcutta. So here dominant location in the query, *i.e.*, QDL is `Calcutta`. In their paper [12], Wang et al. have classified queries into the following four types by presence of location keywords and QDL:

- **Type -I** This type of queries do not have any location name explicitly and also no location context is implicit here. So this type of queries do not have a QDL. Some of the examples of this type of queries are: "Natural Language Processing", "Simpson" *etc.*.

- **Type -II**. This type contains location name explicitly and also it is QDL of the query. An example is "Durga Pujo Kolkata." It is obvious that its QDL is Kolkata, West Bengal, India. Google has started to provide local search by identifying locations from this type of queries.

- **Type -III**. Location names are implicit in this type of queries and this type of queries have an associated QDL. An example is "Dominos". Though there ia no location keywords in the query, this query is locally bound to user who enters the query.

- **Type -IV**. Location name is explicit in this type of queries. But this type of queries do not have a QDL, *i.e.*, there is no local context attached to this type of queries. In this type, the location name may be part of a well known phrase, such as in "Oh! Calcutta", but here the word `Calcutta` is not dominant location of the query.

Here the amount of challenge depends on the variation of information and the documents in which the information is embedded.

3 Variation Of Information and Documents

There is no consistency among the way they are embedded in different web documents. If we take into consideration the structures of the web pages from which we are extracting these information then, this problem of information information extraction can be divided into two categories: first is information extraction from web pages having no particular template. For example, there are many isolated pages from different sites containing the geotemporal information. Most of the times templates of all such pages are different from each other. Second category is information extraction from web pages generated from a specific template. Each of these problems can be further categorized based on the type of information: it may be structured or it may be unstructured.

Address may be **structured** or **unstructured**. Addresses of USA, Canada, Europe and other developed nations follow a particular format, *i.e.*, they are structured. This can be clarified with a few examples as follows:

- **317 Dundas St W Toronto, ON M5T 1G4, Canada**,
- **242 E 14th St NY 10003, USA**,
- **268 Augusta Ave Toronto, ON M5T 2L9, Canada**,
- **2402 Bay Area Blvd., Houston, TX 77058**

Unlike these addresses of USA, Canada, Europe and developed nations, addresses of India does not follow any particular format. Here essential address attributes like `house number`, `street name`, `area name`, `city name`, `zip code` are not always present and even if they are present they do not always follow a certain order.
Example

- **Opp. IIT Main Gate, Powai**,
- **Junction Of MG Road and NSC Bose Road, Howrah**,
- **Behind Ram Mandir, Ramrajatala**,
- **G.T. Road North, Near Belur Math, Howrah**

Like address, timing information of a place may be structured or unstructured, as it can appear in numerous different forms and even sometime it can appear in continuous text.

Example

- **Hours of Operation: 9 am -11 pm**
- **Monday CLOSED Tuesday 10 am-5 pm Wednesday 10 am-8 pm Thursday 10 am-5 pm Friday 10 am-5 pm Saturday 10 am-5 pm Sunday 10 am-5 pm**
- **From Noon To Sunset; In Winter, operation is closed**

4 Identifying Geotemporal Context In Query

In this survey mainly we have given emphasis on identification of geographic context associated with a query. Now a days many electronic devices like PCs, laptops, mobiles have built in GPS devices. With the help of those we can identify current location and time of the user while she is typing the query. But this is not sufficient because user may not always search places nearby her only, but she may search also for places in other cities, in other areas. So identification of geographic context from query is also very important task. As normally the query terms are all in lowercase so NLP tools such as Named Entity Recognizer, Parts of Speech tagger may not be effective. In this context, Wang et al. in their paper [12] have proposed different approached to detect the dominant location in query. Most of the time, search queries are too short to identify location in it. So they have identified the geographic context, *i.e.*, the dominant location associated with query from three sources, they are: queries, search results, and query logs. We discuss each of them here.

4.1 Location Identification From Search Queries

Here they have relied on top results of a good search engine on the basis of the notion that, top results of such search engines are very relevant and up-to-date. In this context they have come up with a tokenization algorithm to tokenize the query into tokens by usage of the query in top search results. In the result, if the location name itself is not an atomic token, then it must be part of a name of well known phrase, that we have discussed earlier. Authors formulated the problem as follows:

Given a query Q, split it into most , split it into the most probable token list $TL = t_1, t_2, \dots, t_n$, in order to maximize the conditional probability $Pr(TL|Q)$ [12]. This can be written by applying Bayes' law as follows [12]:

$$\hat{TL} = \arg\max_{TL} Pr(TL|Q) = \arg\max_{TL} \frac{Pr(Q|TL)Pr(TL)}{Pr(Q)} \tag{1}$$

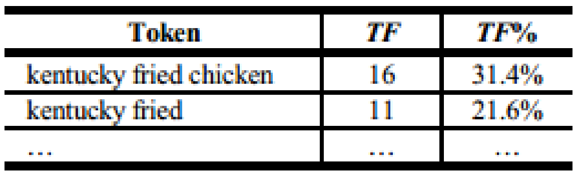
Here $Pr(Q)$ is same for all possible token list for a given query.For a given query Q, $Pr(Q)$ is same for all possible TLs. It is clear that always $Pr(Q|TL)$ is equal to 1. So the problem reduces to maximizing $Pr(TL)$. $Pr(TL)$ is estimated as follows:

$$Pr(TL) \approx \frac{\sum_{i=1}^{n} TF(s_i)}{\sum_{j=1}^{m} TF(t_j)} \tag{2}$$

In this above equation $TF(t_j)$ and $TF(s_i)$ are frequencies of token t_j or s_i in the result snippets respectively, m is the number of all possible tokens for a given query and n is the number of tokens in TL. For example, m is 15 for a query "kentucky fried chicken in seattle" and n is 3 if it is split into "Kentucky fried chicken | in | seattle" ("|" denotes token boundary) [12]. We now present the algorithm with the same example query "kentucky fried chicken in seattle" as given in the paper.

The algorithm suggested by them in this context is as follows [12]:

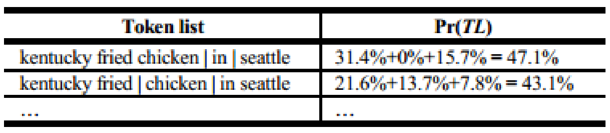
- **Step 1** The query is submitted to the search engine and a list of tokens from top result snippets returned from it are collected in a list in descending order of $TF\%$ where TF is number of occurrences of a token divided by total number of occurrences of all tokens in the top search result.



Token	***TF***	***TF%***
kentucky fried chicken	16	31.4%
kentucky fried	11	21.6%
...	...	...

Figure 2: Top tokens from search results [12]

- **Step 2** Tokens from the list starting from top are assembled into the query without reusing any token.



Token list	**Pr(*TL*)**
kentucky fried chicken | in | seattle	31.4%+0%+15.7% = 47.1%
kentucky fried | chicken | in seattle	21.6%+13.7%+7.8% = 43.1%
...	...

Figure 3: Different assemblies for the example query [12]

- **Step 3** The top token list from Step 2 are picked. For this example, it is "kentucky fried chicken | in | seattle" [12].
- **Step 4** For each token in outcome of Step 3, Steps 1-3 are repeated until it is not further breakable. For this example, it is found that "kentucky fried chicken" is not further breakable [12].
- **Step 5** The final token list only contains atomic tokens and has the largest $Pr(TL)$. For this example, the final output of the algorithm is: "kentucky fried chicken | in | seattle" [12].

With this tokenization algorithm they have suppressed false positives which is evident from the above example. Here though "kentucky" is associated with "fried chicken" it is not QDL. The token "seattle" is atomic and not ambiguous, so is the QDL of this query [12]. As this algorithm is based on live search results so it breaks the queries correctly [12].

4.2 Location Identification From Query Logs

If no QDL is found from the last step, then they mine query logs to get user IPs and clicked URLs which are used in their solution. They first map user IPs to user locations and treat the collection of user locations as a web document, so the location detection problem reduces to identifying location associated with the document [7, 1]. Here one issue is that there may be location names at different levels of hierarchy attached to a document from which a location at an appropriate level has to be found. If simply the most frequently occurring location name is returned as QDL [12], it would be wrong. They proposed following algorithm:

- **Step 1** They map all the extracted location names to a hierarchy, where location names are in different levels such as country, state and city.
- **Step 2** Here Wang et al. have used the concepts pf *power* and *spread* from [7] where power of a location node l is calculated as [12]:

$$Power(l) = f(l) + \sum_{l_i \in offspring(l)} f(l_i) \tag{3}$$

  where $f(l)$ refers to the frequency of l. Here they have adopted the entropy definition of spread which can attain best performance according to [7].

- **Step 3** After computing power and spread, the final locations are at nodes with appropriate levels of hierarchy so that both spread and power values of associated nodes are beyond given thresholds.

Here Wang et al. retrieved content of the pages at clicked URLs and merge them into one page. They extracted all location names from the page with Gazetteer, and then used the above algorithm to find the dominant location. Finally they combined the QDL found from clicked urls and QDL found from user ip by applying the following formula:

$$f(l) = \alpha f(l, QDL - log - URL) + (1 - \alpha) f(l, QDL - log - IP) \tag{4}$$

where $0 < \alpha < 1$, $f(l, QDL - log - URL)$ denotes the frequency of l in the clicked pages and $f(l, QDL - log - IP)$ denotes the corresponding frequency in the IP locations.

4.3 Location Identification From Search Results

If no query dominant location is found from either queries or query logs Wang et al. consider the search results. This is same as before except instead of query log here input is result snippets or result pages [12].

4.4 Location Sensitive Query Identification Using Click Logs

Vadrevu et al. [11] in their paper have presented approaches to identify localizable queries with the help of click logs. An enormous amount of information about users' behavior is found in click logs. Based on this information for each query, authors have constructed a set of regional sensitive features which includes:

- **Local switch rate (swr)** [11], which is defined as the user switch rate from whole web to country search for the same query in a query session log, and
- **Regional click rate (rcr)** [11], which is defined as the ratio of the number of regional results being clicked over the number of non-regional results being clicked for a given query.

They extracted the switching rate feature **swr**, by analysing each session as this feature precisely reflects the users' regional intent. For a given regional intended query, the users only make a switch if the top results returned does not satisfy users' information need. Hence, from the discussion so far, it is clear that this feature does not take all clicks into consideration, so coverage is very low. In comparison, other feature, **rcr**, takes all clicks available into consideration, so its coverage is better [11].

5 Temporal and Spatial Information Extraction From Web Documents

Tazuka et al. in their paper [10] have discussed about temporal and spatial attribute extraction from Web Documents. With the help of this extracted information they have developed a time-specific regional web search engine. In this search engine users can retrieve both spatially and temporally restricted information from the Web. This can be clarified with a simple example. Say a researcher has come to IIT Bombay to participate COLING conference and in the evening she wants to have dinner at some nearby restaurant of IIT Bombay. So by just typing "restaurant" as query she can get the information about nearby restaurants opened at that time.

5.1 Temporal Interval Extraction

Here in this survey we have dealt with simple timing expressions as listed below:

- From 9:00 a.m. to 11:00 p.m.
- 10 AM -11PM
- 10:00 hrs to 23:00 hrs
- 10 o'clock to midnight *etc.*

In their paper Tazuka et al. have extracted the timing information from web documents by constructing regular expression. In some cases a business hours of a place depend on the day of the week. Also, some places mention business hours and lunch hours separately. Even there are pages those specify morning and afternoon hours separately. If all these possibilities are taken into consideration, then the possible variations in which timings can appear becomes huge. They have used a pair that forms a 1-1 match which is a state where one address and one temporal interval are found in a specific location on a web page [10] because a page might contain this timing and address information for multiple entities.

5.2 Address Extraction

As we have told in the beginning of this report that we have given emphasis on extraction of various type of addresses from various types of web documents. First we start with discussing the address extraction algorithm as suggested by Tazuka et al. [10]. Unlike temporal interval extraction, they have not used any special regular expressions but extracted the required address by matching with addresses stored in a database and handled special cases with care.

Yu in his thesis [13] presented various approaches to extract structured addresses from web pages. He suggested rule based approaches based on regular expression and Gazetteer. Then he suggested a machine learning based approach. He built a classifier for each of the words in the parsed text of the web pages, that classifies the words in four classes, as whether it is start, middle or end of the address or not associated with the address at all. Thereby he has identified start, end boundary of the address as well as other words that belong to the address and thereby extracted structured addresses from web pages. In the end he has also suggested a hybrid approach combining these rule based and machine learning based approaches for extraction.

So far we have been discussing the timing and address information extraction from unstructured web pages which have no similar structure to each other. Now we shall discuss the scenario where such pages from which we want to extract information, share similar template with each other and the information embedded in these pages may be either structured or unstructured. In this context `wrapper` and `Xpath` comes to picture. In the following sections we discuss basic concepts of `wrapper` and `Xpath` and then discuss how to use them to extract information from a web site.

5.2.1 Wrapper

A wrapper [8] is a procedure for extracting tuples from particular information source. Formally a wrapper is a function from a page to the set of tuples it contains [8]. For explaining this we consider the following simple example [8]:

```
<html>
<title>some country codes</title>
<body>
<b>some country codes</b>
<b>congo</b><i>242</i><br>
<b>belize</b><i>501</i><br>
<hr><b><end></end></b>
</body>
```

```
</html>
```

Many kinds of wrappers could be written for extracting information. One kind of wrapper is that uses the positions of particular strings to delimit the extracted text. From this above example we can see that, country names are surrounded by $< \text{b} >$ and $< /\text{b} >$ and country codes are surrounded by $< \text{i} >$ and $< \text{i} >$. So one wrapper can depend on these four tag. This kind of wrapper is called LR wrapper [8]. The LR wrapper can find longest common strings that precede and follow each of the examples, if the given set of examples are labeled. This LR wrapper consists of this pair of strings. Thus, such wrapper obtains the node that consists of all the minimal strings that are delimited by these pairs of strings. While extracting multiple types, there is a pair of delimiters for each type.

But this simple LR strategy does not work always, for example, in the above example all occurrences of $< \text{b} > \ldots < /\text{b} >$ does not delimit a country. It is clear from the above example that, the string $< \text{p} >$ can be used to distinguish the head of the page and start of tuples. Similarly $< \text{hr} >$ separates the last tuple from the tail. By using this observation, an extension to this simple LR wrapper can be used in this case, which is called HLRT wrapper. HLRT wrapper is same as a LR wrapper with some additional features, *i.e.*, , it has strings H and T that limit the context under which LR wrapper can be applied. In this above example H and T are $< \text{p} >$ and $< \text{hr} >$ respectively. So formally for a HLRT wrapper for a domain with K attributes per tuple can be encoded as a vector of $2k+2$ strings [8].

5.2.2 Xpath

XPath [2] is a language for traversing DOM trees. A well-formed XML and HTML document can be represented by a DOM tree. A DOM tree is an ordered tree. Each node of a DOM tree is assigned a name and a value. Nodes of a DOM tree are divided in two classes: element nodes and text nodes. Text nodes may only occur as leaves of DOM tree. Names of all text nodes are same, *i.e.*, "#text". The node value of a text node is an arbitrary string. The name of an element node E can be chosen arbitrarily. The value of an element node is evaluated by concatenating the value of its text node descendants, if any. In addition, each element node is also associated to a set of attribute-value pairs. This concept can be clarified with the following simple example [2]:

```
<BODY>
<IMG SRC="filename.gif"/>
<H1>Playing tonight:</H1>
<P>
```

```
Star Wars
<HR/>
</P>
</BODY>
```

Following figure [2] shows the resulting DOM tree that is found after parsing the the above small web document. Here for sake of simplicity only nodes are shown and attribute, value pairs are omitted.

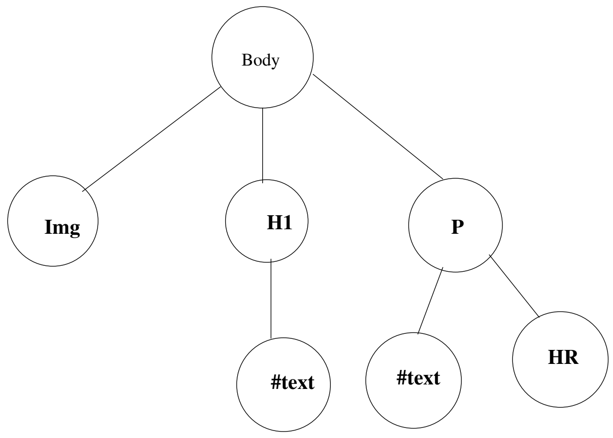


Figure 4: A sample DOM tree

XPath statement (or path, synonymously) defines a traversal through a DOM tree [2]. A path is a sequence of steps, separated by slashes. Each step has three parts: a search direction (*i.e.* called "axis" also), followed by "::", a node filter and a (possibly empty) sequence of predicates [2].

The direction of document traversal is described by the axis. This direction of traversal may be either to the parent, to the child, to following or preceding neighbors. Node filters restrict the set of nodes for which they will be applicable along the search direction. Finally, predicates are XPath expressions in square brackets. These expressions are used to further restrict the set of results. Examples of a few XPath statements [2] are as follows:

- **1.** /descendant-or-self::H1
- **2.**/child::BODY/child::H1
- **3.**/child::BODY/child::H1[following-sibling::P[child::HR]]
- **4.**/descendant-or-self::H1[preceding-sibling::IMG[@src='filename.gif']]/child::text()

Here each step in a path is evaluated on a set of `DOM` tree nodes. This evaluation results another set of `DOM` tree nodes. Such a path is evaluated by computing the first step on the initial set of `DOM` tree nodes and then using each step's result as the input of the following step in this path. For example, statement 1 evaluates to a set of nodes containing all $H1$ elements in the document. If those statements are applied to the sample DOM tree in Figure 1, statements 1,2 and 3 result in the set of nodes consisting of the single $H1$ element and statement 4 yields the text node inside the same $H1$ element.

In general, every step in an `XPath` statement is evaluated with respect to a set of input nodes as follows [2]:

- **1.** Output is an empty set of DOM tree nodes.
- **2.** For each node *N* in the set of input nodes:
  - Let *ResultN* be the sequence of nodes that is encountered if traverse the DOM tree in the current search direction relative to *N*
  - All elements that do not match the current node filter are removed from *ResultN*
  - Each element that does not satisfy any predicate is removed from *ResultN*
  - *ResultN* is added to Output
- **3.** Output is returned.

Node filters match those elements for which tag names corresponds to the value of the node filter. "*" is special node filter that matches all element nodes, but no text nodes, in contrast, "text()" matches all text nodes, but no element nodes. Node filter "node()", matches both element and text nodes.

If the node filter is a tag name, the axis and the double colon may be dropped [2]. In that case, the axis of the step is considered to be "child". Expressions inside predicates are computed in boolean context. Specifically, for non-empty result sets path expressions are true and in contrast arithmetic expressions and functions are true when they have a nonzero value. Now in the following section we shall discuss about the papers related to different approaches for wrapper induction for large scale web extraction.

5.2.3 Extraction Of Information From Large Websites

Several websites use common templates to generate HTML pages. The structural similarity of such template generated web pages is a powerful feature. Utilizing this feature information extraction systems can be made to use wrappers to effectively extract information from these pages. To clarify this concept, we consider a snippet of such a template generated web page as shown below:

```
<div class= 'restaurant links'>
<tr>
<td>
<u>Pizza Hut</u><br>
201, LBS Marg East<br>
Powai,Mumbai 400076
</td>
...
<tr>
<td>
<u>Sweet Bengal</u><br>
42, Hiranandani Gardens<br>
Galleria , Mumbai 400077
</td>
...
</tr>
</div>
```

If the DOM tree of above HTML document is considered, then with the following simple xpath all the restaurant names from this web page can be extracted:

//div[@class =' restaurant links']/tr/td/u/text()

According to this rule, all the dealer names can be found under u tag that is inside a td tag which is inside a tr tag, and this tr tag is within a div tag whose class is restaurant links. This rule above works on any web page generated by this form. Such a rule is called a wrapper. Here we discuss papers related to techniques for extracting data from such templatized HTML pages having large amounts of data, by the use of such automatically generated wrappers. Earlier wrappers were built manually for each Web sites separately. There are several problems with such manually coded wrappers: first of all, sometimes writing them is a very difficult and laborious task. second if the web site structure changes, then these manually coded wrappers are supposed to break. As a result manually constructed wrappers are difficult to maintain. Here the method of learning regular expression will not work also as they cannot be learned from positive examples alone, and it is very complex to learn them even in the presence of additional information [4]. This observation has limited the applicability of the traditional grammar inference techniques to extract data from large Websites and has motivated a several wrapper induction techniques for HTML pages [4].

Now we shall discuss wrapper generation systems [4, 5, 6]. First we shall start with ROAD-RUNNER [4]. It does not require user-specified examples, and any interaction with the user during the wrapper generation process. In other words wrappers are generated and data are extracted in a completely automatic way. This wrapper generator has no a priori knowledge about the page contents and page structure. This wrapper generation system starts off with the entire first input page as its initial template. Then, for each subsequent page it checks if the page can be generated by the current template. If it cannot be, it modifies its current template so that the modified template can generate all the pages seen so far. But there are several limitations to this approach [3]:

One limitation [3] is that, ROADRUNNER assumes that every HTML tag in the input pages is generated by the template. This assumption is crucial in ROADRUNNER because each time a new page appears, it needs to check if this new input page can be generated by the current template. This assumption is clearly invalid for pages in many web-sites since HTML tags can also occur within data values. This can be illustrated by following hypothetical scenario: if address is large, for better readability, one or more `br` or `p` tags are inserted in the address. Also sometimes a review in a site could contain tags inside: the review could be in several paragraphs, in which case it contains `p` tags, or some words in the review could be highlighted using `i` tags. When the input pages contain data values in such formats, ROAD RUNNER will either fail to discover any template, or produce a wrong template. Another limitation [3] of ROAD RUNNER is that, it assumes that the grammar of the template used to generate the pages is union-free, which is equivalent the assumption that there are no disjunctions in the input schema, ROADRUNNER might fail to produce any output if there are disjunctions in the input schema.

N. Dalvi et al. in their paper [5] on information extraction developed an approach based on probabilistic tree edit model to induce wrapper from web pages of similar structures. Their approach is supervised. They considered the fact that the page structure may change continuously in future. So the resulting wrapper is quite robust considering the fact that the wrapper can extract information even if page structure changes in future. In a later paper N. Dalvi et al. [6] presented a generic framework for making supervised wrapper induction noise-tolerant. We now discuss about main idea of this paper. Here given set of labels may be noisy. First all possible wrappers which can be generated by subsets of these labels are enumerated. Then each of the wrapper is ranked according to its quality. This quality is measured by how much the wrapper is likely to generate the (noisy) labels according to annotation model. The likelihood of being a good wrapper is guided by web publication model. Both annotation and web publication models have been discussed in detail in this paper. In short, web publication model is a model for generating a set of web pages in a given domain of interest [6]. Here a set H of HTML web pages are obtained by applying a rendering script over elements of the schema S picked over a set of types T. Annotation model is as follows: Let H be a set of web pages containing a set of text elements and a set of tags where each annotator labels a subset of these text elements. Here in this paper it is assumed that annotators are noisy, *i.e.*, annotator for a type t might miss some nodes of type t and might incorrectly label some nodes that are not of type t . Annotators are characterized by their precision p_t and recall r_t.

All these wrappers and Xpath based approaches can work properly only when the information embedded in the page has at least some structure so that system that annotates them, works at some satisfactory level. But in case of Indian addresses which have no structure as such, no automatic annotation system is possible, so only option is to manually label the pages of each of the site that contains this information, which is a huge task, when there will be hundreds of such sites and each sites has thousands of such pages.

Ma et al. in their paper [9] have shown an approach for extracting unstructured data from template generated web documents. Their approach is based on maintaining a map of text chunks that keeps track of document frequency of text chunk between each two tags of all web pages in the whole collection. Text chunks with document frequency above a particular threshold are identified as part of the template and the part of the text that remains after the process of template identification, is the required extracted text. But this approach is not suitable in case of unstructured address extraction from such template generated pages because a part of the address may be identified as a part of the template and in such case the desired address may not be recovered properly. This can be illustrated with a simple example as follows: say, we are extracting addresses from web pages, all of which are related to restaurants and hotels at `Kolkata`. Here it is very likely that `Kolkata` is a part of the address in most of the documents. So with this approach `Kolkata` is identified as part of the template and as a result addresses do not get recovered properly.

References

[1] Einat Amitay, Nadav Har'El, Ron Sivan, and Aya Soffer. Web-a-where: geotagging web content. In *Proceedings of the 27th annual international ACM SIGIR conference on Re-*

search and development in information retrieval, SIGIR '04, pages 273–280, New York, NY, USA, 2004. ACM.

[2] Tobias Anton. Xpath-wrapper induction by generalizing tree traversal patterns. 2005.

[3] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 337–348, New York, NY, USA, 2003. ACM.

[4] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[5] Nilesh Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 335–348, New York, NY, USA, 2009. ACM.

[6] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proc. VLDB Endow.*, 4(4):219–230, January 2011.

[7] Junyan Ding, Luis Gravano, and Narayanan Shivakumar. Computing geographical scopes of web resources. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 545–556, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[8] Nicholas Kushmerick. *Wrapper induction for information extraction*. PhD thesis, 1997. AAI9819266.

[9] Ling Ma, Nazli Goharian, Abdur Chowdhury, and Misun Chung. Extracting unstructured data from template generated web documents. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 512–515, New York, NY, USA, 2003. ACM.

[10] Taro Tezuka and Katsumi Tanaka. Temporal and spatial attribute extraction from web documents and time-specific regional web search system. In *Proceedings of the 4th international conference on Web and Wireless Geographical Information Systems*, W2GIS'04, pages 14–25, Berlin, Heidelberg, 2005. Springer-Verlag.

[11] Srinivas Vadrevu, Ya Zhang, Belle Tseng, Gordon Sun, and Xin Li. Identifying regional sensitive queries in web search. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1185–1186, New York, NY, USA, 2008. ACM.

[12] Lee Wang, Chuang Wang, Xing Xie, Josh Forman, Yansheng Lu, Wei-Ying Ma, and Ying Li. Detecting dominant locations from search queries. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 424–431, New York, NY, USA, 2005. ACM.

[13] Zheyuan Yu. High accuracy postal address extraction from Web pages. Master's thesis, Dalhousie University, Halifax, Nova Scotia, March 2007.