ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

Enhanced Information Retrieval Over Warehouse

Mr.S.Gopalakrishnan[1], Mr.S.Deepankumar[2]

Assistant Professor, Department of CT, Sri Krishna Arts and Science College, Coimbatore[1]

Assistant professor, Department of CSA, Sri Krishna Arts and Science College, Coimbatore[2]

Abstract: Retrieving information is an active research area over the years. The main focus is to improve the accuracy of the retrieval systems, and it is seen as one time execution process. It is inadequate for real-world applications when this is seen as long running processes. One of the drawbacks of retrieval is that it needs to be started from the scratch to entire warehouse when there is an interruption. To overcome this Parse Tree Query Language is high level extraction query language is used that enables information retrieval over parse trees. Parse Tree Query Language is an extension of the linguistic query language and intermediate output of each text processing component is stored so that only the improved component has to be deployed to the entire warehouse. Retrieval is performed on both the previously processed data from the unchanged components as well as the updated data generated by the improved component. Performing such kind of enhanced retrieval can result in a huge reduction of processing time and provides quality to the retrieval.

Keywords: Text mining, query languages, Information Retrieval

1. INTRODUCTION

It is estimated that each year more than 600,000 articles are published in the biomedical literature, with close to 20 million publication entries being stored in the Medline database. To uncover information from such a large corpus of documents, it is vital to address the information needs in an automated manner. The field of information Retrieval (IR) seeks to develop methods for fetching structured information from natural language text. IR is typically seen as a one-time process for the extraction of a particular kind of relationships of interest from a document collection. IR is usually deployed as a pipeline of special-purpose programs, which include sentence splitters, tokenizers, named entity recognizers, shallow or deep syntactic parsers, and extraction based on collection of patterns. The high demand of IR in various domains results in the development of frameworks such as UIMA [1] and GATE [2], providing a way to perform extraction by defining workflows of components. This type of retrieval frameworks is usually file based and the processed data can be utilized between components. In this traditional setting, relational databases are typically not involved in the extraction process, but are only used for storing the extracted relationships. While file-based frameworks are suitable for one-time extraction, it is important to notice that there are cases when IR has to be performed repeatedly even on the same document collection.

To realize this new information extraction framework, we propose to choose database management systems over file-based storage systems to address the dynamic retrieval needs. Our proposed information retrieval is composed of two phases:

- **Initial Phase:** We perform a one-time parse, entity recognition, & tagging (identifying individual entries as belonging to a class of interest) on the whole corpus based on the current knowledge. (The generated syntactic parse trees and semantic entity tagging of the processed text is stored in a relational database, called parse tree database (PTDB)).

To express extraction patterns, we designed and implemented a query language called parse tree query language (PTQL) that is suitable only for generic extraction. Note that in the event of a change to the extraction goals (e.g., the user becomes interested in new types of relations between entities) or a change to an extraction module (e.g., an improved component for named entity recognition becomes available), the responsible module is deployed for the entire text corpus and the processed data are populated into the PTDB. Queries are issued to identify the sentences with newly recognized mentions. Then extraction can be performed only on such affected sentences rather than the entire corpus. Thus, we achieve incremental extraction, which avoids the need to reprocess the entire collection of text unlike the file-based pipeline approaches.

Using database queries instead of writing individual special-purpose programs, information retrieval becomes generic for diverse applications and becomes easier for the user. However, writing such queries may still require many users effort. To further reduce users' learning burden, we propose algorithms that can automatically generate PTQL queries from training data or a user's keyword queries.

We highlight the contributions of this paper.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

- Novel Database-Centric Framework for Information Extraction. Unlike the traditional approaches, where IR is achieved by special-purpose programs and databases are only used for storing the extraction results, we propose to store intermediate text processing output in a database, parse tree database. This approach minimizes the need of reprocessing the entire collection of text in the presence of new extraction goals and deployment of improved processing components. -Query Language for Information Retrieval. Information Retrieval is expressed as queries on the parse tree database. As query languages such as XPath and XQuery are not suitable for extracting linguistic patterns [6], we designed and implemented a query language called parse tree query language, which allows a user to define extraction patterns on grammatical structures such as constituent trees and linkages. Since extraction is specified as queries, a user no longer needs to write and run special- purpose programs for each specific extraction goal.
-Automated Query Generation. Learning the query language and manually writing extraction queries could still be a time-consuming and labor-intensive process. Moreover, such an ad hoc approach is likely to cause unsatisfactory extraction quality. To further reduce a user's effort to perform information extraction, we design two algorithms to automatically generate extraction queries, in the presence and in the absence of training data, respectively.

2.1 Information Retrieval

IR has been an active research area that seeks techniques to uncover information from a large collection of text. Examples of common IR tasks include the identification of entities (such as protein names), extraction of relationships between entities (such as interactions between a pair of proteins) and extraction of entity attributes (such as co reference resolution that identifies variants of mentions corresponding to the same entity) from text. Readers are referred to [7] for a detailed survey of IR.
The examples and experiments used in our paper involve the use of grammatical structures for relationship extraction. A co-occurrence of entities is a typical method in relationship extraction, but often leads to imprecise results. Consider that our goal is to extract relations between drug and proteins from the following sentence:
Quetiapine is metabolized by CYP3A4 and sertindole by CYP2D6. (PMID: 10422890)
By utilizing our grammatical knowledge, a human reader can observe that CYP3A4, metabolise, quetiapine and CYP2D6, metabolise, sertindole are the only correct triplet relations for the above sentence. This simple example highlights the need of [; grammatical knowledge in relationship retrieval.
A typical IR setting involves a pipeline of text processing modules in order to perform relationship retrieval. These include

- Sentence splitting: identifies sentences from a Para graph of text,
- Tokenization: identifies word tokens from sentences,
- Named entity recognition: identifies mentions of entity types of interest,
- Syntactic parsing: identifies grammatical structures of sentences,
- Pattern matching: obtains relationships based on a set of extraction patterns that utilize lexical, syntactic, and semantic features.

Retrieval patterns are typically obtained through manually written patterns compiled by experts or automatically generated patterns based on training data. Different kinds of parsers, which include shallow and deep parsers, can be utilized in the pipeline. In our work, the Link Grammar parser [8] is utilized as part of our extraction approach. We describe the basic terminologies involved in Link Grammar in the next section.

2.2 Link G r a m m a r

The Link Grammar parser is a dependency parser based on the Link Grammar theory [8]. Link Grammar consists of a set of words and linking requirements between words. A sentence of the language is defined as a sequence of words such that the links connecting the words satisfy the following properties: 1) the links do not cross, 2) the words form a connected graph, and 3) the links satisfy the linking requirements of each word in the sentence. The output of the parser, called a linkage, shows the dependencies between pairs of words in the sentence. Fig. 1 shows an example for the sentence "RAD53, which activates DNA damage, positively regulates the DBF4 protein" (PMID: 10049915). The linkage contains several links, which include link S connecting the subject-noun RAD53 to the transitive verb regulates, the O link connecting the transitive verb regulates to the direct object DBF4 and the MX*r link connecting the relative pronoun which to the noun RAD53. For a complete description of links, we refer the reader to [3].Besides producing linkages, the Link Grammar parser is also capable of outputting constituent trees. A constituent tree is a syntactic tree of a sentence with the nodes represented by part-of-speech tags and words of the sentences in the leaf nodes. For instance, the corresponding constituent tree for the above sentence is illustrated in Fig. 1. In the constituent tree, S stands for a sentence/clause, SBAR for a clause introduced by a subordinating conjunction, WHNP for a clause containing a relative pronoun, NP for a noun phrase, VP for l verb phrase, and ADVP for an adverb phrase. The leaf nodes of the constituent tree represent the words of the sentence and their part-of-speech tags. For words that are not recognizable by the parser, the tag U is given for such words for unknown part-of-speeches.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

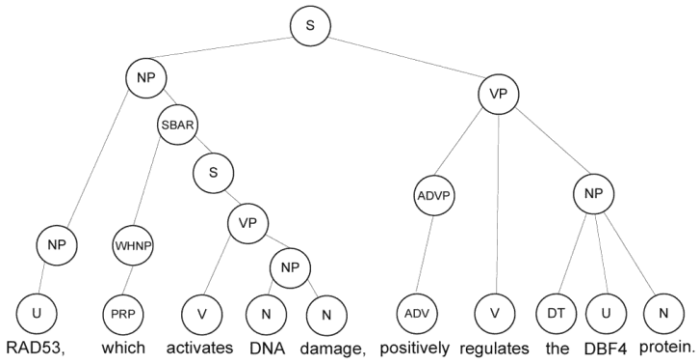

Fig.1 Constituent tree of the sentence "RAD53, which activates DNA damage, positively regulates the DBF4 protein"

3. SYSTEM

We first give an overview of our approach, and discuss each of the major components of our system in this section. Our approach is composed of two phases: initial phase for processing of text and extraction phase for using database queries to perform extraction. The Text Processor in the initial phase is responsible for corpus processing and storage of the processed information in the Parse Tree Database (PTDB). The retrieval patterns over parse trees can be expressed in our proposed parse tree query language. In the extraction phase, the PTQL query evaluator takes a PTQL query and transforms it into keyword-based queries and SQL queries, which are evaluated by the underlying RDBMS and information retrieval (IR) engine. To speed up query evaluation, the index builder creates an inverted index for the indexing of sentences according to words and the corresponding entity types. Fig. 2 illustrate the system architecture of our approach.

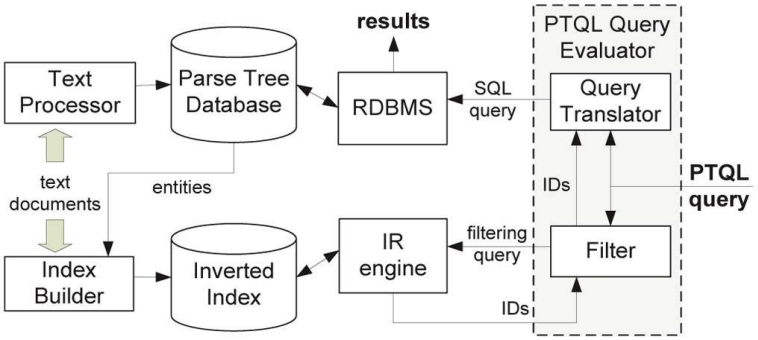

Fig.2.System architecture of the PTQL framework

document collection with information drawn from a problem-specific database. This step necessitates a method for precise recognition and normalization of protein mentions. From this labeled data, initial phrases referring to interactions are extracted. These phrases are then refined to compute consensus patterns and the resulting PTQL queries are generated by the query generator. However, training data are not always readily available for certain relationships due to the inherent cost of creating a training corpus. In that regards, our approach provides the pseudo-relevance feedback driven approach that takes keyword- based queries, and the PTQL query generator then finds common grammatical patterns among the top-k retrieved sentences to generate PTQL queries. We first describe the parse tree database and the syntax of PTQL before we provide details of how PTQL queries are processed. We describe retrieval algebra and demonstrate the effectiveness of our techniques in providing orders of magnitude reduction in the running time of complex retrieval tasks.

3.1 Parse Tree Database and Inverted Index

The Text Processor parses Medline abstracts with the Link Grammar parser [3], and identifies entities in the sentences using BANNER [9] to recognize gene/protein names and MetaMap [10] to recognize other entity types that include disease and drug names. Each document is represented as a hierarchical representation called the parse tree of a document, and the parse trees of all documents in the document collection constitute the parse tree database. A parse tree is composed of a constituent tree and a linkage. A constituent tree is a syntactic tree of a sentence with the nodes represented by part-of-speech tags and leafs corresponding to words in the sentence. A linkage, on the other hand, represents the syntactic dependencies (or links) between pairs of words in a sentence. Each node in the parse tree has labels and attributes capturing the document structure (such as title, sections, sentences) part-of-speech tags, and entity types of corresponding words.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

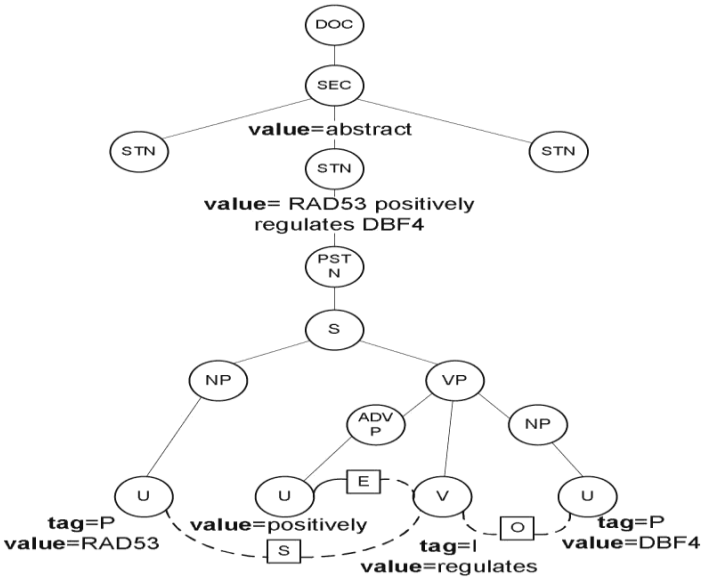

Fig. 3. An example of a parse tree for a document, which includes sections of the document, sentences, and the corresponding parse trees.

Fig. 3 shows an example of a parse tree for a Medline abstract. The parse tree contains the root node labeled as DOC and each node represents an element in the document which can be a section (SEC), a sentence (STN), or a parse tree for a sentence (PSTN). A node labeled as STN may have more than one child labeled with PSTN to allow the storage of multiple parse trees. The node below the PSTN node indicates the start of the parse tree, which includes the constituent tree and linkage of the sentence. A solid line represents a parent-child relationship between two nodes in the constituent tree, whereas a dotted line represents a link between two words of the sentence. In the constituent tree, nodes S, NP, VP, and ADVP stand for a sentence, a noun phrase, a verb phrase, and an adverb phrase, respectively. The linkage contains three different links: the S link connects the subject-noun RAD53 to the transitive verb regulates, the O link connects the transitive verb regulates to the direct object DBF4 and the E link connects the verb-modifying adverb positively to the verb regulates. The square box on a dotted line indicates the link type between two words. Each leaf node in a parse tree has value and tag attributes. The value attribute stores the text representation of a node, while the tag attribute indicates the entity type of a leaf node. For instance, a protein is marked with a tag P, a drug name with a tag D, and an interaction word is marked with I.....

3.2 Parse Tree Query Language

A fundamental design criterion for the query language is the ability of expressing linguistic patterns based on constituent trees. Standard XML query languages such as XPath and XQuery seem to be the ideal candidates for querying parse trees. However, the inability of expressing immediate- following siblings and immediate-preceding siblings in these standard XML query languages, leads to the development of LPath as a query language for linguistic queries on constituent trees. An additional design criteria for the query language is the ability to express linguistic patterns based on dependency grammar, such as Link Grammar [8]. Links and link types can be useful in linguistic patterns, such as the type MXsr connects a relative pronoun to its corresponding noun. However, languages such as XQuery and LPath can only express ancestor- descendant and sibling relations between nodes. One of the novel features of our proposed query language PTQL is the ability to express links and link types between pairs of nodes, so that PTQL can be used to express linguistic patterns based on constituent trees and links, as well as link types. We propose a high level extraction query language called PTQL. PTQL is an extension of the linguistic query language LPath that allows queries to be performed not only on the constituent trees but also the syntactic links between words on linkages. A PTQL query is made up of four components:

1. tree patterns,
2. link conditions,
3. proximity conditions, and
4. return expression.

A tree pattern describes the hierarchical structure and the horizontal order between the nodes of the parse tree. A link condition describes the linking requirements between nodes, while a proximity condition is to find words that are within a specified number of words. A return expression defines what to return. The EBNF grammar for PTQL.

3.3 Query Evaluation

Our approach for the evaluation of PTQL queries involves the use of IR engine as well as RDBMS. The role of the IR engine in query is to select sentences based on the lexical features defined in PTQL queries, and only the subset

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

of sentences retrieved by the IR engine are considered for the evaluation of the conditions specified in the PTQL queries by RDBMS. Our approach does not discard sentences that should otherwise be included for extraction. Using sample query Q1 as an example, the lexical features defined in the query imply that only sentences with at least one gene name together with the keyword "regulates" should be considered for extraction. We summarize the process of the evaluation of PTQL queries as follows.

1. Translate the PTQL query into a filtering query.
2. Use the filtering query to retrieve relevant documents D and the corresponding sentences S from the inverted index.
3. Translate the PTQL query into an SQL query and instantiate the query with document id d 2 D and sentence id s 2 S.
4. Query PTDB using the SQL query generated in Step 3.
5. Return the results of the SQL query as the results of the PTQL query.

In step 2, the process of finding relevant sentences with respect to the given PTQL query requires the translation of the PTQL query into the corresponding filtering query. Query q is translated into a keyword-based filtering query using the following steps:

1. Generate query terms for each of the node expressions that are in the tree pattern of q.
2. Form phrases if consecutive node expressions are connected by "immediate following" horizontal axes (i.e., "->").
3. Form phrases followed by the proximity operator if the corresponding nodes are defined in the proximity condition of q.

4. EXPERIMENTAL EVALUATION

We first illustrate the performance of our approach in terms of query evaluation and the time savings achieved through incremental extraction. Then we evaluate the extraction performance for our two approaches in query generation.

4.1 Time Performance for PTQL

We performed experiments in finding the time performance of the evaluation of PTQL queries, as well as experiments to illustrate the amount of time saved in the event of change of an extraction goal and deployment of an improved module. All experiments were performed using a 2.2-GHz Intel Xeon QuadCore CPU running in Red Hat Linux. Only a single process was used to perform the experiments. The parse tree database is stored as a relational database managed by MySQL.

4.1.1 Query Evaluation

A set of 25 PTQL queries that involves in the extraction of drug-gene metabolic relations was applied to a large corpus of 17 million abstracts to measure the time performance for PTQL evaluation. Specifically, given a drug, the goal is to find which genes are involved in the metabolic relations with the drug. In our experiments, we specified a single drug ("1-drug"), a set of 5 drugs ("5-drugs") and a set of 10 drugs ("10-drugs") in each of the 25 PTQL queries. The keyword "metabolized" is involved in queries Q1-Q7, while the keywords "metabolism" and "substrate" are involved in queries Q8-Q20 and Q21-Q25. In the experiments, each query was evaluated with five different sets of drugs and repeated for five runs. The time performance indicates that our proposed framework is acceptable for real-time IR. We observed that queries specified with a larger set of drugs require a longer time to complete the evaluation. In addition, the evaluation on the set of queries with the keyword "metabolism" (Q8-Q20) generally takes longer than the other sets of queries to complete. This is due to the fact that a higher number of sentences matches with the keyword "metabolism." The complexity of the This experiment showed a tremendous decrease of 89.64 percent when a new module is deployed for text processing as compared to the pipeline approach. Such significant reduction of time is largely due to the fact that the Link Grammar parser is not utilized in the reprocessing, which can take an PTQL queries also plays a role in the amount of time it takes for evaluation.

4.1.2 Change of Modules

We performed experiments to show the time savings for our incremental extraction approach. The scenario behind our experiment is that the initial goal is to perform extraction for drug information from a text collection. The extraction goal is then changed into the extraction of drug- protein relations that requires the deployment of a gene named entity recognizing to identify gene mentions in the text collection. To illustrate the amount of time savings, a collection of 13 K Medline abstracts was initially processed with the Link Grammar parser and a dictionary-based tagger for drug names. This process took about 62.38 hours. We then deployed a statistical-based tagger for gene names to process the corpus. With the pipeline approach, the whole process had to be started from scratch by running the Link Grammar parser, the drug name tagger and the newly deployed gene name tagger. This took another 64.8 hours to complete. With our approach, the intermediate processing data produced by the Link Grammar parser and the drug name tagger were populated into the parse tree database. The gene name tagger was then deployed to process

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

the corpus and SQL insert statements were issued to update the parse tree database. This process took only 6.71 hours to complete. Average of 7.6 seconds to parse a sentence when the timeout limit was set as 15 seconds for the parser. The reduction of time comes at the expense of disk space. The processed data for the 13 K Medline abstracts results in a disk usage of about 110 MB.

4.2 Data and Execution Model

Since our algebra is designed to extract annotations from a single document at a time, we define its semantics in terms of the current document being analyzed. The current document is modeled as a string called doctext .Our algebra operates over a simple relational data model with three data types: span, tuple, and relation. A span is an ordered pair (begin, end) that denotes the region of doctext from position begin to position end. A tuple is a finite sequence of w spans (s1, ..., sw); we call w the width of the tuple. A relation is a multiset of tuples with the constraint that every tuple must be of the same width. Each operator in our algebra takes zero or more relations as input and produces a single relation as output.

4.3 Algebra Operators

Based on their functionality, the set of operators in our algebra fall into the following two categories

Relational Operators: Since our data model is a minimal extension to the relational model, all of the standard relational operators (select, project, join, etc.) apply without any change. The main addition is that we use a few new selection predicates applicable only to spans.

Span Extraction Operators: A span extraction operator identifies segments of text that match a particular input pattern and produces spans corresponding to each such text segment.

5. DISCUSSION AND FUTURE WORK

In this section, we discuss the main contributions of our work as well as their limitations. Existing retrieval frameworks do not provide the capabilities of managing inter- mediate processed data such as parse trees and semantic information. This leads to the need of reprocessing of the entire text collection, which can be computationally expensive. On the other hand, by storing the intermediate processed data as in our novel framework, introducing new knowledge can be issued with simple SQL insert statements on top of the processed data. With the use of parse trees, our framework is most suitable for performing extraction on text corpus written in natural sentences such as the biomedical literature. As indicated in our experiments, our increment extraction approach saves much more time compared to performing extraction by first processing each sentence one-at-a-time with linguistic parsers and then other components. This comes at the cost of overheads such as the storage of the parse trees and the semantic information, which takes up 1.5 TB of space for 17 million abstracts for the parse tree database. In the case when the parser fails to generate parse tree for a sentence, our system generates a "replacement parse tree" that has the node STN as the root with the words in the sentence as the children of the root node. This allows PTQL queries to be applied to sentences that are incomplete or casually written, which can appear frequently in web documents. Features such as horizontal axis and proximity conditions can be most useful for performing extraction on replacement parse trees. For future work, we will extend the support of other parsers by providing wrappers of other dependency parsers and scheme, such as Pro3Gres and the Stanford Dependency scheme, so that they can be stored in PTDB and queried using PTQL. We will expand the capabilities of PTQL, such as the support of regular expression and the utilization of redundancy to compute confidence of the extracted information.

REFERENCES

[1]. D. Ferrucci and A. Lally, "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment," Natural Language Eng., vol. 10, nos. 3/4, pp. 327-348, 2004.
[2]. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications," Proc. 40th Ann. Meeting of the ACL, 2002.
[3]. D. Grinberg, J. Lafferty and D. Sleator, "A Robust Parsing Algorithm For link Grammars," Technical Report CMU-CS-TR- 95-125, Carnegie Mellon Univ. 1995.
[4]. F. Chen, A. Doan, J. Yang and R. Ramakrishnan, "Efficient Information Extraction over Evolving Text Data," Proc IEEE 24th Int'l Conf. Data Eng. (ICDE '08), pp. 943-952, 2008.
[5]. F. Chen, B. Gao, A. Doan, J. Yang and R. Ramakrishnan, "Optimizing Complex Extraction Programs over Evolving Text Data," Proc 35th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09), pp. 321-334, 2009.
[6]. S. Bird et al., "Designing and Evaluating an XPath Dialect for Linguistic Queries," Proc 22nd Int'l Conf. Data Eng. (ICDE '06),2006.
[7]. Sarawagi, "Information Extraction," Foundations and Trends in Databases, vol. 1, no. 3, pp. 261-377, 2008.
[8]. D.D. Sleator and D. Temperley, "Parsing English with a Link Grammar," Proc Third Int'l Workshop Parsing Technologies, 1993.
[9]. R.Leaman and G. Gonzalez,"BANNER:An Executable Survey of Advances in Biomedical Named Entity Recognition," Proc. Pacific Symp. Biocomputing, pp. 652-663, 2008.
[10]. [10A.R. Aronson, "Effective Mapping of Biomedical Text to the UMLS Metathesaurus: The MetaMap Program," Proc. AMIA Symp., p. 17, 2001.
[11]. M.J. Cafarella and O. Etzioni, "A Search Engine for Natural Language Applications," Proc. 14th Int'l Conf. World Wide Web (WWW '05), 2005.