A Novel Incremental Information Extraction Using Parse Tree Query Language And Parse Tree Databases

Rajula Srilatha (M.Tech)
CSE Dept, CMRTC, Hyderabad,

K. Murali, M.Tech.
Assistant Professor, CSE Dept,
CMRTC, Hyderabad,

Abstract: — Mining is nothing but retrieving the information from various resources .We have different approaches to retrieve these information one of them is traditional pipeline approach. As of increasing technologies it became more complicated to workout with these traditional approach the main drawback in these pipeline approach is if any modifications are done or any module is developed newly then we have to reapply the extraction .So we are developing the different approach for data mining in this paper is through database queries . These are optimized by databases that make this as efficient approach.

Index Terms: Text mining, query languages, information storage and retrieval.

I. Introduction

In this paper, we propose an effective and adjustable optimization of queries is critical in database management systems and the complexity involved in finding optimal solutions has led to the development of heuristic approaches. Answering data mining query involves a random search over large databases. Due to the enormity of the data set involved, model Simplification is necessary for quick answering of data mining queries. In this paper, we propose a hybrid model using rough sets and genetic algorithms for fast and efficient query answering. Rough sets are used to classify and summarize the datasets, whereas genetic algorithms are used for answering association related queries and feedback for adaptive classification. Here, we consider three types of queries, i.e., select, aggregate and classification based data mining queries. The field of information extraction (IE) seeks to develop methods for fetching structured information from natural language text. Examples of structured information are the extraction of entities and relationships between entities. IE is typically seen as a one-time process for the extraction of a particular kind of relationships of interest from a document collection. IE is usually deployed as a pipeline of special-purpose programs, which include sentence splitters, tokenizes, named entity recognizers, shallow or deep syntactic parsers, and extraction based on a collection of the development of frameworks such as UIMA and GATE , providing a way to perform extraction by defining workflows of components. This type of extraction 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 IE has to be performed repeatedly even on the same document collection. Consider a scenario where a named entity recognition component is deployed with an updated ontology or an improved model based on statistical learning. Typical extraction frameworks would require the reprocessing of the entire corpus with the improve identity recognition component as well as the other unchanged text processing components. Such reprocessing can be computationally intensive and should be minimized. For instance, a full processing for information extraction on 17 million Medline abstracts took more than 36 K hours of CPU time using a Single-coreCPUwith2-GHzand2GBofRAM.2 Work by, addresses the needs for efficient extraction of evolving text such as the frequent content updates of web documents but such approaches do not handle the issue of changed extraction components or goals over static text data. In this paper, we propose a new paradigm for information extraction. In this extraction framework, intermediate output of each text processing component is stored so that only the improved component has to be deployed to the entire corpus. Extraction

is then 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 incremental extraction can result in a tremendous reduction of processing time. To realize this new information extraction framework, we propose to choose database management systems over file-based storage systems to address the dynamic extraction needs.

II. System study

Our proposed information extraction is composed of two phases:

Initial Phase We perform a one-time parse, entity recognition, and 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).

Extraction Phase Extraction is then achieved by issuing database queries to PTDB. To express extraction patterns, we designed and implemented a query language called parse tree query language (PTQL) that is suitable 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 extraction 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.

Novel Database Centric Framework for Information Extraction. Unlike the traditional approaches, where IE 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 Extraction. Information extraction 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.

Information Extraction IE has been an active research area that seeks techniques to uncover information from a large collection of text. Examples of common IE 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. The examples and experiments used in our paper involve the use of grammatical structures for relationship extraction. Co occurrences of entities are a typical method in relationship extraction, but often lead 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 hCYP3A4, metabolize, quetiapinei and hCYP2D6, metabolize, sertindolei are the only correct triplet relations for the above sentence. However, if we consider co occurrences of entities as a criteria to extract relationships, incorrect relationships such as hCYP3A4, metabolize, sertindolei and hCYP2D6, metabolize, quetiapinei would also be extracted from the above sentence.

III. System evalution

Sentence Splitting: In the first module the documents contain sentences. The sentences are in the unstructured manner. The module converts sentences to structured sentences with index. This process is applied on the existing corpus.

Word Indexing: In this module each sentence of a document is made up with different words.

Example: S1= {w1, w2, w3…….wn}

The module splits all the indexed sentences by words.

Word Tagging: In this module, the words will be presented in the document in different forms such as present, past, future etc…The words has to be n-grammed to find out the possible equivalence of root words. The root words can be grouped together (or) clustered for special group of interests.

Example: {“cricket”, “football”} can be grouped together to special interests called “sports” category. Identifying group of words of similar category can have relationship. Building the relational words together is called word-net.

Parse Tree Database (PTDB) Construction: The word-net is a semantic relational network. The word-net is store in the database as PTDB. The module provides an interface to the user to search the PTDB of the corpus. The user’s query will be in the form of natural language (or) can be with stop words.

Execution Phase:

- The module provides as efficient way to query the PTDB
- The module provides an interface to the user to search the PTDB of the corpus.
- The user’s query will be in the form of natural language (or) can be with stop words.

User’s Query Preprocessing: In this module, user’s query has to be preprocessed against stop words elimination. The query words have to be n-grammed for possible root words.

Query Word Tagging (PTQL): In this module, all the n-grammed words may not be the root words. Find out the possible root words for each query word. Find the semantically words for each word of query root word. Find the appropriate

Tag with their relevancies (or) Frequencies.

IV. Related work

The main focus so far has been on improving the accuracy and runtime of information extractors. But recent work has also started to consider how to manage such extractors in large-scale IE-centric applications. While we have focused on IE over unstructured text, our work is related to wrapper construction, the problem of inferring a set of rules (encoded as a wrapper) to extract information from template-based Web pages. Since wrappers can be viewed as extractors (as defined in Section 3), our techniques can potentially also apply to wrapper contexts. In this context, the knowledge of page templates may help us develop even more efficient IE algorithms. Our work is also related to the problem of wrapper maintenance over evolving Web data. The problem of finding overlapping text regions is related to detecting duplicated Web pages. Many algorithms have been developed in this area. But when applied to our context they do not guarantee to find all largest possible overlapping regions, in contrast to the suffix-tree based algorithm developed in this work. Once we have extracted entity mentions, we can perform additional analysis, such as mention disambiguation. Thus, such analyses are higher level and orthogonal to our current work.

Numerous rule-based extractors and learning-based extractors have been developed. Delex can handle both types of extractors Much work has tried to improve the accuracy and runtime of these extractors. But recent work has also considered how to combine and manage such extractors in large-scale IE applications. Our work fits into this emerging direction. In terms of IE over evolving text data, Cyclex is the closest work to ours. But Cyclex is limited in that it considers only IE programs that contain a Single IE black box, as we have discussed. Also considers evolving text data, but in different problem contexts. They focus on how to incrementally update an inverted index, as the indexed Web pages change. Recent work has also exploited overlapping text data, but again in different problem contexts. These works observe that document collections often contain overlapping text. They then consider how to exploit such overlap to "compress" the inverted indexes over these documents, and Compile Project. Two pages p and q of the same URL, retrieved at different times. A matcher has found that regions u1 and u2 of page p match regions v1 and v2 of page q, respectively how to answer queries efficiently over such compressed indexes. In contrast, we exploit the IE results over the overlapping text regions to reduce the overall extraction time.

The Cyclex Solution Approach

To describe Cyclex, we begin with two notions:

A region r *in a data page* p *of snapshot* Pn+1 *is an old region if it occurs in a page* q *of snapshot* Pn. r *is a maximally old region if it cannot be extended on either Side and still remains an old region.*

To extract mentions fromPn+1, Cyclex then considers each page p in Pn+1 and "matches", i.e., compares p with pages in Pn, to find old regions of p. It then applies extractor E only to the new regions of p, and copies over the mentions of the old regions. Since pages retrieved (in consecutive snapshots) from the same URL often share much overlapping data, to find old regions of p, Cyclex currently matches p only with q, the page in Pn that shares the same URL with p. (If q does not exist, then Cyclex declares that p has no old regions.) Section 8 shows that the choice of matching pages with the same URL already significantly reduces IE time. Considering more complex choices (e.g., matching p with all pages in Pn) is an ongoing research. We call algorithms that match p and q to find old regions in p *page matchers*. Sections 5 shows that such matchers span an entire spectrum, trading off result completeness for runtime, and that no matcher is always optimal. For example, the ST matcher described below returns all maximally old regions, thus providing the most opportunities for recycling past IE results. But it may also incur more runtime than matchers that return only some old regions. So, a priori we do not know if it would be better than these other matchers.

The Page Matchers

Recall from Section 4 that a page matcher compares pages p and q to find old regions of p. We have provided the current Cyclex with three page matchers: DN, UD, and ST (more matchers can be easily plugged in as they become available). DN incurs zero runtime, as it immediately declares that page p has no old region. Cyclex with DN thus is equivalent to applying IE from scratch to Pn+1. UD employs a Unix-diff-command like algorithm [11], which splits pages p and q into lines, then employs a heuristic to find common lines. Thus, UD is relatively fast (takes time linear in |p| + |q|), but finds only some old regions. We omit further description for space reason, but refer the reader to [11]. ST is a novel suffix-tree based matcher that we have developed, which finds *all maximal old regions* of p uSing time linear in |p| + |q|. ST and DN thus represent the two ends of a spectrum of matchers that trade off the result completeness for runtime efficiency, while UD represents an intermediate point on this spectrum. In the rest of this section we describe ST in detail. Roughly speaking, ST inserts all suffixes of q and p into one suffix tree T [7]. As we insert each suffix of p, T helps us identify the longest prefix of this suffix that also appears in q. To realize this intuition, however, we must handle a number of intricacies, so that we can locate all maximal old regions without slowing down ST to quadratic time.

Suffix Tree Basics

The suffix tree for a string q is a tree T with |q| leaves, each describing a suffix of q. T must satisfy the followings: (1) each non-root internal node has at least two children. (2) Each edge is labeled with a nonempty substring of q, and no two edges out of a node can have labels beginning with the same character. (3) The *path label* of a node is the concatenation of all edge labels on the path from the root to this node; each suffix of q corresponds to the path label of a leaf. (4) Each non-root internal node with path label _u (where _ is a Single character and u is a string) has *a suffix link* to the node with path label u; the root has a suffix link to itself. Figure 3(a) shows the suffix tree for "ababbabaab$," where symbol $ terminates the string. Suffix links are showed as dotted lines.

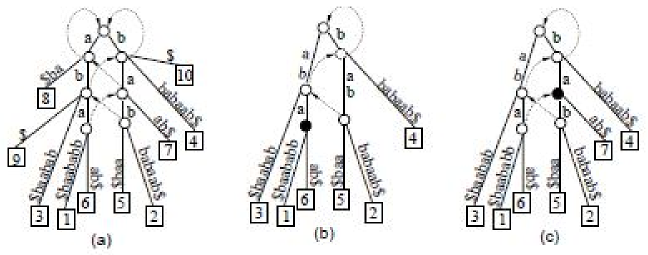

Figure 3. An example of inserting a suffix.

To construct a suffix tree for q, we insert all suffixes of q one by one into an initially empty tree. For example, the suffixes of "ababbabaab$" are "ababbabaab$," "babbabaab$," "abbabaab$," . . ., "b$." Let Si denote q [i..|q|], the Ith suffix of q. Conceptually, to insert Si, we first look up Si, matching Si against edge labels as we go down the tree until no more characters can be matched. If lookup stops at a node, we insert Si as a leaf below that node; if lookup stops in the middle of an edge, we add a new node to split the edge right before the point where it diverges from Si, and then insert Si as a leaf of the new node. Unfortunately, if we insert every Si by starting the lookup from the root, we would end up with a quadratic time algorithm.

V. CONCLUSION

In this section, we discuss the main contributions of our work as well as their limitations.

Extraction framework: Existing extraction frameworks do not provide the capabilities of managing intermediate 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 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. . Parse tree query language. One of the main contributions of our work is PTQL that enables information extraction over parse trees. While our current focus is per-sentence extraction, it is important to notice that the query language itself is capable of defining patterns across multiple sentences. By storing documents in the form of parse trees, in which the node DOC is represented as the root of the document and the sentences represented by the nodes STN as the descendants. PTQL has the ability to perform a variety of information extraction tasks by taking advantage of parse trees unlike other query languages. Currently, PTQL lacks the support of common features such as regular expression as frequently used by entity extraction task. PTQL also does not provide the ability to compute statistics across multiple extractions such as taking redundancy into account for boosting the confidence of an extracted fact. 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] S. 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] A.R. Aronson, "Effective Mapping of Biomedical Text to the UMLS Metathesaurus: The MetaMap Program," Proc. AMIA Symp., p. 17, 2001.

First A. Author: ***Rajula Srilatha*** *woasrking* as Assistant Professor in Narayana Engineering College Nellore from 2009. She received B.Tech in Computer Science and Information Technology from KMCE affiliated to JNTU, Hyderabad. She is currently doing M.Tech in Computer Science and Engineering from CMRTC, Hyderabad. And research interests include logic programming, information extraction, information retrieval, knowledge representation, keyword search on structured and semi structured data.

Second B. Author: ***Mr. K.Murali*** working as Assistant Professor of CSE in CMR Technical Campus, Hyderabad from 2012, he has worked as Asst. Professor of CSE in BIES, Warangal for last 4 years. Ratified by JNTUH (2012). Teaching Experience: 5 years. Guided many B.Tech and M.Tech students in their project work.