GenerIE: Information Extraction Using Database Queries

Luis Tari[1], Phan Huy Tu[1], Jörg Hakenberg[1], Yi Chen[1], Tran Cao Son[2], Graciela Gonzalez[3], Chitta Baral[1]

[1]*Department of Computer Science and Engineering, Arizona State University*
Tempe, AZ 85287, USA

[2]*Department of Computer Science, New Mexico State University*
Las Cruces, NM 88003, USA

[3]*Department of Biomedical Informatics, Arizona State University*
Phoenix, AZ 85004, USA

Abstract— **Information extraction systems are traditionally implemented as a pipeline of special-purpose processing modules. A major drawback of such an approach is that whenever a new extraction goal emerges or a module is improved, extraction has to be re-applied from scratch to the entire text corpus even though only a small part of the corpus might be affected. In this demonstration proposal, we describe a novel paradigm for information extraction: we store the parse trees output by text processing in a database, and then express extraction needs using queries, which can be evaluated and optimized by databases. Compared with the existing approaches, database queries for information extraction enable generic extraction and minimize reprocessing. However, such an approach also poses a lot of technical challenges, such as language design, optimization and automatic query generation. We will present the opportunities and challenges that we met when building GenerIE, a system that implements this paradigm.**

I. Introduction

Information extraction (IE) is typically realized by special-purpose programs that perform a sequence of processing modules, including sentence splitters, tokenizers, named entity recognizers, shallow or deep syntactic parsers, and finally extraction based on a collection of patterns. However, such a framework is inflexible and expensive in face of dynamic application needs. Consider a biology-oriented scenario when the original information extraction goal is to extract interactions among proteins from a corpus of text. Suppose later on we are interested in finding gene-disease associations from the same corpus. Existing approaches would have to develop a new extraction system specifically for this new extraction goal, and run that extraction system on the entire corpus from scratch, which is very expensive. Consider another application scenario where the extraction goal remains the same, but an improved named entity recognizer becomes available. This would also require extraction to be performed from scratch on the entire corpus. However, we observe that only a portion of the corpus is affected with newly recognized entities, as the majority of the entities are overlaps between the original and the improved recognizers. Such expensive re-computation should be minimized. This is particularly true for extraction in the biomedical domain, where a full processing of all 17 million Medline abstracts took about more than 36K hours of CPU time using a single-core CPU with 2-GHz and 2 GB of RAM. In this case, the Link Grammar parser [3] contributes to a large portion of the time spent in text processing.

In this demonstration, we propose a new paradigm of information extraction in the form of database queries. We present a general-purpose information extraction system, GenerIE, in the context of biomedical extraction, which can efficiently handle diverse extraction needs and keep the extracted information up-to-date incrementally when new knowledge becomes available. The insight of GenerIE is that changes in extraction goals or deployment of improved processing modules hardly affects all sentences in the entire collection. Thus we differentiate two phases of processing.

- *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 current knowledge. The generated syntactic parse trees and semantic entity tagging of the processed text is stored in a parse tree database (PTDB).
- *Extraction Phase:* Extracting particular kinds of relations can be done by issuing an appropriate query to PTDB. As query languages such as XPath and XQuery are not suitable for extracting linguistic patterns [2], we design and implement a query language called PTQL for pattern extraction which effectively achieves diverse IE goals [6]. To ease the extraction tasks for users, our system not only allows a user to issue PTQL queries for extraction, but it can also automatically generate queries for high-quality extraction based on user input keyword-based queries and feedback.

There are several advantages of the proposed approach, which have been demonstrated in our initial experimental evaluation. First, using database queries instead of writing individual special-purpose programs, information extraction becomes generic for diverse applications and becomes easier for the user. The user can express and analyze an extraction pattern by issuing a database query. When a user has a new extraction goal, the user only needs to write another query on PTDB without developing and running new programs.

Second, upon new extraction goals, the two-phase approach

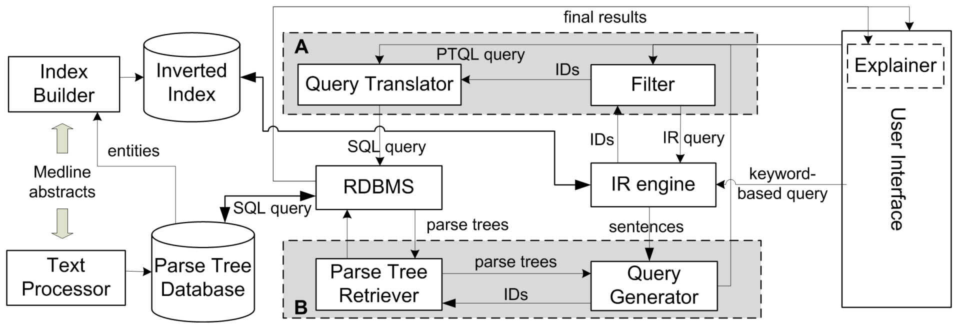


Fig. 1. Architecture of GenerIE. (A) Query-specified Extraction: evaluation of PTQL queries through filtering and translation to SQL queries. (B) Pseudo-relevance Feedback Query Generation: generation of PTQL queries based on the common grammatical patterns among the top-ranked sentences relevant to the user keyword-based queries.

avoids performing the initial phase again, an extremely expensive phase that has be to performed by existing approaches.

Third, with the use of databases, GenerIE only needs to perform extraction *incrementally* on the sentences that are affected by an improved module, and thus it is much more efficient than running the whole extraction programs from scratch as required by existing systems. Suppose an improved named entity recognizer that can discover a more extensive list of protein names becomes available. We only need to perform a delta extraction on the database with respect to the newly recognized protein names using queries.

Indeed, the ability of expressing information extraction and exploiting database optimizations for the process is also observed in [7]. While [7] proposes to use Datalogs for extracting facts from “relational tables”, we focus on extracting meaningful “tables” from “parse trees” of text documents. Due to the variety of extraction needs, the existence of hierarchical data structure and the lack of a relational schema, this involves a new set of technical challenges as outlined in Section IV.

II. System

Figure 1 illustrates the system architecture of our GenerIE system. The *Text Processor* performs the Initial Phase for corpus processing and stores the processed information in the *Parse Tree Database* (PTDB). The extraction patterns over parse trees can be expressed in our proposed *parse tree query language* (*PTQL*). 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 IR engine. The *index builder* creates an inverted index for the corpus as part of the query evaluation by the IR engine.

The *user interface* provides two input modes: *query-specified extraction mode* and *pseudo-relevance-feedback extraction mode*. A user can directly specify PTQL queries for extraction in *query-specified extraction mode*. The user interface also provides the capability for users to input a keyword-based query. When a user keyword query is issued, relevant sentences are retrieved using an existing IR keyword search engine. With the top-ranked sentences, their corresponding grammatical structures are retrieved from PTDB. The *PTQL query generator* then uncovers the common grammatical patterns by considering the parse trees of the top-ranked sentences to automatically augment the initial keyword-based queries and generate PTQL queries. Extracted results are presented to the users once the queries are evaluated. Furthermore, an *explainer* module is available to illustrate the provenance of query results by showing the syntactic structures of the sentences involved in the extracted results. This helps the users understand, and enhance their queries accordingly.

A. Text Parsing and Parse Tree Database (PTDB)

The *Text Processor* parses Medline abstracts with the Link Grammar parser [3], and identifies entities in the sentences. Each document is represented as a hierarchical representation called the *parse tree of a document*. 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. Figure 2 shows a sample parse tree of a sentence, where the solid lines indicate parent-child relationships in the constituent tree and the dotted lines represent the linkage. Each leaf node in a parse tree has a value and a `tag` attribute. The `tag` attribute indicates the entity type of a leaf node. The *Parse Tree Database* is a relational database for storing parse trees and semantic types provided by the *Text Processor*.

B. Information Extraction Using PTQL Queries

To perform information extraction, we propose a query language, PTQL, to specify linguistic patterns on parse trees. An example of a PTQL query is shown in Figure 3. A PTQL query consists four components delimited by colons: (i) tree patterns, (ii) a link condition, (iii) a proximity condition, and (iv) a return expression. A *tree pattern* describes the hierarchical structure and the horizontal order of the nodes in a linguistic extraction pattern. XPath axis are used for expressing node relationships. In the example for Figure 3, the tree pattern specifies that there is a node labeled as `S` as the root of a subtree that contains three nodes represented by variables `i1`, `v` and `i2`. A *link condition* describes the

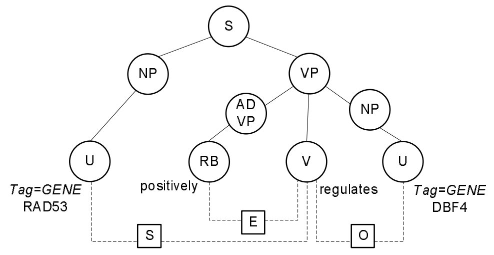


Fig. 2. An example of a parse tree

```
//S{//?[tag='GENE'](i1)=>//V[Value='regulates']
  (v)=>//?[tag='GENE'](i2)}: i1 !S v and
    v !O i2 :: i1.value, v.value, i2.value
```

Fig. 3. An example of a PTQL query

linking dependencies between nodes. In the example for Figure 3, `i1 !S v` represents that the node denoted by `i1` has to be connected to the node denoted by `v` through an `S` link. In other words, `i1` is the subject and `v` is the corresponding verb. Similarly, the link term `v !O i2` indicates that `i2` is the object and `v` is the corresponding verb. A *proximity condition* specifies words that are within a specified word distance in the sentence. A *return expression* defines the list of elements to be returned. In the example, `i1.value, v.value, i2.value` indicate to return the bindings to the variables `i1, v, i2` (i.e. two interactors and the interaction verb) for sentences that satisfy the query. The parse tree in Figure 2 satisfies the query. The details of the PTQL query language and its implementation can be found in [6].

C. Pseudo-relevance Feedback Query Generation

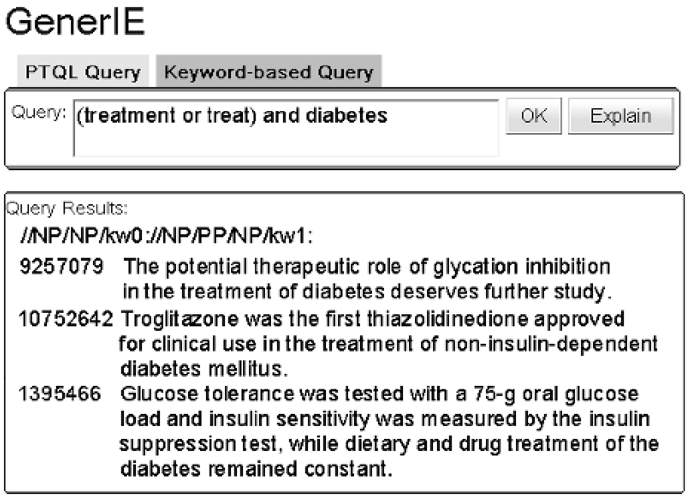
To ease the the learning curve in issuing PTQL queries for the users, GenerIE allows a user to issue simple keyword-based queries, and automatically generates PTQL queries based on the user keyword query.

To achieve this, it first performs an initial retrieval from the inverted index of the corpus with the user keyword query. Among the top-$k\%$ of the retrieved sentences S_k, the parse trees of S_k are retrieved from PTDB to find the common grammatical patterns among S_k. Intuitively, a sentence that bears the common grammatical patterns among the top-ranked sentences are likely to be relevant. Second, for each parse tree of the relevant sentence GenerIE extracts the subtree that is rooted at the LCA (lowest common ancestor) lca of the query terms. Third, to efficiently compare and find the common patterns, GenerIE generates m-th level string encodings for each subtree [5]. When $m = 0$, the string encodes the exact linguistic pattern in the subtree, and thus the retrieved sentences have the exact pattern as the relevant sentences, potentially with a high precision. With the increase value of m, the string encodes a more generalized linguistic pattern, and is likely to retrieve more sentences that leads to a higher recall with possible compromise on precision. Fourth, identical m-th level string encodings form clusters of common grammatical patterns C_m. Finally, a PTQL query is generated for each of the clusters in C_m.

D. Query Evaluation and Optimization

To evaluate PTQL queries on PTDB, the *Query Translator* generates SQL queries from PTQL queries. Efficiency is a key requirement for query evaluation. One of our optimizations is that for each PTQL query, the *Filter* module first generates an keyword-based query to efficiently prune irrelevant sentences, and then the *Query Translator* generates a SQL query equivalent to the PTQL query, and performs the actual extraction only on relevant sentences. The keyword-based query captures keywords in the PTQL query, while the extraction query captures both the structural patterns and keywords. The keyword-based and SQL queries are evaluated using an IR engine and a relational database, respectively. For efficient query processing, the *Index Builder* creates an inverted index that indexes sentences according to the words, named entities and entity types.

III. Demonstration



Query: (treatment or treat) and diabetes

Query Results:
/NP/NP/kw0://NP/PP/NP/kw1:

9257079	The potential therapeutic role of glycation inhibition in the treatment of diabetes deserves further study.
10752642	Troglitazone was the first thiazolidinedione approved for clinical use in the treatment of non-insulin-dependent diabetes mellitus.
1395466	Glucose tolerance was tested with a 75-g oral glucose load and insulin sensitivity was measured by the insulin suppression test, while dietary and drug treatment of the diabetes remained constant.

Fig. 4. A screenshot for the GenerIE system showing the query results that share the same m-th level string encoding.

What will be shown in the demo? Our web-based demonstration , as shown in Figure 4, will illustrate how the GenerIE system enables generic extraction.

• *Query-specified Extraction.* In the demonstration, the user can input a PTQL query to express an extraction pattern or select one of the PTQL query examples. We will show that to perform extraction, a user no longer needs to write specific extraction programs.

• *Pseudo-relevance Feedback Query Generation.* The user can express extraction patterns in the form of keyword-based queries. This scenario illustrates the feasibility of generating PTQL queries from keyword-based queries through a mechanism inspired by the pseudo-relevance feedback approach commonly found in IR. In addition, the user can achieve extraction results for optimal precision or recall by adjusting the value of m.

• *Two Phase Extraction and Incremental Evaluation.* We will illustrate the efficiency of GenerIE when new extraction goals or improved processing components emerge. For instance, assume that NER1 is a currently deployed gene name recognizer, and NER2 is an improved version NER1 to be adopted by GenerIE. The user can browse the sentences that are affected, i.e. sentences with genes that are recognized by NER2 but not NER1, and vice versa. Then the user can see that the extraction is incrementally performed on the affected sentences only, and thus it is very efficient.

• *Provenance of Query Results and Query Explanation.* To help users develop and test their queries, upon click, the *provenance* of the query results will be displayed, which includes the original sentence along with its parse tree. GenerIE also illustrates the flow of every step of the query generation and PTQL query evaluation.

IV. Discussion

Significance of Our Approach. The significance of our approach lies in three aspects.

• *Novel Database-Centric Framework for Information Extraction.* Information extraction is traditionally realized by writing special-purpose programs for each specific extraction goal. In this demonstration, we will illustrate a new extraction framework, where extraction is formulated as queries on a database that stores the parsed data. The benefits of such a framework for information extraction include: (i) incremental evaluation is achieved in the presence of new extraction goals and deployment of improved processing components; (ii) database query optimization is leveraged for efficiency.

• *Proven Success of Information Extraction in Biomedical Domain with Promises to General Domains.* The underlying framework of the GenerIE system has been tested on information extraction of biomedical literature [5], and performed among the top in the BioNLP'09 shared task on event extraction [4]. Our two-phase extraction framework and query generation are not specific to only the biomedical domain, but can be adapted to information extraction in general domain.

• *Performing Diverse Extraction Goals without Training Data.* Typical IE systems, such as Snowball [1], adopt the supervised learning approach that takes annotated data in generating extraction patterns. However, training data is scarce and it is known to be expensive to assemble. This can limit the opportunity for a trained IE system to perform another extraction goal. Our automated query generation approach forms PTQL extraction queries by exploiting the linguistic features of the top-ranked relevant results. Without the use of training data, our approach is readily available to extract different kinds of extraction goals. Such approach serves diverse information needs among different users.

Database Challenges. Our general framework for two-phase information extraction opens up a lot of new opportunities and challenges for data management research.

• *Languages for Information Extraction.* The parse tree database is complex, and extraction patterns involve traversals of paths in constituent trees, as well as links and link types between node pairs. Without user-defined functions, existing query languages fail to specify required extraction patterns due to missing axes (XPath, XQuery) or unable to traverse linkages as a first class citizen (XPath, XQuery, LPath [2]). The design of query languages for information extraction on parsed documents demands investigation.

• *Optimization Challenges for Query Optimization on Large-scale Data.* GenerIE handles 1.5 terabytes of parsed text data. Thus efficiency and scalability are essential elements of the system. During prototyping, we found that directly evaluating SQL queries translated from PTQL queries was very slow due to the complexity of the extraction patterns. In GenerIE, we significantly improved the efficiency by leveraging keyword-based queries for pruning. However, further query optimization is essential to handle cases when only a small number of sentences can be filtered by keyword-based queries.

• *Automated Query Generation.* Query generation is critical so that casual users can specify their information needs without learning a query language. Although our current attempts of automated query generation already show promises, many further technical challenges need to be addressed. For instance, how to strike the balance of precision and recall when generating PTQL queries that may generalize the linguistic tree patterns in relevant sentences? How to estimate the "quality" of the generated PTQL queries before the execution?

GenerIE presents our attempts in providing a versatile approach for information extraction. The elegance of our approach is that unlike typical extraction frameworks, introducing new knowledge in our framework does not require the reprocessing of all modules. Simple SQL insert statements can be issued to store the new entities in PTDB. We believe that studying fundamental database management issues on information extraction – a well-known important problem – opens up a lot of new opportunities and challenges.

V. Acknowledgments

We appreciate the comments made by the reviewers. Authors CB, JH, LT and PHT acknowledge the support from NSF 0412000, 0950440, SFAZ CAA 0289-08 and an IARPA contract. YC acknowledges the support of NSF through the CAREER grant number IIS-0845647.

References

[1] E. Agichtein and L. Gravano Snowball: extracting relations from large plain-text collections. In *ACM Digital libraries*, 2000.
[2] S. Bird, Y. Chen, *et. al.* Designing and Evaluating an XPath Dialect for Linguistic Queries. In *ICDE*, 2006.
[3] D. Grinberg, *et. al.*. A Robust Parsing Algorithm For LINK Grammars. CMU-CS-TR-95-125, Pittsburgh, PA, 1995.
[4] J. Hakenberg, *et. al.* Molecular event extraction from Link Grammar parse trees. In *Proc. of BioNLP'09*, 2009.
[5] L. Tari, *et. al.*. Querying parse tree database of Medline text to synthesize user-specific biomolecular networks. In *PSB'09*, 2009.
[6] P. H. Tu, *et. al.*. Generalized text extraction from molecular biology text using parse tree database querying. TR-08-004, Arizona State University, 2008.
[7] S. Warren, *et. al.* Declarative IE using datalog with embedded extraction predicates. In *VLDB '07*, 2007.