Integrating querying and retrieval for biomedical information extraction

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
[#]luis.tari@asu.edu

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

[3]*Department of Biomedical Informatics, Arizona State University*
Tempe, AZ 85287

***Abstract*— Biomedical natural language processing (BioNLP) involves the automatic processing of documents in the biomedical domain for the purpose of extracting information of interest from them. A common approach to BioNLP is through sequential application of a selection of modules (such as a named entity recognizer and a grammar parser followed by extraction) where the aggregated output is stored in an application-specific format. A major drawback of this approach is that if a particular module is improved, then one needs to re-process the data from that point forward in order for the improvement to have an impact on the final output. We propose a data-centric approach to minimize such re-computation by using a database approach where the result of processing some general modules are stored in an appropriately designed database, and specific modules are implemented as database queries in a high-level query language called PTQL. Our resulting protein-protein interaction extraction system relies on 11208 linguistic patterns that are collected automatically from training data, and we demonstrate that the combination of information retrieval and database techniques is necessary to be able to execute such a large number of complex linguistic patterns. Our experimental results also show that such approach is capable of achieving high precision and recall for the extraction of protein-protein interactions. In particular, our resulting protein-protein interaction extraction system is currently the best when evaluated with respect to the BioCreative 2 dataset.**

I. Introduction

The focus of experimental research in molecular biology has shifted from data collection to data-driven generation of hypothesis in the past years [1]. Millions of facts are stored in a large amount of databases. Still, the most recent experimental findings and, more importantly, valuable hypotheses and opinions, can be found in semi-structured scientific articles only. Facts described in publications on seemingly unrelated topics further add to the vast amount of 'hidden' knowledge. Searching for particular facts (say, gene-disease or protein-protein associations) forces researchers to scan thousands of articles. To satisfy the information needs of biologists, biomedical natural language processing (*BioNLP*), has been an active research area over the past years [2]. It aims at making information contained in literature accessible in ways known from structured databases, thereby enabling knowledge integration over a plentitude of entities (genes, drugs) and their relationships.

A common technique for the extraction of such relations is to sequentially apply various task-specific modules (such as named entity recognizer, named entity identifier, parsing, etc.) where the result of applying each module feeds into subsequent modules. In this approach, if a particular module is improved then one needs to re-process the data through subsequent modules from that point onward. For example, using an improved named entity recognition module means that the whole extraction process has to be repeated from scratch. It is essential to minimize such expensive recomputation. We propose here an approach to BioNLP that minimizes such re-processing by replacing the text handling pipeline model with a data-centric approach, where the output of each module is stored in a central database and the final information extraction step is implemented as a series of database queries. Our proposed framework stores parse trees and recognized entities in a relational database and a high-level query language that is capable of expressing linguistic patterns to retrieve information from the database, such as extracting protein-protein interactions. With the query language, reprocessing the corpus for extraction of protein-protein interactions can be avoided by issuing queries to the newly added protein names.

Extensions of XPath [3] and XQuery [4] to handle linguistic queries on parse trees have been proposed in the past [5], [6], [7], [8]. In particular, [7] described a framework that expresses linguistic patterns with their query language and applied the framework to question-answering using Medline abstracts. However, their evaluation is based on only 8 queries. It is not clear how their system would perform on a large number of complex queries, which can occur when extraction patterns are learned from full-text articles, as sentences in full-text are typically more syntactically complex than the sentences in the abstracts, which are written in a concise manner.

In this paper, we develop a parse tree query language called PTQL and an efficient query processing method that utilizes

an existing information retrieval system. We demonstrate that the combination of information retrieval and database provides an efficient way to process large number of queries for the extraction of protein-protein interactions. Our approach in utilizing information retrieval for extraction goes beyond the extraction systems proposed in [9], [10], [11]. Unlike these earlier efforts, which are based on text literal patterns for extraction, our approach is capable of expressing grammatical patterns that utilize syntactic parse trees and dependency grammar in an efficient manner. Our results show that the use of information extraction techniques, particularly the pre-selection of relevant sentences by partial matching of the semantic aspects of the patterns, achieves high precision and recall for the extraction. We also illustrate that such framework is ideal for the development of BioNLP applications other than text extraction. For instance, information in the text surrounding a gene name, such as the mention of species, can be seen as linguistic patterns for finding the correct gene identifiers in the task of gene name normalization. For the recognition of gene names from text, neighboring words of gene name mentions, such as the word "protein", can be used as linguistic patterns to provide clues for recognizing gene names. These patterns can be used as features for training machine learning approaches in the recognition of gene names.

The main contributions of this paper can be summarized as follows.

- We have developed a protein-protein interaction extraction system that is far superior than the existing systems in terms of its accuracy with respect to the BioCreative 2 dataset.
- As part of this system development we developed and implemented a parse tree query language that can express queries corresponding to or useful in information extraction as well as other BioNLP tasks such as named entity recognition, entity normalization, and question answering.
- We also show how a combination of information retrieval and database query processing leads to an efficient implementation of our main task of protein-protein interaction extraction when we need to apply a large number of linguistic patterns, 11208 to be exact, to 358 full-text articles.

II. Related Work

The problem of information extraction has been an active research area over the years, especially for extracting protein-protein interactions (see [12], [2]). The main focus has been on improving the accuracy of the extraction systems, and much fewer work in developing query languages for extraction. DIAL [13], [14] was originally designed to be applied to extraction for the non-biological domain, but further extension has been proposed to extract biological relationships. The extraction rules written in DIAL can be seen as a logic program, describing the entities and the specific patterns associated with types of the relationships. However, DIAL does not support the use of deep parsers to query dependencies between words. Another text analysis engine called TLM [15] is based on the idea of a retrieval system that allows retrieval of documents, sentences, or phrases based on queries composed of words, punctuations and tags. TLM is based on a highly expressive query language, and its queries are based on textual patterns. However, TLM does not allow querying parse trees, which prohibits the representation of grammatical dependencies between words in the queries. OpenDMAP [16] is an extraction system that is based on an extraction language capable of expressing textual patterns. While it is possible to express dependent relations between words, it does not have features that allows navigating parse trees for extraction. [7] proposed an expressive query language that supports parse tree navigation based on the HPSG parser. Our work takes a similar direction with our parse tree database and a more expressive query language. Our framework incorporates an information retrieval system for efficient query processing. Such integration is critical for processing large number of complex linguistic queries that involve navigation of constituent trees and dependency relations between pairs of words.

The aspect of using information retrieval system on information extraction can be found in several systems for the non-biological domain. QXtract [9] uses a machine-learning approach in deriving queries to retrieve promising documents before extraction. Ours focuses on a narrower aspect that assumes majority of the the biomedical articles provided to our extraction system are articles about protein-protein interactions. KnowItNow [10] is an extraction system based on a natural language search engine [17] that indexes part-of-speech tags to support variabilized queries. An extension of their work includes neighbor and multigram indices [11]. These indices can be seen as a way to store syntactic relations in a statistical manner. Our work stores parse trees of text, which captures the complete syntactic relations among the words in the sentences. [18] considers efficiency of extraction over evolving text, but our work considers the contents of the biomedical articles for extraction to be static. [19] finds execution plans over different document retrieval and extraction strategies with a single extraction system (and later extended to multiple extraction systems [20]) for efficiency-quality tradeoffs. Our work does not consider such tradeoff issue.

III. Background - Link Grammar

The Link Grammar parser is a deep syntactic parser based on the Link Grammar theory [21]. Link Grammar consists of a set of words and linking requirements between words. A sequence of words is a *sentence* in the language if there is a way to connect words together such that (i) the links do not cross, (ii) the words form a connected graph, and (iii) 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.

We use the output of the Link Grammar parser for the sentence "*RAD53 positively regulates DBF4*" as an illustration. The linkage contains three different links: the `S` link connects the subject-noun `RAD53` to the transitive verb `regulates`,

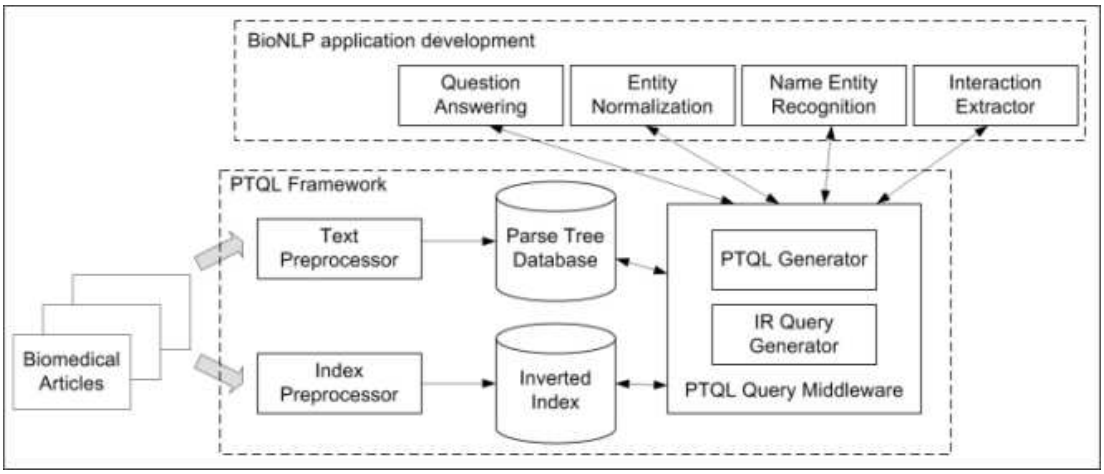

Fig. 1. System architecture of the PTQL framework that involves the parse tree database and inverted index. The PTQL query middleware generates PTQL and IR queries based on the linguistic patterns that are provided by the BioNLP applications under development, and retrieve information from the database and the index.

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 `DBF4`.

```
  +---------S--------+
  |        +----E----+----O---+
  |        |         |        |
RAD53 positively regulates.v DBF4
```

In addition to these links, Link Grammar also has many other links to connect different types of words in sentences. For example, link `M` is used to connect a noun to modifiers such as prepositional phrases, link `J` is for connecting a preposition to its object, and link `AN` for connecting an attribute to a noun. For a complete description of links, we refer the reader to [22]. Note that `regulates` is subscripted as `.v` to indicate that this is a verb.

In addition to linkages, the Link Grammar parser also outputs *constituent trees* for sentences. For instance, the following is the corresponding constituent tree for the above sentence:

```
(S (NP RAD53)
   (VP (ADVP positively)
       regulates
       (NP DBF4)))
```

where `S` stands for a sentence, `NP` stands for a noun phrase, `VP` stands for a verb phrase, and `ADVP` stands for an adverb phrase.

IV. System Architecture

We illustrate an overview of our system in Figure 1. The core component of our framework is the *parse tree database*, which is a relational database for storing parse trees and other information from processed text provided by the *text preprocessor*, and the information retrieval (IR) system for *efficient query processing*. The role of the text preprocessor includes parsing the biomedical text with the Link Grammar parser and named entity recognizers to identify biomedical entities in the sentences. Our current implementation utilizes a dictionary-based approach [23] in recognizing entities, but more advanced statistical-based name entity recognizers can be used as well. The *index preprocessor* utilizes dictionaries of different word categories, such as interaction verbs that are frequently used in biomedical text to describe interactions between proteins. Terms that belong to particular categories are replaced with identifiers in each sentence, and all sentences with identifiers are indexed.

Since standard SQL queries are not ideal for expressing queries that involve linguistic patterns, we develop a high-level query language called *parse tree query language* (*PTQL*) that can be used by various BioNLP applications to express linguistic patterns for their own needs. The *PTQL query middleware* handles the communication between a BioNLP application and the parse tree database. The middleware takes linguistic patterns and generates IR and PTQL queries based on the patterns. The PTQL queries are translated into standard SQL queries before querying the parse tree database. We summarize the process of translating PTQL queries into SQL queries by the PTQL query middleware as follows:

1) Generate an IR query based on a linguistic pattern that is provided by a BioNLP application.
2) Use the IR query to retrieve relevant documents D and the corresponding sentences S.
3) Translate the PTQL query into an SQL query and instantiate the query with document id $d \in D$ and sentence id $s \in S$.
4) Query the parse tree database using the SQL query generated in Step 3.
5) Return the results to the BioNLP application.

The goal of the approach in combining both IR with the parse tree database is to process PTQL queries efficiently. The idea is to limit the database to search for linguistic patterns only on sentences retrieved by the IR system. We will demonstrate the efficiency of this approach with the extraction of protein-protein interactions in later sections. We

doc_id	*id*	*sent_id*	*pid*	*depth*	*left*	*right*	*word_order*	*label*	*value*	*tag*
...	...	...	...	...	...	...	...	...	...	...
8611	15	15	14	3	13	18		STN	RAD53 positively regulates DBF4	
8611	16	15	15	4	13	18		PSTN		
8611	17	15	16	5	13	18		S		
8611	18	15	17	6	13	14		NP		
8611	19	15	18	7	13	14	1	U	RAD53	P
8611	20	15	17	6	14	17		VP		
8611	21	15	20	7	14	15		ADVP		
8611	22	15	21	8	14	15	2	U	positively	
8611	23	15	20	7	15	16	3	V	regulates	I
8611	24	15	20	7	16	17		NP		
8611	25	15	24	8	16	17	4	N	DBF4	P
8611	26	15	17	6	17	18		U		
...	...	...	...	...	...	...	...	...	...	...

TABLE I

RELATIONAL REPRESENTATION OF THE CONSTITUENT TABLE FOR THE PARSE TREE IN FIGURE 2

first describe the parse tree database and the syntax of PTQL before we provide details of how IR is utilized in PTQL query processing.

doc_id	*f_id*	*t_id*	*type*
...	...	...	...
8611	19	23	S
8611	22	23	E
8611	23	25	O
...	...	...	...

TABLE II

RELATIONAL REPRESENTATION OF THE LINKAGE TABLE FOR THE PARSE TREE IN FIGURE 2

A. Parse Tree Database

Documents are represented in a hierarchical format to store sentences and their corresponding parse trees in a relational database. We call the hierarchical representation format as the *parse tree of a document*, and the database as the *parse tree database*. Figure 2 shows an example of a parse tree for a Medline abstract. In this figure, 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.

A parse tree for a document can be viewed as a tree in which the root node is labeled with `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 with `STN` may have more than one child labeled with `PSTN` to allow the storage of multiple parse trees that can be generated by the Link Grammar parser. In addition to the `label` attribute that stores the name of the constituent, each node in the parse tree has other attributes, such as `doc_id` `id`, `pid`. The attribute `doc_id` indicates the document in which a node is originated, `id` serves as an unique identifier for a node, and `pid` represents the `id` of the parent of a node. The attribute `sent_id` indicates the `id` of the sentence that a node belongs to, and `word_order` stores the position of the word in which a node represents. The attribute `value` stores the text representation of a node, while `tag` indicates the entity type of a node. A protein is marked with a tag `P`, disease with a tag `D`, species with a tag `S` and a word referring to an interaction is marked with a tag `I`. The attributes `left` and `right` are identifiers assigned by a labeling scheme that captures the hierarchical structure and the horizontal relationships between nodes. The details of the labeling scheme can be found in [24]. Table I shows an example of the `Constituent` table that stores the relational representation of the parse trees of documents. The table `Linkage` is used to store the links of the linkages of sentences. Table II shows an example of the `Linkage` table, and the attribute `type` indicates the types of the links between pairs of nodes, which are represented by the identifiers stored in the attributes `f_id` and `t_id`.

B. PTQL: Parse Tree Query Language

A fundamental design criteria for the query language is the ability of expressing linguistic patterns based on constituent trees. Standard XML query languages such as XPath [3] and XQuery [4] 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, as shown in [25], leads to the development of LPath [5], [6] 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 [21]. 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. While the XML-like query language proposed in [7] is based on another kind of dependency grammar called head-driven phrase structure

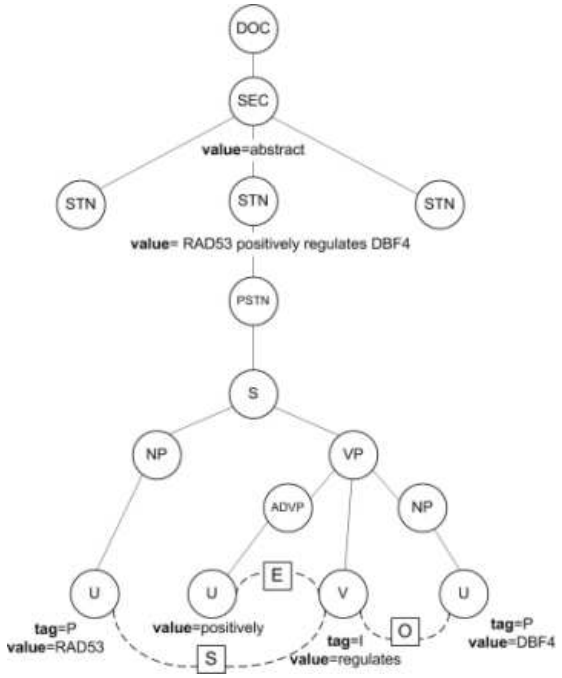

Fig. 2. An example of a parse tree for a document, which includes sections of the document, sentences and the corresponding parse trees (both constituent trees and dependency linkages)

grammar (HPSG), link types cannot be expressed with this query language.

We propose a high level extraction query language called PTQL, which is designed for the purpose of simplifying the utilization of linguistic patterns in BioNLP applications. PTQL is an extension of the linguistic query language LPath [5], [6] 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: (i) tree patterns (ii) link conditions, (iii) proximity conditions, and (iv) 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. Before going into details of the definition of PTQL queries and its usage, we start with the basic element of PTQL queries called node expressions.

Definition 1: A *node expression* is an expression of the form `X[<pred exp>](x)` where `X` is a node name, e.g., a sentence (`STN`), a parse sentence (`PSTN`), a noun phrase (`NP`), or `?` (a node of any name), `<pred exp>` (predicate expression) is a boolean formula of expressions of the form `<attribute> <op> <value>`, and `x` is a variable name. Intuitively, `X[<pred exp>](x)` represents a node of type `X` that satisfies the condition specified by `<pred exp>` and this node is denoted by the variable `x`. When the predicate expression is empty (resp. variable binding is empty) we can omit the square brackets (resp. parenthesis). As an example, `N[tag='P'](x)` is a node expression describing a node labeled with `N` (noun) and tagged with `P` (i.e., a protein name) and this node is denoted by the variable `x`. A wildcard `'?'` denotes a node of any name. For instance, `?(y)` represents a node of any name and this node is denoted by the variable `y`.

To describe a tree pattern, we use two types of axis. A vertical axis `/` (parent-child relationship) or `//` (ancestor-descendant relationship) describes the hierarchical order of nodes in the parse tree. A horizontal axis `->` (immediate following) or `=>` (following) describes the horizontal order of nodes[1]. Formally, a tree pattern is defined recursively as follows.

Definition 2: If `e` is a node expression then `/e` and `//e` are tree patterns. If `e` is a node expression and $q_1, \ldots, q_n$ are tree patterns then `/e{` q_1 `<ha`$_1$`>` $\ldots$ `<ha`$_{n-1}$`>` q_n `}` and `//e{` q_1 `<ha`$_1$`>` $\ldots$ `<ha`$_{n-1}$`>` q_n `}` are tree patterns, where `ha`$_i$ can be `->`, `=>`, or `<=>`.

A parse tree matches a pattern `/e` (resp. `//e`) if one of the children (resp. descendants) of the root node satisfies the node expression `e`. A parse tree matches a pattern `/e` { q_1 `<ha>` $\ldots$ `<ha>` q_n } (resp. `//e` { q_1 `<ha`$_1$`>` $\ldots$ `<ha`$_{n-1}$`>` q_n }) if there is a node `X` with children Y_1, Y_2, $\ldots$, Y_n of `X` such that (i) `X` is a child (resp. descendant) of the root node of the parse tree, (ii) `X` satisfies the node expression `e`, (iii) each tree T_i with `X` being the root node and the subtree of the parse tree rooted at Y_i being the only subtree of T_i matches the pattern q_i, and (iv) if `ha`$_i$ `= '->'` (resp. `ha`$_i$ `= '=>'`) then Y_{i+1} immediately follows (resp. follows) Y_i. Y_{i+1} follows or precedes Y_i if `ha`$_i$ `= '<=>'`. For instance, the parse tree in Figure 2 matches the pattern `//S{/NP->/VP}` as (i) the node labeled with `S` is a descendant of the root node of the parse tree, (ii) the first noun phrase and the verb phrase are two children of that node, and (iii) the verb phrase immediately follows the noun phrase. This tree also matches the pattern `//S{//?[tag='P']=>//VP{/V->//?[tag='P']}}`. However, it does not match the pattern `//S{//NP->//V}` as the verb does not immediately follow either the first noun phrase (there is an adverb in between) or the second noun phrase. A link condition is defined as follows.

Definition 3: A *link term* is an expression of the form `x !<link> y`, where `x` and `y` are variable names and `<link>` is a link name in the linkage. A *link condition* is a boolean expression of link terms.

For instance, `x !S y` is a link term representing the fact that the node denoted by `x` connects to the node denoted by `y` through an `S` link. Similarly, `y !O z` is a link term representing the fact that the node denoted by `y` connects to the node denoted by `z` through an `O` link. `x !S y AND y !O z` is a link expression whose meaning should be clear.

Definition 4: A *proximity term* is an expression of the

[1]A node `X` is said to (immediately) follow a node `Y` in a parse tree if the right most leaf of `X` (immediately) follows the left most leaf of `Y`.

	Query	PTQL query
Q1	Extract interactions based on the pattern <subject>-<verb>-<object>, in which the subject and object correspond to protein names (tag='P': protein name, tag='I': interaction word)	//S{//?[tag='P'](i1)=>//V[tag='I'](v)=> //?[tag='P'](i2)}: i1 !S v and v !O i2 :: i1.value, v.value, i2.value
Q2	Extract interactions based on the pattern <interaction-noun> between <protein> and <protein>	//NP{//N[tag='I'](v)->//?[value='between'](x)-> //?[tag='P'](i1)->//?[value='and']-> //?[tag='P'](i2)} : x !J i1 and i1 !J i2 :: i1.value,v.value,i2.value
Q3	Find sentences that contain the pattern <protein1> .. <verb> .. <protein2>, and <protein1> is connected to a relative pronoun	//STN(x){//PSTN//?[tag='P'](i1)=>//?(y)->//V[tag='I']=> //?[tag='P']} : i1 !MXsr y :: x.value
Q4	Find sentences that contain the pattern <protein1> .. <verb> <protein2> with an optional arbitrary word between <verb> and <protein2>	//STN(x){//PSTN//?[[tag='P']=>//V[tag='I'](v)=> //?[tag='P'](i2)} :: 1[v i2]2 : x.value

TABLE III

EXAMPLES OF PATTERNS AND THEIR CORRESPONDING PTQL QUERIES

form $m\{x_1 \ \ldots \ x_k\}n$ or $m[x_1 \ \ldots \ x_k]n$, where x_1, ..., x_k are variable names and m, n are integers. A *proximity condition* is a boolean expression of proximity terms.

We use an example to illustrate the definition of proximity terms. 1{x y}2 is a proximity term representing the fact that the nodes denoted by x, y are at least 1 node but not more than 2 nodes apart with respect to the sentence that contains words represented by x and y. In the case of 1[x y]2, an additional constraint is imposed such that words represented by x have to appear before words represented by y in the sentence.

Definition 5: A *return expression* is a list of elements of the form <var>.<attr> separated by ',', possibly preceded by the keyword DISTINCT, where <var> is a variable name and <attr> is an attribute name.

As an example, DISTINCT x.value, y.value, z.value is a return expression that returns the distinct value attributes of nodes denoted by variables x, y, z in the tree pattern. We now define the syntax of PTQL queries.

Definition 6: A PTQL *query* is an expression of the form <pattern> : <link cond> : <proximity cond> : <return exp> where <pattern> is a tree pattern, <link cond> is a link condition, <proximity cond> is a proximity condition and <return exp> is a return expression.

A parse tree matches a PTQL query if it matches the tree pattern of the query and the links between nodes satisfies the link condition of the query. The *return expression* of the query defines what information we want to return.

We illustrate the PTQL queries with examples, as shown in Table III. For query Q1, the tree pattern

```
//?[tag='P'](i1)=>//V[tag='I'](v)=>
    //?[tag='P'](i2)}
```

represents that a protein name (denoted as i1) is followed by a verb (denoted as v), which is followed by another protein name (denoted as i2), while the link condition

```
i1 !S v and v !O i2
```

specifies that i1 has to be the subject of the sentence and v is the corresponding verb, and v and i2 has verb-object relation. Query Q2 extracts patterns of the form "interaction between A and B", where A and B are two protein names. The link J connects a preposition to its object, and the scope of this query is within a noun phrase (NP). Query Q3 illustrates the power of link conditions. The link type MXsr connects a noun to its relative pronoun. This query matches the sentence "*RAD53 regulates DBF4, which binds to ABC1*". In this case, *DBF4* is the noun that connects to *which* through the MXsr link. Proximity conditions are useful for optional matches, as illustrated in query Q4. The proximity condition in Q4 specifies that the verb and the second protein name should be at least 1 but no more than 2 words apart. While this query can be expressed in two PTQL queries, the proximity conditions are useful when multiple optional matches are specified in the query.

The details of the algorithm for translating PTQL queries to SQL queries can be found in [24], which takes a similar approach of translation as described in [5], [26]. We illustrate the translation with an example in Figure 3.

C. Query Processing with Information Retrieval

With the enormous amount of data stored in the parse tree query database, retrieving answers from the database with PTQL queries can be slow due to the number of joins involved in the corresponding SQL queries. Improving the efficiency of query processing is essential to applications that involve a large number of queries, such as the extraction of protein-protein interactions. Poor query processing performance can be expected especially when the number of results returned by the most inner subquery is large. For instance, the most inner subquery of the example in Figure 3, i.e.

```
SELECT docid docid1,..,pid pid1,..,v v1
     FROM C WHERE d>2 AND C.n='S'
```

returns all nodes that have the label as S, which essentially involves the root nodes of every single parse trees in the database. This is clearly infeasible when the parse tree database contains a large number of parse trees. Query

```
PTQL query:
   //S{//?[tag='P'](x)=>//V[tag='I'](y)=>//?[tag='P'](z)}:
          x !S y and y !O z : 1[x y]2 : x.value,y.value,z.value
SQL query:
 SELECT v2,v3,v4
 FROM (SELECT docid docid4,..,id id4,..,v v4,id3,..,v3,..,id1,..,v1
    FROM C,(SELECT docid docid3,..,id id3,..,v v3,id2,..,v2,id1,..,v1
        FROM C,(SELECT docid docid2,..,id id2,..,v v2,id1,..,v1
            FROM C,(SELECT docid docid1,..,pid pid1,..,v v1
                FROM C WHERE d>2 AND C.n='S') C2
            WHERE C.docid=C2.docid AND C.l>=C2.l1 AND C.r<=C2.r1
                  AND C.d>C2.d1 AND (C.t='P')) C3
        WHERE C.docid=C3.docid AND C.l>=C3.l1 AND C.r<=C3.r1
           AND C.d>C3.d1 AND (C.t='I' AND C.n='V') AND C.l>=C3.r2) C4
    WHERE C.l>=C4.l1 AND C.r<=C4.r1 AND C.d>C4.d1 AND (C.t='P')
        AND C.l>=C4.r3) T
 WHERE ((T.id2,T.id3) IN (SELECT f_id,t_id FROM L WHERE TYPE='S'))
     AND ((T.id3,T.id4) IN (SELECT f_id,t_id FROM L WHERE TYPE='O'))
     AND (((T.wo3 - T.wo2) >= 1 AND (T.wo3 - T.wo2) <= 2 AND (T.wo3 >= T.wo2)))
```

Fig. 3. An example of a translated SQL query

optimization for the translated SQL queries is likely to have only limited improvement in the performance.

Our approach in boosting the query processing performance is to utilize an IR system to select documents and the relevant sentences, and then apply the translated SQL queries only to the sentences retrieved by the IR system. This approach is intuitive since the PTQL queries issued by different BioNLP applications are unlikely to be involved in the entire document collection in the database but rather a subset of the collection. In the case of question-answering systems (QA), using PTQL queries to express the target linguistic patterns have to be relevant to the documents of interest. Using the question "Which *mutations* in the *Raf* gene are associated with *cancer*?" as an example, a QA system can first retrieve sentences that have mentions of keywords in close proximity using the following IR query:

```
+"mutations Raf"~5 +cancer
```

The IR query indicates that the keywords *mutations* and *Raf* have to appear within 5 words and cooccur with the keyword *cancer* in the retrieved sentences. A PTQL query can then be applied to the retrieved sentences to improve the precision by selecting sentences that *Raf* acts as a noun-modifier to the word *mutations* through the `AN` link.

```
//DOC[DOC_ID=D]{//STN[ID=S](x){
  //?[tag='P' and value='Raf'](y)=>
  //N[value='mutations'](z)}}:y !AN z:: x.value
```

In the PTQL query, `D` and `S` are document and sentence identifiers for the sentences that are retrieved by the IR system.

In the case of text extraction when queries are more dependent on the concepts, such as proteins, rather than particular instances of concepts, the IR inverted index has to be extended in order to process PTQL queries for text extraction. This approach, as demonstrated in Figure 4, relies on a text preprocessor to recognize entities and replace the entities with identifiers in the sentences. Each sentence in the documents are indexed on its own so that each IR query retrieves a sentence rather than the entire document. Assuming that the concepts of interest are protein names and interaction verbs in present tense, and the identifiers are `PROTNAME` and `IVERB-S` respectively, the sentence "*ABC1 phosphorlyates ABC2*" is indexed as `PROTNAME IVERB-S PROTNAME` under the field name `sent_proc` in the inverted index. The approach of indexing sentences with replaced identifiers is similar to [17], [27], [28]. Unlike [17], our approach requires no modification to the structure of the inverted index in order to process variabilized queries. On the other hand, our query language PTQL is capable of navigating parse trees, which is much more expressive than the query languages proposed in [17], [10], [11], [27], [28]. This extension allows efficient processing of PTQL queries that involve matching of entity types. We describe in details on how this is utilized in the section about evaluation of protein-protein interaction extraction.

V. BioNLP Applications using the PTQL framework

In this section, we elaborate on how the PTQL framework can be applied to the development of various BioNLP applications with examples. We provide details on how the text extraction module utilizes the PTQL framework.

A. Question-answering

Consider the query "Find information about *glycan modification*", in which *glycan modification* is a keyword that spans multiple words. It is important to notice that sentences that match *glycan modification* do not necessarily have both words together as a phrase, as shown in sentences S1 and S2. The sample sentences show that using *glycan modification* as a phrase to find matching sentences could miss out relevant information.

> <S1> It is possible that PSGL-1 requires *modification* of only a few specific O-linked *glycans* to bind optimally to P-selectin.
>
> <S2> Changes in the normal expression profile of glycosyltransferases in various cell types can alter

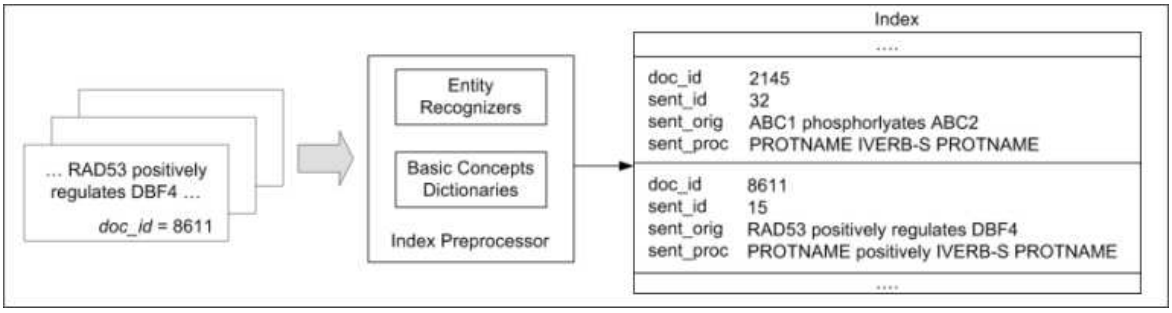

Fig. 4. An extended inverted index to handle queries that involve concepts rather than just instances

glycan branching and influence terminal modifications in the golgi.

However, having both words *glycan* and *modification* in close proximity do not necessarily mean that they refer to the meaning of "*glycan modification*". In fact, sentence S2 is incorrect, as the words *glycan* and *modification* cooccur in the sentence without any actual syntactic relation to each other.

One solution is to first retrieve sentences that have the keywords in close proximity, and then apply linguistic patterns to refine the retrieved sentences. As an example, a PTQL query can be used to find sentences that have *modification* acting as a modifier to the word *glycan* through links `M` and `J`. Such query can be expressed as follows:

```
//STN(x){//N[value='modification'](y)=>
   //?(z)=>N[value='glycan'](w)}:
   y !M z and z !J w :: x.value
```

B. Entity normalization

The task for gene mention normalization is to recognize and identify genes that are discussed in a text. While recognition deals with finding genes on a word–basis (*i.e.*, decide if *p21* in the current text refers to a gene), normalization finds the exact identifier for this gene, for instance, from GenBank. One of the challenges of gene name normalization is to identify which species a gene name belongs to before an identifier can be assigned to the gene name. One solution is to find gene-species relations based on some grammatical patterns, such as finding gene and species that appear in the same noun phrase.

```
{//NP{//N[value='human']=>//?
    [tag='GENE'](x)}} ::: x.value
```

C. Named entity recognition

In general, one may not assume that all entity classes of interest in a text database can be recognized by named entity recognition (NER) systems, such as the ones used by the PTQL implementation. There can even be instances of the entity classes which are recognized but not tagged correctly. PTQL can be useful in complimenting the ability of a NER system.

As NER systems trained by machine learning techniques are typically based on the features of sentences, there can be occasions that a particular gene name is recognized in some documents but not the others. To increase the recall of the system, a NER system can utilize PTQL queries to find sentences in which gene names are not tagged when they are in other sentences. Suppose *p53* has been tagged as a gene name in some documents, find *p53* such that *p53* is not tagged as a gene name.

```
//DOC(x){//STN(y){//?[tag IS NOT 'GENE'
    and value='p53']}}::: x.value, y.value
```

Once the sentences with *p53* are retrieved by the PTQL query, the NER system can add the `GENE` tag to *p53* in those sentences. An interaction extraction module can perform extraction of interactions only to these affected sentences. This is very different from the traditional approach that the entire corpus has to be reprocessed when a component module in the extraction pipeline has been improved.

D. Relationship Extraction

We illustrate our PTQL framework with the an application of protein-protein interaction extraction, as shown in Figure 6. To generate a set of patterns for information extraction, we first automatically annotate an unlabeled 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, we extract initial phrases referring to interactions. We refine these phrases and compute consensus patterns. We then use different strategies to match patterns against new text. At the end, we try to validate each protein–protein interaction predicted by filtering likely false positives.

1) Collecting training data: We use the annotations for PPIs assigned to a publication (but without pointing to the exact evidence in the text); we consider this as *partially labeled data*. One source for such information is the MINT database[2], which contains PPIs and references to the literature (PubMed IDs) for most of them. We can assume that each referenced publication will contain the PPI somewhere in the full–text (taking aside articles on large–scale experiments where particular PPIs might not be discussed in the text, but in tables or maps). While MINT refers to a protein by UniProt ID, the authors of a publication will use one or more names for that protein. We thus employ named entity recognition and normalization to find sentences that contain all partners of an

[2]See http://mint.bio.uniroma2.it/mint/

interaction (most consist of two proteins, some involve three or more proteins, some are auto–interactions). For the purpose of this paper, we restrict the interactions and publications to the data contained in the BioCreative 2 IPS training set, which was created based on entries from MINT [12].

As many sentences might contain coincidental mentions of proteins and not describe an interaction ("We study the proteins A, B, and C."), we reduce these initial candidate evidence by a number of refinement steps. As a first step, we search for words typically referring to protein–protein interactions ("binds", "association", "–mediated"). We currently use a set of 123 verbs, 126 nouns, and 8 adjectives, plus corresponding word forms. Such words have to appear between the two proteins under consideration, or precede/follow them in a short distance, which is parameterizable. We then reduce the full sentence to the snippet that likely conveys the information about an interaction; therefore, we may extract the shortest snippet that contains both proteins and an interaction–indicating word, or include additional words from the left and right of this snippet.

2) Generating patterns from initial data: We consider each snippet found according to Section V-D.1 as an *initial pattern*; it could be directly used to find similar (parts of) sentences in new text. The more the snippet extends to the left and right, the more precise the pattern will be. Shorter snippets, on the other hand, will typically increase the recall when using that pattern.

To increase the recall of the initial patterns, we pursue the following strategy to generalize them: We substitute tokens belonging to certain word categories with a higher–level concept, similar to part–of–speech tags. It shows that many natural language sentences that describe certain events, like protein–protein interactions, exhibit a certain 'behavior' that makes them similar to other such sentences. We exploit this fact in two ways. The first is based on the observation that often, only a few words in the (relevant part in the) sentence are interchangeable with others, without losing the initial meaning of the sentence (at least, in a broader sense). Consider Figure 5, where the words 'binds', 'interacts', etc. are replaced with each other. Thus, in our first method, we treat each initial pattern acquired in step V-D.1 as a template; we sort words into groups corresponding to the following concepts:

- protein, species, cell line (biological entities)
- domain, residue, mutation, phage (other biological categories)
- preposition, auxiliary, modal verb, adverb, adjective, proper noun (part–of–speech)
- bracketed expression
- numeral, number range, percentage (other word categories)
- i-attachment, i-noun-singular, i-verb-past-tense, ..., i-adjective (interaction–indicators)

The first two sets of concepts are clearly geared towards protein–protein interactions, and will have to be adapted for other applications. The last set also refers to words that are useful for finding PPIs, but not necessarily other associations (although there are overlaps, *e.g.*, for bonds between drugs/compounds and proteins).

1	P	–	P	complex	
2	P	and	P	synergism	
3	P	/	P	complex	
A	P	{conjunction}	P	{i-attachment}	
4	P	interacts	with	the	P
5	P	binds	to		P
6	P	bound	to		P
B	P	{i-verb}	{preposition}	{determiner}?	P

{conjunction} := –, /, and;
{determiner} := a, an, the, these, this, those, ...;
{preposition} := between, for, to, with, ...;
{i-attachment} := bond, complex, heterodimer, synergism, ...;
{i-verb} := binds, bound, interacts, interacted, ...;

Fig. 5. Multiple initial patterns (1–3 and 4–6) lead to the same general patterns A and B, respectively, after words have been replaced with concepts (in curly brackets; names of proteins have been replaced with 'P'). For example, the words 'complex' and 'synergism' are instances of the same concept, termed 'i-attachment' (see text).

Figure 5 shows two examples for initial patterns in which replacement of words with concepts leads to identical patterns. Note that concepts are defined so that all instances 'behave' similar from a linguistic point of view, meaning they share similar syntactic distributional and morphological properties; such groupings do not necessarily make sense from a biological perspective (see 'complex' and 'synergism' in Figure 5). When creating patterns, we include some of the aforementioned concepts as optional: species, adverbs, adjectives (normal and interaction–indicating), determiners, bracketed expressions.

3) Query Processing: With the generated patterns, a straightforward approach is to directly translate these patterns to PTQL queries. However, with the large number of PPI patterns, executing all queries can be time-consuming. Our approach of query processing is based on the observation that not all of the generated patterns are applicable to the testing dataset, as some PPI patterns can be over-specific. Using an IR system provides an efficient way in identifying sentences that are relevant to the patterns as an "upper-bound". With the relevant sentences identified by the IR system, a PTQL query translated from the PPI pattern is made specific to these sentences. The whole extraction process is illustrated in Figure 6.

VI. Experimental Results

We evaluate our PTQL framework by applying to the problem of extracting protein-protein interactions (PPI) using the BioCreative 2 dataset [12]. The implementation was done using Java, and we used JavaCC for the translation of PTQL queries to SQL queries. The parse tree database was implemented using the MySQL relational database, and Lucene[3] was used for the IR system of the PTQL framework.

The experiments were performed on a machine with a 2.4-GHz Pentium Xeon QuadCore CPU and 3GB of RAM

[3]see http://lucene.apache.org

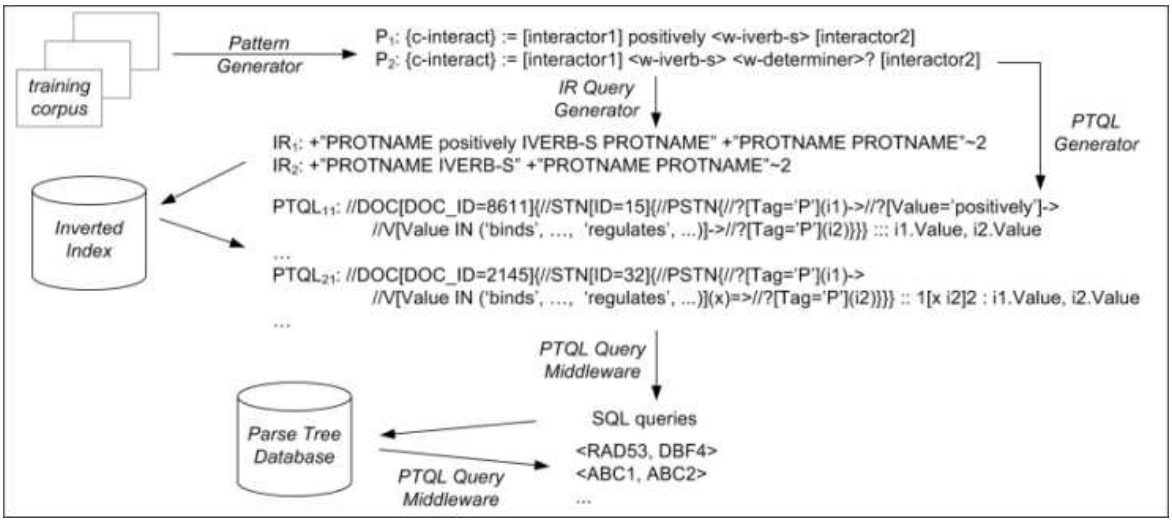

Fig. 6. An example to illustrate the process of extraction that involves query processing using an information retrieval system.

running Windows XP operating system. Each query reported in the later subsections were performed with repetitions of 5 times, and the average query processing time was reported, disregarding the maximum and minimum processing time in the calculation.

A. Experiment Settings

For the evaluation of our extraction system for protein–protein interactions, we use the BioCreative 2 IPS test data [12] as a benchmark. This data set consists of 358 full–text articles, which we transformed into 98,209 sentences. This set was reduced by us to include only sentences that contain at least one protein, resulting in 71,631 sentences. In addition, all proteins were mapped to corresponding identifiers in the UniProt database[4]. We describe the initial recognition of protein names and their mapping to identifiers (called *entity mention normalization*) in [29].

The task in the BioCreative 2 IPS benchmark is to find protein–protein interactions for which a text provides evidence for a physical interaction between the proteins. The benchmark actually combines multiple steps in biomedical NLP: named entity recognition (NER; for proteins), named entity normalization (EMN; to map protein names to UniProt identifiers), and relationship extraction (for protein–protein interactions). In this paper, we focus on the last step; however, the results of the former steps have a large influence on the overall outcome. Typically, NER and EMN both yield performances in the 80–90% range (each); this, for a pair of proteins, creates an upper bound for the overall system, ranging between 50 and 70%.

B. Evaluation of Protein-Protein Interaction Extraction

Using the 11208 patterns generated from the BioCreative 2 training dataset through the pattern generator component, we used two different strategies in utilizing IR to reduce the number of PTQL queries that are applied to the testing dataset. *Approach 1* is to recognize gene names and replace them with an identifier before the sentence is indexed, while *Approach 2* recognizes gene names and other base concepts that are utilized by the pattern generator. The recognized gene names and base concepts are replaced in the sentences before they are indexed. We use the sample sentence "*RAD53 positively regulates DBF4*" to illustrate the two approaches. In this sample sentence, protein names *RAD53* and *DBF4* are replaced with the the identifier `PROTNAME`, and *regulates* is recognized as a interaction verb in present tense, denoted by the identifer `IVERB-S`. Approach 1 replaces only the gene names in the sentence, and the replaced sentence `PROTNAME positively regulates PROTNAME` is indexed, and `PROTNAME positively IVERB-S PROTNAME` for approach 2.

Translating the PPI patterns into PTQL queries and then querying the parse tree database with 11208 PTQL queries can be very time-consuming. To maximize the efficiency of query processing, each of the PPI patterns are transformed into IR queries. The IR queries are then utilized to retrieve sentences from the testing dataset through the IR system. With approach 1, only 3071 IR queries retrieve at least 1 sentence from the testing dataset. Applying approach 2 to generate IR queries from the PPI patterns results in achieving 1314 IR queries that retrieve at least 1 sentence from the testing dataset. The PPI patterns that correspond to these reduced sets of IR queries are translated into PTQL queries, and the document and sentence identifiers returned by the IR system are used to instantiate the PTQL queries. As shown in Figure 7, we can see that the number of sentences involved are much fewer for approach 2 than approach 1, which means that the time to retrieve all results is much shorter for approach 2. It is interesting to observe that with the fewer number of queries in approach 2, the extraction results for approach 2 is better than for approach 1 both in precision and recall. The extraction results are summarized in Table IV. Our results also show

[4]See http://beta.uniprot.org

TABLE IV

PERFORMANCE OF VARIOUS APPROACHES ON THE BIOCREATIVE 2 IPS TEST DATA (VERSION: "SWISSPROT ONLY"). MEAN PRECISION, RECALL, AND F1–MEASURE IN %. THE SYSTEMS MARKED * ARE BASICALLY THE SAME; FOR ONE, WE REPLACED THEIR NER/EMN WITH OUR OWN TO GET RESULTS THAT ARE BETTER COMPARABLE.

System	P	R	F
PTQL with IR (approach 1 with 3071 queries)	76.1	57.6	61.3
PTQL with IR (approach 2 with 1314 queries)	83.6	58.6	64.2
* Reported by [30]	39.1	29.7	28.5
* 74 manually created patterns [16], our NER/EMN	59.7	37.9	41.8
Reported by [31]	37.0	32.7	30.4
Reported by [32]	25.2	23.3	24.2

far better performance than the previously top-performing PPI systems [30], [31], [32]. To make a fair comparison, we performed another experiment that utilized the 74 manually curated patterns reported in [16] on their OpenDMAP system, but using the same gene normalization and named entity recognition that we used. We observed that our PTQL with IR system still achieves significantly better results.

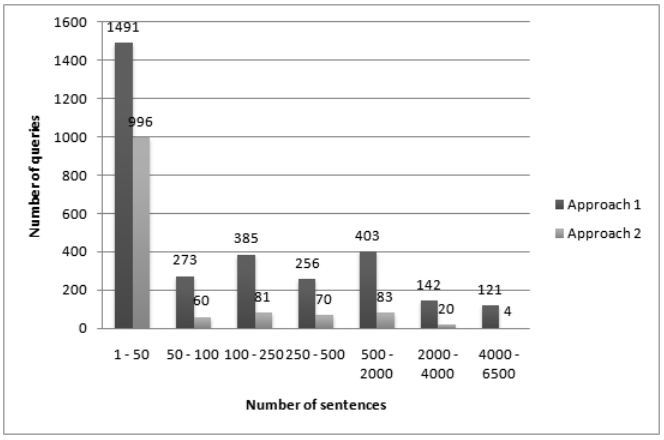

Fig. 7. Distributions of the number of sentences for the reduced sets of queries that retrieve at least 1 sentence in the testing data. Approach 1 only replace gene names with an unique identifier during indexing, and approach 2 extends into replacing base concepts with identifiers.

C. Query Processing Time

Figure 8 illustrates the query execution time for a set of 23 queries that were selected from the 1314 queries used in approach 2. While some of the queries are computationally expensive in retrieving all results, majority of the queries involve fewer than 50 sentences as indicated in Figure 7. Execution time for this kind of queries, for instance queries 1-4, is reasonably fast. The execution time reported in Figure 8 is the execution of the translated SQL queries, as the processing time is neligible for generating IR queries from PPI patterns, retrieving from the IR index and translating PTQL queries to SQL queries. We report the execution time for returning all results, since it is critical to get all extraction results. Based on our analysis, we observed that the query execution time of the translated SQL queries can be influenced by a number of factors: number of horizontal axis in the query, first node of the tree pattern and number of sentences involved. The factors of the 23 queries are listed in Table V. The number of horizontal axis in the PTQL queries is a critical factor, since this implies the number of joins that are needed in the SQL queries. The first node of the tree pattern also affects the efficiency of joins, as the subsequent joins depend on the number of results returned by the first join. For instance, queries 7 and 8 are involved in similar number of sentences, but query 7 is much more efficient. This can be explained by observing that the first node is a literal in query 8, while the first node of query 9 is of type `PROTEIN`, which is composed of over 40000 protein names. Interaction-indicating concepts, such as `i-attachment`, `i-noun-regulation`, `i-verb-s` and `i-verb-ing`, contain much fewer instances than the type `PROTEIN` with over 100 instances for each concept. By comparing queries 2 and 3, we can observe that both the number of sentences involved in the query and number of horizontal axis can have a major impact in the query execution time.

We also evaluate the performance of the 23 queries by applying to all sentences in the queries rather than only the sentences retrieved by the IR queries. This is the same as executing the PTQL queries without utilizing IR. We observe that none of the 23 queries return all answers in fewer than 2 hours. We conclude that the IR system has a major impact in the efficiency of the query execution.

Query	1st node	# axis	# sent	result size
1	`PROTEIN`	3	1	0
2	`PROTEIN`	2	1	2
3	`PROTEIN`	7	6	0
4	`PROTEIN`	5	1	0
5	`PROTEIN`	3	73	2
6	`PROTEIN`	4	81	0
7	*text literal*	5	75	0
8	`PROTEIN`	2	60	564
9	`PROTEIN`	3	187	12520
10	`i-noun-regulation`	3	114	0
11	*text literal*	2	230	0
12	`PROTEIN`	3	134	58
13	`i-verb-s`	4	458	138
14	`i-attachment`	4	498	2980
15	*text literal*	3	259	0
16	`PROTEIN`	3	364	65
17	`PROTEIN`	2	1332	1009
18	`i-noun-regulation`	3	1407	0
19	`i-attachment`	3	531	0
20	`i-noun-regulation`	5	688	0
21	`i-attachment`	4	2250	4986
22	`i-verb-ing`	5	4440	0
23	`i-verb-ing`	3	6359	0

TABLE V

PROPERTIES OF THE 23 QUERIES USED FOR COMPUTING THE QUERY EXECUTION TIME. "1ST NODE": FIRST NODE OF THE TREE PATTERN IN THE PTQL QUERY; "# AXIS": NUMBER OF HORIZONTAL AXIS IN THE QUERIES; "# SENT": NUMBER OF SENTENCES INVOLVED; "RESULT SIZE": NUMBER OF RETURNED ROWS

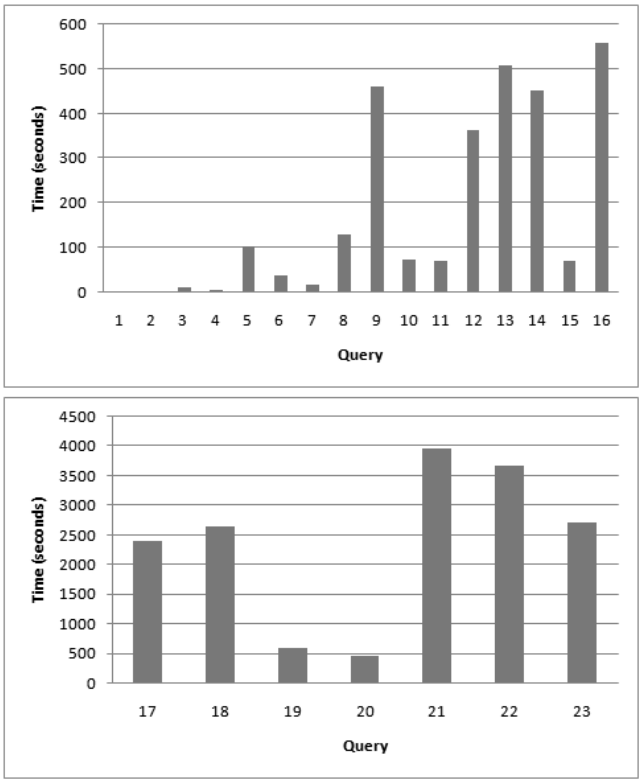

Fig. 8. Processing time for a set of PTQL queries that are selected from the 1314 queries in approach 2

VII. Conclusion

Our results showed that our protein-protein interaction extraction system based on the PTQL with IR framework achieved far superior performance than the existing systems in terms of accuracy with respect to the BioCreative 2 dataset. We also showed that a combination of information retrieval and database query processing has a major impact on the performance of query execution. This is important especially when we have a large number of linguistic patterns for the extraction of protein-protein interaction. We illustrate that our PTQL framework is also suitable for the development of other BioNLP applications.

References

[1] M. V. Blagoskionny and A. B. Pardee, "Unearthing the gems," *Nature*, vol. 416, p. 373, 2002.

[2] P. Zweigenbaum, D. Demner-Fushman, H. Yu, and K. B. Cohen, "Frontiers of biomedical text mining: current progress," *Brief Bioinform*, vol. 8, no. 5, pp. 358–375, 2007.

[3] J. Clark and S. DeRose, "XML Path language (XPath)," November 1999, http://www.w3.org/TR/xpath.

[4] "XQuery 1.0: An XML query language," June 2001, http://www.w3.org/XML/Query.

[5] S. Bird, Y. Chen, S. B. Davidson, H. Lee, and Y. Zheng, "Extending xpath to support linguistic queries," in *Workshop on Programming Language Technologies for XML (PLAN-X)*, 2005.

[6] ——, "Designing and evaluating an XPath dialect for linguistic queries," in *ICDE '06*, 2006, p. 52.

[7] Y. Miyao, T. Ohta, K. Masuda, Y. Tsuruoka, K. Yoshida, T. Ninomiya, and J. Tsujii, "Semantic retrieval for the accurate identification of relational concepts in massive textbases," in *Proceedings of ACL '06*, 2006, pp. 1017–1024.

[8] C. Lai and S. Bird, "Lpath +: A first-order complete language for linguistic tree query," in *Proc. 19th Asia-Pacific Conference on Language, Information and Computation (PACLIC)*, 2007, pp. 1–12.

[9] E. Agichtein and L. Gravano, "Querying text databases for efficient information extraction," in *ICDE*, 2003, pp. 113–124.

[10] M. Cafarella, D. Downey, S. Soderland, and O. Etzioni, "Knowitnow: Fast, scalable information extraction from the web," in *HLT-EMNLP-05*, 2005, pp. 563–570.

[11] M. J. Cafarella, C. Re, D. Suciu, and O. Etzioni, "Structured querying of web text data: A technical challenge," in *CIDR*, 2007, pp. 225–234.

[12] M. Krallinger, F. Leitner, and A. Valencia, "Assessment of the second biocreative PPI task: Automatic extraction of protein–protein interactions," in *Proc 2nd BioCreative Challenge Evaluation*, 2007, pp. 41–54.

[13] H. Shatkay and R. Feldman, "Mining the biomedical literature in the genomic era: an overview." *J Comp Biol*, pp. 821–855, 2003.

[14] R. Feldman, Y. Regev, E. Hurvitz, and M. Finkelstein-Landau, "Mining the biomedical literature using semantic analysis and natural language processing techniques," *Drug Discovery Today*, vol. 1, no. 2, 2003.

[15] J. D. Martin, "Fast and furious text mining," *IEEE Data Eng. Bull.*, vol. 28, no. 4, pp. 11–20, 2005.

[16] L. Hunter, Z. Lu, J. Firby, W. Baumgartner, H. Johnson, P. Ogren, and K. B. Cohen, "OpenDMAP: An open source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression," *BMC Bioinformatics*, vol. 9, no. 1, p. 78, 2008.

[17] M. J. Cafarella and O. Etzioni, "A search engine for natural language applications," in *Proc. of WWW 2005*, 2005.

[18] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan, "Efficient information extraction over evolving text data," in *ICDE*, 2008, pp. 943–952.

[19] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "Towards a query optimizer for text-centric tasks," *ACM Trans. Database Syst.*, vol. 32, no. 4, p. 21, 2007.

[20] A. Jain, A. Doan, and L. Gravano, "Optimizing SQL queries over text databases," in *ICDE*, 2008, pp. 636–645.

[21] D. D. Sleator and D. Temperley, "Parsing English with a link grammar," in *Third International Workshop on Parsing Technologies*, 1993.

[22] D. Grinberg, J. Lafferty, and D. Sleator, "A robust parsing algorithm for LINK grammars," Pittsburgh, PA, Tech. Rep. CMU-CS-TR-95-125, 1995. [Online]. Available: citeseer.ist.psu.edu/grinberg95robust.html

[23] C. Plake, T. Schiemann, M. Pankalla, J. Hakenberg, and U. Leser, "AliBaba: PubMed as a graph," *Bioinformatics*, vol. 22, no. 19, pp. 2444–2445, 2006.

[24] P. H. Tu, C. Baral, Y. Chen, and G. Gonzalez, "Generalized text extraction from molecular biology text using parse tree database querying," Arizona State University, Tech. Rep. TR-08-004, 2008.

[25] C. Lai, "A formal framework for linguistic tree query," Department of Computer Science and Software Engineering, University of Melbourne," Master's thesis, 2005.

[26] D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu, "A comprehensive XQuery to SQL translation using dynamic interval encoding," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 623–634.

[27] T. Cheng and K. C.-C. Chang, "Entity search engine: Towards agile best-effort information integration over the web," in *CIDR*, 2007.

[28] H. Bast and I. Weber, "The CompleteSearch Engine: Interactive, efficient, and towards IR& DB integration," in *CIDR*, 2007, pp. 88–95.

[29] J. Hakenberg, C. Plake, R. Leaman, M. Schroeder, and G. Gonzalez, "Inter–species normalization of gene mentions with GNAT," in *ECCB'08*, Cagliari, Italy, September 22–26 2008.

[30] W. Baumgartner, Z. Lu, H. Johnson, J. Caporaso, and *et al.* J. Paquette, "An integrated approach to concept recognition in biomedical text," in *Proc. of the Second BioCreative Challenge Evaluation Workshop*, 2006.

[31] M. Huang, S. Ding, H. Wang, and X. Zhu, "Mining physical protein-protein interactions by exploiting abundant features," in *Proc 2nd BioCreative Challenge Evaluation Workshop*, 2007, pp. 237–245.

[32] J. Hakenberg, C. Plake, L. Royer, H. Strobelt, U. Leser, and M. Schroeder, "Gene mention normalization and interaction extraction with context models and sentence motifs," *Genome Biology, Special Issue on BioCreative Challenge Evaluation*, 2007, to appear.