TR Discover: A Natural Language Interface for Querying and Analyzing Interlinked Datasets

Dezhao Song[1](✉), Frank Schilder[1], Charese Smiley[1], Chris Brew[2], Tom Zielund[1], Hiroko Bretz[1], Robert Martin[3], Chris Dale[1], John Duprey[3], Tim Miller[4], and Johanna Harrison[5]

[1] Research and Development, Thomson Reuters, Eagan, MN 55123, USA
{dezhao.song,frank.schilder,charese.smiley,chris.brew,thomas.zielund, hiroko.bretz,robertd.martin,chris.dale,john.duprey,tim.miller, johanna.harrison}@thomsonreuters.com
[2] Research and Development, Thomson Reuters, London, UK
[3] Research and Development, Thomson Reuters, Rochester, NY 14694, USA
[4] Intellectual Property and Science, Thomson Reuters, London, UK
[5] Intellectual Property and Science, Thomson Reuters, Philadelphia, PA 19130, USA

Abstract. Currently, the dominant technology for providing non-technical users with access to Linked Data is keyword-based search. This is problematic because keywords are often inadequate as a means for expressing user intent. In addition, while a structured query language can provide convenient access to the information needed by advanced analytics, unstructured keyword-based search cannot meet this extremely common need. This makes it harder than necessary for non-technical users to generate analytics. We address these difficulties by developing a natural language-based system that allows non-technical users to create well-formed questions. Our system, called TR Discover, maps from a fragment of English into an intermediate First Order Logic representation, which is in turn mapped into SPARQL or SQL. The mapping from natural language to logic makes crucial use of a feature-based grammar with full formal semantics. The fragment of English covered by the natural language grammar is domain specific and tuned to the kinds of questions that the system can handle. Because users will not necessarily know what the coverage of the system is, TR Discover offers a novel auto-suggest mechanism that can help users to construct well-formed and useful natural language questions. TR Discover was developed for future use with Thomson Reuters Cortellis, which is an existing product built on top of a linked data system targeting the pharmaceutical domain. Currently, users access it via a keyword-based query interface. We report results and performance measures for TR Discover on Cortellis, and in addition, to demonstrate the portability of the system, on the QALD-4 dataset, which is associated with a public shared task. We show that the system is usable and portable, and report on the relative performance of queries using SQL and SPARQL back ends.

Keywords: Natural language interface · Question answering · Feature-based grammar · Auto-suggestion · Analytics

1 Introduction

Organizations adopt Linked Data because they want to provide information professionals with seamless access to all the relevant data that is present in the organization, irrespective of its arrangement into database tables and its actual physical location. Many organizations now have effective strategies for ensuring that there are well-designed links between related records in different tables. Technical professionals, such as database experts and data scientists, will use a mix of traditional and novel database query languages to access this information. But non-technical information professionals, such as journalists and patent lawyers, who cannot be expected to learn a database query language, still need a fast and effective means for accessing the data that is relevant to the task at hand.

Keyword-based search allows non-technical users to access large-scale linked data, and it can be applied in a uniform fashion to information sources that may have wildly divergent logical and physical structure. But it does not always allow precise specification of the user's intent, so the result sets that are returned may be unmanageably large and of limited relevance. If non-technical users could write good SPARQL queries, they would get smaller and more relevant result sets. In addition, because database query languages impose structure on the result set, they can readily be used to provide dynamically generated analytics. This is not so easy to do when the results are less structured, as they will be when they come from keyword-based search.

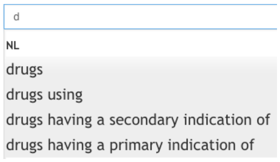
Our system, *TR Discover*, is designed to bridge the gap between keyword-based search and structured query. In our system, the user creates natural language questions, which are mapped into a logic-based intermediate language. A grammar defines the options available to the user and implements the mapping from English into logic. An auto-suggest mechanism guides the user towards questions that are both logically well-formed and likely to elicit useful answers from the available databases. A second translation step then maps from the logic-based representation into a standard query language such as SPARQL or SQL, allowing the database query to rely on robust existing technology. Since all professionals can use natural language, we retain the accessibility advantages of keyword-based search, and since the mapping from the logical formalism to the query language is information-preserving, we retain the precision of query-based information access. We also retain the ability to generate useful structured analytics.

TR Discover[1] is composed of the following components: Web User Interface, Auto-suggest, Question Understanding, FOL Parsing and Translation, Query Execution, and Analytics Generation. We present the details for each of the components in the rest of the paper: Section 2 describes use cases for *TR Discover* with Cortellis. We then formally present the different components of *TR Discover* in Sections 3 to 5, and evaluate our system in Section 6. We discuss related work in Section 7 and conclude in Section 8.

[1] Beta version available at: http://cortellislabs.com (free sign-up)

2 Use Cases

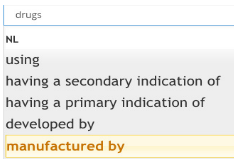
In this section, we present use cases of *TR Discover*, targeting different types of users. We describe the use cases in the context of Thomson Reuters Cortellis[2] (Cortellis). Cortellis is a data integration and search platform developed for professional users in the Pharmaceutical industry. It relies on a linked dataset that covers a variety of domains, including Life Sciences, Intellectual Property, Legal and Finance. A keyword-based query interface allows users to obtain information relevant to a wide range of tasks, such as drug repurposing, target finding, legal research about a specific drug, or search for patents owned by a particular company.



d

NL
drugs
drugs using
drugs having a secondary indication of
drugs having a primary indication of

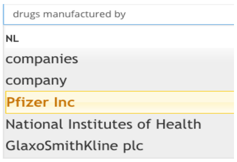
(a) "d" is typed



drugs

NL
using
having a secondary indication of
having a primary indication of
developed by
manufactured by

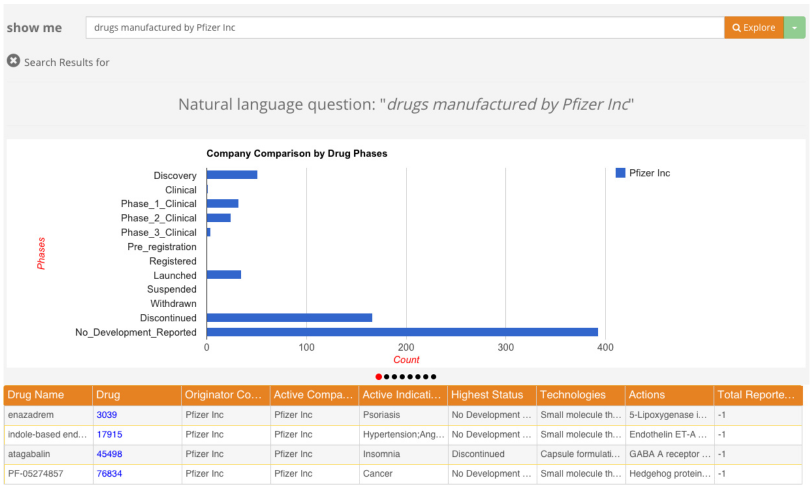
(b) "drugs" is selected and suggestions are provided



drugs manufactured by

NL
companies
company
Pfizer Inc
National Institutes of Health
GlaxoSmithKline plc

(c) "manufactured by" is picked and "Pfizer Inc" can be chosen to complete a question



drugs manufactured by Pfizer Inc

Search Results for

Natural language question: "*drugs manufactured by Pfizer Inc*"

Drug Name	Drug	Originator Co...	Active Compa...	Active Indicati...	Highest Status	Technologies	Actions	Total Reporte...
enazadrem	3039	Pfizer Inc	Pfizer Inc	Psoriasis	No Development ...	Small molecule th...	5-Lipoxygenase i...	-1
indole-based end...	17915	Pfizer Inc	Pfizer Inc	Hypertension;Ang...	No Development ...	Small molecule th...	Endothelin ET-A ...	-1
atagabalin	45498	Pfizer Inc	Pfizer Inc	Insomnia	Discontinued	Capsule formulati...	GABA A receptor ...	-1
PF-05274857	76834	Pfizer Inc	Pfizer Inc	Cancer	No Development ...	Small molecule th...	Hedgehog protein...	-1

(d) Query Results and Analytics

Fig. 1. Use Case: First-time Users of TR Discover

[2] http://thomsonreuters.com/en/products-services/pharma-life-sciences/pharma-business-development/cortellis-data-fusion.html

The second use case targets expert professional users (e.g., medical professionals, financial analysts, or patent officers). This user, User B, understands the domain, and has specific questions in mind that may require material from multiple slices of data. She need not be concerned with how the data is partitioned across database tables because she is sheltered from this level of implementation detail. Suppose User B works for a pharmaceutical company and is interested in searching for patents relevant to a particular line of drug development. Guided by our structured auto-suggest, she could pose the detailed question, *patents filed by companies developing drugs targeting PDE 4 inhibitor using Small molecule therapeutic that have already been launched.* Our system returns 12 patents for this question and from the generated analytics (Figure 2), she can immediately see a general view of the competitive field. User B can then drill further into the patents, and begin to develop a strategy that navigates around potential infringements of her competitors' protected rights, for example.

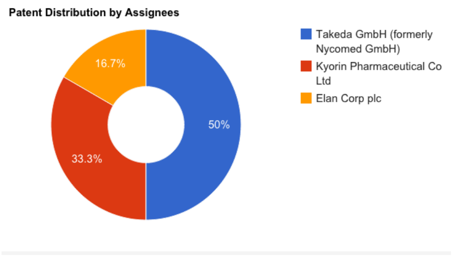


Fig. 2. Analytics for Complex Question from Professional Users

Our first use case targets first-time users of *TR Discover* or users with limited knowledge of the underlying data. This user, User A, may be interested in broad, exploratory questions; however, due to lack of familiarity with the data, guidance (from our auto-suggest module, Section 3.2) will be needed to help him build a valid question in order to explore the underlying data. Figures 1(a)-1(c) demonstrate this question building process. Assuming that User A starts by typing in *d*, *drugs* will then appear as a possible completion. He can either continue typing *drugs* or select it from the drop down list on the user interface. Upon selection, suggested continuations to the current question segment, such as *using* and *manufactured by*, are then provided to User A. Suppose our user is interested in exploring drug manufacturers and thus selects *manufactured by*. In this case, both the generic type, *companies*, along with specific company instances like *Pfizer Inc* and *Glaxo Smith Kline Plc* are offered as suggestions. User A can then select *Pfizer Inc* to build the valid question, *drugs manufactured by Pfizer Inc* thereby retrieving answers from our data stores along with the corresponding analytics (Figure 1(d)).

3 Question Understanding

3.1 Natural Language Question Parsing

In *TR Discover*, we use a feature-based context-free grammar (FCFG) for parsing natural language questions. Our FCFG consists of phrase structure rules on non-terminal nodes and lexical entries for leaf nodes. The large majority of the phrase structure rules are domain independent allowing the grammar to be portable to new domains. The following shows a few examples of our grammar rules: *G1* - *G3*. Specifically, Rule *G3* indicates that a verb phrase (*VP*) contains a verb (*V*) and a noun phrase (*NP*).

```
G1:  NP → N
G2:  NP→ NP VP
G3:  VP → V NP
Lex1: N[TYPE=drug, NUM=pl, SEM=<λx.drug(x)>] → 'drugs'
Lex2: V[TYPE=[drug,org,dev], SEM=<λX x.X(λy.dev_org_drug(y,x))>, TNS=past, NUM=?n]
→ 'developed by' /*In general, TYPE=[subject_constraint, object_constraint, predi-
cate_name]*/
Lex3: V[TYPE=[org,country,hq], NUM=?n] → 'headquartered in'
```

Each entry in the FCFG lexicon contains a variety of domain-specific features that are used to constrain the number of parses computed by the parser preferably to a single, unambiguous parse. *Lex1-Lex3* are examples of lexical entries. For instance, *Lex1* is the lexical entry for the word, *drugs*, indicating that it is of TYPE *drug*, is plural, and has the semantic representation $\lambda x.drug(x)$. Verbs (V) have an additional feature tense (TNS), as shown in *Lex2*. The TYPE of verbs specify both the potential subject-TYPE and object-TYPE, which can be used to filter out nonsensical questions like *drugs headquartered in the U.S.*

Disambiguation relies on the unification of features on non-terminal syntactic nodes. We mark prepositional phrases (PPs) with features that determine their attachment preference. For example, we specify that the prepositional phrase *for pain* must attach to an NP rather than a VP; thus, in the question *Which companies develop drugs for pain?*, *for pain* cannot attach to *develop* but must attach to *drugs*. Additional features constrain the TYPE of the nominal head of the PP and the semantic relationship that the PP must have with the phrase to which it attaches. This approach filters out many of the syntactically possible but undesirable PP-attachments in long queries with multiple modifiers, such as *companies headquartered in Germany developing drugs for pain or cancer*. In rare instances when a natural language question has multiple parses, we always choose the first parse. Future work may include developing ranking mechanisms in order to rank the parses of a question.

3.2 Enabling Question Completion with Auto-suggestion

Traditional question answering systems often require users to enter a complete question. However, often times, it may be difficult for novice users to do so, e.g., due to the lack of familiarity and an incomplete understanding of the underlying data. One unique feature of *TR Discover* is that it provides suggestions in order to help users to complete their questions. The intuition here is that our auto-suggest module guides users in exploring the underlying data and completing a question that can be potentially answered with the data. Unlike Google's query auto-completion that is based on query logs [3], our suggestions are computed based upon the relationships and entities in the dataset and by utilizing the linguistic constraints encoded in our grammar.

Our auto-suggest module is based on the idea of left-corner parsing. Given a query segment qs (e.g., *drugs*, *developed by*, etc.), we find all grammar rules whose left corner fe on the right side matches the left side of the lexical entry of qs. We then find all leaf nodes in the grammar that can be reached by using the adjacent element of fe. For all reachable leaf nodes (i.e., lexical entries in our grammar), if a lexical entry also satisfies all the linguistic constraints, we then treat it as a valid suggestion.

We rank the suggestions based upon statistics extracted from an RDF graph. Each node in the RDF graph represents a lexical entry (i.e., a potential suggestion), including entities (e.g., specific drugs, drug targets, diseases, companies, and patents), predicates (e.g., *developed by* and *filed by*), and generic types (e.g., Drug, Company, Technology, etc.). The 'popularity' (i.e., ranking score) of a node is defined as the number of relationships it is involved in. For example, if a company filed 10 patents and is also involved in 20 lawsuits, then its popularity will be 30. Our current ranking is computed based only upon the data; in future work, it may be possible to tune the system's behavior to a particular individual user by mining our query logs for similar queries previously made by that user.

There are (at least) two ways of using the auto-suggest facility. Users can work in steps: they could type in an initial question segment, like *patents*, and wait for the system to provide suggestions. Then, users can select one of the suggestions to move forward. By repeating this process, users can build well-formed natural language questions (i.e., questions that are likely to be understood by our system) in a series of small steps guided by our auto-suggest. Alternatively, users can type in a longer string, without pausing, and our system will chunk the question and try to provide suggestions for users to further complete their question. For instance, given the following partial question *drugs developed by Merck using ...*, our system first tokenizes this question; then starting from the first token, it finds the shortest phrase (a series of continuous tokens) that matches a suggestion and treats this phrase as a question segment. In this example, *drugs* will be the first segment. As the query generation proceeds, our system finds suggestions based on the discovered query segments, and produces the following sequence of segments: *drugs, developed by, Merck*, and *using*. At the end, the system knows that *using* is likely to be followed by a phrase describing a drug technology, and is able to offer corresponding suggestions to the user. In general, an experienced user might simply type in *drugs developed by Merck using*; while first-time users who are less familiar with the data can begin with the stepwise approach, progressing to a more fluent user experience as they gain a deeper understanding of the underlying data.

4 Question Translation and Execution

In contrast to other Natural Language Interfaces (NLI) [11,18], *TR Discover* first parses a natural language question into a First Order Logic (FOL) representation. The FOL representation of a natural language question is further translated to other executable queries (e.g., SPARQL and SQL). This intermediate logical representation provides us the flexibility to develop different query translators for various types of data stores.

The process of translating FOL to SPARQL/SQL can be divided into two steps. In the first step, we parse the FOL representation into a parse tree according to an FOL parser. This FOL parser is implemented with ANTLR[3] (a parser development tool). The FOL parser takes a grammar and an FOL representation as input, and generates a parse tree for the FOL representation. Figure 3 shows the parse tree of the FOL representation for the question "Drugs developed by Merck".

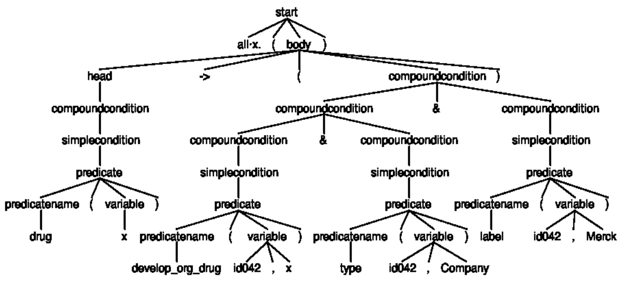


Fig. 3. The Parse Tree for the FOL of the Question "Drugs developed by Merck"

We then perform an in-order traversal (with ANTLR's APIs) of the FOL parse tree and translate it to executable queries. While traversing the tree, we put all the atomic logical conditions and the logical connectors into a stack. When we finish traversing the entire tree, we pop the conditions out of the stack to build the correct query constraints; predicates in the FOL are also mapped to their corresponding attribute names (SQL) or ontology properties (SPARQL).

The following summarizes the translation from a natural language question to a SQL and SPARQL query via a FOL representation:

Natural Language Question: `Drugs developed by Merck`
FOL: `all x.(drug(x) → (develop_org_drug(id0,x) & type(id0,Company) & label(id0,Merck)))`
SQL Query: `select drug.* from drug where drug.originator_company_name = 'Merck'`
SPARQL Query (prefixes are omitted):

```
    select ?x
    where {
    ?id0 rdfs:label 'Merck'.
    ?id0 rdf:type example:Company .
```

[3] http://www.antlr.org/

```
    ?x rdf:type example:Drug .
    ?id0 example:develops ?x .
    }
```

We execute the translated SQL queries using SQLite, a light weight relational database management system that allows us to store the entire database as a single file on disk. We run the SPARQL queries in a Jena TDB triple store[4]. An additional query is issued for SPARQL queries to retrieve all attribute values of the entities in the result set.

5 Analytics Generation

This section details two different kinds of analytics that are generated for the result set for a given question. We developed these analytics for the Cortellis dataset (Section 2) and future work will include generating more analytics based on the content type and the questions derived from different use cases.

Descriptive and Comparative Analytics. Query results are analyzed according to the counts of the results, as shown in Figure 2. In addition to the counts of the result set, descriptive analytics are also shown across different dimensions (e.g., indication, technology, action, etc.) for drugs. Moreover, result sets can be compared across these dimensions via related entities, such as companies. Figure 4 demonstrates a concrete example of comparative analytics. Here, companies in the result set are compared against various phases of the drug development. This chart shows that *Signature Therapeutics* has the most drugs in the Discovery phase even though *Pfizer* has the most drugs across all phases.

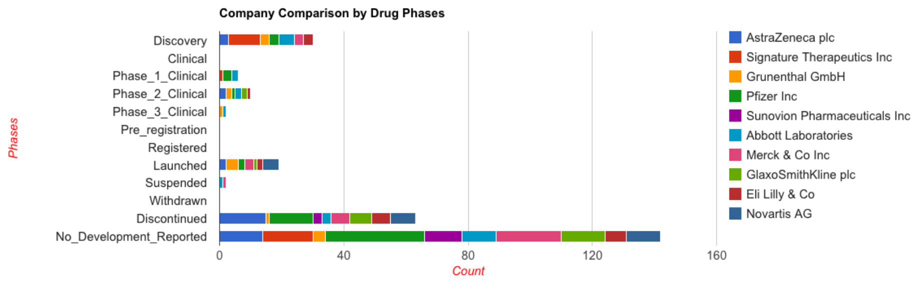


Fig. 4. Comparing companies on the dimension of drug development phase for "Drugs having a primary indication of pain"

Content Analysis. We also developed content-based analytics. To support this type of analytics, we applied named entity recognition (NER) and sentiment analysis. For NER, we used the Stanford CoreNLP toolkit [12] to recognize person, organization, and locations from the Reuters News Archive (RNA). There

[4] https://jena.apache.org

are about 14 million documents and 147 million sentences in the entire RNA dataset and the named entity recognition is done in a distributed environment using Apache Spark. The entire process took roughly 48 hours and discovered about 280 million entities. As a second step, we ran an open-source sentiment

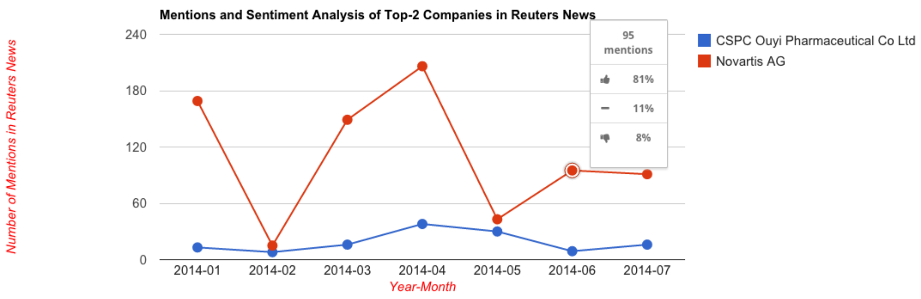


Fig. 5. NER and Sentiment Analysis Results for Top-2 Companies in Question "companies developing drugs having a primary indication of hypertension"

analysis tool over the entire corpus [14]. Given an entity from the outcome of the NER process, we retrieve documents from RNA that contain this entity. For each document, we then find all sentences that contain this entity and perform sentiment analysis on each of the sentences. The outcome for each sentence could be: Positive, Neutral, or Negative. Finally, we determine the overall sentiment of this document on the given entity by majority vote on the sentence-level results.

Figure 5 demonstrates our sentiment analysis results for the question *Companies developing drugs having an indication of Hypertension*. Here, we pick the top two companies that are most frequently mentioned in RNA in order to avoid overwhelming users with a dense chart. When the mouse hovers over a data point, the system displays the sentiment results. In this example, we can see that in June 2014, there are 95 news articles that mention "Novartis AG", with 81% of such documents having a positive sentiment towards the company.

6 Evaluation

In this section, we present the evaluation results of *TR Discover*. First, we evaluate the response time of the different components of *TR Discover*. Then, we apply our proposed system to Task 2 (Biomedical question answering over interlinked data) of QALD-4 [19] to compare its precision and recall to state-of-the-art question answering systems in the biomedical domain.

6.1 Runtime Evaluation

Dataset. We evaluate the runtime of the different components of *TR Discover* on the Cortellis dataset, which consists of about 1.2 million entities. Our dataset is actually integrated from three sources: Cortellis drug data, a Thomson Reuters patent dataset, and DrugBank. Using string matching, we linked Cortellis' companies to patent assignees, and the drugs names between Cortellis and DrugBank. Thus, complementary information from different sources can be presented to users. The different entity types include *Drug, Drug_Target, Company, Technology, Patent*, etc. Various types of relationships exist between the entities, including *Using* (Drug uses Technology), *Developing* (Company develops Drug), *Headquartered in* (Company headquartered in Country), etc.

Since we can translate the logical representation of a natural language question to both SPARQL and SQL, we prepared two different data stores for our dataset. We store the Cortellis dataset into a relational database using SQLite; and, in order to be able to run SPARQL queries, we convert the relational data to triples and store them into a Jena TDB triple store. Take the above examples again: *Drug, Drug_Target, Company, Technology*, and *Patent* all become classes, while *Using, Developing*, and *Headquartered in* become predicates in our RDF data. This data transformation process produces about 12 million triples.

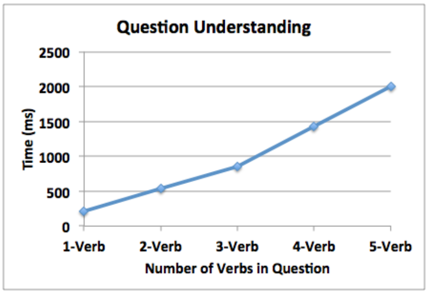
Random Question Generation. In order to evaluate the runtime of our proposed system, we randomly generated a total number of 5,000 natural language questions using our auto-suggest component (Section 3.2). Recall that our auto-suggest module provides suggestions as potential next steps in order to help users to complete their questions, thus making it also useful for generating random testing questions. We give the auto-suggest module a starting point, e.g. *drugs*, and then perform a depth-first search to uncover all possible questions. At each depth, for each question segment, we randomly select b suggestions; we then continue this search process with each of the b suggestions. By setting different depth limits, we generate questions with different levels of complexity (or different number of verbs). Using this random question generation process, we generated 1,000 natural language questions for each number of verbs from 1 to 5, thus 5,000 questions in total.

Runtime Results. We evaluate on a 16-core RedHat machine with 2.90GHz CPU and 264GB of memory. Figure 6(a) shows the parsing time of natural language questions. Although we adopt NLTK[5] for parsing natural language questions in our system (by supplying NLTK with our own grammar and lexicon), this evaluation is to show the practicability of the overall approach in potential real-world scenarios. According to Figure 6(a), unless a question becomes truly complicated (with 4 or 5 verbs), the parsing time is generally under 1 second. One example question with 5 verbs could be *Patents granted to companies headquartered in Australia developing drugs targeting Lectin mannose binding protein modulator using Absorption enhancer transdermal.* Experts on the Cortellis

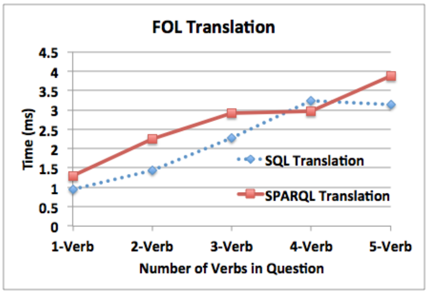
[5] http://www.nltk.org/

Team assure us that questions with more than 5 verbs are rare, thus we did not evaluate questions beyond this level of complexity. Although the parsing time increases as a question becomes more complicated, we did not observe an exponential increase in our experiments.

Figure 6(b) shows the runtime for translating the FOL of a natural language question to SPARQL and SQL queries. In general, for both SPARQL and SQL, the translation time increases as the questions become more complicated, as the FOL translation module needs to traverse bigger FOL parse trees. However, in general, only a few milliseconds are needed for performing each translation, it should not add much burden to the overall runtime of our system.



(a) Question Understanding



(b) FOL Translation

Fig. 6. Runtime Evaluation: Question Understanding and FOL Translation

Finally, we demonstrate the query execution time of both SPARQL and SQL in Figure 7. Generally speaking, SQL queries run faster than SPARQL queries

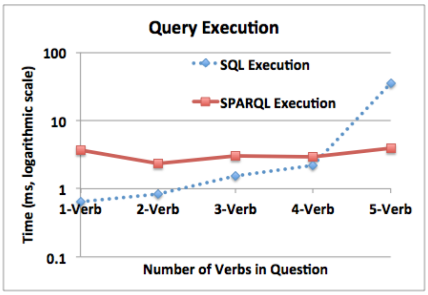


Fig. 7. Runtime Evaluation: SQL and SPARQL Query Execution

(for questions with up to 4 verbs). However, for really complicated questions (i.e., those with 5 verbs), SQL queries took much longer to finish than SPARQL queries did. One potential reason is that many joins are usually performed for the SQL query of 5-verb questions, which could greatly slow down the query execution process. Consider the above 5-verb question again, its SQL Translation actually contains 6 joins between different tables.

6.2 Evaluation on the QALD-4 Benchmark

We also evaluate *TR Discover* on Task 2 (i.e. Biomedical question answering over interlinked data) of QALD-4 [19], using the FOL to SPARQL translation.

Dataset. The QALD-4 competition provides an RDF dataset, training and testing questions, and ground truth answers to the questions[6]. We loaded the data into a Jena TDB triple store. We also materialized the triples based upon *owl:sameAs* statements. For example, given explicit triples: *a owl:sameAs b. b hasPossibleDrug c.*, we then add the following triple *a hasPossibleDrug c* into our triple store. By adding such additional triples, we then do not have to explicitly add *owl:sameAs* triple patterns when performing SPARQL translation. We use the competition's online evaluation tool to calculate the precision and recall. Our evaluation did not involve any human evaluators.

Evaluation Results. In Table 1, for the training questions, *TR Discover* was able to achieve a comparable recall to the state-of-the-art systems but with a much lower precision. The main reason for having low precision is that we added the materialized triples according to the *owl:sameAs* statements in the original dataset. Many of the returned results of *TR Discover* are actually correct, but they are not in the provided ground truth. They are linked (either explicitly or implicitly) to the instances in the ground truth via *owl:sameAs* statements. In order to report on a more realistic precision number, we augmented the ground truth by adding the equivalent instances to those already in the ground truth and achieved higher performance: 80% of precision and 92% of recall, i.e., *TR Discover+*. Similar to the training set, *TR Discover+* also achieves a much better precision by using the augmented ground truth on the testing set. For *TR Discover+*, we implemented an evaluation tool in Java; our evaluation tool produces the exact results for *TR Discover* as the online evaluation tool does.

Table 1. Evaluation Results on Task 2 of the QALD-4 Benchmarck

Dataset	System	Precision	Recall	F1	Dataset	System	Precision	Recall	F1
Training	*TR Discover*	0.44	0.88	0.58	Testing	*TR Discover*	0.34	0.80	0.48
	TR Discover+	0.80	**0.92**	**0.85**		*TR Discover+*	0.75	0.84	0.79
	GFMed [13]	N/A	N/A	N/A		*GFMed* [13]	**1.00**	**0.99**	**0.99**
	POMELO [8]	**0.83**	0.87	**0.85**		*POMELO* [8]	0.82	0.87	0.85
	RO_FII [19]	N/A	N/A	N/A		*RO_FII* [19]	0.16	0.16	0.16

* We did not find the publication of *RO_FII*. Please refer to the QALD-4 paper for details.

In Table 1, we employ fuzzy string matching on literal values. Here, we also study the impact of adopting exact string matching on the performance of our system. In Table 2, by employing fuzzy matching, we achieve higher recall.

[6] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=task2&q=4

Table 2. Fuzzy String Matching (Fuzzy) vs. Exact String Matching (Exact) on Literals

Dataset	System	Precision	Recall	F1	Time (s)	Dataset	System	Precision	Recall	F1	Time (s)
Training	Exact	**0.87**	0.88	**0.87**	**7**	Testing	Exact	0.55	0.67	0.60	**3**
	Fuzzy	0.80	**0.92**	0.85	50		Fuzzy	**0.75**	**0.84**	**0.79**	20

Although fuzzy matching results in lower precision on the training questions, the overall F1-score was only slightly impacted. Different from the training questions, fuzzy matching leads to a much better precision and recall on the testing questions, because the identified entities and literal values in the testing questions often times do not match exactly with the underlying data. We also measure the runtime of the two matching approaches, and it is natural to observe that fuzzy matching requires a much longer time to perform the translated SPARQL queries. When running queries on even larger datasets, the trade-off between precision, recall, and runtime needs to be taken into account.

6.3 Discussion

We presented *TR Discover* that can be used by non-technical professionals to explore complex interlinked datasets. *TR Discover* relies on a feature-based context free grammar for question parsing. The grammar represents about 2 months of design and experimentation, with approximately 60 grammar rules and 1 million lexical entries. The lexical entries are automatically created using the attribute values in our database, and only the grammar rules are handcrafted for question types used in *TR Discover*. Our grammar covers conjunctions (*and/or*), noun phrases with optional quantifiers (*all* and *some*), nominal and adjectival modifiers, and verbal constructions. Given a new domain, the core grammar remains stable and new lexical entries can be automatically added. The adaptation to QALD-4 took roughly 30 person hours, including adding the types (e.g., enzymes, side-effects) and their relationships (e.g., associatedWith(gene, gene)), and grammar rules to cover syntactic differences in the QALD-4 questions.

Furthermore, we perform some error analyses to study the questions that *TR Discover+* failed to answer properly. Our system has very low precision for Training Question 5: "Which genes are associated with breast cancer?". Our system finds two *Disease* instances with the exact *rdfs:label* "breast cancer" and returns their associated genes. However, the gold SPARQL query uses another *Disease* instance http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/1669 with *rdfs:label* "Breast cancer-1" and returns its associated genes. Consequently, our system gives many false positives to this question.

Our system also has difficulties understanding Training Question 14: "Give me drug references of drugs targeting Prothrombin". Our system interprets two predicates from this question as follows: *drug_reference* between *drug* and *drug_reference*, and *target* between *drug* and *target*. However, the gold SPARQL query indicates that the *drug_reference* predicate is actually between *target* and *drug_reference*. For such questions, additional domain expertise may be required in order for us to build our question understanding module to achieve more accurate parsing. A similar situation also applies to Testing Question 21: "Give me the drug categories of Desoxyn". Through our natural language question understanding process, we interpret one predicate from this question: *brandName* between *drug_category* and the literal value "Desoxyn"; however, the gold SPARQL query indicates that *drug_category* should be an object property rather than a class. This misinterpretation results in an empty result set for this question.

We also notice a potential error in the ground truth for Training Question 19: "Which are the drugs whose side effects are associated with the gene TRPM6?", which looks for drugs that satisfy certain constraints. However, in the ground truth, a *Disease* instance is given as the answer. The ground truth SPARQL query also looks for diseases rather than drugs, thus we think there might be an error in the ground truth for this question.

One limitation of *TR Discover* is that it lacks the ability to translate natural language questions with quantifiers, e.g., Testing Question 3: "which drug has the highest number of side effects?" Testing Question 14 and 23 are also of this category. Also, negation is only supported with our SPARQL translation.

7 Related Work

Keyword search [4,6,17] and faceted search [7,23] have been frequently adopted for retrieving information from knowledge bases (KB). However, users may have to figure out the most effective queries in order to retrieve relevant information. Furthermore, without appropriate ranking methods, users may be overwhelmed by the information available in the search results. In contrast, our system allows users to ask questions in natural language format, which enables users to express their search requests in a more intuitive way.

Much of the prior work on question answering over linked data parses a natural language question with various NLP techniques, maps the identified entities, concepts and relationships to instances, classes and properties in an ontology to construct a SPARQL query, and retrieves answers from a triple store [9,11,15,18,21]. In addition to adopting fully automatic query parsing, CrowdQ also incorporates crowd sourcing techniques for understanding natural language questions [5]. Instead of only using structured data, HAWK [20] utilizes both structured and unstructured data for answering natural language questions.

Significant efforts have also been devoted to developing question answering systems in the biomedical research field [1]. Questions are first analyzed to identify entities (e.g., person, location, disease, gene, etc.) and relationships. The identified entities are then mapped to concepts in a taxonomy/ontology or linguistic resources for query expansion [10,22]. An information retrieval query is finally issued to an index to find matching documents, which are then ranked and summarized to produce the final answer.

Compared to state-of-the-art systems, in this work, we maintain flexibility by first parsing a question into First Order Logic, which is further translated into both SPARQL and SQL. Using FOL allows us to be agnostic to which query language (e.g., SQL and SPARQL) will be used later. We do not incorporate any query language statements directly into the grammar, keeping our grammar leaner and more flexible for adapting to other query languages. Furthermore, one distinct feature of our system is that it helps users to build a complete question by providing suggestions according to a partial question and a grammar. For example, when users type in *drugs*, our system will suggest: *developed by*, *having a primary indication of*, etc., since these are the options to form valid questions according to our grammar. Finally, to the best of our knowledge, none of existing NLIs provide dynamic analytics for the results. Given different types of entities and their dimensions, our system performs descriptive analytics and comparisons on various dimensions of the data, and conducts sentiment analysis. Such analytics would support users to better conduct further analyses and derive insights from the data. Although ORAKEL [2] also maps a natural language question to a logical representation, there is no auto-suggest and analytics provided to the users. Compared to our prior work [16], in this paper, we provide a more fluent user experience of auto-suggest, and we perform a thorough evaluation of our system by examining the runtime of its various components and by comparing to state-of-the-art systems on the QALD-4 Benchmark.

8 Conclusion and Future Work

In this paper, we propose *TR Discover*, a natural language question answering system over interlinked datasets. *TR Discover* was designed with non-technical information professionals in mind in order to allow them fast and effective access to large-scale interlinked datasets. Going beyond keyword-based search, *TR Discover* produces precise result sets and generates analytics for natural language questions asked by information professionals, such as journalists or patent lawyers. Rather than asking users to provide an entire question on their own, *TR Discover* provides suggestions (i.e., auto-suggest) in order to facilitate this question building process. Given a completed natural language question, *TR Discover* first parses it into its First Order Logic (FOL) representation, by using a feature-based grammar with full formal semantics derived from interlinked datasets. By further translating the FOL representation of a natural language question into different executable queries (SPARQL and SQL), our system retrieves answers from the underlying data stores and generates corresponding analytics for the results. Through our runtime evaluation, we show that the overall response time of *TR Discover* is generally acceptable ($<$ 2 seconds). *TR Discover* also achieves comparable precision and recall to that of state-of-the-art question answering systems on the QALD-4 benchmark. In future work, we plan to develop personalized auto-suggestion by using user query logs, and apply *TR Discover* on more and larger datasets to examine the response time of its various components. Furthermore, it would be interesting to seek feedback from real users on the performance and usability of our system. Finally, we plan to better handle synonyms, e.g., "medicines" for "drugs".

References

1. Athenikos, S.J., Han, H.: Biomedical question answering: A survey. Comput. Methods Prog. Biomed. **99**(1), 1–24 (2010)
2. Cimiano, P., Haase, P., Heizmann, J., Mantel, M., Studer, R.: Towards portable natural language interfaces to knowledge bases - the case of the ORAKEL system. Data Knowl. Eng. **65**(2), 325–354 (2008)
3. Cornea, R.C., Weininger, N.B.: Providing autocomplete suggestions (February 4, 2014). US Patent 8,645,825
4. d'Aquin, M., Motta, E.: Watson, more than a semantic web search engine. Semantic Web Journal **2**(1), 55–63 (2011)
5. Demartini, G., Trushkowsky, B., Kraska, T., Franklin, M.J.: CrowdQ: crowdsourced query understanding. In: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research (2013)
6. Ding, L., Finin, T.W., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the 2004 ACM International Conference on Information and Knowledge Management, pp. 652–659 (2004)
7. Hahn, R., Bizer, C., Sahnwaldt, C., Herta, C., Robinson, S., Bürgle, M., Düwiger, H., Scheel, U.: Faceted wikipedia search. In: Abramowicz, W., Tolksdorf, R. (eds.) BIS 2010. LNBIP, vol. 47, pp. 1–11. Springer, Heidelberg (2010)
8. Hamon, T., Grabar, N., Mougin, F., Thiessard, F.: Description of the POMELO system for the task 2 of QALD-2014. In: Working Notes for CLEF 2014 Conference, pp. 1212–1223 (2014)
9. Lehmann, J., Furche, T., Grasso, G., Ngomo, A.N., Schallhart, C., Sellers, A.J., Unger, C., Bühmann, L., Gerber, D., Höffner, K., Liu, D., Auer, S.: DEQA: deep web extraction for question answering. In: 11th International Semantic Web Conference, pp. 131–147 (2012)
10. Lin, R.T.K., Chiu, J.L., Dai, H., Tsai, R.T., Day, M., Hsu, W.: A supervised learning approach to biological question answering. Integrated Computer-Aided Engineering **16**(3), 271–281 (2009)
11. Lopez, V., Pasin, M., Motta, E.: AquaLog: an ontology-portable question answering system for the semantic web. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 546–562. Springer, Heidelberg (2005)
12. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pp. 55–60 (2014)
13. Marginean, A.: GFMed: Question answering over biomedical linked data with grammatical framework. In: Working Notes for CLEF 2014 Conference, pp. 1224–1235 (2014)
14. Narayanan, V., Arora, I., Bhatia, A.: Fast and accurate sentiment classification using an enhanced naive bayes model (2013). CoRR abs/1305.6143
15. Shekarpour, S., Ngomo, A.N., Auer, S.: Question answering on interlinked data. In: 22nd International World Wide Web Conference, pp. 1145–1156 (2013)
16. Song, D., Schilder, F., Smiley, C., Brew, C.: Natural language question answering and analytics for diverse and interlinked datasets. In: The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 101–105 (2015)

17. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: weaving the open linked data. In: Aberer, K., et al. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 552–565. Springer, Heidelberg (2007)
18. Unger, C., Bühmann, L., Lehmann, J., Ngomo, A.N., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: Proceedings of the 21st World Wide Web Conference, pp. 639–648 (2012)
19. Unger, C., Forascu, C., Lopez, V., Ngomo, A.N., Cabrio, E., Cimiano, P., Walter, S.: Question answering over linked data (QALD-4). In: Working Notes for CLEF 2014 Conference, pp. 1172–1180 (2014)
20. Usbeck, R., Ngonga Ngomo, A.C., Bühmann, L., Unger, C.: HAWK - hybrid question answering over linked data. In: 12th Extended Semantic Web Conference (2015)
21. Yahya, M., Berberich, K., Elbassuoni, S., Weikum, G.: Robust question answering over the web of linked data. In: 22nd ACM International Conference on Information and Knowledge Management, pp. 1107–1116 (2013)
22. Yu, H., Cao, Y.G.: Using the weighted keyword model to improve information retrieval for answering biomedical questions. Summit on translational bioinformatics, p. 143 (2009)
23. Zhang, X., Song, D., Priya, S., Daniels, Z., Reynolds, K., Heflin, J.: Exploring linked data with contextual tag clouds. Journal of Web Semantics **24**, 33–39 (2014)