Swati Verma, Debasish Bhowmick

1 Introduction

Generally, two types of data are available, structured and unstructured data. In case of unstructured data, data can be of any type, not necessarily follow any format or rule, e.g. text, video, sound etc. On the other hand, in case of structured data, the data is organized in semantic chunks(entities). Similar entities are grouped together to form relations or classes. i.e. in case of structured data, some schema information is available. As an example ExDB,YAGO have schema items numbering in the millions. So, given such huge schema information, writing a structured query is a very hard task. This is called Information Overload Problem.

1.1 Motivation

As a naive approach to the above problem, one can use keyword search. But, this comes at a loss of expressivity. i.e. users can not express desired structure in the query and can not take advantage of schema information. But, it is very flexible, in the sense that a user can query even if he has no knowledge of the underlying schema information. On the other hand, conventional structured query is expressive but lacks flexibility. So, the authors propose a new approach that combines flexibility of keywords as well as expressivity of structured query, to provide a new query language, called keyword-based structured query language. e.g. say the information need is “find all people of German nationality who have won a Nobel award”. Then, the structured query would be “q(x):- GERMAN,PEOPLE(x), hasWonPrize(x, y), NOBEL_PRIZE(y)”. The keyword query would be “german, has won nobel award”, and the keyword based-structured query would be “german, has won(nobel award)”.

1.2 Contribution

The keyword based structured query language has both expressivity and flexibility. But, it introduces ambiguity as it has keywords as the basic level constructs. So, the authors also propose a model for efficient disambiguation(as well as for partial disambiguation). They also integrate full-text search when full disambiguation can not be achieved. They verify the above by extensive experiments.

1.3 Previous Work

Mapping keywords into structured data graphs has been studied in the context of keyword search over relational databases. Candidate network graphs are generated based on location of keyword matches in this problem [4]. This previous work was based on efficiently generating and ranking the candidate networks while, authors fix the shape of the disambiguation graph by the structure of the query. The WebTables project [2] also processes keyword queries over large web-extracted data sets, but don’t consider the structured queries. Also, different forms of structured query languages over web-extracted data, like NAGA[5], falls into the previous work part. The EntityRank system[3], on the otherhand, incorporates keywords into structured queries as so called context keywords. But, here the entity types are explicitly given by the user with no ambiguity. A form of KB-based semantic search over text databases making use of keyword search interfaces, is ESTER[1], which is very successful.
2 Proposed Solution

A user enters keyword-based structured query $Q$ to the system. Each keyword $k$ in $Q$ is matched to a set of schema items using some syntactic similarity measure, with the help of the $KB$. Each such set forms a partition. Now, a disambiguation graph $G$ is generated from these. Any induced subgraph of $G$, that spans all the partitions, corresponds to a concept query interpretation of $Q$. Now, these interpretations are ranked based on some score function that combines semantic and syntactic similarities with the original query $Q$, given the $KB$. Next, top $k$ of those structured queries are evaluated to find potentially relevant entities and their corresponding documents. The documents are then returned to the user ranked by some relevance metric.

2.1 System Overview

Here, we focus only on the query language and the query disambiguation phase. Some preprocessing is required for these. So, first, the information extraction tool is run over the document corpus to mine structured information from the text (e.g., entities, relationships etc.). Then, an inverted index is built over the entities and documents. Also, the knowledge base is indexed to support efficient entity search within the knowledge base.

A document query specifies documents of interest based on entities that qualify for the structured keyword queries mentioned in it. So, a document query $DQ$ is either a structured keyword query $Q_i$ or a conjunction of the $Q_i$s. If $k$ is a keyword phrase, then a structured keyword query $Q$ is either a primitive concept or a relationship or a conjunction of these two.

Given a structured keyword query $Q$. Let $M(k)$ denotes the set of possible schema item (entity/concept/relation) mappings of keyword $k$. Now, a disambiguation graph is generated, where the vertex set contains all the $M(k)$s. The edges are generated using an edge-generating function. Now, any induced subgraph of $G$ that spans all partitions $M(k)$s corresponds to a concept interpretation of $Q$.

2.2 The Scoring Model

Now, the interpretations are ranked based on some scoring model. A scoring model computes some score based on syntactic and semantic similarities. For syntactic matching edit-distance, q-gram distance, keyword occurrence etc. can be used. For semantic matching Jaccard coefficient, dice coefficient etc. can be used. Now, given a disambiguation graph, the score (or weight) of each edge changes depending on which other edges are part of the subgraph forming the candidate concept interpretation. This leads to high precomputations. So, we need some optimization. We define approximate score of a concept $C$ represented by a subgraph $G$ with respect to query $Q$, knowledge base $KB$ and binary aggregation operator $\bigoplus$ as,

$$\text{score}(G, Q, KB) = (\sum_{(v_1, v_2) \in E} w((v_1, v_2))) \bigoplus \sum_{v \in V} w(v),$$

where the first and second component in the RHS corresponds to semantic and syntactic similarities respectively.

2.3 Solving the Disambiguation Problem

Here, the goal is to find out top-$k$ maximally scoring subgraphs (corresponding to concept interpretations of the original query) which are single connected components. If we only take the maximum scoring subgraph for each subquery, then the resulting graph may not be connected, i.e., we may have no concept interpretation. If our scoring function is monotonic, then we can implement a rank-join algorithm to efficiently find the top scoring subgraphs, given that we have access to vertices in sorted order and edges in sorted order or has random access to edges.

2.4 Integrating Full Text Search

It may happen that no subgraph is found that spans all partitions in a disambiguation graph for some query. This may occur due to a failure to generate good candidate partitions, or due to a lack of coverage in the underlying KB. So, we can go for partial interpretations, and favor those documents which contain the additional keywords by re-ranking the document results. For efficiency, we can also use the document index to do a keyword search, and then boosting the documents which are relevant to the entities returned by the partial interpretations. So, in worst case, it reduces to a keyword search system.
2.5 Query Results and Ranking

The entities could be ranked by their relationships to the query in the KB, or based on statistics over the document corpus. We can also rank a document-list by the relevance of the corresponding entities to the documents. We can combine the above two approaches by ranking groups of documents per entity, and ranking the groups based on an overall ranking of entities.

3 Experimental Evaluation

- **QUICK**: It takes a structured query and finds the entities of interest and returns the documents relevant to the entities. It uses OpenNLP library as the information extraction component. For indexing purpose it uses the Lucene library. It also uses YAGO for KB items. They have also considered extended version of YAGO, so that it can evaluate disambiguations in the case where the knowledge base is incomplete as well as the case when full disambiguations are possible.

- **IR-AND**: Here, keywords are encoded using AND semantics and the qualifying documents must contain all of the given keywords in the query.

- **IR-OR**: Same as the previous, except we have to use OR semantics instead of the AND semantics.

- **Syntax-Only**: The closest syntactic match for each relation, entity/concept are chosen for disambiguation to a formal query (by replacing the keywords), and semantic similarities are unused.

- **Concept-Query**: It uses hand-coded (i.e., manually disambiguated) structured concept queries.

As a dataset, 2008 version of YAGO is used. Also, to form keyword queries, TREC 2007 Question Answering system is used. A total of 22 queries is being used.

3.1 Experimental Results

- **Disambiguation Task**: We consider a failed (or empty) disambiguation as a correct behaviour when no disambiguation is possible due to a lack of coverage in the underlying KB. By taking this assumption, the system computes a correct disambiguation around 95% of the time using the Raw KB. But, in case of Extended KB, it drops to about 90%. This decreased performance is due to the fact that, it tries to give a disambiguation for every query. So, it commits errors.

- **Entity Search Task**: Here, three variations of QUICK are used, based on keyword occurrence, edit distance and q-gram distance for syntactic similarity matching. All the results are compared against the baseline Concept-Query system which returns correct results by definition. It is found that all of the configurations of QUICK greatly outperform the simple IR and Syntax-Only baselines in terms of precision, recall and $F_1$-score.

- **Document Retrieval Task**: Here, the goal is to find documents relevant to the entities described by the query. The authors pooled the top-10 results from each system and manually assessed relevance. Relevance scores were assigned on a 0 to 4 scale. Scores 0 to 2 is marked as irrelevant and scores 3 to 4 is marked as relevant for precision measures. Here, also the QUICK systems perform better than any other existing systems, in terms of NDCG, MAP and precision at 10 measures.

- **System Performance**: It is found that the disambiguation phase took on average 0.2 seconds, while the full system took on average 0.81 seconds for answering user queries. Generally, KB-search is the most expensive task. Sometimes, document retrieval task dominates. This happens when a large number of documents need to be retrieved for a large number of qualifying entities for some queries.

4 Discussion

- Authors have used the TREC system to formulate the concept queries. But it does not contain any variable term (x) that was initially present while giving an example of a structured query.
• Authors used some specific systems like YAGO, TREC, QUICK etc which are open-source. What happens if these underlying systems are replaced with some other systems. Will the suggested strategies outperform in those cases too. Also, the system highly relies on the indexing, information extraction and Knowledge base components. What if any of these components are changed and what would be the corresponding impact to the overall system performance.

• Only 22 queries are taken into consideration which may not be quite sufficient to evaluate system performance. Also, what is the reason for considering 22 queries,basically why it is 22. Also, what happens or how the system performs in a real world scenario,where various types of inputs may be given by users, that also remains questionable.

• The size of $M(k)$ is varied from 10 to 100 during the experiments. But, it is seen that it has almost no impact to the quality of the result. So, top 10 schema items matching, corresponding to a keyword is sufficient. If this holds, then why to use a rank join algorithm to find top-k results. We can efficiently find top-k results by using a naive approach.

• If the user enters any keyword based structured query which contains similar keywords, then the partition sets $M(k_i)$s would no longer be disjoint, and so we can not apply the specified approach as there may be cycles in the resulting disambiguation graph.

• In the system performance curve, in fig.11 in the paper, authors do not mention what is the value of $k$, and how many queries they evaluate. Because, in the x-axis, they mention it as query, but what it actually signify, is not very clear. So, the system performance remains a little vague.

• In case of document ranking, authors have considered only manual evaluation. So, how do we compare the system with the existing systems when presented it online. How to rank the documents in that case. What would be the proper approach.

• Why the precision of IR-OR is less than that of precision of IR-AND. Because, documents retrieved by IR-AND will be a subset of the documents retrieved by IR-OR. So, precision of IR-OR should be larger than the precision of IR-AND theoretically. So, how come the reverse happens in simulations thats a question.

References


