Provenance for Natural Language Queries

DEVELOPMENT EFFORT

By

Krunal Vipinchandra Sevak [A20405875]

Parth Girishkumar Patel [A20405581]

Tai Nguyen [A20356881]

Group 6

Spring 2018

Presented to the

College of Science

Illinois Institute of Technology

ACKNOWLEDGMENTS

The support of Illinois Institute of Technology's Center for Computer Science is gratefully acknowledged. Thanks are also due to Prof. Boris Glavic for many helpful discussions, to Xing Niu for reviewing this on an earlier draft of this paper.

ABSTRACT

Multiple lines of research have developed Natural Language (NL) interfaces for formulating database queries. We build upon this work, but focus on presenting a highly detailed form of the answers in NL. The answers that we present are importantly based on the provenance of tuples in the query result, detailing not only the results but also their explanations. We develop a novel method for transforming provenance information to NL, by leveraging the original NL query structure. Furthermore, since provenance information is typically large and complex, we present two solutions for its effective presentation as NL text: one that is based on provenance factorization, with novel desiderata relevant to the NL case, and one that is based on summarization. We have implemented our solution in an end-to-end system supporting questions, answers and provenance, all expressed in NL. Our experiments, including a user study, indicate the quality of our solution and its scalability.

INTRODUCTION

Developing Natural Language (NL) interfaces to database systems has been the focus of multiple lines of research (see e.g. [35, 4, 34, 48]). In this work we complement these efforts by providing NL explanations to query answers. The explanations that we provide elaborate upon answers with additional important information, and are helpful for understanding why does each answer qualify to the query criteria. Even with this constraint, there may still be exponentially many (in the size of the provenance expression) compatible factorizations, and we look for the factorization with minimal size out of the compatible ones; for comparison, previous work looks for the minimal factorization with no such "compatibility constraint". propose summarized explanations by replacing details of different parts of the explanation by their synopsis, e.g. presenting only the number of papers published by each author, the number of authors, or the overall number of papers published by authors of each organization. We have implemented our solution in a system prototype called NLProv [15], forming an end-to-end NL interface to database querying where the NL queries, answers and provenance information are all expressed in Natural Language. We observe a tight correspondence between factorization and summarization: every factorization gives rise to multiple possible summarizations, each obtained by counting the number of sub-explanations that are "factorized together". As observed already in previous work [8, 42], different assignments (explanations) may have significant parts in common, and this can be leveraged in a factorization that groups together multiple occurrences. Here again, while provenance summarization has been studied before (e.g. [3, 44]), the desiderata of a summarization needed for NL sentence generation are different, rendering previous solutions inapplicable here. Importantly, we impose a novel constraint on the factorizations that we look for (which we call compatibility), intuitively capturing that their structure is consistent with a partial order defined by the parse tree of the question. The corresponding decision problem remains coNP-hard (again in the provenance size), but we devise an effective and simple greedy solution. This constraint is needed so that we can translate the factorization back to an NL answer whose structure is similar to that of the question. In typical scenarios, a single answer may have multiple explanations (multiple authors, papers, venues and years in our example). We provide a robust solution, allowing to compute NL summarizations of the provenance, of varying levels of granularity.

THE ANALOGIES

We start by recalling some basic notions from NLP, as they pertain to the translation process of NL queries to a formal query language. A key notion that we will use is that of the syntactic dependency tree of NL queries:
We focus on a sub-class of queries handled by NaLIR, namely that of Conjunctive Queries, possibly with comparison operators (=,>,<) and logical combinations thereof (NaLIR further supports nested queries and aggregation). The corresponding NL queries in NaLIR follow one of the two (very general) abstract forms described in Figure 3: an object (noun) is sought for, that satisfies some properties, possibly described through a complex sub-sentence rooted by a modifier (which may or may not be a verb, a distinction whose importance will be made clear later). After compiling a formal query corresponding to the user's NL query, we evaluate it and keep track of provenance, to be used in explanations. To define provenance, we first recall the standard notion of assignments for CQs.
Basic Solution : We follow the structure of the NL query dependency tree and generate an answer tree with the same structure by replacing/modifying the words in the question with the values from the result and provenance that were mapped using the dependency-to-query-mapping and the assignment. Yet, note that simply replacing the values does not always result in a coherent sentence.
Logical Operators: Logical operators (and, or) and the part of the NL query they relate to will be converted by NaLIR to a logical predicate which will be mapped by the assignment to one value that satisfies the logical statement. To handle these parts of the query, we augment Algorithm 1 as follows: immediately following the first case (before the current Line 5), we add a condition checking whether the node object has a logical operator ("and" or "or") as a child. If so, we call Procedure HandleLogicalOps with the trees TQ and TA, the logical operator node as u, the dependency-to-query-mapping τ and the assignment α. The procedure initializes a set S to store the nodes whose subtree needs to be replaced by the value given to the logical predicate. Procedure HandleLogicalOps first locates all nodes in TQ that were mapped by the dependency-to-query-mapping to the same query variable as the sibling of the logical operator (denoted by u). Then, it removes the subtrees rooted at each of their parents, adds the value (denoted by val) from the database mapped to all of them in the same level as their parents, and finally, the suitable word for equality is added as the parent of val in the tree by the procedure AddParent.

Factorization : How do we measure whether a possible factorization is suitable/preferable to others? Standard desiderata [42, 17] are that it should be short or that the maximal number of appearances of an atom is minimal. On the other hand, we factorize here as a step towards generating an NL answer; to this end, it will be highly useful if the (partial) order of nesting of value annotations in the factorization is consistent the (partial) order of corresponding words in the NL query. We will next formalize this intuition as a constraint over factorizations. We start by defining a partial order on nodes in a dependency tree:

Computing Factorizations : Recall that our notion of compatibility restricts the factorizations so that their structure resembles that of the question. Without this constraint, finding shortest factorizations is coNP-hard in the size of the provenance (i.e. a boolean expression) [27]. The compatibility constraint does not reduce the complexity since it only restricts choices relevant to part of the expression, while allowing freedom for arbitrarily many other elements of the provenance. Also recall (Example 4.8) that the choice of which element to "pull-out" needs in general to be done separately for each part of the provenance so as to optimize its size (which is the reason for the hardness in [27] as well). In general given a dependency tree T, a provenance expression provT and an integer k, deciding whether there exists a T-compatible factorization of provT of size ≤ k is coNP-hard.

From Factorizations to Summarizations : So far we have proposed a solution that factorizes multiple assignments, leading to a more concise answer. When there are many assignments and/or the assignments involve multiple distinct values, even an optimal factorized representation may be too long and convoluted for users to follow.

CONCLUSION

We have studied in this paper, for the first time to our knowledge, provenance for NL queries. We have devised means for presenting the provenance information again in Natural Language, in factorized or summarized form. There are two main limitations to our work. First, a part of our solution was designed to fit NaLIR, and will need to be replaced if a different NL query engine is used. Specifically, the "sentence generation" module will need to be adapted to the way the query engine transforms NL queries into formal ones; our notions of factorization and summarization are expected to be easier to adapt to a different engine. Second, our solution is limited to Conjunctive Queries. One of the important challenges in supporting NL provenance for further constructs such as union and aggregates is the need to construct a concise presentation of the provenance in NL (e.g. avoiding repetitiveness in the provenance of union queries, summarizing the contribution of individual tuples in aggregate queries, etc.).

REFERENCES

- https://github.com/navefr/NL_Provenance/