NLProveNAns: Natural Language Provenance for Non-Answers

Daniel Deutch Nave Frost Amir Gilad Tomer Haimovich
danielde@post.tau.ac.il {navefrost, amirgilad, tomerh}@mail.tau.ac.il
Tel Aviv University

ABSTRACT

Natural language (NL) interfaces to databases allow users without technical background to query the database and get the results. Users of such systems may be surprised by the absence of certain expected results. To this end, we propose to demonstrate `NLProveNAns`, a system that allows non-expert users to view explanations for non-answers of interest. The explanations are shown in an intuitive manner, by highlighting parts of the original NL query that are intuitively "responsible" for the absence of the expected result. Our solution builds upon and combines recent advancements in Natural Language Interfaces to Databases and models for why-not provenance. In particular, the systems can provide explanations in one of two flavors corresponding to two different why-not provenance models: a short explanation based on the *frontier picky* model, and a detailed explanation based on the why-not *polynomial* model.

PVLDB Reference Format:
Daniel Deutch, Nave Frost, Amir Gilad, Tomer Haimovich. NLProveNAns: Natural Language Provenance for Non-Answers. *PVLDB*, 11 (12): 1986-1989, 2018.
DOI: https://doi.org/10.14778/3229863.3236241

1. INTRODUCTION

Natural Language (NL) interfaces to database systems are often used as easy-to-understand gateways for accessing complex databases. In this work we build upon such NL Interface called NaLIR [6][1], and enrich it with support for explanations of non-answers.

Explaining non-answers, also termed why-not provenance, has been the focus of multiple previous works (e.g. [3, 1]). This type of provenance is used to explain why a certain piece of information, that the user expected to see, was not returned in response to the user query. Such information is crucial for understanding the result, and debug and improve the query and/or the input database.

To our knowledge, presenting why-not provenance to non-experts has not been previously studied. We claim that it is even of greater importance in the context of interfaces for non-experts, because of the cumulative errors that arise in such systems: first, the user has to specify her intent in a question; a failure to specify a certain condition or a too specific query might result in tuple loss, simply because the user performed an error in the query formulation. Then, an NL query engine needs to parse the supplied sentence and construct an SQL query. Failing to perform this task correctly. Since the user has no means of viewing or understanding the SQL query, this error may go unnoticed.

There are multiple approaches and models for capturing why-not provenance; we focus here on explaining non-answers through the parts of the query (a query operator or a set thereof) that were responsible for the answers omission. Our main observation is that when the original question was given in NL, we can in many cases trace back a given query operator to the part of the NL question that corresponds to it. The latter is then natural to present to non-experts – in our implementation, we simply highlight the relevant parts of the NL question. This provides insights on the criteria that had led to the answer being discarded.

As an illustration, consider the natural language query depicted in Figure 3a, translated (by `NaLIR`) into the Conjunctive Query depicted in Figure 3b. When this query is evaluated and returned to the user, she might further wonder why the specific author Marge, is not part of the result set. The underlying reason for the absence of this name from the results is that the author Marge did not publish papers after 2005. Using our system, the user will be shown her original natural language question, with a emphasis on the relevant words that caused this absence (see Figure 3c). This then enables the user to better understand the query, validate the translation process and the credibility of the database, and reformulate her natural language query accordingly.

Our approach may be applied for any choice of query operators that serve as why-not explanations. In particular, we incorporate two previously proposed models, namely the *frontier picky* model [3] and the *polynomials* model [1, 2]. The former explains a non-answer using a single query operator, intuitively the last to contain in its input a matching tuple. The latter provides a more extensive explanation, accounting for all operators that could independently serve as reasons for the non-answer (along with additional information on the number of relevant tuples for each operator).

[1]We are extremely grateful to Fei Li and H.V. Jagadish for generously sharing with us the source code of NaLIR, and providing invaluable support.

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/8.
DOI: https://doi.org/10.14778/3229863.3236241

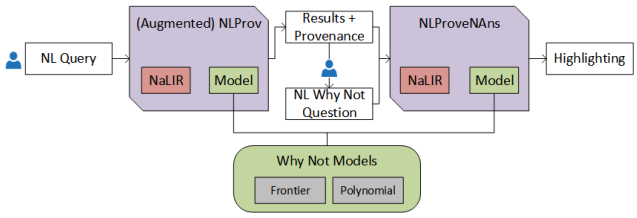
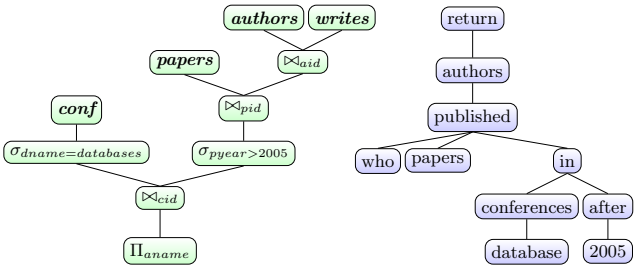


Figure 1: System Architecture

Our solution leverages and extends multiple previously proposed models and systems for NLIDBs and provenance, and our high-level system architecture is depicted in Figure 1. We start by using NaLIR to parse the NL query, and use our previously proposed implementation NLProv [4] to further track provenance for the query during evaluation. We augment NLProv (originally tracking "positive" provenance) so that it also tracks provenance for non-answers, using one of the two why-not models mentioned above. After viewing the results, the user specifies a why-not question. This may simply be a single hypothetical output tuple whose non-existence had "surprised" the user, or a more general specification of such queries (e.g. "why not authors from Springfield"). For the latter we again leverage NaLIR, allowing the user to phrase the why-not question in NL which is translated into a formal selection query. Finally, we leverage the obtained provenance along with mappings from question parts to query operators computed by NLProv, to highlight parts of the NL question that have intuitively contributed to the non-existence of the tuples she was interested in.

Related Work. Why-not provenance has been studied by several lines of work (e.g. [3, 1]), suggesting different models to capture the reason for non-answers, and focusing on the query operators, rather than classic provenance that focuses on the data. Explaining non-answers to users was also demonstrated in [2], with the major difference being the target users. While [2] shows the raw SQL query and its operators, `NLProveNAns` shows the explanations in natural language, assuming users do not necessarily understand SQL. This constraint gives rise to several algorithmic challenges we detail in the sequel.



(a) Query Plan (b) NL Query Parse Tree

Figure 2: Query Trees

2. TECHNICAL DETAILS

The architecture is presented in Figure 1 and we next overview its main components.

```
return authors who published papers in database
conferences after 2005
```

(a) NL Query

```
query(aname) :- conf(cid, dname, cname),
pub(wid, aid, cid, ptitle, pyear), author(aid, aname, oid),
dname = 'Databases', pyear > 2005
```

(b) Corresponding CQ

```
return authors who published papers in database
conferences after 2005
```

(c) Explanation for "Why not Marge?"

Figure 3: NL Query, CQ, and why-not explanation

2.1 Problem Statement

In the following let Q be a query whose non-answers we wish to explain.

Why-Not Question. We define a why-not question W as a selection query over the schema of the output of Q. Intuitively each hypothetical output tuple of Q that matches the criteria of W is a tuple whose non-existence in the answer we wish to explain.

Word Highlights. The input to our problem is then an NL query Q_{NL} and an NL why-not question W_{NL}. Its output is a set of word indices in Q_{NL}, where each index corresponds to the position of a word that is intuitively "responsible" for filtering out the tuples captured by W_{NL}, leading to their non-existence in the result set of Q_{NL}.

2.2 Provenance Generation

We explain next the processing of an NL query to obtain a result set as well as why-not provenance. We focus here, for ease of presentation, on the why-not model of Frontier Picky [3]; then we explain the incorporation of the polynomial model [1].

Frontier Picky. The why-not model of [3] explains non-answers in the following manner. Based on the query plan, the frontier picky operator is the last query operator such that its input contains some tuples satisfying W. Note that if there are multiple tuples satisfying W, the frontier picky is the last operator out of the frontier picky operators for each of them and is thus uniquely defined.

EXAMPLE 2.1. *Consider the query evaluation plan in Figure 2a, for the query in Figure 3a. It queries 4 tables, authors and conf, including author and conference information respectively, papers including publications, and writes which links authors and papers (this is a simplification of the actual schema of [7]). Consider three authors "Homer", "Marge" and "Lisa", where "Homer" is not a published author, so his frontier picky operator is the join operator between the authors and writes tables. "Marge" is a published author but did not publish after* 2005*, so for her,* $\sigma_{pyear>2005}$ *is the frontier picky operator. "Lisa" is an author who published after* 2005 *but not in database conferences, thus having the join on the cid key as the frontier picky operator.*

Provenance-Aware Query Evaluation. Algorithm 1 then implements query evaluation while storing why-not provenance. It starts by translating the NL query to a formal one, via the mechanism of `NaLIR` [6] augmented so that we keep track of which word in the original NL question has been mapped to which operator of the formal query, as done in `NLProv` [4]. Then, in line 2 the query is evaluated. During evaluation, whenever a tuple is removed (due to a selection operator or as part of a filtering join), we update the mapping M_{filter}, which maintains the relation between the query operators and the tuples that were removed by them.

EXAMPLE 2.2. *Re-consider our running example query, and now consider the NL query in Figure 3a which is translated by* `NaLIR` *to the query in Figure 3b. First, algorithm 1 stores the mappings between words in the original sentence to their respective operators:* $M['after\ 2005'] : pyear > 2005$, $M['database'] : dname = \text{“}Databases''$ *and* $M['authors'] : query(aname)$.

During evaluation, M_{filter} *includes tuples such as* $(Marge, paper_1, 1988)$, $(Lisa, paper_2, 2007, VLDB)$ *and their respective picky operators* $\sigma_{pyear>2005}$, $\bowtie_{cid}$ *(the two tuples include more attributes such as pid, which are omitted for brevity).*

Algorithm 1: EvaluateWithProvenance

input : NL query Q_{NL}, Database D
output: Result set and provenance for Q_{NL}

1 $(Q, M) \leftarrow NLToFormal(Q_{NL})$;
2 $(R, M_{filter}) \leftarrow ProvEval(Q, D)$;
3 **return** (R, M_{filter});

Answering Why-Not. Algorithm 1 prepared the necessary data structures. After viewing the evaluation results, the user now formulates a why-not question in Natural Language. We use `NaLIR` again, to convert this question into a formal why-not selection query W. Its output is the set of word indices to be highlighted.

Algorithm 2: Highlight

input : M, M_{filter}, W, R
output: Word highlight set

1 **if** *R contains tuple t such that* $W(t)$ **then**
2 **return** ϕ;
3 $reason \leftarrow$ $GetFrontierPicky(M_{filter}.where(W(tuple)))$;
4 **if** $reason == null$ **then**
5 $reason \leftarrow GetProjectionOperator(M_{filter})$;
6 **return** GetWords(reason);

Algorithm 2 operates as follows. In lines 1–2 it checks whether the why-not question is valid, i.e. if there are indeed no output tuples satisfying the criteria.

If this is not the case, it finds the frontier picky operator (line 3) for each of the tuples satisfying the why-not predicate W and returns the last of them which is the frontier picky operator for W. We may obtain NULL as the operator in the case where there is no match to the why-not question in the input; in this case we consider the last projection operation that took place to be the “reason” (lines 4–5). We then (line 6) invoke algorithm 1 to track back the words corresponding to the frontier picky operator. Once the words are found, they are marked as highlighted.

Algorithm 3: GetWords

input : Query operator *op*
output: Set of word indices relevant to *op*

1 **if** *op is* $\sigma(A = x)$ **then**
2 **return** $map(A) \cup map(x)$;
3 **if** *op is* $T \bowtie R$ **then**
4 **if** $map(R) \neq \phi$ **then**
5 **return** $map(R)$;
6 **else**
7 (left, right) = GetJoinedRelations(R);
8 **while** *map(left)*$==\phi$ **do**
9 (left, _) = GetJoinedRelations(left);
10 **while** *map(right)*$==\phi$ **do**
11 (_, right) = GetJoinedRelations(right);
12 **return** GetPath(left, right);

Algorithm 3 operates as follows. The *map* function exploits the M data structure, and gets as input variable names, values or table names that can be mapped back into words in the original query. Generally, a selection operator is straightforward to map to words because the selection itself was expressed in the query. Join operations pose further challenges. A join frontier picky operator may be mapped directly into a word if both the joined relations are words in the query, but may also be due to an in indirect mapping, where the two joined relations are not part of the query. Lines 7-12 are used to help link back the join operators which could not be directly mapped to words. The main idea is to exploit the typical scenario where unmapped joins are used as means for the query generation engine to link two other relations, which are directly referenced in the NL query. The algorithm traverses the join operations before and after the given join operator (by repeatedly calling $GetJoinedRelations()$), until two operators, one before and one after the join, which can be mapped into words $(left, right)$, are found. The returned indices are not of these two words, but rather of the words between $left$ and $right$ (by calling $GetPath()$), corresponding to the intuition that the answer to the why-not question is an unmapped operator between the two mapped operators.

EXAMPLE 2.3. *Recall that the author “Homer” has no published papers, so despite being contained in the authors table he has no relevant tuple in the writes table. The frontier picky operator would then be the join operation between the two relations, as the only tuple relevant to this author is dropped because no matches were found for it in the writes table. Unfortunately, the writes table cannot be mapped into a word in the original query because the word “writes” does not even appear in it. The reason for the join operation being included is that the NL-to-SQL engine* `NaLIR` *has used this table as a link between the authors table and the papers table. In this case, lines 7-12 are used to trace back the joined relations. Left and right would be authors and papers respectively, and the returned path would include only the*

*word "published" as seen in Figure 2, thus our word highlight answer would be: return authors who **published** papers in database conferences after 2005.*

2.3 Plugging in the Polynomial Model

The frontier picky approach is natural and easy to understand, but (1) it returns only a single explanation for why-not where there may be multiple relevant operators and (2) it is sensitive to the query plan that is used. The latter is problematic in our setting where the original question is phrased in NL and is then translated into a formal one (with the non-expert user being oblivious to the precise translation), and different engines may translate and execute it differently.

To this end we may use a different explanation model for why-not queries: the polynomial model described in [1]. Instead of answering why-not questions with a single operator, the output of this model is a polynomial of operators, in which each additive term (addend) acts as an explanation for a missing tuple described by the why-not predicate, and consists of several operator-related reasons for not including this tuple in the result set.

EXAMPLE 2.4. *For instance, consider a different author "Maggie", who published 3 papers in DB conferences before 2005 and 5 papers in NLP conferences before 2005. The question "Why not Maggie" will be answered in the polynomial model with the polynomial* $3op_1 + 5op_1op_2$. *This answer should be read as "3 tuples matching the why-not predicate were excluded because they represented papers which were published before 2005 (*op_1*), and 5 other tuples were excluded because they represented papers which were published before 2005 and in a non-database conference (*op_2*)".*

The resulting polynomial is invariant to query rewriting[1], but on the other hand is much more complex than the frontier picky. The polynomial may consists of many different addends and each addend may include many operators.

Our algorithms all extend to support explanations based on this model. We show the user several different word highlights based on the different addends in the polynomial. Instead of computing the frontier picky operator in Algorithm 2, we calculate the polynomial. Then, for each explanation (addend), instead of starting with one possible operator in Algorithm 3, we iterate over the set of relevant operators and return the union of the returned sets.

The explanations are then ranked: firstly by the number of operators in the addend, and then by the integer coefficient of the addend, indicating how many tuples might be affected by modifying the relevant operators thus reflecting on the importance of the corresponding explanation.

EXAMPLE 2.5. *The natural language word highlight explanations for the predicate "Why not Maggie", using the polynomial model are:*

1. *"return authors who published papers in database conferences **after 2005**"*
2. *"return authors who published papers in **database conferences after 2005**"*

*In the frontier picky model, only one explanation is displayed to the user: "return authors who published papers in database conferences **after 2005**". The explanation depends upon the query plan, though, and different query plans may lead to non-intuitive answers.*

3. SYSTEM OVERVIEW

`NLProveNAns` is implemented in JAVA, and runs on Windows 10. It uses MySQL as its underlying database system and uses two previously developed system prototypes, namely `NaLIR` [6] and `NLProv` [4]. Figure 1 depicts the system architecture. First, the user inputs a query in natural language and chooses between the two provenance models. This query is fed to the modified `NaLIR` implementation which parses the sentence and generates an SQL query. This query is then evaluated by `SelP` [5].

The user is then presented with the results of the initial query along with natural language explanations for the result set tuples, generated by `NLProv`. In this step a why-not question may be asked in order to assist the user in understanding why certain tuples are not visible in the result set. This question is parsed by `NLProveNAns` and using the information stored while processing the initial query, it computes an answer for the why-not question using the chosen provenance model, and uses this answer to produce a word highlighting answer. The relevant words in the initial query are highlighted based on the answer, and the user is invited to refine the query and observe its effect on the result set.

4. DEMONSTRATION SCENARIO

We will demonstrate the system prototype using a sample of the publications database of Microsoft Academic Search [7]. We will design multiple complex questions over the database of the flavour exemplified above, execute them over the database and show the results along with their natural language provenance. We will then demonstrate why-not questions in Natural Language with respect to the query results, and show the obtained why-not explanations for both the Frontier Picky model and the why-not polynomials model. We will allow the users to browse through the obtained explanations. We will then allow the users to pose their questions of choice in NL (using a pre-prepared template), both for the original query and for the why-not questions. Finally, we will allow the audience to look "under the hood", showing the intermediate results of the different components, including the obtained SQL queries and a sample of the internal provenance representation.

Acknowledgments. This research was partially supported by the Israeli Science Foundation (ISF, grant No. 1636/13), and by ICRC - The Blavatnik Interdisciplinary Cyber Research Center. The contribution of Amir Gilad is part of a Ph.D. thesis research conducted at Tel Aviv University.

5. REFERENCES

[1] N. Bidoit, M. Herschel, and A. Tzompanaki. Efficient computation of polynomial explanations of why-not questions. In *CIKM*, 2015.
[2] N. Bidoit, M. Herschel, and K. Tzompanaki. EFQ: why-not answer polynomials in action. *PVLDB*, 8(12):1980–1983, 2015.
[3] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.
[4] D. Deutch, N. Frost, and A. Gilad. Nlprov: Natural language provenance. *PVLDB*, 9(13):1537–1540, 2016.
[5] D. Deutch, A. Gilad, and Y. Moskovitch. Selective provenance for datalog programs using top-k queries. *PVLDB*, 8(12):1394–1405, 2015.
[6] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, 2014.
[7] MAS. `http://academic.research.microsoft.com/`.