Constructing an interactive Natural Language Interface for Relational Databases

- Fei Li and H.V. Jagadish

Report by: [redacted]

ABSTRACT:

The architecture of Interactive Natural Language interface for relational databases is briefly described. A logically complex English language query is translated into an SQL query which can include Aggregation, nesting and joins.

5½/6

INTRODUCTION:

Querying data in relational databases is challenging until the user knows the exact schema of the database. NaLIR(Natural Language Interface for Relational Databases) have been built towards this goal. The chief contributions of this paper include:

what goal?

1. **Interactive Query Mechanism**- Enable users to ask complex queries and interpret them with little interaction.

2. **Query Tree**- To represent the interpretation of natural language query and after verification translated into SQL.

What is the proposed mechanism for doing this?

3. **System Architecture**- NaLIR (Natural Language Interface to Relational databases).

4. **User Study**- The usability of NaLIR in practice.

SYSTEM ARCHITECTURE

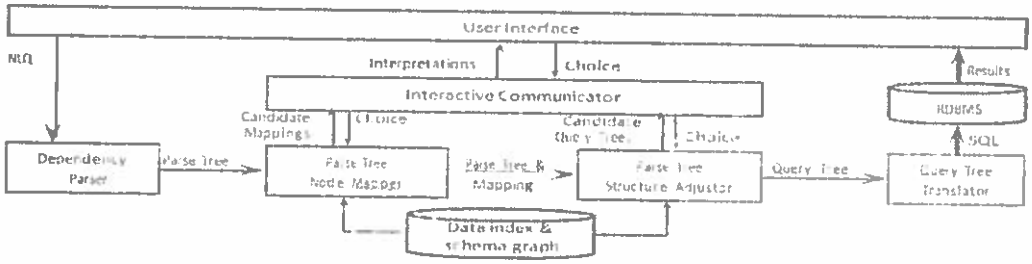
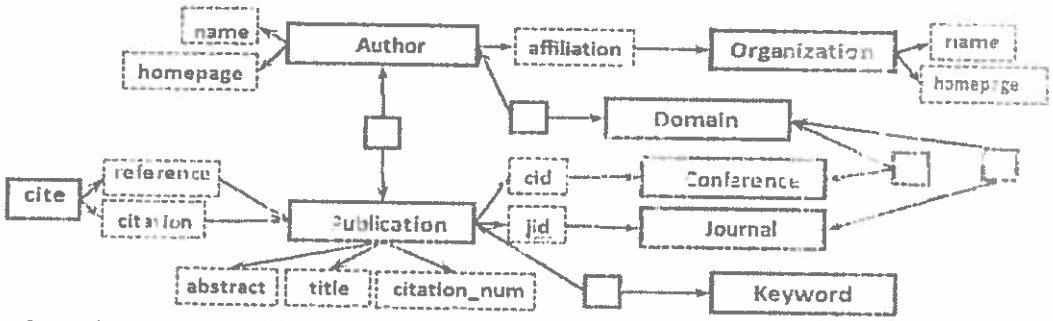


Figure 2: System Architecture

EXAMPLE

Consider the example with different set of tables and its relationships, three queries given below will be explained using this model discussed in the paper.



Query*1:"Return"the"average"number"of"publications"by"Bob"in"each"year.""
Query*2:"Return"authors"who"have"more"papers"than"Bob"in"VLDB"after"2000.""
Query*3:"Return"the"conference"in"each"area"whose"papers"have"the"most"total"citations.""

Figure 1: A Simplified Schema for Microsoft Academic Search and Sample Queries.

Consider Query 1, word "publication" may be stored in different tables. Hard to be identified by NLIDB. Ambiguity in Complex Natural Lang. Query is handled by giving multiple interpretations to the user and allows them verify it. This is done through the Query

Given a complex Natural lang query, how do they translate it to SQL?

interpretation part in the system. When a natural language query is provided it goes to the dependency parser where a **Stanford parser** is used to generate a linguistic parse tree fig 3.a Parse tree node mapper generates the best mapping for each node in fig 3.b. and reports to the user, in e.g. node " VLDB" maps to "VLDB conference" and "VLDB Journal" in the database and the system has chosen "VLDB conference" as the default mapping.

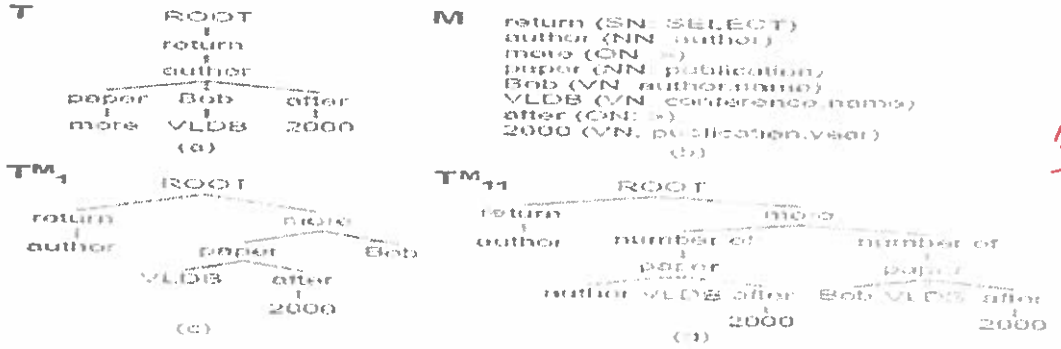


Figure 3: (a) A Simplified Linguistic Parse Tree from the Stanford Parser. (b) A Mapping Strategy for the Nodes in the Parse Tree. (c) A Valid Parse Tree. (d) A Query Tree after Inserting Implicit Nodes.

Parse tree adjustor reformulates the structure of tree T and generates the top K valid parse trees T_i^M in natural language for user to choose from, for example T_1^M is explained as "return the authors, where the number of papers of the author in VLDB after 2000 is more than the number of papers of the author in VLDB after 2000 is more than Bob" whereas after **inserting implicit nodes** T_{11}^M the parse tree becomes more clear where it "return the authors, where the number of papers of the author in VLDB after 2000 is more than the number of papers of the author in VLDB after 2000 is more than the number of paper Bob in VLDB after 2000" is in which the underlined part can be cancelled by the user. Parse tree node interpretation identify the parse tree that can be mapped to SQL components as per fig4.

Node Type	Corresponding SQL Component
Select Node(SN)	SQL keyword SELECT
Operator Node(ON)	An operator, eg. =, >, contains&
Function Node(FN)	An aggregation function, e.g., AVG
Name Node(NN)	A relation name or attribute name
Value Node(VN)	A value under an attribute
Quantifier Node(QN)	ALL,ANY,EACH
Logic Node(LN)	AND,OR,NOT

Figure 4: Different Types of Nodes

Parse tree reformulation algorithm is shown below:

Algorithm 1: QueryTreeGen(parseTree)

```
1:  results = ∅; HT = ∅;
2:  PriorityQueue.push(parseTree)
3:  HT.add(h(tree))
4:  while PriorityQueue ≠ ∅ do
5:      tree = PriorityQueue.pop()
6:      trees[] = adjust(tree)
7:      for all tree' ∈ trees[] do
8:          if tree' not exists in HT && tree'.edit < t then
9:              tree'.edit = tree.edit+1;
10:             HT.add(h(tree'));
11:             if evaluate(tree') >= evaluate(tree) then
12:                 PriorityQueue.add(tree')
13:                 if tree' is valid
14:                     results.add(tree')
15: rank(results)
16: Return top-k results
```

Figure 5: Parse Tree Reformulation Algorithm

Each time the adjust(tree) to generate all the possible parse trees in sub tree move operation(line6). Each new generated parse tree are filtered in (line 11-12), also we hash each parse tree into a number and store all hashed numbers in a hash table (line 10). We make sure that each parse tree will be processed at most once (line 8). System record all the valid process appeared in reformulation process (line 13-14) and return the top results from the (line 15-16). The next step involves converting the query tree into SQL statement. Consider the query. The schema element mapped by NV node under the SELECT node is added to the SELECT clause. Each value node (together with its operation node if specified) is translated to a selection condition and added to the WHERE clause. Finally, a FK-PK join path (schema graph), to connect each NV node and its neighbors. Later translated to series of FK-PK join conditions and are added to the FROM clause.

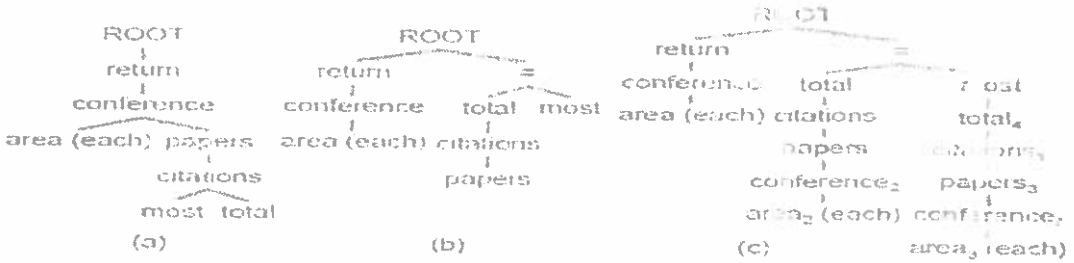


Figure 7: (a) A Simplified Linguistic Parse Tree for Query 3 in Figure 1. (b) A Valid Parse Tree. (c) A Query Tree after Inserting Implicit Nodes.

The SQL statement for the above parse tree representation is

```
1. Block 2: SELECT SUM(Publication.citation_num) as sum_citation,
2.                  Conference.cid, Domain.did
3.          FROM Publication, Conference, Domain, ConferenceDomain
4.          WHERE Publication.cid = Conference.cid
5.                  AND Conference.cid = Conference.cid
6.                  AND ConferenceDomain.did = Domain.did
7.          GROUP BY Conference.cid, Domain.did

8. Block 3: SELECT MAX(block4.sum_citation) as max_citation,
9.                  block4.cid, block4.did
10.         FROM (CONTENT OF BLOCK4) as block4
11.         GROUP BY block4.did

12.Block 1: SELECT Conference.name, Domain.name
13.         FROM Conference, Domain, ConferenceDomain
14.                 (CONTENT OF BLOCK2) as block2
15.                 (CONTENT OF BLOCK3) as block3
16.         WHERE Conference.cid = ConferenceDomain.cid
17.                 AND ConferenceDomain.did = Domain.did
18.                 AND block2.citation_num = block3.max_citation
19.                 AND Conference.cid = block2.cid
20.                 AND Conference.cid = block3.cid
21.                 AND Domain.did = block2.did
22.                 AND Domain.did = block3.did
```

Figure 8: Translated SQL Statement for the Query Tree in Figure 7 (c).

CONCLUSION

Innovation is interactive communicator, experiments conducted even turning off and verified that system always chose the default option. Thus Natural language query is translated into a SQL statement and are evaluated in RDMS. Some of the advantages are high reliability by explaining each query processing, ambiguity avoided by multiple likely interpretations.

The WEKA Data Mining Software: An Update

- Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten.

INTRODUCTION

To introduce the WEKA workbench, review the history of the WEKA project and discuss the new features in the latest version (Weka 3.6) at the time of writing. A homogenous framework on which algorithms could be implemented was needed -- WEKA (Waikato Environment for Knowledge Analysis).

FUNCTIONS

To run data Mining Algorithms (e.g. Association rule mining, clustering and classification. Visualization, Preprocessing and Statistical evaluation.

FEATURES

Easy and quick implementation of algorithms. Modular and extensible architecture and Plug-in mechanisms.

USER INTERFACE

The Explorer- Main components in the explorer include:

- Preprocess Panel – It is responsible for loading and transforming data using filters. Data can be loaded from different sources in different formats.

- Classify Panel – User can run classification and regression algorithms.
- Cluster Panel – Clustering algorithms were found in this panel and could be run from here.
- Associate Panel – Able to run association rule mining algorithms from this panel.
- Select Attribute Panel – Algorithms for feature engineering. Mostly used for exploratory data analysis.
- Visualize Panel – colour-coded scatter plot matrices could be obtained from this panel.

Explorer Window

Some other graphical user interfaces include:

The Knowledge flow

- Suitable for large datasets that would crash the Explorer.
- Supports incremental model building.
- This can be used from a command line interface.

The Knowledge Flow Environment

The Experimenter

Wasting space

- It Evaluates performance of algorithm.
- Experiments can be saved in XML or binary format.
- Configured experiments can be run from command-line.

nothing useful on this page

The Experiment Window

PROTOCOL TO RUN AN ALGORITHM

Could demonstrate with a simple example to increase Tech Content

- Load your data set from the Preprocess panel.
- Perform some filtering and visualization also on the Preprocess panel.
- Choose the algorithm you want to run your data on from the Classify, Cluster or Associate Panels.
- Set the parameters for the algorithm e.g. percentage of data to be used for training, number of iterations, etc.

but good

- Click on the start button to run the algorithm
- When the algorithm has finished running an output would be displayed on the right of the screen.
- Save the model which was just built.

CORE VERSION 3.6 UPDATE

v3.4 has 690 Java class files with a total of 271,447 lines of code while v3.6 has 1,081 class files with a total of 509,903 lines of code. The Core update involves support for XML data format and specification of instance weights in ARFF files, "Capabilities" meta-data facility which allows filters and algorithms to declare what data algorithms they can handle and "Technical Information" which is a meta-data facility that provides citation details for

algorithm. Central log file that captures all information written to any graphical logging panel in WEKA. Some User Interfaces improvements include new GUI tools like SQL Viewer and Bayes network editor. Standards and Interoperability that support for PMML (Predictive Modeling Markup Language).

PREPROCESSING FILTERS UPDATE IN VERSION 3.6

Addition of new preprocessing tools including Add Classification which adds the predictions of a classifier to a data set. Add ID that adds an ID attribute to a data set which is useful for keeping track of instances. Add Values to adds the labels from a given list to an attribute if they are missing. Attribute reorder that changes the order of the attributes in a data set. Numeric cleaner to "cleanse" numeric values that exceed a threshold or are too close to a certain value by replacing them with user-supplied defaults. Numeric to nominal which converts a numeric attribute to nominal by simply adding all the observed numeric values to the list of nominal values and finally Random subset which allows users to select a random subset of attributes.

USER INTERFACE UPDATES

Some user interface updates include Visualization menu to include Scatter Plots, ROC Curves, Trees and graphs, Tools menu in SQL viewer to enter SQL commands and Bayes network editor. In addition to that some plug-in mechanisms has been added.

ADDITIONAL INFORMATION ON WEKA

The WEKA development team received the SIGKDD Data Mining and Discovery Service Award in 2005 in recognition of the longevity and widespread adoption of WEKA. Members of the WEKA team wrote a book about data mining that explains mining techniques and their implementation with WEKA – Data Mining: Practical Machine Learning Tools and techniques. The authors the success of WEKA to the fact that it is open source. As at the 2000, WEKA had over 1.4 million downloads on source forge. Between 2000 and March 2015, it had 6 million downloads.

CONCLUSION

According to the authors, the WEKA project has come a long way since its inception in 1992. Releasing WEKA as open source software and implementing it in Java has played a small part in its success. These two factors ensure that it remains maintainable and modifiable irrespective of the commitment or health of any particular institution or company.