Final Report

Samy Simon (SS21CN) Brendan Gressel (BCG16B)

I. Research Problem/Contributions

For our project, we chose the approach implementation-flavor for the paper "Constructing an Interactive Natural Language Interface for Relational Databases" by Fei Li and H. V. Jagadish (both from the University of Michigan). This paper was published at the VLDB conference in 2015.

The problem the research paper is trying to solve is the unavailability of Natural Language Interfaces to Databases (NLIDB). These interfaces are very hard to implement for the very reason that understanding natural language is normally challenging (with slang words, dialect, etc.), thus causing the absence. Though complicated, these interfaces have many advantages that can be useful on a large scale, including the accessibility for normal users who are unfamiliar with SQL to access and query databases without any training needed. Schema knowledge will not have to be so extensive like what is currently required by visual query builders.

Queries can quickly become complicated when accessing large data banks with many relations, and with the way databases continue to expand beyond recognition, it's important to ensure that whoever needs to access the stored information should be able to.

The workload distribution for our project was as 50/50 as possible. We had multiple meetings for outlining our strengths and weaknesses. From these, we believe we have split the work successfully. Brendan Gressel was responsible for the write-ups (Project Proposal, Literature Survey, and the Final Report).

Samy Simon was responsible for the source code and implementation. He also aided in reading over and adding to all the write-ups. Both were equally hands-on in terms of the experiments and drawing conclusions on the project's effectiveness with participants and performance against Microsoft's algorithm.

II. Related Work & Improvements

A. MAS

One of the most notorious implementations of an NLIDB is Microsoft's. This is the system that we will be comparing our work to in the experimental phase of this project measured by performance time.

Essentially, the schema for the Microsoft Academic Search (or MAS) contains eight relations respectfully (as seen in *Figure 1*): author, organization, publication, cite, domain, conference, journal, and keyword. These all contain their own primary keys, as well as associated foreign keys to reference each other. Containing these eight specific relations, the overall system can accept a wide array of supported queries.

This system is ingrained within MAS website, and the comparison we will make will be performed on the website itself. This website contains a total of 196

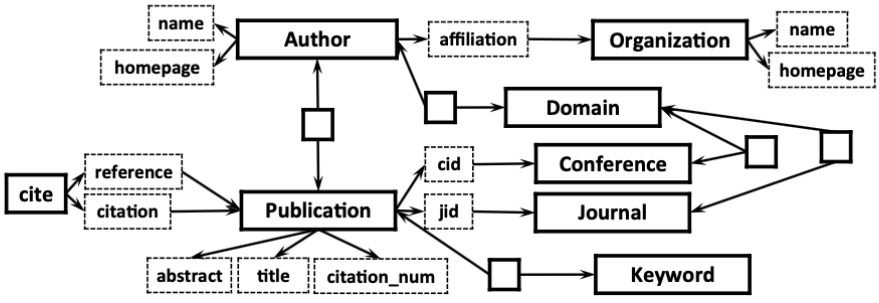

Figure 1: A Simplified Schema for Microsoft Academic Search and Sample Queries.

directed supported queries, which means that once entered, the result can be obtained in a single webpage instead of promoting multiple possible answers to the query.

These queries contain a specific difficulty ranking that the authors of the research paper acquired through exhaustive aggregation of the system. The ranking of a query is determined based on the number of relations it must access to determine an answer. A query is considered “Easy” if it only must access one, “Medium” if it must access two, and “Hard” if it must access three or more. The authors found that out of the 196 directly supported queries, 63 are considered easy, 68 are considered medium, and 65 are considered hard, presenting an evenly split database. These are important for the experimental phase of the project.

When compared to our implementation, MAS’s effectiveness suffers. This is strictly due to our user interface which will have user interaction and verification until the results are projected correctly. Without this interaction, our implementation still holds better results with simple and medium queries. However, with those terms, the harder queries are parsed better using MAS. That is why our implementation uses the user interaction to our advantage, giving us the better performance we expect. In terms of time cost, our implementation does fair better when the user interaction is disabled- however, when our entire system is running, the harder queries do have larger time costs than MAS. Using the author’s findings, the usability of our system never caused participants to become frustrated with the user interaction.

B. QUEST

QUEST is designed to showcase a query generator for structured sources, which acts as a search engine for a relational database. This, very much like MAS, will take in keywords and transform them into meaningful SQL queries for the machine to process. This was spawned in 2013 by a group of researchers from Italy.

QUEST uses two methods for this transformation: forward and backward. The forward will take these specific keywords provided by the user and match them to database terms like attributes and the names of tables. The backward will work to compute particular paths that will connect the structures provided from the forward. These methods are then combined within a probabilistic framework that is based on the Dempster-Shafer Theory.

The full framework follows this pattern: QUEST will first learn the database schema using metadata to create a wrapper. Then it will find the top-k configurations associated with the user keyword queries and use a Steiner Tree-based technique to find the top-k paths that join the database schema elements provided. It will aggregate these results and develop an SQL query that then provides the results to that query on-screen. This implementation uses a similar technique as our implementation via the tree structure, however our implementation will improve upon QUEST by having our additional user interface to guarantee a correct query. As it stands, QUEST does not verify the query with the user, thus causing issues when the suspected query is displaying the wrong results. There is no way around this, as their interface provides no user support after their natural language query is executed.

C. DBXplorer

DBXplorer, like QUEST and MAS, is used as a search engine based on keywords. This one, like MAS, was developed by Microsoft. This implementation takes a set of keywords given by the user and matches them to all the rows which contain those keywords (note: the rows must contain all the keywords). This can be done via single tables or by joining tables that share foreign keys. This is done in a two-step process: publish and search.

The publish step identifies a database and all the tables and columns within it. Using this, auxiliary tables will be made to aid in the keyword searches. This step will also create a symbol table that is used at search time to determine the locations of the keywords in the database.

The search step will use the created symbol table to locate the existing tables, columns, and rows that contain the keyword as well as subsets of tables that, if joined, will contain all the keywords. These subsets need to somehow be connected already in the schema via a join tree. For each join tree, an SQL query is built that will then connect the tables in the tree and selects the rows that contain all the keywords. These rows are then compared and delivered to the user.

Our implementation will improve tremendously when compared to DBXplorer. Firstly, this implementation is far too strict as each tree will not be created unless every keyword from the natural language query is present. This causes tremendous performance issues whenever the queries become too complex and will make developing accurate results extremely hard. This will not only jeopardize correct query results but also query time.

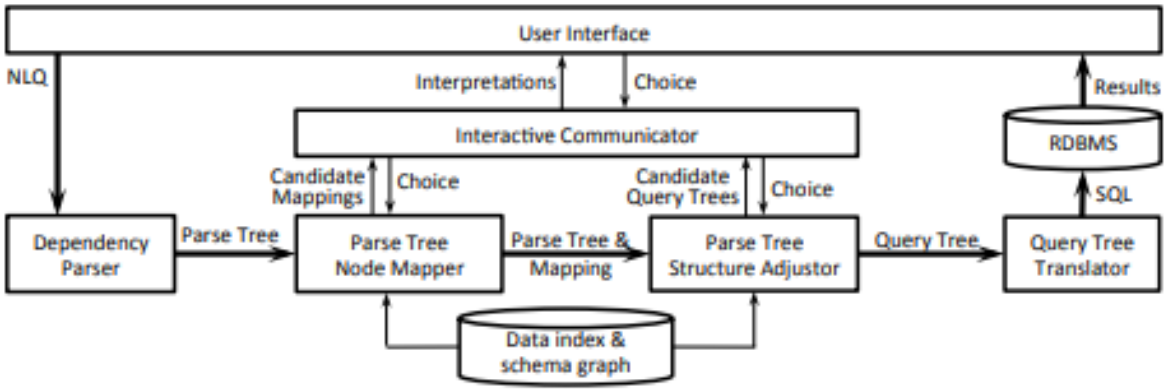

Figure 2: System Architecture.

Moreover, like MAS and QUEST, this implementation does not verify any part of the query with the user. So, if a query is too hard for DBXplorer to provide a correct display of results, then it is on the user to change their natural language query until they get the results they were after.

III. Our Implementation

Figure 2 contains the actual system we implemented. This system contains three components that work very linearly. The first is called the Dependency Parser, and this takes the natural language input that the user inputs into the user interface and transforms it into a parse tree.

Given this parse tree, the second component- which contains the bulk of the system- turns this into a query tree. This component of the system contains three different parts, the first two being the Parse Tree Node Mapper and the Parse Tree Structure Adjustor. These are responsible for interpreting the query and representing the interpretation as a query tree. Whenever these parts need the user to make a clarification on an aspect of the query, it communicates with the third part of this component, the Interactive Communicator, which gives the user choices on how to make the query more suitable to their request. Then, with a correct query tree completed, the system will move onto the third component.

The last component is called the Query Tree Translator. This takes the query tree and turns it into a singular SQL query. This finished query then is passed into the RDBMS and the results are displayed onto the User Interface.

The resources we used to complete this project are Eclipse to code our system in Java, MySQL for our RDBMS in which we will build our NLP interface on, and the Microsoft Academic Search database which will be used as our dataset.

IV. Experimental Design

We compared our system against Microsoft Academic Search Website (MAS). This website claims to have 196 directly supported queries- meaning they are one pass and should return a single webpage instead of multiple when inputted into the website. These queries are separated into three groups: easy, medium, and hard.

Note: For our purposes with these experiments, we removed all the queries labeled hard. This is because, although we were able to get some of them to work, we had many non-conclusive results with others. For this type, the main component that failed was the structure adjuster, which had problems with translation. Including these would have been useless for the time cost evaluation, as the ones which had problems would ramp up the already existing time costs we had and would influence the numbers dramatically- ultimately making our numbers unrepresentable of the actual system.

This brought our query number down from 196 useable queries to 160. From these queries, we separated them randomly into 28 groups, all containing at least one from the easy category and one from the medium category.

Next, we found our participants outside of Strozier Library and vetted them to make sure they had no experience with SQL in any sense and made sure they have used an academic and/or a scientific website before. We immediately dismissed any STEM majors, as SQL can be prevalent within any of those departments (i.e., Biology, Engineering, Computer Science, etc.).

	Mapper	Reformulation	Insertion	Translation
Without Interactive Communicator	22	25	0	0
With Interactive Communicator	0	12	0	0

Figure 3: Failures in each component

	With interaction	Without interaction	MAS
Simple	25/33	19/33	20/33
Medium	20/22	15/22	18/22

Figure 4: Effectiveness

	With interaction	Without interaction	MAS
Simple	55	42	52
Medium	78	45	65

Figure 5: Average Time Cost (s)

The participants received two of the 28 groups of queries- one containing six to ten queries to test with our system and one with three to five to test with the MAS website.

For our system, roughly three to five of the queries from the first group were labeled easy. Half of these were used to test accuracy with the Interactive Communicator enabled and the other half were used to test with it disabled. The other queries (also about three to five) were labeled medium and were tested the same way.

For the MAS website, roughly two to three queries from the second group were labeled easy. Since MAS does not have an Interactive Communicator, these were just tested with the existing system. The other queries (about one to two) were labeled medium and were tested the same way.

We measured for correctness, which components caused the failures, and time costs. Finally, we had the participants rate out of five the easiness of using our system and the MAS website with each type of query.

V. Results & Analysis

For our experiments, we first compared the effectiveness of our system with the interactive communicator enabled and disabled with the MAS website. The results obtained with the interactive communicator enabled was significantly better compared with the disabled interactive communicator. Indeed, we registered 43 wrong answers generated without the interactive communicator interactions for queries that were more complex to be interpreted with our system. Out of the 43 wrong answers, we noted 21 that were unnoticed by the user while generating the query result since the participant were unfamiliar with the SQL language structure.

On the other hand, when the interactive communicator was enabled, we registered a total of 12 wrong answers out of 55 queries. We then compared our system results with the result obtained for the queries tasks ran on the MAS website. In contrast, participants only accomplished 21 out of the 55 queries using the MAS website to find the queries results. We noticed that when the query was wrongly interpreted, 12 out of the 14 participants did not manage to simply find the appropriate result after several pages lookup on the MAS website result pages.

We show the result of our effectiveness query comparison tests in Figure 4. We also gather the statistics results of the specific components that cause the failures which are presented in Figure 3.

Our results shows that the query tree was correctly generated and the translation results to the correct SQL statement when interaction is enabled. We also noted that our system always detects and inserts the implicit parse tree nodes even when the interaction communicator is not enabled for the user, during the parse tree structure adjuster. We also noticed that the accuracy for the parse tree node mapper improved significantly when the interactive communicator is enabled since it asks the user to choose the correct mapping for each node with the database schema objects. Indeed, the parse tree node

mapper generated at least one correct node mapping parse tree for the user to choose from and most of the participants were able to find the correct mapping for each node. We noted that the accuracy in our parse tree structure adjuster was greatly improved by the possible generated adjusted parse tree choices, since participants were allowed to choose the correct parse tree and therefore improve the accuracy of the generated SQL statement result. We also noted that for more complex queries, valid adjusted parse tree could not always be generated for the user.

We then evaluated the usability of our system by measuring the average time needed for each query tasks to be completed. Our results are presented in the Figure 5, which shows that when the interactive communicator is enabled, participants took more time to resolve the query task since they had to make choices during the node mapping process in which the user chooses the mapping of each node with the database schema objects and structure adjuster system components to choose the correct adjusted translated query tree.

We also noted that the time constraint for each participant did not impact the usability of our systems, since all participants manage to find their query results in an average of 55 seconds for simple query tasks and 78 seconds for medium query tasks. Meanwhile, when using the MAS website, we noticed that for 15 of the query tasks, participants did not manage to complete their result search since it took too much time (more than 120 seconds) and the interface required the user to browse on many pages of the website.

Finally, we questioned each participant to evaluate different aspects of our system and the MAS website. The questionnaire results showed that participants felt that the query tasks results were simpler to find using our system interactive communicator compared to the MAS website. However, they noted that the MAS website appearance was more convenient to represent the query data results. In addition, for some complex queries tasks participants found difficulties in finding a correct result since it requires more parse tree adjustment to return a possible valid query tree. On the other hand, participants noted that for complex queries, the MAS website did not take in consideration their parameters and returned only partial results. The evaluation resulted in an average satisfaction score of 5/5 for simple queries, and for 4/5 for medium queries for our system, where 5 reflects a highly easy to use interface. For the MAS website, the result obtained are 4/5 for simple queries and 3/5 for medium queries.

VI. SUMMARY

We can conclude that our interactive NLIDB system provides concrete improvement to current existing systems that are used to simplify relational database querying for non-technical users.

Firstly, our system introduces a well-defined architecture that assures that each phase of the parse tree is processed with the actual database system schema. This assures to the user that the data result can be accurately retrieved without having to understand the database relationship, schema, and structure.

Secondly, our systems provide an innovative interactive user interface to accurately map each element of the parse tree with the database schema during the parse tree node mapping phase. In addition, our system provides to the user other potential parse tree results to guarantee that the user natural language query is well represented in our parse tree in order to be processed by the query translator component.

Thirdly, our system can easily be improved and maintained since its architecture is structured on separated different components that process each step independently which allows the addition of other steps and rules to be easily implemented to further improve our system. In addition, our system can be implemented on any relational database that uses SQL structure language which improves its usability and flexibility.

However, we noted that our system still needs to be improved to process more complex query structures, since these types of queries can generate many possible adjusted parse trees to the user and limits its potential accuracy. For example, user queries that contain complex sub-query joins conditions cannot always be returned to the user since many possible adjusted query trees can be generated for these types of queries. Therefore, this will increase the average time cost for users to find the correct adjusted tree.

On the other hand, our system can be limited by the current natural language word semantics since certain keywords are not always mapped to the already defined rules. A potential improvement to this limitation, can be the implementation of a machine learning component in order for our system to learn and add potential keywords to the current defined mapping rules.

Finally, compared to the evaluation results obtained when our system is tested against the MAS website, our system can use a better and friendlier user interface to

represent the process of its phases of our NLIDB system. This will improve its usability and would also improve the average time cost spent by each user during the interactive phases.

VII. References

S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In ICDE, pages 5–16, 2002.

S. Bergamaschi, F. Guerra, M. Interlandi, R. T. Lado, and Y. Velegrakis. Quest: A keyword search system for relational data based on semantic and machine learning techniques. PVLDB, 6(12):1222–1225, 2013.

MAS, Microsoft Academic Search
https://academic.microsoft.com/home