Disambiguation and Result Expansion in Keyword Search over Relational Databases

Niousha Hormozi
dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
nhormozi@di.uoa.gr

***Abstract*—Recently, there has been a significant growth in keyword search due to its wide range of use-cases in people's everyday life. While keyword search has been applied to different kinds of data, ambiguity always exists no matter what data the query is being asked from. Generally, when users submit a query they need an exact answer that perfectly meets their needs, rather than wondering about different possible answers that are retrieved by the system. To achieve this, search systems need a Disambiguation functionality that can efficiently filter out and rank all possible answers to a query, before showing them to user. In this paper, we are going to describe how we are improving state of the art in various stages of a keyword-search pipeline in order to retrieve the answers that best match the user's intent.**

***Index Terms*—Disambiguation, Relational Databases, Expansion, keyword search, Information Retrieval, Data Extraction**

I. Introduction

In the last few decades, searching over (semi-)structured data has become an important topic of research due to the vast growth in using such data. However, writing queries in a structured query language such as SQL or SPARQL is challenging for non-expert users, who are not familiar with such languages. Moreover, in real scenarios, the underlying dataset schema is not exposed to them. For these reasons, modern search interfaces over databases support free-form queries and return results that are ranked according to how well they match the input query. Several early approaches support keyword-based queries [1]–[5], while latest approaches, such as NALIR [6], support queries that are written in some form of natural language. In their core, these approaches assume some kind of graph, either a *schema graph* that represents the database relations as nodes and the joins between them as edges [1]–[5] or a *tuple graph* where the tuples become the nodes [7]–[13]. Then, answers to a free-form query are defined as subgraphs over the complete graph, comprising a subset of the relations and tuples that contain the query keywords and are connected by the joins between them. An important challenge faced by these approaches is that many candidate answer subgraphs could be retrieved in response to a user query, each of them representing a different interpretation of the query. Having more than one candidate answer (i.e. getting multiple interpretations in response to a query) is a problem called *ambiguity*. Ambiguity hinders the system from answering a query accurately as well as efficiently because generating too many possible answers is more time-consuming.

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642795.

Example: Let us imagine the following scenario. Alice is looking for italian restaurants and enters a query: "italian restaurant". Based on Yelp dataset, her query could be interpreted in two ways: "a business that is categorized as *italian* and has *restaurant* in its name", and "a business that is categorized as *italian* and *restaurant* in its address". The reason behind getting two different interpretations from this query is that *restaurant* has been found in two columns, i.e. in restaurant = Business(name) and restaurant = Business(address). It is clear that the second mapping (having "restaurant" in address of a business) is not a likely match because single column mappings that carry less/un-related information will finally result in interpretations that are not commonly asked by users.

Problem Statement In this work, we are focusing on the problem of query disambiguation in the context of free-form queries over structured data. Our goal is to distinguish between different meanings a keyword according to database columns and also the interpretations of a query as a whole, which means removing the ones that have a very low chance of being the user's intention and ranking the rest of them in an order that shows their importance. In what follows, we will first give a brief overview of how existing approaches deal with ambiguity and then we will outline our approach. We will present our progress and outline the work lying ahead.

II. Dealing with ambiguity

A. Existing approaches

Free-form search over relational databases has been extensively explored in the literature. Several methods for ranking the answers to a query have been proposed ranging from probabilistic [14] to information retrieval style [15]. However, disambiguation, as a very important part of query understanding, is still a big challenge.

Existing approaches have dealt with ambiguity in different ways. For example, Discover [15] has used the size of the answer graphs as a means to prune "big", less likely, query interpretations. Precis [5] finds all tables that contain at least one query keyword (Initial Relation, IR). Then, it computes all combinations of IRs and connects IRs together via edges that exist in the Schema Graph (SG). Potential subgraphs are pruned by putting constraints based on the edge weights of the subgraph. To select the most likely attribute mapping for a keyword, Demidova et al. in [16] assign a score to each mapping. Their scoring function 1) gives an attribute a higher score if more entities from an attribute have a non-zero value, and 2) gives preference to single-valued attributes that appear only once in an entity description. MeanKS [17] uses a user-interface to ask the user to specify the IR and Initial Attribute of each keyword (they refer to IR as role). They also limit the number of nodes in an answer subgraph by setting a constant number. Finally, they rank the generated subgraphs (they call it MJNSs) according to importance of nodes and edges to produce the top-k tuples.

These approaches to disambiguation have several shortcomings. Using heuristics like the size or weight of an answer graph (e.g., [15] [5]) provides a very coarse pruning criterion based on structure rather than semantics. Secondly, these methods use constant predefined thresholds to prune answer subgraphs, which will not give equally good results for all queries and contexts. And finally, considering all possible attribute mappings and building subgraphs for all combinations is extremely time-consuming as the number of possible query interpretations grows.

B. Our Approach

In our work, we aim to address ambiguity and make free-form search over structured data faster and more content-based. We build on Précis [5]. Précis is a schema-based system (i.e., it uses a schema graph like works [1], [2], [3], [4]). The answer to a Précis query is a logical subset of the database that contains not only the tuples that contain the keywords but also information “around” the keywords that may be of interest to the user. For example, if the query is ”Woody Allen” over IMDB, the system will return not just the tuple “director Woody Allen” but possibly some of his movies too.

The basic system architecture of Precis is the following:

1) *Initial Relation/Attribute Extraction.* The first step is to map query keywords to tables and columns containing them. Précis defines initial relation (IR) a relation that contains at least one tuple with a keyword. Indexes are used to find matches of keywords in relations.
2) *Initial Subgraph Creation.* By connecting IRs through edges that exist in the schema graph Initial Subgraphs (IS) are formed. If no direct edge exists between two IRs, they should get connected via an intermediate relation. Multiple Initial Subgraphs may be created, each one representing a different interpretation of the query.
3) *Subgraph Expansion.* After finding all candidate answers (subgraphs) to a particular query, we could enrich answers by adding extra information that exists in the dataset and is semantically related. For this purpose, each Initial Subgraph is expanded by connecting other relations or attributes (*Expanded Subgraphs*), in order to provide additional information about the query [5].
4) *SQL Query Execution.* Each expanded subgraph is mapped to a SQL query that will return the matching results. Results could be ranked using ranking methods as in [1], [17], [2], [3].

Figure 1 shows our system’s flowchart. White boxes represent Précis modules that are used in our pipeline without any changes and black boxes represent our added modules or enhanced modules of Précis.

1) *Initial Relation/Attribute Extraction.* We map query keywords into columns and tables.
2) *Attribute Disambiguation.* Given the query keyword mappings to database attributes, we differentiate between them by removing the less important ones. For example, having ”italian” in the name of a business is not equivalent to having ”italian” in the ”address”.
3) *Attribute Combination Creation.* Based on the remaining column mappings, we calculate all possible attribute combinations (AC) that have all query keywords.
4) *Initial Subgraph Creation.* For each AC, we find their corresponding Initial Subgraph by connecting their Initial Relations using the edges that exist in the SG.
5) *AC Disambiguation 1 (Cut-off).* Each Attribute Combination is a different interpretation of the query. We semantically differentiate between them and to remove the less important or less likely ones. For this purpose, we ensure that the answers with lower relevancy to the query are pruned safely and the ones with a higher chance of being what the user expects are kept.
6) *AC Disambiguation 2 (Ranking).* Then, we need to rank the rest of Attribute Combinations, based on their meaning and the estimated number of tuples that they will generate. There are two challenges here. First, Attribute Combinations with lower number of estimated tuples are more likely to actually produce no tuples simply because there is no join between their corresponding Initial Relations, while Attribute Combinations with higher estimated number of tuples are more probable to generate tuples. Second, even Attribute Combinations with lower number of tuples might contain information that is interesting to the user and is different from the rest of the answers at the same time (i.e. they need to be in the result set for the sake of diversity).
7) *Expansion.* Given the disambiguated and ranked Initial Subgraphs, we now need to find extra information in the database that are related to the main topic and could help to show more interesting data to the user. For example, when they ask about “capital of Greece”, they may want to see more than just “Athens” in the answer, such as cafes, museums and main attractions in Athens. To the best of our knowledge, only two state-of-the-art systems show expanded results to users [5], [12]. In Precis [5], weights, added on tables and attributes in an offline phase by an expert, enable the system to decide which extra features should be added to the answer. Fakas [18] assigns importance weights to each table

based on a formula that takes only structural features such as relative cardinality, schema connectivity and distance between tables into consideration. We are going to improve state-of-the-art expansion methodologies by making them smarter, content-based, and query-driven.

8) *SQL Query Generator.* Finally, Attribute Combinations will be executed in order of relevancy to the query and diversity, while the ACs that are more probable to be zero-combinations are removed.

III. Planned Methodologies

In Precis, Initial Relation Extraction is responsible to find tables that mention at least one query keyword. Additional to this, we need to keep track of Initial Attributes (IA) and term frequency (TF) in that column for further process. We use the following concepts as defined in Precis:

1) Initial Tuple: An initial tuple is one that has at least one column containing a query term.
2) Initial Relation (IR): An initial relation is one that contains at least one initial tuple.

Additional to Precis concepts, we defined Initial Attribute as an attribute that contains at least one query term.

Based on Figure 1, Initial Relation Extraction is reading the user query and finding relations and columns in the database that contain these query keywords. To do this, we use fulltext indexes which are built on each individual relation (a functionality in MySQL). Generally, these indexes carry information about table and column names for each space separated word in the record, as well as its primary key. After applying fulltext Indexes on tables, this information is saved and is reusable for any inquiry in the future. We take a keyword query Q = $\{w_1, \ldots, w_n\}$, and retrieve Initial Relations and Initial Attributes using Indexes in MySQL. Here, our system checks if all query terms are found in Initial Relations. The algorithm tries to find the keywords and if a keyword is not found, the algorithm does not stop because we assume OR semantics.

Example: Assume that a query "italian restaurant" is submitted over Yelp dataset, which is described by the schema in Figure 2. Initial Relation Extraction module takes as input query keywords as well as the indexes. Here, we list some Initial Relations and Initial Attributes that are found for our example as well as their term frequency:

1) $< italian, Category >=< category, 4411 >$
$< italian, Business >=< name, 492 >$
$< italian, Business >=< fullAddress, 1 >$
2) $< restaurant, Category >=< category, 51625 >$
$< restaurant, Business >=< name, 4476 >$
$< restaurant, Business >=< fullAddress, 3 >$

A. Attribute Disambiguation

After extracting Initial Relations and Initial Attributes for a query, we observed that query keywords may appear in multiple tables and columns. Now, the question is "which of the mappings best represent the user's intention for each keyword?" In the above example, by submitting "italian restaurant" query, the user is looking for answers that refer to "italian" as "Category"/"Name" of a business and not in "Address" of a business. We also noticed that in most cases, there is a connection between term frequency of a keyword in the mapping, and "how much its interpretation is common". In the above example, we can see that "italian" appeared 4476 times in Category(category), but appeared only once in Business(fullAddress), while "italian" as category is considered as a common interpretation and "italian" as part of an address is not. Therefore, our first intuition is that the number of times that a word appears in a column could be related to its commonality as an interpretation, and remove mappings that map to "less common interpretations". By removing non-common mappings here, we solve one level of ambiguity and save time later in the Attribute Combination Creation module which will create combinations on fewer Initial Attributes. For the sake of calculations, we normalized TF dividing by the number of words inside a column. As a result, we calculate the probability of a query keyword, w_i, in initial attribute IA_j as follows:

$$p(w_i) = \frac{\text{Term Frequency of } w_i \text{ in } IA_j}{\text{Number of all terms in } IA_j} \tag{1}$$

Where Query = $\{w_1, \ldots, w_n\}$, *n* being the number of query keywords, *m* being the number of IAs, and $\{AI_j : 1 \leq j \leq m\}$.

Given the set of Initial Attributes and the probabilities of query terms in those columns, a classifier should decide between various mappings of keywords to columns of the dataset. Consequently, this module will only send filtered attributes to our "Attribute Combination Creation module", which correspond to the most frequent interpretations and would eventually increase the overall system's efficiency since this means less processing and calculations in further stages. Additional to probability, we used other statistical profile of the data in our feature set namely, order based on sorted probabilities for each keyword, number of space separated words in column, term frequency of keyword in the database, and exclusivity of keyword in the database to feed into a Logistic Model Tree and achieved 81% accuracy. The fact that we extracted these statistics from the data itself and not from constant predefined conditions makes our approach robust and generalizable to any kind of knowledge graphs in the back-end.

B. Attribute Combination Creation

After removing Initial Attributes that would lead to "less common interpretations", we create all possible subsets of the Initial Attributes (equal to the size of the query) that contain all query keywords.

Example. Interpretation1: [italian = Category(category), restaurant = Category(category)]
which means *a business that is categorized as both restaurant and italian*

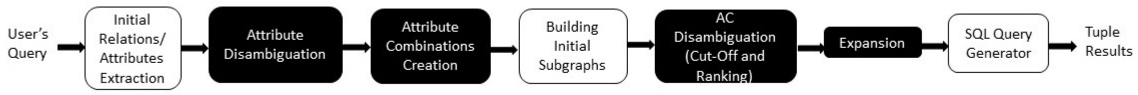
User's Query
Initial Relations/ Attributes Extraction
Attribute Disambiguation
Attribute Combinations Creation
Building Initial Subgraphs
AC Disambiguation (Cut-Off and Ranking)
Expansion
SQL Query Generator
Tuple Results

Figure 1. Our keyword search pipeline.

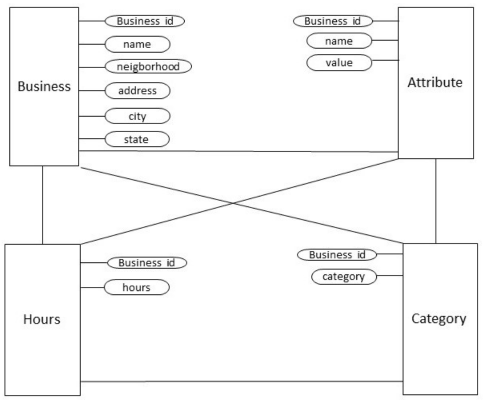

Figure 2. Yelp Schema Graph

Interpretation2: [italian = Category(category), restaurant = Business(fullAddress)]
which means *a business categorized as italian and having restaurant as part of its address*

After Initial Relation Extraction, Precis directly builds ISs by connecting Initial Relations via edges that exist in the SG. However, we apply “Attribute Combinations Creation” module before creating ISs. It takes as input Initial Attributes, Initial Relations and query terms and creates all possible combinations of Initial Attributes that cover all query terms. This new function enables the system to go one level deeper than just tables (i.e. into their columns) and decide which columns to include in the answer, according to statistical profile of the data such as probability, estimated number of tuples, histograms etc. Apart from that, it gives us a higher level of flexibility to evaluate the potential answers in terms of their meaning and commonality, before we generate the final results.

C. Initial Subgraph Creation

Given Initial Relations and Initial Attributes that ”Attribute Combination Creation” module returned, the next step is to find out how their Initial Relations are connected using the SG. The main difference between our ISCreation module and Precis’s is that, when there are different ways to connect Initial Relations together (i.e. different interpretations), Precis chooses the path that leads to higher weight for the whole graph. However, in our case, we have different modules for Disambiguation that are responsible to choose between different interpretations. As a result, in ISCreation module, we only connect the Initial Relations together (whether directly or through an intermediate table). In other words, for us, this module is only responsible to find the foreign/primary key connections for Initial Relations, in order to build the SQL query later on.

Example.

$AttributeCombination1:$ [italian = category(category), restaurant = category(category)]

IS1 = $< Category >$

$AttributeCombination2:$ [italian = business(name), restaurant = category(category)]

IS2 = $< Business - Category >$

In Attribute Combination1, since both keywords are mapped to the same column of a table, IS would only consist of one table (i.e. Category). In Attribute Combination2, as Business and Category tables are directly connected via primary-foreign key (Business-id), their IS would be <Business-Category>.

D. AC Disambiguation (Cut-Off and Ranking)

Each Attribute Combination corresponds to a different interpretation of the query. Oftentimes, more than one interpretation exist for a query. However, Attribute Combinations are not equivalent when it comes to their meaning. Obviously, in our example, interpretation 1 is more probable to be asked compared to interpretation 2. Now the question is: should we keep all interpretations of a query? Should we remove the less frequent interpretations to increase the overall efficiency of the system? What if one user in one thousand asks for it? In this case, should we rank the interpretations? If yes, what measures should be taken into account in order to have a more accurate disambiguated answer?

Our purpose is to use machine learning to learn which attribute combinations are more likely to be meant by the query. We introduce a rich set of features to feed into a machine learning model. This set of features should be able to represent the importance of each Attribute Combination based on statistical profile of the data and rank them accordingly, so that the most important combinations are shown first while ACs that are highly probable to be zero-combinations (not generating tuples) are removed. We applied a Random Committee Model with 4 features namely, Minimum Probability in AC, Number of Tables in AC, Number of Columns in AC and an IR-based formula used by [15]. This model was trained on 2033 samples and scores 0 and 1 as keeping or removing and AC and successfully resulted with 75% accuracy. This module is partially completed and is still in progress.

E. Expansion

Now that we have the disambiguated results in form of subgraphs, we can improve the answers by adding extra information that might be either interesting to user or helpful to him to search with alternative words to find information about what he is looking for. Mutual Information and Entropy are two well known measurements that could help us to distinguish between the rest of column values without adding values with overlapping information.

- Example1: *Address*, *State*, and *City* are columns with overlapping information.
- Example2: *Category* and *AttributeValue* is another set of columns with more or less similar information.

It seems that having both MI and Entropy criterias combined, would give us a good insight into this problem so that having Initial Attributes already in the answer, system decides automatically not to add another column of that set with similar information. To the best of our knowledge, only two papers have already worked on this issue, but not in a semantic and intelligent way. Precis uses predefined weights assigned by expert in an offline phase. Fakas [12] decides between Relations based on an "Importance Formula". However, their importance formula is only based on structural constraints such as relative cardinality, reverse relative cardinality, distance and schema connectivity of relations. This formula distinguishes between tables (and not columns) and is not able to choose semantically. Although we have worked on some experiments for Expansion, it is still in progress.

F. SQL Builder

Finally, all disambiguated, ranked and expanded subgraphs will be sent to a SQL Builder. SQL Builder is responsible to build SQL queries equivalent to each subgraph and execute them in an ordered manner. This module takes Attribute Combinations and their corresponding Initial Relations, along with Primary/foreign keys that we retrieved from indexes. Using this information, each time the system will generate a SQL query dynamically, before executing it on top of the dataset. As a result of this stage, disambiguated, ranked and expanded tuple results would be retrieved and shown to the user. This module is still in progress as we are trying to improve the way that tuples will be shown to the user.

IV. Conclusion

We propose a comprehensive pipeline that consists of separate modules, ranging from processing the keyword query, to different levels of Disambiguation and Expansion. One key advantage of our approach in comparison to the literature, is its capability to prune a number of low-ranked answers in every stage of the work-flow starting at the very beginning and hence, avoid carrying unwanted answers in memory and unnecessary computations on them in each individual module. Another advantage is its adaptability to various queries and contexts. In other words, the decisions are made on the fly and based on submitted keywords and their statistical features in the back-end data. More importantly, our algorithms are completely generalizable to any kind of data structure as long as they are presented as a graph.

References

[1] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 670–681.

[2] S. Agrawal, S. Chaudhuri, and G. Das, "Dbxplorer: A system for keyword-based search over relational databases," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 5–16.

[3] Y. Luo, X. Lin, W. Wang, and X. Zhou, "Spark: top-k keyword query in relational databases," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 115–126.

[4] S. Tata and G. M. Lohman, "Sqak: doing more with keywords," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 889–902.

[5] A. Simitsis, G. Koutrika, and Y. Ioannidis, "Précis: from unstructured keywords as queries to structured databases as answers," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 17, no. 1, pp. 117–149, 2008.

[6] F. Li and H. V. Jagadish, "Nalir: an interactive natural language interface for querying relational databases," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 709–712.

[7] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan, "Banks: Browsing and keyword searching in relational databases," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 1083–1086.

[8] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 305–316.

[9] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 836–845.

[10] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, "Star: Steiner-tree approximation in relationship graphs," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. IEEE, 2009, pp. 868–879.

[11] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 431–440.

[12] G. J. Fakas, Z. Cai, and N. Mamoulis, "Size-l object summaries for relational keyword search," *Proceedings of the VLDB Endowment*, vol. 5, no. 3, pp. 229–240, 2011.

[13] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 505–516.

[14] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic ranking of database query results," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 888–899.

[15] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient ir-style keyword search over relational databases," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, pp. 850–861.

[16] E. Demidova, I. Oelze, and P. Fankhauser, "Do we mean the same?: disambiguation of extracted keyword queries for database search," in *Proceedings of the First International Workshop on Keyword Search on Structured Data*. ACM, 2009, pp. 33–38.

[17] M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu, "Meaningful keyword search in relational databases with large and complex schema," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015, pp. 411–422.

[18] G. Fakas, Z. Cai, and N. Mamoulis, "Diverse and proportional size-l object summaries for keyword search," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 363–375.