Meaningful Keyword Search in RDBMS

Mehdi Kargar, Aijun An, Parke Godfrey, Jaroslaw Szlichta and Xiaohui Yu

Technical Report CSE-2013-03

February 4 2013

Department of Computer Science and Engineering
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

Meaningful Keyword Search in RDBMS

Mehdi Kargar, Aijun An, Parke Godfrey, Jaroslaw Szlichta and Xiaohui Yu
Department of Computer Science and Engineering
York University, Toronto, Canada
{kargar,aan,godfrey,jszlicht}@cse.yorku.ca and xhyu@yorku.ca

ABSTRACT

Keyword search over relational databases offers an alternative way to SQL to query and explore databases that is effective for lay users who may not be well versed in SQL or the database schema. This becomes more pertinent for databases with large and complex schemas. An answer in this context is a join tree spanning tuples containing the query's keywords. As there are potentially many answers to the query, and the user is often only interested in seeing the top-k answers, how to rank the answers based on their relevance is of paramount importance.

We focus on the relevance of join as the fundamental means to rank answers. We devise means to measure relevance of relations and foreign keys in the schema over the information content of the database. This can be done offline with no need for external models. We compare the proposed measures against a gold standard we derive from a real workload over TPC-E and evaluate the effectiveness of our methods. Finally, we test the performance of our measures against existing techniques to demonstrate a marked improvement, and perform a user study to establish naturalness of the ranking of the answers.

1. INTRODUCTION

1.1 Motivation

Much of the world's high-quality data remains under lock and key in relational databases. Access is gained through relational query languages such as SQL. This can suffice for people who are well versed in both SQL and in the schemas of the databases in which they have an interest. However, a lay user—anyone who does not know SQL *or* who is not well versed in the given schema—is effectively locked out. As the schemas of the databases that organizations field become more complex, we all effectively become lay users. *Keyword search over relational databases* was proposed a decade ago [1, 9] to offer an alternative way to query a database that neither requires mastery of a query language such as SQL, nor deep knowledge of the database's schema. While the general approach has merit, many refinements have been needed, and are needed still, before it can be truly effective. As such, this remains a relevant, active research area. [3, 4, 17].

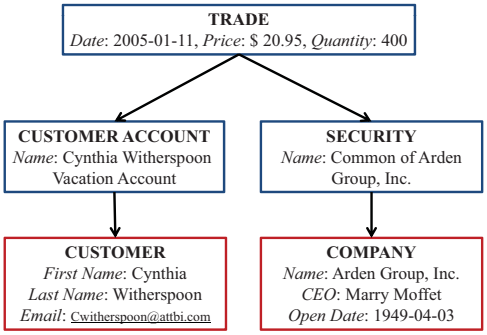

Figure 1: An answer considered non-minimal by DISCOVER.

For keyword search over databases, the database schema is thought of as a *graph*: the relations are the *nodes*, and the foreign key relationships between them are the directed *edges*. A *query* is simply a set of words (*keywords*). The concept of what constitutes an *answer*, however, is more involved. For an answer, we want to convey elements from the database—namely tuples—that *cover* the keywords of the query, and a natural *structure*—a sub-graph of the database's schema—that spans those elements. This sub-graph is commonly called a *network* in the literature [9, 8], but is effectively a *tree*, called a *join tree* in [1].

What is an admissible answer is usually further restricted. We are not interested in *any* tree; some may only very loosely connect the tuples containing our keywords. Previous work restricts answers over *minimal* trees [9, 8], meaning there is no answer over a sub-tree of the tree in question. However, there is another aspect of an answer: relevance to the query. By restricting answers to a minimal structure, some relevant answers may be missed.

Consider the keyword query *"Cynthia Arden"* over the TPC-E[1] schema. The TPC-E benchmark simulates the *online transaction processing* (OLTP) workload of a brokerage firm.[2] The keywords *Cynthia* and *Arden* may appear in dif-

[1]http://www.tpc.org/tpce/
[2]The TPC-E schema contains 33 tables and is shown in Fig. 5. The database models information about financial transactions such as traded companies, fees of brokers, customer accounts and their related holdings, and the type of

ferent relations. Each could refer to the name of a *customer*, *broker*, *company*, or CEO of a *company*, or be in the title of a *news item*. Of course, each of these different relations for *Cynthia* and for *Arden* potentially lead to rather different answers. For example, Fig. 1 shows an answer for the keyword query *"Cynthia Arden"* on TPC-E, which says that a customer *Cynthia* buys the stocks of a company named *Arden Group*. This is an interesting and relevant relationship between *Cynthia* and *Arden* assuming the user wants to find out the relationship between customer *Cynthia* and company *Arden Group*. However, such an answer may be missed if we restrict the answer to be structurally minimal while covering all the query keywards. This is because word *Cythia* also appears in an intermediate node of the tree and thus the *customer* node is pruned. The same problem arises for word *Arden*.

Another issue in keyword search is to score answers for *relevance*. Some answers are more relevant than others, and thus the answers should be presented in order of descending relevance. The relevance of an answer depends on many factors: the tuples in the answer and how the keywords appear in them; the bridging tuples that do not contain keywords; and the join tree. How to measure the value of each of these is open to question. How then to combine these measures into a single relevance score is also open to debate.

Prior work has addressed relevance. In [9], they offer the simplistic solution of scoring relevance as the reciprocal of the number of edges in an answer's tree. This heuristic assumes that fewer joins involved mean the tuples are related more closely. This approach might work for a small and simple schema, but it fails to return relevant answers when the schema is large and complex. Consider the query {*Anderson Joseph*} over the TPC-E schema. Also, assume that *Anderson* refers to a company name (e.g., Andersen Group, Inc.) and *Joseph* refers to a customer name (e.g., Joseph Coronado). Fig. 2 shows four possible join trees of different sizes that could produce answers that connect company *Anderson* and customer *Joseph*.

If we rank the trees according to their size (i.e., the number of edges or nodes), the first tree (a) which connects a customer and a company when they have the same status gets the highest rank. However, based on the TPC-E schema description, this is not a strong relationship; *status type* is a dimension table which is connected to six tables in the schema and stores the status value for other entities (such as companies, customers and trades). An answer derived from this tree would say that both *Anderson* and *Joseph* are *active*. Looking at the *customer* and a *company* tables in TPC-E, it turns out that all the customers and companies have the *Active* status. Therefore, any given customer and any given company in the TPC-E database share the same status through the *status type* table. Thus, the first found relationship (a) between *Anderson* and *Joseph* is, in fact, not that interesting or relevant. The second tree (b) does not reveal a strong relationship between *Anderson* and *Joseph* either. It states that the security traded on the company (in this case, *Anderson*) has the same status as the queried customer (i.e. *Joseph*). This is similar to the first tree (a). The third tree (c) states that company *Andersen*'s stock is traded by customer *Joseph*. This is one of the strongest relations between a customer and a company and is related to the purpose of the TPC-E schema. In addition, it usually produces few results since trading the stocks of one specific company by one specific customer is not common. The fourth tree (d) is the largest and therefore it is not straightforward to be interpreted easily. The purpose of the keyword search is to help users to understand and explore the database more conveniently. This tree states that the company *Anderson* has the same status as a security which is related to a company that has the same address as customer *Joseph*. Two relations, *address* and *status type*, that are involved in this tree makes it less meaningful and more difficult to interpret for the user.

In [8], the authors take a different tact: they apply information retrieval (IR) measures to the answers to determine an individual measure per answer. This leverages approaches from IR that work well in other domains of keyword search (e.g., keyword search on the web). An answer in this case, however, need not cover *all* the query's keywords to score well and the approach de-emphasizes the importance of the tree. We believe that for keyword search in relational databases the relevance of the tree from which answers derive is paramount. We hypothesize that relevance as measured by adapted IR techniques is much less effective. For example, for the trees in Fig. 2, assume that the IR scores of the tuples that contain keywords are the same (e.g., *Joseph* is the first name of a customer and the related column contains only "*Joseph*"). Then, applying the ranking technique of [8], the first tree (a) gets the highest score. As we discuss previously, this is not a relevant answer for the given query.

Previous work on relevance has been tested over simple schemas. For application over more complex schemas, the importance of the schema (thus, answers's trees) becomes more pronounced. It is *how* the tuples are related (via the joins) that is meaningful for search in the relational domain.

1.2 Objectives and Contributions

In this work, we address the above problems by (1) defining answers that incorporate the user's interest and (2) devising meaningful relevance scores for answers.

We address the first issue by identifying the meaning of each query keyword (such as whether keyword *Joselph* is a customer's name or a company's CEO's name) either through user interactions or using an automatic method for catching the meaning of a keyword query. A first step to keyword query evaluation in a relational database is to determine the relations in the database that contain the query keywords. We call a relation that contains a keyword a potential *role* for that keyword. Finding the potential roles for the query's keywords can be done efficiently, by using an inverted index structure or existing built-in support for full-text keyword search in the RDBMS [17].

Identifying the most relevant role of a query keyword from its potential roles is beneficial to keyword search. First, the query is focused on roles of interest. Second, this will admit meaningful answers that would not otherwise be found. The answer in Fig. 1 is *not* found by DISCOVER [9] because of the minimality constraint they add to their definition of "answer", and that roles are not part of the definition. This minimality issue is addressed in depth in Section 2.

To address the second problem, to devise meaningful relevance scores for answers, we seek to measure importance of edges and nodes (foreign keys and relations, respectively) in the graph (the database schema). The relevance of an answer's tree can be then determined based on the impor-

traded securities.

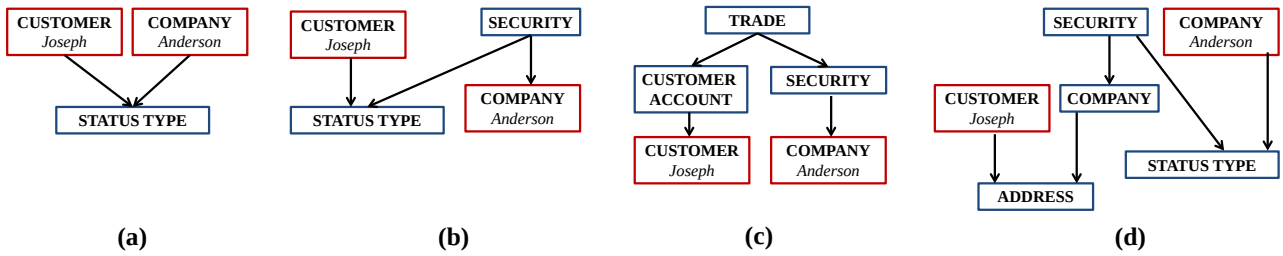

Figure 2: Four possible trees for the query *"Andersen:company Joseph:customer"*.

tance value of its nodes and edges. Our relevance measures only rely on the database's schema and data. We do not rely on other models, workloads and logs, or other external information. Our node and edge relevance can be computed offline and efficiently.

Our contributions are as follows.

1. *Model for keyword queries in relational databases.*

   Use schema-based ranking and keyword roles to solve the problem of keyword search over relational databases.

   Redefine *answer* via roles (as discussed above and defined in Section 2) that captures important answers missed by previous techniques.

2. *Schema-based ranking.*
   (a) Devise importance measures for nodes, importance measures for edges, and a hybrid measure of the two.
   (b) Devise relevance measures for join trees derived from the schema relevance. Consider the effect of penalizing larger trees.
   (c) Construct a *gold standard* for relevance of answers from an extensive workload of real SQL queries. This is used to evaluate the effectiveness of our measures which do not require such external information.

3. *Evaluation.*

   Establish the efficacy of our approach by evaluation.
   (a) Perform a comprehensive evaluation based on TPC-E to demonstrate the viability of our methods and to compare against existing methods. The experiments are over a larger and more complex schema (TPC-E) than the experiments in previous work.[3]
   (b) Run a user study to establish the meaningfulness of answers and ranking generated by our system.

The paper is organized as follows. In Section 2, we present our framework. In Section 3, we devise our measures for relevance of nodes and edges, and for join trees. We show how to compute these efficiently. In Section 4, we discuss the comparison of the methods against a gold standard, we evaluate our methods against existing methods, and we present the results of the user study. In Section 5, we discuss related work. In Section 6, we conclude.

2. FRAMEWORK

[3]In [4], keyword search over the Credit Suisse data-warehouse is considered, which has a complex schema. However, the authors use meta-data and patterns to build a model. We assume no extra information.

2.1 Formal Problem Definitions

A relational database schema consists of a set of n relation schemas, denoted as $\{R_1, R_2, \ldots, R_n\}$, where each R_i is described by a set of attributes. Two relation schemas, R_i and R_j, may be related by a *foreign key relationship*, denoted as $R_i \leftarrow R_j$, where the primary key of R_i is referenced by the foreign key of R_j. Thus, a relational database schema can be considered as a graph $G = \langle V, E \rangle$, where nodes are relation schemas and edges represent the foreign key relationships.

A relational database is an instance of a relational database schema G. It consists of a set of relations, where each relation $r(R_i)$ is a set of tuples conforming to the relation schema R_i in G. Given a relational database D and a set of l (≥ 2) keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), the problem of keyword search over D is to find a set of tuples that are connected via foreign key relationships and cover all the keywords in Q. The most representative algorithm for this problem is DISCOVER [9], which finds *minimal total joining networks of tuples* defined as follows.

DEFINITION 1. ***Minimal Total Joining Network of Tuples (MTJNT)**: Given a database with schema graph G and a query Q containing a set of keywords, a minimal total joining network of tuples is a tree T of tuples that satisfy the following conditions.*

- ***Joinable**. For each edge (t_i, t_j) in T, where $t_i \in r(R_i)$ and $t_j \in r(R_j)$, there is an edge $R_i \leftarrow R_j$ or $R_i \rightarrow R_j$ in G and $t_i \bowtie t_j \in r(R_i) \bowtie r(R_j)$.*
- ***Total**. Each keyword in Q is contained in at least one tuple in T.*
- ***Minimal**. If a tuple in T is removed, T is either not joinable or not total.*

The DISCOVER algorithm generates all the MTJNTs given a database and a query. It does not specify the role of a query keyword (i.e., it does not care in which relation a query keyword appears). Thus, an answer that contains a tuple with a keyword in an interesting relation may not be found if the answer is not *minimal*. This occurs when an intermediate tuple connecting the tuples in the interesting relations with the query keywords also contains a query keyword, which leads to a tuple in an interesting relation being pruned to make the network minimal. This was illustrated in Fig. 1.

In our framework, the role of a query keyword is identified through user interactions or detected automatically using a method such as the one in [3]. For each query keyword, our algorithm first finds the list of columns (and relations) that contain the keyword using an inverted index or the built-in support for full-text keyword search in RDBMS [17], and

then a relation is chosen (either through interaction with the user or automatically) from the list as the ***role of the keyword***. Role selection is discussed in Section 2.3. With specified keyword roles, our algorithm searches for answers that are defined as follows.

DEFINITION 2. ***Minimal Joining Network of Tuples Covering Roles***. *Given a database D with schema graph G and a query Q containing a set of keywords $\{k_1, k_2, \ldots, k_l\}$ and their respective roles $\{r_1, r_2, \ldots, r_l\}$ (where r_i's are relations in D), a minimal joining network of tuples for query Q is a tree T of tuples that satisfy the following conditions:*

- ***Joinable***. *For each edge (t_i, t_j) in T, where $t_i \in r(R_i)$ and $t_j \in r(R_j)$, there is an edge $R_i \leftarrow R_j$ or $R_i \rightarrow R_j$ in G and $t_i \bowtie t_j \in r(R_i) \bowtie r(R_j)$.*
- ***Role and keyword covering***. *For each query keyword role r_i, there exists a node t_j in T such that $t_j \in r_i$ and t_j contains keyword k_i.*
- ***Minimal***. *If a tuple in T is removed, T is either not joinable or does not cover all the roles or all the keywords.*

For brevity, we refer to answer as defined in Definition 2 as ***final answer*** in this paper. Given a database, there may be many final answers to a query. Instead of producing all the answers which may overwhelm the user, the goal of our algorithm is to produce the top-k most meaningful final answers.

2.2 Methodology

The final answers defined above can be generated through a sequence of join operations on the database. To generate such answers, we first generate the *minimal joining networks of schemas (MJNSs)* that represent the join operations for producing the final answers. First, we define MJNS, and then present our method for generating them.

DEFINITION 3. ***Minimal Joining Network of Schemas (MJNS)***: *Given a database D with schema graph G and a set r of query keyword roles $\{r_1, r_2, \ldots, r_l\}$ (where r_i's are relations in D), a minimal joining network of schemas that cover r is a tree T of relation schemas in G that satisfy the following conditions.*

- ***Joinable***. *Each edge in T is an edge in G. That is, each edge in T represents a foreign key relationship.*
- ***Role covering***. *For each query keyword role r_i, its schema is in T.*
- ***Minimal***. *If a relation schema in T is removed, T is either not joinable or does not cover all the roles in r.*

Note that our MJNSs bear similarity to the *candidate networks (CN)* used in the DISCOVER algorithm [9], but with the following differences. First, a CN is defined as a network of tuple sets, while our MJNS is defined at the schema level. Second, our MJNS must cover a set of specified roles (i.e., it must contain the schemas of a set of specified relations), while a CN does not need to. This second difference allows us to find interesting final answers that DISCOVER misses as discussed earlier.

To generate MJNSs, we use a breadth-first search algorithm similar to the CN generator of DISCOVER. Our algorithm starts with a role schema as an initial tree T and extends T with a relation schema in G that has a foreign key relationship with a node in T. The expansion of T stops once the schemas of all the roles are covered in T. To avoid generating duplicate trees, each generated tree is assigned an ID based on tree isomorphism during the execution of the algorithm. The ID of a tree is checked with the existing IDs that are generated so far and the current tree is accepted if it is not generated previously. Fig. 2 shows four MJNSs generated for query $\{Andersen, Joseph\}$ and their respective roles $\{Company, Customer\}$ over the TPC-E database.

After MJNSs are generated, final answers can be produced by creating an execution plan to evaluate the MJNSs. Note that an MJNS may or may not produce a final answer[4]. But a final answer can be produced by one and only one MJNS. The union of the final answers produced by all the MJNSs is the set of all possible final answers.

Since many final answers can be generated for a query but some answers may not be interesting, we aim at producing top-k most interesting final answers. To achieve this, we first limit the number of nodes in an MJNS (which is a strategy taken by DISCOVER as well). This limits the size of final answers too. The rationale is two-fold. First, if two tuples in a final answer are far away from each other, it is not easy to interpret the answer. Second, executing the query associated with a large MJNS is time consuming. Thus, a size control parameter, D_{max}, is used to specify the maximum number of allowed nodes in an MJNS. In addition and **more importantly**, we rank the generated MJNSs according to an interestingness measure so that final answers from the top-ranking MJNS are produced first. If the number of final answers produced so far is less than k, the next MJNS is used to produce more final answers until k final answers are produced.

The overall procedure of our search method is as follows. Given a database D, a query containing keywords $\{k_1, \ldots, k_l\}$, an answer size control parameter D_{max}, and a maximum number k of final answers to be returned.

1. For each keyword k_i, find the relations in D containing k_i using an inverted index or the built-in support for full-text keyword search in RDBMS [17].
2. Select a relation containing k_i as the role of k_i either through user interaction or automatically using a role-ranking method in [3].
3. Generate the MJNSs that cover all the selected roles and whose size is no more than D_{max}.
4. Rank the generated MJNSs according to an interestingness measure.
5. For each MJNS m_i in the ranked list, evaluate m_i to generate a set s_i of final answers, rank the answers in s_i according to a content-based IR-style ranking measure, and add the answers in the ranked order into the

[4]In [9] it is claimed that all the candidate networks (CN) generated by DISCOVER lead to generation of an MTJNT (the final answer of DISCOVER). But its CN generation algorithm works at the schema level without checking whether a join in CN can produce an answer. Thus, there is no guarantee that a CN can produce at least one final answer.

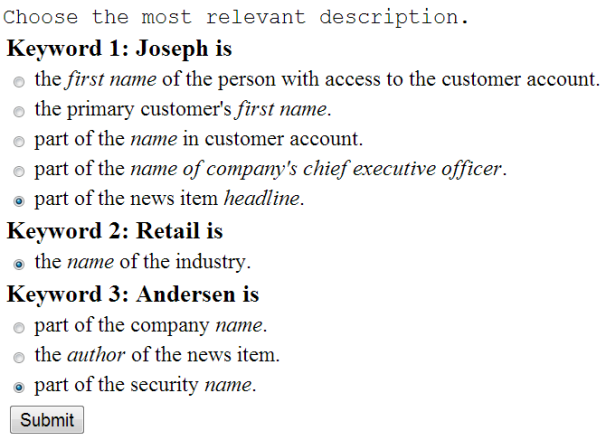

Figure 3: Role selection by the user for query "Joseph Retail Andersen".

final answer set A. This procedure stops until either A contains k answers or all m_i's have been evaluated.

The main focus of this paper is on Step 4: how to rank MJNSs so that the most interesting answers will be presented to the user first and less interesting ones can be pruned. Note that the IR-style ranking measure in Step 5 is a secondary ranking measure for locally ranking the final answers generated from each MJNS, which is exactly the same as the method in [8]. The role selection (Step 2) is presented in the next subsection. In Section 3, we present a number of measures for ranking MJNSs in Step 4.

2.3 Role Selection

A keyword may appear in multiple relations and columns. The role of a keyword is a relation with a column that contains the keyword and also matches the user's intention for the keyword. As discussed before, identifying the role of each query keyword and requiring the final answer to cover the roles of all the query keywords can avoid missing interesting answers and can also cut down the search space. Below we briefly describe two role selection approaches that can be used in our framework.

In the first approach, a user-interface is designed to allow the user to specify the role of each keyword without needing to know the database schema. A short natural language description of each attribute (i.e., table column) in the database is stored in the keyword search system. Such descriptions tell the meanings of the attributes and can be provided by people (such as DBAs) familiar with the database schema[5]. During keyword search, once the columns that contain a keyword in their values are found using the inverted index or the built-in DBMS keyword search function, the short descriptions of such matched columns are shown to the user in the user interface [13]. The user can then choose from them the most relevant description for each query keyword. For example, Fig. 3 shows the user-interface that lists the descriptions of the matched columns for query *"Joseph Retail Andersen"* over the TPC-E database. In this example, five columns contain *Joseph*, one contains *Retail* and three contain *Andersen*. For each keyword, the user can select the description that best matches his/her intention.

[5]In our experiments on the TPC-E database, the short attribute descriptions are taken from the TPC-E document.

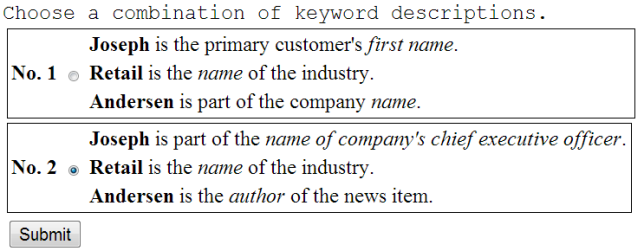

Figure 4: Role selection from the top-2 results from the automatic method in [3] for query "Joseph Retail Andersen".

Based on the user's selection, the most relevant column (and hence its relation, i.e., the role) of each keyword is identified.

The second approach assumes that the meaning/role of each keyword depends on the meaning/role of other keywords in the query. It evaluates combinations of columns, each containing one matched column for each keyword. Such an evaluation can be done automatically using a method introduced in [3], which ranks the combinations based on the likelihood that the role of each keyword represents the intended semantics of the keyword. In [3] the authors study the inter-dependencies across the ways that different keywords are mapped into the database values and schema components. They extend the Hungarian algorithm [5] for generating and ranking different interpretations of keyword queries. Based on the rankings from [3], we can either take the top-ranking combination of roles (if fully automatic role selection is preferred) or let the user select one from the top-k combinations (if user interaction is desired). Fig. 4 shows the user interface that allows the user to select a combination of roles from the top-2 results returned from the method in [3]. Since role selection is not the focus of this paper, we omit the details of the method in [3] and further discussion on role selection due to the space limit.

3. RANKING MODELS

We propose several methods for ranking minimal joining networks of schemas (MJNSs). The proposed methods work just based on the database schema and its given instance, assuming that no extra information (e.g., query logs or inheritance relationships among the tables) is available. The methods use information-theoretic measures to evaluate the importance of a relation and/or an edge in the given database, and rank the MJNSs based on the importance of the relations and/or edges in an MJNS. We classify the proposed methods into three categories: (1) ranking based on the importance of the nodes in an MJNS; (2) ranking based on the importance of the edges in an MJNS; and (3) ranking based on the importance of both nodes and edges in an MJNS.

3.1 Ranking by Importance of Nodes

Given a database D, this type of methods assumes that the importance of an MJNS M is related to the importance of the tables in D that instantiate the schemas in M. Here, we propose a ranking method called *key entropy transfer* (KE) that ranks the tables in D. Consider $G_D = (V_D, E_D)$ as an undirected graph representing a relational database D, where V_D is the set of nodes representing relations (i.e., tables) in D and E_D is the set of edges representing foreign key relationships. The KE method builds a node-to-node

transition probability matrix M based on the entropies of table attributes, and performs a random walk on G_D with M. The steady-state probabilities of the random walk are then assigned as the importance scores of the tables.

Let $r.A$ denote an attribute A in table r, and let a represent a value of $r.A$. The *entropy* of $r.A$ is defined as follows:

$$H(r.A) = -\sum_{\forall a \in r.A} p(a) \times \log p(a) \quad (1)$$

where $p(a)$ is the probability that a occurs in column $r.A$ (i.e. $P(r.A = a)$).

For each table in D, a primary key is created that consists of all of the attributes in the table, and a self-loop using this key is added to the corresponding node in G_D. This is done even when the table already has a primary key. The purpose of the self-loop is to keep some of the table's information within the table during the random walk (i.e., to make the random walk have more probability to stay at a node).

In [18] it is suggested that the importance of a table is equivalent to the entropy of all of its attributes, including the numeric attributes. To compute the entropy for numeric attribute, discretization should be performed first in order to compute the probabilities involved in the entropy computation. Thus, the entropy value of a numeric attribute greatly depends on the discretization method used or the parameter value used in a discretization method (such as the number of intervals in the equi-width discretization method). To avoid such dependencies, for a numeric attribute, we set its entropy to the maximum value, which is $log|r|$. We further discovered in our experiments that for a non-key attribute (i.e., an attribute that is not a primary or foreign key), using its true entropy or the maximum entropy (i.e., $log|r|$) does not make much difference in terms of finding meaningful MJNSs (which will be shown in our experimental results). Thus, to speed up the entropy computation, we use $log|r|$ as the entropy value for all the non-key attributes. Since the entropies of only primary and foreign keys are truly computed, this method is referred to as the *key entropy (KE)* method.

We define the *information content* of a table r as follows:

$$IC(r) = (|C_r^{nj}| + 1) \times \log |r| + \sum_{A \in C_r^j} H(A) \quad (2)$$

where C_r^{nj} and C_r^j denote the set of non-key and key columns of table r, respectively. (Each column in C_r^j is part of at least one edge and is either a primary or foreign key attribute.) In this equation, each non-key column has the $\log |r|$ contribution to the importance of the table. The more non-key columns each table has, the more important it is. The number of non-key columns are summed up with one to reflet the self primary key join. The IC value of a table measures the importance of the table without considering the foreign key relationships between tables.

To take into account such relationships, the *information transfer* rate on a join edge starting from $r.A$ (no matter if $r.A$ is a primary or foreign key in the connection) is defined as follows:

$$T(r.A \to r'.A') = \frac{H(r.A)}{IC(r) + \sum_{X \in C_r^j}[(n_{r.X} - 1) \times H(r.X)]} \quad (3)$$

where $n_{r.X}$ denotes the total number of join edges involving attribute $r.X$, including the self loop primary key. Since the self loop contains all the attributes of r, $n_{r.X} \geq 1$. Note that $T(r.A \to r'.A')$ does not depend on $r'.A'$, which means that the same amount of information is transferred from $r.A$ along any edge starting from A.

A *transition probability matrix* for the random walk is then built by

$$\Pi(r, r') = \sum_{r.A - r'.A'} T(r.A \to r'.A') \qquad r \neq r' \quad (4)$$

where the sum ranges over all the join edges between r and r' in the database graph G_D. Additionally, $\Pi(r, r)$ is defined as:

$$\Pi(r, r) = 1 - \sum_{r \neq r'} P(r.A \to r'.A') \quad (5)$$

The probability matrix is an $n \times n$ matrix, where n is the number of tables in the database. It is a matrix with numbers between 0 and 1 and each row sums up to 1. If two tables r and r' are connected together in G_D, $\Pi[r, r']$ is greater than zero and its value determines the amount of information transferred along the (r, r') edge. If more than one join edge exists between two tables, the information on each join edge is summed up.

The importance of a table r is defined as its stable-state probability of a random walk on G_D with the probability matrix Π. For any given probability matrix Π over a connected and non-bipartite graph G, there exits a unique stationary distribution Ξ [16]. Therefore, the table's importance in the above model is well defined. The stationary distribution vector can be obtained by applying an eigenvector calculation method. We use an iterative method used in [18]. The method starts with an arbitrary non-zero vector Ξ_0[6]. Then, $\Xi_{i+1} = \Xi_i \times \Pi$ is computed repeatedly until the distance between Ξ_i and Ξ_{i+1} is less than a given threshold. Setting the threshold to zero results in stopping the method when the stationary distribution is reached.

The importance scores of the tables in a database can be computed offline before a keyword search starts[7]. During the keyword search we use the scores to rank the minimal joining networks of schemas (MJNSs). Given a set of query keywords and their corresponding roles, all the MJNSs generated in the procedure described in Section 2.2 share the same set of role relation schemas. Thus, to rank the MJNSs, we only need to consider the non-role relation schemas in each MJNS. We use the average importance score of the tables associated with these non-role relation schemas to compute an importance score for an MJNS M, as defined below:

$$Score_{table}(M) = \frac{\sum_{r \notin Roles} TblScore(r)}{n} \quad (6)$$

where $Roles$ is the set of role relations, n is the number of non-role relation schemas in M, and $TblScore(r)$ is the importance score of table r computed using the KE method. Note that if a non-role relation schema appears more than once in the given MJNS, it is counted more than once (equal

[6]The final value of stationary distribution Ξ does not depend on the initial value Ξ_0. However, we set Ξ_0 to $IC(r)$.

[7]The scores should be updated periodically if the database content changes.

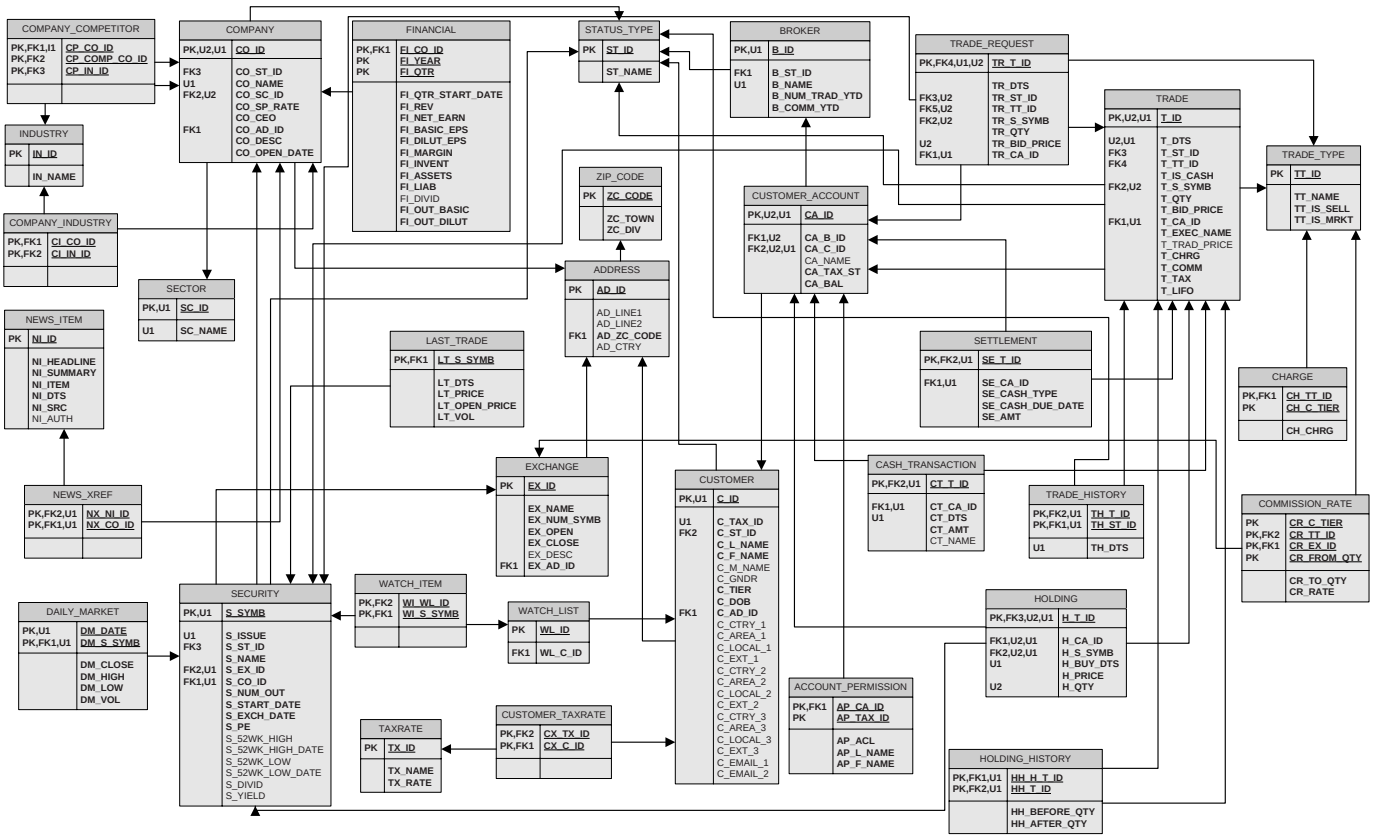

Figure 5: The TPC-E schema graph. The direction of the edges are from the foreign key to the primary key.

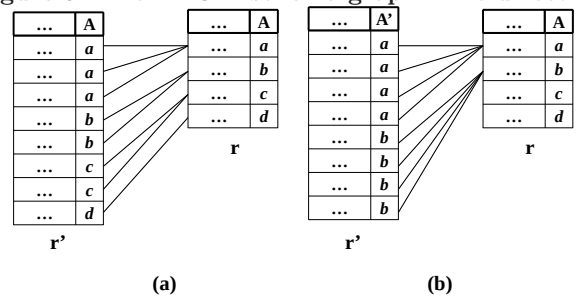

Figure 6: Two different instances of the foreign key connections between tables r and r'

to the number of occurrences in the MJNS). This is reasonable since the more a relation schema appears, the more important it is for connecting the role relation schemas together in the MJNS.

3.2 Ranking by Importance of Edges

Another approach to ranking the MJNSs is based on the importance (strength) of the edges (i.e. foreign key connections) that connect the nodes in an MJNS. Below we present two measures for ranking the edges in the database schema.

Intuitively, edge strength can also be measured by the fraction of the join key values being instantiated. The more fraction of primary key values are instantiated, the more important the edge is [18]. However, in [18], the authors also assumed that by increasing the number of connections between two tables, the importance of the edge decreases. As we will show in the experiments, this assumption does not work for finding meaningful MJNSs. We propose the following measure, called *instantiation fraction* (IF), to quantify the importance of an edge based on the fractions of instantiated key values.

$$ST_{IF}(r.A, r'.A') = \frac{N^{inst}_{r.A}}{N_r} \times \frac{N^{inst}_{r'.A'}}{N_{r'}} \quad (7)$$

where $N^{inst}_{r.A}$ is the number of tuples in r that instantiates the edge between $r.A$ and $r'.A'$, and N_r is the total number of tuples of table r. $ST_{IF}(r.A, r'.A')$ of the the left and right instances in Fig. 6 is equal to 1 and 0.5, respectively. We believe the left instance represents a stronger association between r and r'.

A modification of the above model is to consider the entropy of each column. By adding the entropies, the information content of each column is taken into account. This version of the IF measure, denoted as IF_Ent, is defined as

$$ST_{IF\_Ent}(r.A, r'.A') = \frac{N^{inst}_{r.A}}{N_r} \times \frac{r^{inst}_{r'.A'}}{N_{r'}} \times H_{norm}(r.A) \times H_{norm}(r'.A') \quad (8)$$

where $H_{norm}(r.A)$ and $H_{norm}(r'.A')$ are normalized entropies of $r.A$ and $r'.A'$, respectively. The values of $\frac{N^{inst}_{r.A}}{N_r}$ and $\frac{N^{inst}_{r'.A'}}{N_{r'}}$ lie between zero and one. Thus, to make instantiation fractions and entropies equally important to the strength of the edge, normalized entropies in range $[0, 1]$ are used.

The same as the table importance scores, edge importance scores defined above can be computed offline. During a keyword search, to rank the MJNSs by edge importance,

we compute a score for each MJNS using its average edge importance score, defined below:

$$Score_{edge}(M) = \frac{\sum_{\forall(r.A,r'.A')\in M} EdgeScore(r.A, r'.A')}{m} \quad (9)$$

where M is an MJNS, $EdgeScore(r.A, r'.A')$ is an edge strength function and can be either ST_{IF} or ST_{IF_Ent}, and m is the number of edges in the MJNS M. The same as in ranking MJNSs based on the node importance, if one edge appears more than once, it is counted more than once. Since the edge strength scores are pre-computed, computing $Score_{edge}$ is fast and efficient.

3.3 The Hybrid Ranking Model

Our last approach to ranking MJNSs is to rank them based on the importance of both nodes and edges. In this *hybrid model*, we consider node importance when measuring the edge strength. The new edge strength is computed as follows:

$$EdgeScore(r.A, r'.A') \times \frac{TblScore(r) + TblScore(r')}{2} \quad (10)$$

If we use ST_{IF} for edge strength and the KE method for computing node importance, the hybrid formula becomes:

$$ST_{IF\_KE}(R.A, R'.A') = ST_{IF}(R.A, R'.A') \times \frac{KE(R) + KE(R')}{2} \quad (11)$$

Since the value of $ST_{IF}(R.A, R'.A')$ lies between zero and one, $KE(R)$ and $KE(R')$ should be normalized into range $[0, 1]$.

To rank the MJNSs, the score of an MJNS is computed using Equation 9 with ST_{IF_KE} as the $EdgeScore$ function.

3.4 Penalizing Larger MJNSs

In all the MJNS ranking methods we described above, the average of node/edge importance scores is used to compute a score for an MJNS. Thus, the size of the MJNS (i.e., the number of nodes) is ignored in these ranking methods. Although we have shown that ranking MJNSs purely based on their size does not return satisfactory results in databses with large and complex schema, completely ignoring the size may not be a good strategy either. Generally, interpreting and understanding larger MJNSs is harder than interpreting smaller ones. Thus, for two MJNSs with similar average node or edge scores, the one with the smaller size should be ranked ahead of the larger one. To achieve this purpose, the final score of an MJNS M can be adjusted as[8]:

$$Score_{new}(M) = Score_{old}(M) \times \frac{1}{\log(tree\_size)} \quad (12)$$

where $tree\_size$ is the number of nodes in M, and $Score_{old}$ can be either $Score_{table}$ or $Score_{edge}$.

In the following section, we evaluate all the scoring methods with and without the penalization factor.

3.5 Runtime Discussion

The runtime of our methods are similar to that of DISCOVER I [9]. All MJNSs have to be found for the keyword query under the D_{max} threshold. The primary overhead is evaluating the SQL queries associated with the MJNSs. For us, this overhead is marginally more than DISCOVER I because our networks can be marginally larger due to roles. As with DISCOVER I, though, we can evaluate MJNSs in a pipelined fashion from highest to lowest relevance until we have achieved enough answers. Thus, we usually stop early, not having needed to evaluate most of the MJNSs. We are not directly comparable with DISCOVER I on a query by query basis, since the networks between the two may be ordered quite differently. Therefore, the number of networks needed to be evaluated for a given query can vary widely.

Our runtime compares very favorably against DISCOVER II [8]—just as DISCOVER I does. This is because DISCOVER II is comparably quite expensive. It must evaluate *all* its networks before the answers can be ranked. It has no options to stop early.

[8]This formula was used in [14] to penalize large XML trees.

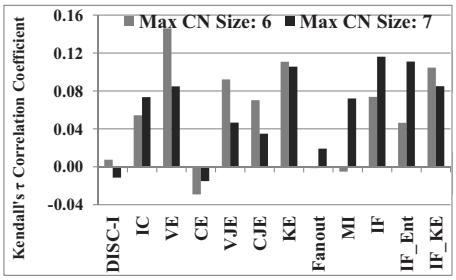

Figure 7: Results of Kendall's τ coefficient without penalizing larger MJNSs.

4. EXPERIMENTAL EVALUATION

We evaluate our proposed ranking methods for finding the most meaningful/relevant MJNSs. All of the evaluated methods are implemented in Java. The experiments were performed on a performance test machine with an Intel(R) Core(TM) i7 2.80 GHz processor and 4GB of RAM.

4.1 Dataset and Experimental Setup

The experiments are conducted over the TPC-E database. TPC provides a transaction log which we use to generate a gold standard for ranking MJNSs (Section 4.2). Since no active transaction is performed, table *trade request* is not loaded with any data. Therefore, in our experiments, table *trade request* is removed from the schema along with all of its foreign key connections. *EGen*, a package from TPC, is used for generating an instance of the database. The parameters of *EGen* are set to the same as those in [18, 19]. The number of customers, initial trade days and scale factor are set to 1000, 10, and 1000, respectively.

The focus of this work is ranking the MJNSs. As described in Section 2, the input to our MJNS generator is the keyword roles. That is, the main input to our ranking methods and other methods is a set of relation names (i.e., query keyword roles). Table 1 lists ten sets of query keyword roles that we use as the input query templates in our evaluation. For example, the first set of keyword roles specifies the names of two relations: *Customer* and *Company*, which may result from keyword query {*Joseph, Andersen*}. As another example, the fourth query template is meant to find the relationships between two customers.

The MJNS generator also receives the maximum size of the MJNS as input. Since TPC-E has a large schema with

Table 1: List of 10 sets of query keyword roles.

No.	Keyword Roles (Query Templates)
1	Customer, Company
2	Company, Broker
3	Customer, Broker
4	Customer, Customer
5	Customer, Company, Industry
6	Customer, Company, Trade Type
7	Customer, Company, Broker
8	Customer, Company, Exchange
9	Customer, Company, Broker, Security
10	Customer, Company, Broker, Customer Account

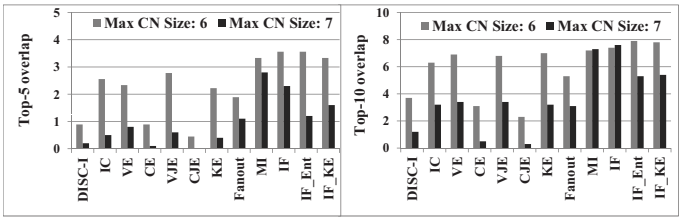

Figure 8: Results of the top-5 and top-10 without penalizing larger MJNSs.

dimensional tables, setting the maximum MJNS size to a value less than six results in generating MJNSs that are mostly connected through dimensional tables. On the other hand, setting the maximum value larger than seven results in generating many MJNSs, whose results are hard to interpret by the users of the system. Thus, the maximum size (i.e., D_{max}) of the MJNSs is set to six and seven in our experiments.

4.2 Gold Standard and Performance Measures

TPC provides twelve transactions along with the TPC-E benchmark. The set of transactions represents the usage of the database. Thus, they could be considered as a query log for the TPC-E benchmark [19]. We parsed the pseudo-codes of transactions and recorded the number of times a join between a pair of attributes $r.A$ and $r'.A'$ is performed. Let us denote this number by $n(r.A, r'.A')$. The importance of the connection between $r.A$ and $r'.A'$ is calculated as $\frac{n(R.A,R'.A')}{N_{total}}$, where N_{total} represents the total number of joins in all the pseudo-codes of all the transactions. Given an MJNS, its average edge importance score is calculated as its *gold standard* importance score. Given a set of MJNSs, their gold standard importance scores are used to rank the MJNSs to generate a *gold stardard ranking*.

Given a query, we use our MJNS generator to produce all the MJNSs for the query and then use each of the ranking methods (listed in Table 2) to rank the MJNSs. The ranked list from a method is then compared with the gold standard ranking of these MJNSs. To see how close a ranked list produced by a ranking method is to the gold standard ranking, we employ two measures used in the IR community [2]. The first one is a statistical measure for comparing two ranked lists, called Kendall's τ coefficeint [12]. It returns a value between +1 and -1, measuring the correlation between the two lists. If two lists are identical, it returns +1. If they are reversely ordered, it returns -1. Generally, a positive value means that two lists are related, and a negative value means that they are reversely related. The statistical tests were performed using SPSS 15.0 for Windows. The second measure is the size of the overlap between the top-k items

	KE	DISC-I	MI	IF	IF_Ent
IF_KE	**0.018**	**0.004**	**0.003**	**0.025**	0.399
	0.000	**0.001**	**0.032**	**0.000**	0.903
	0.001	**0.000**	**0.037**	0.052	0.070
	0.000	**0.0001**	**0.012**	**0.016**	0.656
IF_Ent	0.153	**0.023**	**0.002**	**0.040**	
	0.021	**0.002**	0.071	**0.016**	
	0.193	**0.017**	1.000	**0.004**	
	0.0000	**0.000**	**0.001**	0.207	
IF	**0.001**	**0.000**	0.096		
	0.000	**0.000**	0.541		
	0.000	**0.000**	**0.002**		
	0.000	**0.000**	**0.002**		
MI	**0.000**	**0.000**			
	0.000	**0.000**			
	0.081	**0.012**			
	0.462	**0.009**			
DISC-I	0.443				
	0.021				
	0.096				
	0.033				

top-5 overlap without penalization
top-10 overlap without penalization
top-5 overlap with penalization
top-10 overlap with penalization

Figure 9: p-values of t-tests on the top-5 and top-10 overlap results. MNJS maximum size is 7.

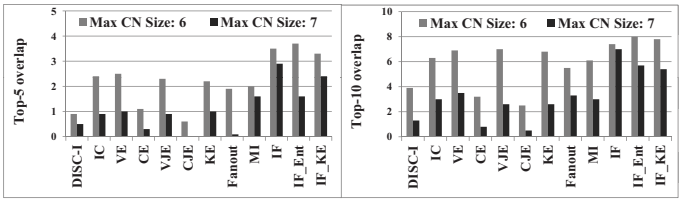

Figure 10: Results of the top-5 and top-10 with penalizing larger MJNSs.

in the two lists. We call this measure *top-k overlap*. We use five and ten as the values of k. In our evaluation, the performance score of a method is measured by the average score over the ten "queries" listed in Table 1.

4.3 Results of Ranking Methods without Size Penalization

We evaluated a total of twelve MJNS ranking methods. They are described in Table 2. We present the results of these ranking methods without penalizing larger MJNSs. The results are not compared with the IR based ranking methods (such as DISCOVER II [8]) since those methods are suitable for ranking the final answers but not the set of MJNSs.

The results of the twelve methods in terms of Kendall's τ rank correlation coefficient are presented in Fig. 7. The results suggest that for the maximum MJNS size of six, the ranking closest to the gold standard is achieved by the VE method. The best results for the maximum MJNS size of seven are produced by IF, KE and IF_Ent. However, the

Table 2: List of methods for ranking MJNSs. Our proposed methods are shown in bold face.

Method	Description
DISC-I	Size of MJNSs [9] (i.e. DISCOVER I)
IC	Node's importance, information content, Equation 2
VE	Node's importance, variable entropy [18]
CE	Node's importance, constant entropy [18]
VJE	Node's importance, variable joinable entropy [18]
CJE	Node's importance, constant joinable entropy [18]
KE	Node's importance, key entropy, Section 3.1
Fanout	Edge's importance, Definition 6 in [18]
MI	Edge's importance, mutual information [19]
IF	Edge's importance, instantiation fraction, Equation 7
IF_Ent	Edge's importance, IF & entropy, Equation 8
IF_KE	Hybrid method, IF & KE, Equation 11

correlation between VE and gold standard is not high when the maximum size of the MJNS is set to seven. This is not the case for IF and IF_Ent when the maximum size of the MJNS is set to six. On the other hand, the correlation between KE and the gold standard is high for both of the maximum MJNS sizes. It achieves the best average and most stable result in terms of Kendall's τ coefficient.

We also observe that ranking MJNSs by their size does not work well (indicated by the result for *DISC-I*). Its Kendall's τ coefficient is close to zero when the maximum MJNS size is 6, and it is negative when the size is 7. Therefore, ranking the results solely based on the number of nodes in the MJNS does not produce satisfactory results. Since only the size has been used to rank joining networks in previous methods, our proposed ranking methods outperform previous ones.

The results also show that some measures for node/edge importance (such as CE and $Fanout$) produce poor results as well. The measures that we present in this paper are much better than those measures.

The results of the top-5 and top-10 evaluations are presented in Fig. 8. By increasing the maximum size of the MJNS from 6 to 7, the top-k overlap with the gold standard decreases. This result is expected since by increasing the maximum size, the number of generated MJNSs increases. As the value of k is a constant (i.e. it is set to 5 or 10) and there are more items (MJNSs) in the list, the chance for overlap in top-k lists decreases.

Generally, the methods that rank MJNSs based on the edge importance work better than the ones based on the node importance. The difference between the two types of methods becomes greater when the maximum MJNS size is 7. In order to see whether different methods are significantly different from each other, we run the t-test. The p-values of the t-tests are presented in Fig. 9 for the maximum MJNS size of 7. The results for the maximum size of 6 is not presented due to space limits. But, they follow the same trend. Also, due to space limits, the t-test results for the node-based ranking methods are not presented.

Among the edge based methods, IF is the best in most of the cases. MI is the second best in general according to Fig. 8. The t-test results show that their performances are not significantly different (with p-values of 0.096 and 0.541) without penalizing larger MJNSs. The other edge based method IF_Ent performs the best when the maximum MJNS size is 6, but not well when the maximum size is 7. Similar observation is found for the hybrid method IF_KE. Thus, these two methods are not stable compared to IF and MI.

Looking at the results for the node based methods in Fig. 8, we observed that the results of VE and KE are very similar. Thus, using KE is better than using VE, since it is faster to evaluate, as discussed in Section 3.1.

4.4 The Effect of Penalizing Larger MJNSs

The results of the top-5 and top-10 evaluations with penalizing larger MJNSs are presented in Fig. 10 and t-test results are shown in Fig. 9. Note that the penalization technique (Equation 12) is applied to the computation of our gold standard as well.

Comparing the results in Fig. 8 and Fig. 10, we observe that the MI method is significantly negatively affected by the use of the size penalization technique, while the performance of other methods remain pretty much the same.

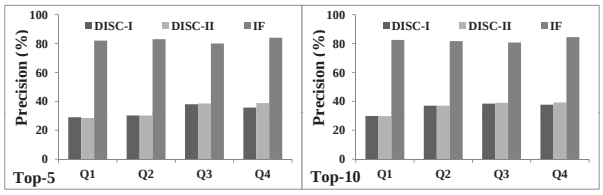

Figure 11: Top-5 and Top-10 precision of answers.

This set of results also suggests that the edge importance based methods are better than the node based methods, and that the edge based method IF is a stable method with the best overall performance. Again, the edge based IF_Ent method and hybrid method IF_KE have the best or close to best performance for the maximum MJNS size of 6, but their performance descreases significantly for the maximum MJNS size of 7. In addition, both Fig. 8 and Fig. 10 suggest that the size based method (*DISC-I*) has the worst performance. The t-test results show that it is significantly worse than other methods.

4.5 Relevance Evaluation of Final Answers by a User Study

To see how effective ranking of MJNSs impacts the *final answers* of the keyword search, we compare the top-k final answers from our keyword search method that uses the IF method for ranking MJNSs with the final answers produced by the method that ranks based on the number of joins (Discover I [9]), and the one based on the IR techniques (Discover II [8]) in terms of how relevant their answers are to the query. A common metric of relevance used in information retrieval is top-k precision, defined as the percentage of the answers in the top-k answers that are relevant to the query. To evaluate the top-k precision of the methods, we conduct a user study. We use four sets of keyword related to the first, second, fifth and sixth queries in Table 1 to evaluate the search results by human user. For example, the first query is "Jacob Insurance" in which "Jacob" is associated with the *customer* table and "Insurance" is associated with the *company* table (i.e., their roles are *customer* and *company*, respectively). In the experiment, top-5 and top-10 answers are produced for each query by each search method.

We ask eight graduate students in computer science, information technology, and mathematics to judge the relevancy of the answers. A user assigns a score between 0 and 1 to each final answer, where 1 means completely relevant and 0 means completely irrelevant to the query. This score may vary among the users. Thus, the average of the relevancy scores from the 8 users is used as the final relevancy score for an answer. The top-k precision is computed as the average relevancy score of the top-k answers.

The top-5 and top-10 precisions for each query are presented in Fig. 11. Clearly, the IF method which ranks the answers based on the edge strength between the associated entities achieves much better precisions than DISC-I and DISC-II in all the queries for both k values. The reason for DISC-I and DISC-II to have a lower precision is that in most of the cases, the tuples of the final answers are connected together by the dimension tables (e.g. *status type*) and fact tables are not involved. Thus, most of the users find the answers not so meaningful and assign them lower scores.

4.6 The Importance of Role Selection

We designed an experiment, in order to show the impor-

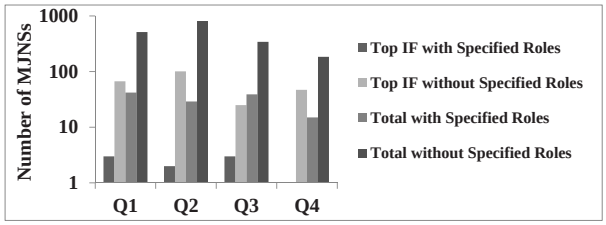

Figure 12: Number of MJNSs with/without specified roles.

tance of role selection in the search procedure. We show that role selection cuts significantly the search space in a database with a large and complex schema by removing irrelevant answers from the final list. It also shows that ranking based on only the importance of nodes/edges in the database schema is not enough to prune irrelevant answers. The four sets of keywords (used in user study in Section 4.5) are taken for this experiment. For each query, we calculate the number of MJNSs with the highest IF score with and without specified roles. For example, for the first query (i.e., "Jacob Insurance"), the highest IF score is 1.0. Therefore, with specified roles, we show the number of MJNSs with the score of 1.0 when the roles are set to *customer* and *company* for the two keywords. Without specified roles, we show the number of MJNSs which covers query "Jacob Insurance" and have the IF score of 1.0. The results are shown in Fig. 12. The total number of MJNSs is also shown in the figure in order to provide intuition about the size of the search space. Without specifying roles, we have large number of MJNSs—note that the scale in Fig. 12 is logarithmic—with the same highest score. For example, for the first query, there are 67 MJNSs ranked at the top without specified roles. However, only three of these MJNSs are relevant assuming the user is interested to see the relations between a *customer* and a *company*. Thus, without specified roles, the top ranking final answers could contain many irrelevant answers.

5. RELATED WORK

Existing approaches to keyword search over relational databases fall into two classes. The first class converts the database into a graph, on which the search is then performed [11]. The foreign key connections correspond to the edges of the graph. One of the challenges of this approach is fitting the graph into main memory [11]. Although some indexing techniques are proposed in the literature [7], converting a relational database with millions of records into a graph is memory consuming. In addition, how to find meaningful sub-graphs as query answers is still an open issue [11].

The second class of approaches considers the relational schema as a graph. The search is directly performed over the relational databases by generating and executing SQL queries on the RDBMS [1, 9, 8, 15]. Therefore, it heavily relies on the database schema and query processing techniques in the RDBMS. The methods for ranking the query answers are divided into two categories. For the first, the final answers are simply ranked based on the number of joins [1, 9]. This follows from the intuition that the smaller the number of joins, the easier to interpret the results. For the second the final answers are ranked based on the IR score of the tuples that contain the input keywords [8, 15]. We propose a series of methods for ranking the final answers first based on the importance of the nodes/edges of their associated join trees, and then based on an IR score as a secondary ranking measure. This approach finds more meaningful and suitable results for the users.

Below we first discuss two recent methods on finding meaningful answers for keyword search over relational databases, pointing out their differences from our work. Then we consider the work for ranking the components of a database schema.

5.1 Finding Meaningful Answers

The authors of [3] propose a framework for keyword search in relational databases. For building the index, traditional keyword search methods require access to the actual data stored in the RDBMS. In contrast, the method proposed in [3] uses intensional knowledge. Therefore, it can be used in applications in which building and maintaining specialized indexes are not feasible. Such systems only allow access to the data through predefined queries, wrappers or web forms. For interpreting the role of each keyword in the query, the authors extend the Hungarian algorithm [5] for finding the tuples that are most likely related to the meaning of the keyword. Our work can be considered as a subsequent step in which we rank the set of interconnected tuple sets for given pairs of keyword-entity.

The goal of the SODA (Search Over DAta warehouse) system, just as for us, is to enable end users to explore large data warehouses with complex schemas by applying the keyword search approach [4]. SODA is based on the idea of applying a graph pattern matching algorithm to generate SQL queries based on the given keywords. The focus of the system is to disambiguate the meaning of words using the joins and inheritance relationships among the matching tables. Our system differs from [4] in that we do not use metadata or other extra information. Our ranking model uses the schema and its instance to find the most meaningful relationships between the roles of query keywords. In addition, our focus is to rank the join trees rather than the roles of the keywords.

5.2 Ranking the Components of the Database Schema

A database schema is composed of a set of tables and connections (i.e., foreign keys) between them. Thus, the schema can be considered as a graph: tables are vertices and the foreign keys are edges. There are different methods for ranking tables and for raking connections. The two works [10, 18] study the problem of measuring table importance in a database. Both define the importance of a table as the steady state probability of the table in a random walk over the database graph G. The purpose of the work in [10] is to generate a form-based database query interface, and in [18], it is to summarize a relational schema with respect to the given database instance.

The method in [10] assumes that the importance of a table is proportional to the number of tuples, the number of attributes that it contains, and the number of connections it has to other tables. Based on this assumption, a probability matrix for the random walk is built. As discussed in [18], this is a reasonable assumption for an XML schema on which the method was evaluated, but it may fail to produce good results for relational databases, especially on data

warehouses that have dimension tables (e.g., *address* and *zip code* in TPC-E). Dimension tables usually have many connections to other tables. However, such connections may not indicate interesting relationships among the tuples they connect. For example, the first tree in Fig. 2 connects customer and company through the *status type* dimensional table. As discussed before, this is not an interesting relation between the keywords in the given query. The way the probability matrix is defined in [10] results in dimension tables gaining more importance than fact tables, which is not desirable in the context of data warehouses. Therefore, we do not choose this method as our baseline due to its poor performance over relational databases.

In [18], four methods for measuring the importance of tables are presented and shown to outperform the method in [10] for *summarizing* relational databases. The authors assume that a table's importance is proportional to the sum of its attribute entropies. Then, a probability matrix composed of the tables of the database schema is built. The importance of a table is then again defined as the steady state probability of the table in a random walk over the database graph G. Each of the four methods follow a different approach for building the probability matrix. All of them are implemented as our baseline in Section 4. Our proposed key entropy transfer model (KE) for measuring the importance of tables in Section 3 follows the spirit of the measures of [18]. Entropy techniques are common tools used for a variety of database problems, such as for modeling data and constraint repair [6].

Two measures for finding the importance (strength) of the edges (foreign keys) in the database schema is recently introduced in [18] and [19]. [18] proposes a method that measures the similarity between two given tables. This method is called $Fanout$ and is based on the ratio of instantiated fraction to the matched average fanouts of the tables along each edge (See [18] for details.). The method in [19] is based on the mutual information between two tables. Both of these works have been applied for *summarizing* schema graphs and are implemented in Section 4 as baselines. One of our edge based ranking methods, the instantiated fraction, adapts the method in [18].

6. CONCLUSIONS & FUTURE WORK

Our goal has been to improve relevance scoring of answers based on their networks (join trees) in light of larger and more complex database schema. We propose a series of measures, and algorithms to compute them based on the importance of nodes and edges to capture the intended semantic of queries. Extensive experiments with respect to a gold standard and a user study on a large and complex TPC-E schema establish that the proposed methods are able to capture well the intended semantics behind queries. We further find that one of the proposed edge based ranking methods (IF) outperforms other methods. While our methods prove to be effective, there is room for further research and improvement. In this work we sought to demonstrate how effective deriving relevance of the nodes and edges of the database schema could be based on just the schema and data, by no means are we advocating that auxiliary information cannot improve it. We would like to explore the use of Linked Data[9] and WordNet[10].

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE'02*, 2002.

[2] J. Bar-Ilan, M. Mat-Hassan, and M. Levene. Methods for comparing rankings of search engine results. *Computer Networks*, 50:1448–1463, 2006.

[3] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis. Keyword search over relational databases: A metadata approach. In *Proc. of SIGMOD'11*, 2011.

[4] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger. SODA: Generating SQL for Business Users. In *Proc. of VLDB'12*, 2012.

[5] F. Bourgeois and J. C. Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of ACM*, 14:802–804, 1971.

[6] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *Proc. of ICDE'11*, 2011.

[7] B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. In *Proc. of VLDB'08*, 2008.

[8] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proc. of VLDB'03*, 2003.

[9] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB'02*, 2002.

[10] M. Jayapandian and H. V. Jagadish. Automated creation of a forms based database query interface. In *Proc. of VLDB'08*, 2008.

[11] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. In *Proc. of VLDB'11*, 2011.

[12] M. Kendall. A new measure of rank correlations. *Biometrika*, 30:91–93, 1983.

[13] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Explaining structured queries in natural language. In *Proc. of ICDE'10*, 2010.

[14] Z. Liu and Y. Chen. Processing keyword search on xml: a survey. *World Wide Web*, 14:671–707, 2011.

[15] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: Top-k keyword query in relational databases. In *Proc. of SIGMOD'07*, 2007.

[16] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge Univ. Press, 1995.

[17] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: The power of rdbms. In *Proc. of SIGMOD'09*, 2009.

[18] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing relational databases. In *Proc. of VLDB'09*, 2009.

[19] X. Yang, C. M. Procopiuc, and D. Srivastava. Summary graphs for relational database schemas. In *Proc. of VLDB'11*, 2011.

[9] http://linkeddata.org

[10] http://wordnet.princeton.edu