International Journal of Computer Applications (0975 – 8887)
Volume 69– No.3, May 2013

A Metadata Search Approach to Keyword Query in Relational Databases

Jarunee Saelee
Software Systems Engineering Laboratory, Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand 10520

Veera Boonjing
Software Systems Engineering Laboratory, Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand 10520

ABSTRACT
This paper proposes an effective approach to keyword query in relational databases. It uses a semantic graph model consisting of database metadata, database values, user terms, and their semantic connections. Keywords of a query determine all possible connected subgraphs of the semantic model. A query answer is a subgraph with the minimum connections. In addition, the approach proposes to rank result tuples of the answer subgraph using the IR-style ranking function. Our experiment results show that queries with metadata terms give more precise answers than queries without them.

General Terms
Database and Information Systems, Information Retrieval, Relational Databases, Keyword-based Search.

Keywords
Keyword Search, Metadata Search, Database Query, Keyword Query, Relational Database.

1. INTRODUCTION
Many services on the Web and advanced applications widely use relational databases as structured information storages. Accordingly, the need for information retrieving is increasing. Traditional relational database search systems require users to know database schemas and query language such as SQL. So keyword search systems as information retrieval systems [1] over relational databases have recently proposed. However, keyword search techniques on the Web cannot directly be applied to databases because data on the Internet and database are in different forms. In databases, the information is viewed as data tables and their relationships, and query results may be a single tuple or joining tuples. Accordingly, the challenge is how to apply keyword-based search to find sorted relevant results in databases.

Existing systems supporting keyword search in relational databases (e.g., [2]–[6]) limit type of keywords to database value terms. In fact, users may query with metadata terms (e.g., attribute name or relation name) or their preferred terms. Consider an instance of a movie database as shown in Figure 1, a query with keyword {Steven} obtains two relevant tuples *r12* and *r53*. A query with keywords {director, Steven}, which {director} is an attribute name, gives only tuple *r12*. Thus, a metadata is useful for giving precise answers. Next, consider a query with keywords {movie, Steven, direct}, keywords {movie} and {direct} are ignored by most of systems, even these metadata terms are meaningful for querying. This is because they are not database values. This means that metadata terms in a query are the semantics of the answer. In addition, users often query using their preferred terms that are not directly matched to any objects in a database. For example, users may refer to the "actor" object of database using a keyword {player}. Moreover, these systems generally assumed that the answer graphs are in horizontal line or instance-level. Hence, the answer graph is a joining tuple tree which consists of all attributes of each tuple.

Directors R_1

	DirectorId	DName
r11	d01	Ridley Scott
r12	d02	Steven Spielberg

Movies2Directors R_2

	DirectorId	MovieId
r21	d01	20001
r22	d02	20021

Movies R_3

	MovieId	Title	Year
r31	20001	Gladiator	2000
r32	20021	Minority Report	2002

Movies2Actors R_4

	MovieId	ActorId	Character
r41	20001	a01	Maximus
r42	20021	a02	Chief John Anderton

Actors R_5

	ActorId	AName
r51	a01	Russel Crowe
r52	a02	Tom Cruise
r53	a03	Steven Seagal

Fig 1: An example of movie database instances

For these reasons, this paper proposes an effective system that allows users to query a database using database value terms, metadata terms, and their preferred terms. To achieve the purpose, it employs a metadata search approach [7], which uses a semantic graph of underlying database to accommodate these terms and database semantics. The semantic graph consists of database metadata, database values, user terms, and their semantic connections. This graph is useful for dealing with relation-level, attribute-level, and value-level in vertical line. An answer to a query is defined as a smallest subgraph containing all query keywords as its nodes. Moreover, we adopt a state-of-the-art IR ranking function to rank result tuples obtained from the answer subgraph.

The rest of this paper is organized as follows. Section 2 briefly introduces related works. Section 3 describes how a relational database is modeled as a semantic graph. Section 4 presents the system architecture. We give preliminary experiment results in Section 5. Section 6 concludes the paper and outlines future works.

2. RELATED WORK

Keyword search systems over relational databases have been extensively proposed. The earlier survey in [8] and [9] overviewed systems such as BANKS [10], DBXplorer [3], DISCOVER [4], ObjectRank [2], and EASE [11] and briefly summarized the key techniques from several aspects.

DBXplorer, DISCOVER, and BANKS share a similar idea but differ from each other in their search algorithms and ranking functions. They return joining tuple trees as answers for a given keyword query. DBXplorer and Discover generate connected tuple trees through primary-key-foreign-key relationships that contain all query keywords called candidate networks (CN). Thein et al. [12] proposed candidate network generation algorithms for reducing the overhead that caused by raising the number of joining tuples for the size of minimal candidate network. BANKS represents all tuples in a database as tuple graphs and generates answer graphs by searching Steiner trees containing all query keywords. However, all of them just assume AND semantics for an answer whereas our approach supports both AND and OR semantics. Hristidis et al. [13] proposed the extension of DISCOVER that handles non-metadata queries with both AND and OR semantics. Kacholia et al. [14] presented the bidirectional strategy to improve backward expanding search in BANKS by allowing forward search strategy. However, it still works by identifying Steiner trees from a whole graph. Furthermore, Ding et al. [15] employed a dynamic programming to improve efficiency of identifying Steiner trees.

DataSpot [16] is a database search system using free-form queries similar to our approach. It represents database content in form of schema-less semi-structured graph called hyperbase. Nodes in hyperbase represent data objects (e.g., relations, tuples, and attributes) and edges represent associations between data objects. Query results are connected subgraphs of hyperbase containing all query keywords. Goldman et al. [17] proposed a simple query language with two sets of keywords in form of *find x near y*. Two sets of objects in a database are found and the result set is ranked based on distance between these two sets. A similar system is proposed by Yin et al. [18]. Their concept is to find *the target objects related to source objects* with AND and OR semantics. The system converts a database schema to a graph. At the query time, it extends shortest join paths to measure the strengths of their relationships. Mragyati [19] is the system to keyword searching and browsing on relational databases. The system maps query keywords to a database schema using metadata as four-level trees and translates answer trees to SQL. The ranking function can be based on user-specified criteria but the default ranking is based on the number of foreign-key constraints. It is similar to our work in supporting synonyms and metadata. However, the implementation does not handle queries with more than 2 solution paths. Dissimilar to the other approaches, Wheeldon et al. [6] proposed a system to keyword search over relational databases which indexed a relational database as virtual documents to querying and navigation. Their approach indexes textual content of each tuple as a web page and their foreign-key constraints are extracted to hyperlink between virtual web pages. This is similar to a text object in EKSO [20] but EKSO provided offline indexing time to significantly reduce query time computation. Given a keyword query, the system in [6] calculates a ranked set of virtual web pages with at least one keyword matched. Then it uses the best trial algorithm to expand a rank set of navigation paths. The relevant results are unnecessary to the SQL translation. However, it does not support numerical queries.

The original ranking of the query results is based on the size of the answer tuple trees [3, 4]. Given a query *Q*, the score assigned to a result tuple tree *T* is:

$$Score(T,Q) = \frac{1}{size(T)} \qquad (1)$$

where *size(T)* is the number of tuples in *T*. More recent approaches have been attentively proposed ranking methods. ObjectRank uses an authority-based ranking strategy to keyword search in relational databases. It returns a set of the individual tuple as an answer. The ranking function is based on link analysis and term frequencies of query keywords. Luo et al. [5] proposed a new IR style method to join-tuple tree ranking. Liu et al. [21] improves the ranking strategy in [13] by identifies four normalization factors, tuple tree size, document length, document frequency, and inter-document weight. Yanwei et al. [22] studies the problem of finding the top-k results in relational databases for a continual keyword query. A set of potential top-k results is computed by evaluating the range of the future relevance score for every query result and a light-weight state is created for each keyword query.

The IR-style relevance ranking function for an individual text-attribute has two sub-functions, *Score* and *Combine*, defined as below:

$$Score(a_i,Q) = \sum_{k \in Q \cap a_i} \frac{1+\ln(1+\ln(tf))}{(1-s)+s\frac{dl}{avdl}} \cdot \ln\frac{N}{df} \qquad (2)$$

where *Score(a_i, Q)* is the relevance score with respect to the keyword query *Q* determined by an IR engine for a single text attribute a_i which is viewed as a text document. *k* is a keyword in *Q*, *tf* is the frequency of *k* in a_i, *df* is the number of tuples in a_i's relation with keyword *k* in this attribute, *dl* is the size of a_i in characters, *avdl* is the average attribute-value size, *N* is the total number of tuples in a_i's relation, and *s* is a constant usually be 0.2. Let *A* be the set of all text attributes of an answer tuple tree *T*. The score assigned to *T* for query *Q* is calculated by aggregate among two functions as below:

$$Score(T,Q) = Combine\ (Score(A,Q),\ size(T)) \qquad (3)$$

$$= \frac{\sum_{a_i \in A} Score(a_i,\ Q)}{size(T)}$$

The most similar in objective to our approach are in [23] and [24]. In [23], it extends keyword search to metadata over relational databases but not also user-terms or synonyms. It designs a data model as tuple graphs that each tuple contains a set of an attribute-value pairs and a new metadata attribute. For example, given a tuple *r31* in Figure 1 (a), this tuple is represented in form of {(MoviedId: 20001), (Title: Gladiator), (Year: 2000), (N4: Movies)} where N4 is a new metadata attribute. Considerably, there is too redundant metadata information in these tuples.

Approaches to keyword query are summarized in Table 1.

Table 1. Representative keyword search systems

Approach	Data Model	Ranking	Top-k Processing
Proximity	Data graph	Distance	N/A
DataSpot	Data graph	Number of edges	N/A
DBXplorer	Schema graph	Number of joins	N/A
BANKS	Data graph	Edge weight, node weight	N/A
DISCOVER	Schema graph	Number of joins	N/A
IR-Style		TF-IDF	Sparse algorithm, (Global) Pipeline algorithm
Effectiveness		Normalization	N/A
ObjectRank	Data graph, Schema graph	Authority rate	Threshold algorithm
Top-k-min-cost	Data graph		GST-k (optimal at top-1)
SPARK	Schema graph	Number of joins, TF-IDF	Skyline sweeping algorithm, Block pipeline algorithm
EASE	Data graph	Structural compactness, TF-IDF	N/A

3. DATABASE SEMANTIC REPRESENTATION

In this section, a data model and related definitions used in a metadata search approach are briefly presented. A database is considered as the semantic model including metadata terms, database value terms, and user terms. Metadata and database value terms intuitively known as relation names, attribute names, and attribute values. User terms are abbreviations, words or phrases that users use to refer to objects in the model. A user term is defined as (class, object), where classes consist of relation, attribute, and value, and objects are instances of these classes.

Informally, the semantic model is viewed as a graph with nodes representing objects of three classes: the relation class, the attribute class, and the value class. Edges represent connections between corresponding objects: relation to relation, relation to its attribute, and attribute to its attribute value. An example of the semantic graph for a movie database is illustrated in Figure 2. The answer graphs should be connected semantic subgraphs containing query keywords. Because of these structures of semantic model, the answer graphs can be additional metadata graphs that can deal separately from instance-level.

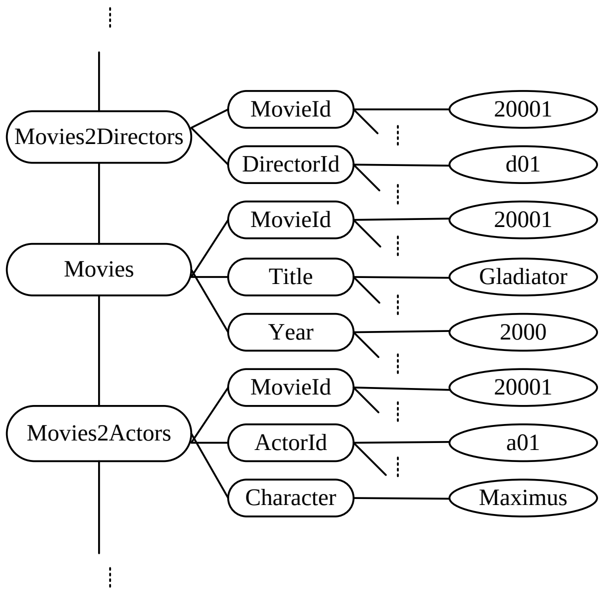


Fig 2: The semantic sub-graph

A formalized semantic graph and necessary definitions used to describe a query model in the next section are showed in the following.

Definition 1 Given a semantic graph *G <V, E>*, Node *V* is a set of metadata (*M*) and Database Values (*D*), and *E* is a set of their connections between relation-relation, relation-attribute and attribute-value.

Definition 2 Given a set of user-terms *U*, each user term $u \notin V$ but *u* is referred to corresponding node *V* in a semantic graph *G*.

Definition 3 A query keyword *K* is a set of $\{k_1, k_2, \dots, k_n\}$, where each *k* is a word or phrase of query *Q* matching some objects in *G* or *U*, and *n* is a number of query keywords.

Definition 4 A keyword node set of a query keyword k_i, denoted V_{k_i}, is a set of nodes in *G* that correspond to k_i.

Definition 5 Given *n* is a number of query keywords and $n \neq 0$, a query image *QI* is a set of keyword nodes v_i, where i = 1, 2, …, *n* and $v_i \in V_{k_i}$.

Definition 6 A basic path p_i of v_i is a minimum set of *V* in *G* that connect v_i to its relation node.

Definition 7 A feasible graph *FG* of *QI* is a sub-graph of *G* that is a minimal collection of basic paths p_i in *QI*, where i = 1, 2, …, *n*.

Definition 8 An answer graph is the shortest feasible graph that consists of <*V, E*> where *V* is a set of objects (relation nodes, attribute nodes, and value nodes) and *E* is a set of connection between them.

4. SYSTEM ARCHITECTURE

This section generally explains the architecture and strategies of the metadata search approach [7]. Figure 3 shows the architecture of our proposed system. It consists of two components, preprocessing and query processing. As described in the previous section, the database semantic representation is primarily explained in the preprocessing module. It automatically indexes resources except user terms, which are updated by administrators. In the second module, the keyword matching process finds database objects in the semantic model corresponding to user inputs. Answer graphs are generated in the query graph generator process. This process filters all possible query graphs to answer graphs just be the informative answers. These answer graphs are translated to SQL statements in the SQL generator process. Finally, the result ranking process evaluates the relevant results with ranking functions and then gives them back to the user. The detail of query processing is explained in the following.

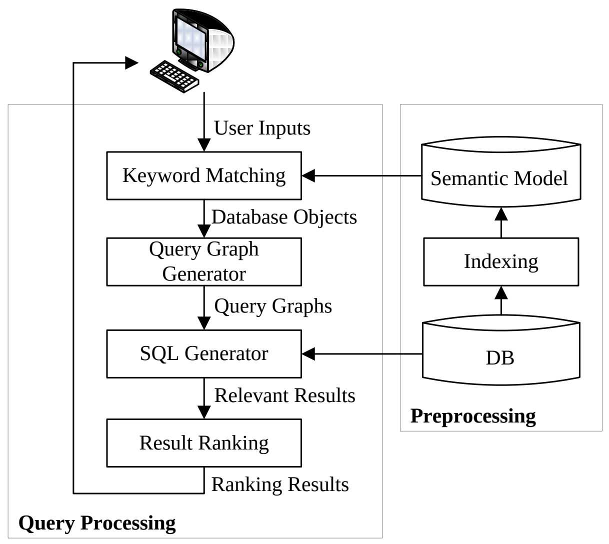


Fig 3: The architecture of a metadata search approach

4.1 Keyword Matching

The purpose of keyword matching is to find corresponding objects in semantic model for each query keyword. In other words, this process examines a query to find query keywords and their nodes in graph *G*. There are two cases in this process. The first is a direct matching that a keyword directly matches with objects in graph *G* (metadata terms or database value terms). For example from Figure 1, a query “director Steven” has two query keywords, {director} and {Steven}. Their resources are the {Directors} relation and the value terms from {Directors, Actors} relations respectively. The second is a synonym-based matching that keyword is a synonym or an abbreviation of objects in semantic graph. Given a keyword {player} for an example for this case, it cannot directly match with any objects but it is a synonym of {Actor}. Consequently, a query keyword {player} is changed to {actor} and its node is a relation {Actors}.

Keyword matching generally associates each query term with senses under the database semantic representation. Therefore, after keyword matching process, a set of resources for each keyword indicates kinds of elements that user wants.

4.2 Query Graph Generator

The purpose of query graph generator is to find semantic answer graphs from many candidate query graphs. To achieve this purpose, it begins with determining all possible query images containing one keyword node from each query keyword. The next is to find basic paths of each query image. The last, all feasible graphs are created. Figure 4 gives an algorithm to generate optimal answer graphs. Figure 5 shows an example of these steps with query keywords {film, Steven, direct}.

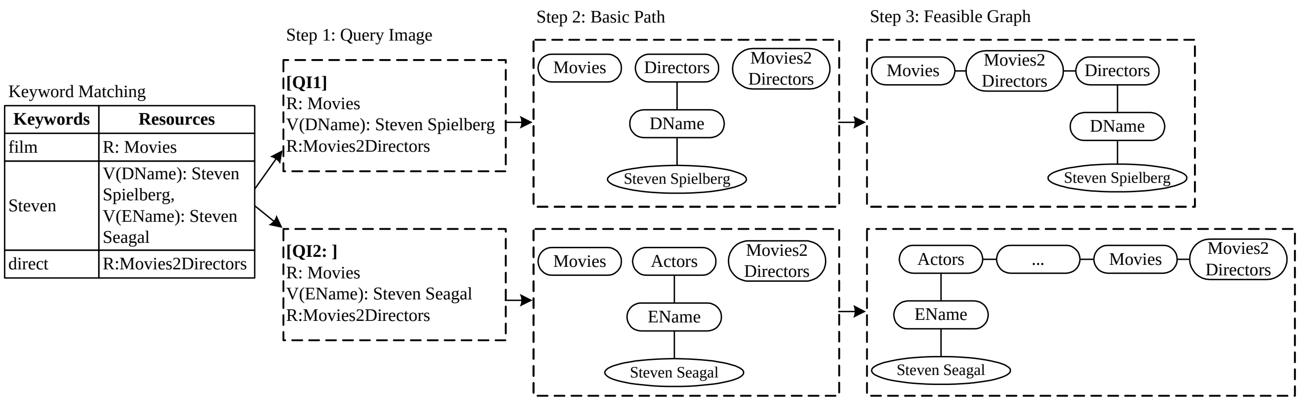
Input: Query *Q*, a semantic graph *G*
Output: a set of answer graphs

1. Determine a set of query keywords *K* and their keyword node sets
   $K = \{k_1, k_2, \ldots, k_n\}$, where *n* is a number of query keywords
   $V_{k_i} = \{ v_{k_{i1}}, v_{k_{i2}}, \ldots, v_{k_{ij}} \}$, where $i = 1, 2, \ldots, n$ and *j* is a number of corresponding nodes in *G* of k_i
2. Determine a set of query images *QI*
   $QI_m = \{v_{k_{1j}} \in V_{k_1}, v_{k_{2j}} \in V_{k_2}, \ldots, v_{k_{nj}} \in V_{k_n}\}$,
   where $m = |V_{k_1}||V_{k_2}|\ldots|V_{k_n}|$
3. Finding a set of basic paths in each *QI*
   Each v_{k_i} in *QI*, find a basic path where
   - If v_{k_i} is in a value class in *G*, then $p_{k_i} = (v^R_{k_i}, v^A_{k_i}, v_{k_i})$ where $v^R_{k_i}$, $v^A_{k_i}$ are relation and attribute nodes of v_i belonging to in *G* respectively
   - If v_{k_i} is in an attribute class in *G*, then $p_{k_i} = (v^R_{k_i}, v_{k_i})$ where $v^R_{k_i}$ is a relation node of v_{k_i} belonging to
   - If v_{k_i} is in a relation class in *G*, then $p_{k_i} = (v_{k_i})$
4. Find a feasible graph and determine a set of answer graphs
   Answer graphs are feasible graphs that have the lowest cost, $min_{i=1\text{-}m}(cost(FG_i))$ where *m* is a number of feasible graphs and $cost(FG) = |V| - 1$ where $|V|$ is a number of nodes in *FG*.

Fig 4: An algorithm for answer graph generation

In this process, branch and bound algorithm is used to find optimal solutions which have a minimum weight based on the number of nodes in a query graph. This is because the likelihood of an informative answer for keyword searching in relational databases based on the relational objects over the component facts. This means that, given two or more query graphs, the informative answer graph will always prefer the shortest graph. Because of our answer graph is a tree, its weight is $|V| -1$ where *V* is a set of nodes in it. For this reason, the optimal solution or the informative answer graph from an example in Figure 5 is a feasible graph from *QI1*.

In summary, a query graph associates a set of objects based on relationships in the database semantic representation and indicates what kinds of collections that users want. Therefore, after the query graph generator process, the best informative answer graphs are converted to SQL in the next process.


Keyword Matching

Keywords	Resources
film	R: Movies
Steven	V(DName): Steven Spielberg, V(EName): Steven Seagal
direct	R:Movies2Directors

Fig 5: An example of query graph generator

4.3 SQL Generator

To generate SQL from an answer graph, objects, relationships, and their joining conditions are determined as follows.

1) The *relation list* consists of all relation nodes in answer graph and indicates that what relations that tuple results originated from.

2) The *attribute list* is a selection list based on the nature of a semantic model. Figure 6 shows the natures of semantic model, relation-level (a), attribute-level (b), and value-level (c). If an answer graph consists of these natures, an attribute list contains all attributes that belong to a relation, a terminal-attribute, and an attribute of value node respectively. Additionally, if an answer graph is a lonely nature (c), an attribute list is all attributes of relation that its value node belongs to.

3) The *joining condition* extracts from foreign-key constraints of a database schema.

4) The *selection condition* is determined from value nodes in answer graph. If value node v_1 belongs to attribute node a_1, v_2 belongs to a_2, and v_n belongs to a_n then the selection condition is *AND* semantics ($a_1 = v_1$ *AND* $a_2 = v_2$ *AND* ... *AND* $a_n = v_n$). If v_1, v_2, …, v_n belong to the same attribute a, the selection condition is *OR* semantics ($a = v_1$ *OR* $a = v_2$ *OR* ... *OR* $a = v_n$).

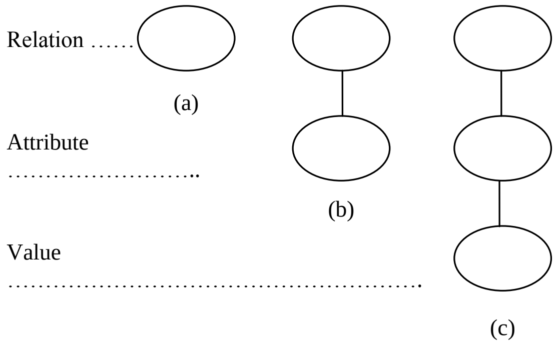


Fig 6: The nature of the semantic model

Consequently, we have SQL statement for each answer graph as follow:

```
SELECT    [attribute list]
FROM      [relation list]
WHERE     [selection condition] [AND]
          [join condition]
```

4.4 Result Ranking

After SQL generator process, multiple consequence result tuples are produced. This causes the question that which top-k tuples are the most likely answers for the end users. The IR-style ranking function as in [13], is applied to rank these tuples. Given a query *Q*, the ranking function is assigned to a tuple answer *T* is:

$$Score(T,Q) = \frac{\sum_{t \in T} Score\,(t,\,Q)}{size\,(T)} \tag{4}$$

where *Score(t, Q)* is the relevance score defined as below with respect to the keyword query Q and *size(T)* is weight of its answer graph.

$$Score(t,Q) = \sum_{k \in Q \cap T} \frac{1+\ln(1+\ln(tf))}{(1-s)+s\,\frac{dl}{avdl}} \cdot \ln\frac{N+1}{df} \tag{5}$$

where *t* is a result tuple that relevant to query *Q*, *tf* is the frequency of a keyword *k* appears in tuple *t*, *df* is the number of tuples that *k* appears, *dl* is the size of *t* in characters, *avdl* is the average length of *T*, *N* is the number of *T*, and *s* is a constant usually be 0.2.

5. EXPERIMENTS

To evaluate the search effectiveness of our approach, we use the Internet Movie Database (IMDB)[1] as the datasets in our experiments. We converted a subset of original files into relational tables as showed in Table 2 and used MySQL v5.0.24a with its default configuration and JDBC connections. All experiments were run on PC with a 1.66GHz CPU and 1G

[1] http://www.imdb.com/interfaces

RAM. The database server and the client were run on the same PC.

Table 2. Internet Movie Dataset Statistics

Relation	#Tuples	Relation	#Tuples
movies	7,485	Actors	10,025
directors	4,296	Editors	2,572
producers	8,556	Writers	7,130
genres	10,030	language	8,054
prodcompanies	8,340	releasedates	17,480
movies2actors	15,475	movies2editors	7,956
movies2directors	8,165	movies2producer	13,768
movies2writers	11,680		
Total number of tuples			**141,012**

In preliminary experiments, the influence of the system does with metadata terms additionally is measured by running ten queries with at least one metadata term in two systems, a metadata search and non-metadata search. Table 3 gives the total number of query results from: 1) all answer graphs of a metadata search (w/m), 2) the best answer graph of a metadata search (w/q), and 3) all answer graphs from non-metadata search (w/o). A metadata search approach gives the number of results much more than non-metadata search because metadata terms alternatively cause the semantic answer graphs and various tuples from each graph.

Table 3. The Total Number of Query Results

Query	#w/m	#w/q	#w/o	Query	#w/m	#w/q	#w/o
1	881	50	220	6	216	215	143
2	157	13	89	7	334	56	118
3	37	24	3	8	696	33	49
4	1745	107	220	9	169	77	106
5	2009	138	239	10	3826	1688	2288

Table 4. The Number of Top-k Results

Query	Top-10		Top-20		Top-30		Top-40		Top-50	
	#w/m	#w/o	#w/m	#w/o	#w/m	#w/o	#w/m	#w/o	#w/m	#w/o
1	10	0	20	2	30	7	40	10	50	17
2	10	1	13	2	13	4	13	6	13	11
3	10	0	20	0	24	0	24	0	24	0
4	10	0	20	0	30	0	40	0	50	0
5	10	0	20	0	30	0	40	0	50	0
6	10	0	20	0	30	0	40	0	50	0
7	10	0	20	0	30	0	40	0	50	0
8	10	0	20	0	30	0	33	0	33	0
9	10	0	20	0	30	0	40	0	50	0
10	10	0	20	0	30	0	40	0	50	0

Moreover, how a metadata search approach has the influence on top-k results is showed in Table 4. The numbers of top-k results from each query are compared between two systems. Top-k columns of Table 4 (where *k* = 10, 20, 30, 40, and 50) show a number of top-k results obtained from metadata search and non-metadata search. It shows that most of all top-k results derive from a metadata search approach, and a little to zero of non-metadata search results also appear in top-k. This is because metadata terms change a kind of result collections that users want. For example in Figure 5, a query with keywords {film, Steven, direct} is considered and means that "What movies did Steven direct?" by a metadata search, but just one value term {Steven} is considered and not distinguished by non-metadata search.

To evaluate answer accuracy, top-k precision, the ratio of the number of answers deemed to be relevant in the first *k* results with the highest scores of a method to *k*, is employed. Answer relevance is judged by discussion of researchers in our software systems engineering laboratory group. Figure 7 illustrates the average top-50 precision on various queries and the average results of the top-k precision with different values of *k* are shown in Figure 8. As expected, a metadata search approach achieves much higher precision than non-metadata search. As discussed previously, this is because metadata terms change a kind of result collections that users want.

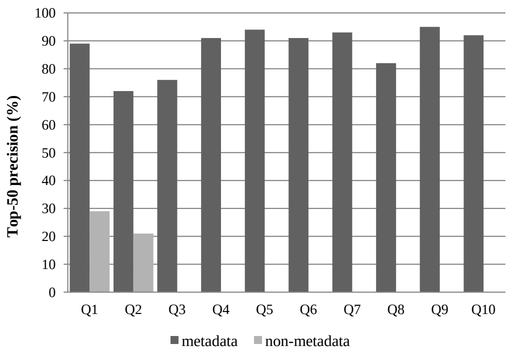


Fig 7: Top-50 precision on various queries

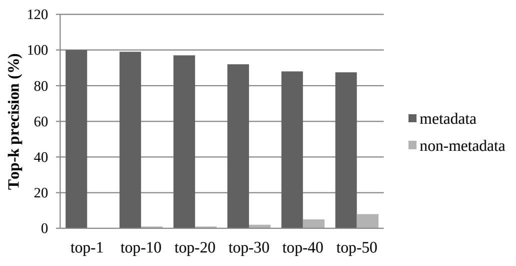


Fig 8: top-k precision with different values of *k*

Furthermore, to analyze the best semantic answer graph, sets of query results are considered as shown in Figure 9. All top-k results were found in R(w/q) first. This means that the best answer graph gives the optimal answers and corresponds to human judgment. All answer graphs of each query were shown to 29 user judges. The judges had to decide that the best answer graphs were relevant collections which they want.

Therefore, the best answer graph for a query would give precise answers.

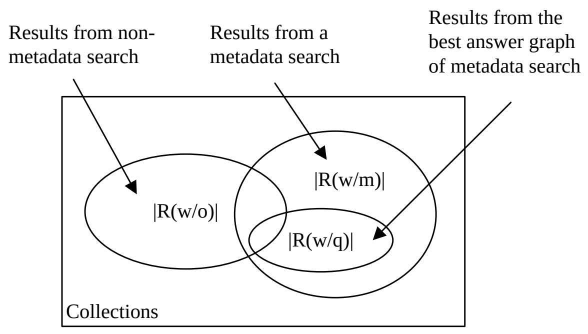

Results from non-metadata search
Results from a metadata search
Results from the best answer graph of metadata search
|R(w/o)|
|R(w/m)|
|R(w/q)|
Collections

Fig 9: The sets of query results

6. CONCLUSIONS

In this paper, a metadata search approach to ranked keyword search in relational databases is proposed. A semantic graph as a data model and strategies is used to find the optimal solutions. Additionally, the IR-style ranking function is applied to rank result tuples from answer graphs. The experiments confirmed that metadata in a semantic representation are useful for giving precise answers in both attribute-level and relation-level. Moreover, user terms can solve ambiguity problem of querying. Answer graphs are optimal solutions and suitable for mapping to corresponding SQL statements.

7. ACKNOWLEDGMENTS

This research was supported by the grant of the National Centre of Excellence in Mathematics, PERDO, Bangkok, Thailand. The authors also would like to thank Commission on Higher Education, Thailand. This support is very gratefully acknowledged.

8. REFERENCES

[1] B. Yates, and R. Neto, *Modern Information Retrieval*, ACM Press Series/Addison Wesley, New York, 1999.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou, “ObjectRank: Authority-Based Keyword Search in Databases”, *In VLDB*, 2004, pp. 564-575.

[3] S. Agrawal, S. Chaudhuri, and G. Das, “DBXplorer: A System for Keyword-Based Search over Relational Databases”, *In ICDE*, 2002, pp. 5-16.

[4] V. Hristidis and Y. Papakonstantinou, “Discover: Keyword Search in Relational Databases”, *In VLDB*, 2002, pp. 670-681.

[5] Y. Luo, C. Yu, W. Wang, and X. Zhou, “SPARK: Top-k Keyword Query in Relational Databases”, *In SIGMOD*, 2007, pp. 115-126.

[6] R. Wheeldon, M. Levene, and K. Keenoy, “DbSurfer: A Search and Navigation Tool for Relational Databases”, *LNCS*, Springer, Heidelberg, 2004, pp. 144-149.

[7] J. Saelee and V. Boonjing, “A Metadata Search Approach to Keyword Search in Relational Databases”, *In ICCIT*, 2008, pp. 571-576.

[8] S. Wang and K. Zhang, “Searching Databases with Keywords”, *J. Computer Science and Technology*, 2005, pp. 55-62.

[9] J. Park and S. G. Lee, “Keyword Search in Relational Databases”, *J. Knowledge and Information Systems*, vol. 26, 2011, pp. 175-193.

[10] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti, “Keyword Searching and Browsing in Databases using BANKS”, *In ICDE*, 2002, pp. 431-440.

[11] G. Li, B.C. Ooi, J. Feng, J. Wang, and L. Zhou. “EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data”, *SIGMOD*, Canada, 2008.

[12] M. M. Thein and M. M. S. Thwin, “Efficient Schema Based Keyword Search in Relational Databases”, *J. Computer Science, Engineering and Information Technology*, vol. 2, no. 6, Dec. 2012, pp. 13-32.

[13] V. Hristidis, L. Gravano, and Y. Papakonstantinou, “Efficient IR-Style Keyword Search Over Relational Databases”, *In VLDB*, 2003, pp. 850-861.

[14] V. Kacholia, S. Pandit, A. Chakrabarti, S. Sudarhan, R. Desai, and H. Karambelkar, “Biderectional Expansion for Keyword Search on Graph Databases”, *In VLDB*, 2005, pp. 505-516.

[15] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, “Finding Top-k-Min-Cost Connected trees in Databases”, *In ICDE*, 2007, pp. 836-845.

[16] S. Dar, G. Entin, S. Geva, and E. Palmon, “DTL's DataSpot: Database Exploration Using Plain Language”, *In VLDB*, 1998, pp. 645-649.

[17] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H.G. Molina, “Proximity Search in Databases”, *In VLDB*, 1998, pp. 26-37.

[18] X. Yin, J. Han, and J. Yang, “Searching for Related Objects in Relational Databases”, *In SSDBM*, 2005, pp. 227-236.

[19] N.L. Sarda, and A. Jain, “Mragyati: A System for Keyword-based Searching in Databases”, *TR CoRR cs.DB*, 2001.

[20] Q. Su, and J. Widom, “Indexing Relational Database Content Offline for Efficient Keyword-Based Search”, *In IDEAS*, 2005, pp. 297-306.

[21] F. Liu, C. Yu, W. Meng, and A. Chowdhury, “Effective Keyword Search in Relational Databases”, *In SIGMOD*, 2006, pp. 563-574.

[22] Y. Xu, Y. Ishikawa, and J. Guan, “Efficient Continual Top-k Keyword Search in Relational Databases”, *J. Information Processing*, vol. 20, no. 1, Jan. 2012, pp. 114-127.

[23] J. Gu, and H. Kitagawa, “Extending Keyword Search to Metadata on Relational Databases”, *In INGS*, 2008.

[24] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis, “Keyword Search over Relational Databases: A Metadata Approach”, *In SIGMOD*, 2011, pp. 565-576.