การค้นหาด้วยวิธีเมทดาตาเพื่อการสืบค้นฐานข้อมูลแบบจัดลำดับ

A METADATA SEARCH APPROACH TO RANKED KEYWORD DATABASE QUERY

จารุณี แซ่หลี

JARUNEE SAELEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาคุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

KMITL-2012-SC-D-002-042

การค้นหาด้วยวิธีเมทดาตาเพื่อการสืบค้นฐานข้อมูลแบบจัดลำดับ

A METADATA SEARCH APPROACH TO RANKED KEYWORD DATABASE QUERY

จารุณี แซ่หลี

JARUNEE SAELEE

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปรัชญาดุษฎีบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2556

KMITL-2012-SC-D-002-042

คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์	การค้นหาด้วยวิธีเมทดาตาเพื่อการสืบค้นฐานข้อมูลแบบจัดลำดับ A Metadata Search Approach to Ranked Keyword Database Query
นักศึกษา	นางสาวจารุณี แซ่หลี
รหัสประจำตัว	49062907
ปริญญา	ปรัชญาดุษฎีบัณฑิต
สาขาวิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.วีระ บุญจริง
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	-

คณะกรรมการสอบวิทยานิพนธ์		**ลายมือชื่อ**
ผศ.ดร.ศรัณย์	อินทโกสุม	
รศ.ดร.จีรพร	วีระพันธุ์	
ผศ.ดร.นวลสวาท	หิรัญสกลวงศ์	
ดร.เฉลิมศักดิ์	เลิศวงศ์เสถียร	
รศ.ดร.วีระ	บุญจริง	

วัน / เดือน/ ปี ที่สอบ 29 พฤษภาคม พ.ศ. 2555 เวลา 14.00 – 16.00 น.
สถานที่สอบ ณ ห้อง 216 ชั้น 2 อาคารจุฬาภรณวลัยลักษณ์ 1

คณะวิทยาศาสตร์รับรองแล้ว

(รองศาสตราจารย์ ดร.ดุษณี ธยะบริพัฒน์)
คณบดีคณะวิทยาศาสตร์

วันที่ 29 **เดือน** พฤษภาคม **พ.ศ.** 55

หัวข้อวิทยานิพนธ์ การค้นหาด้วยวิธีเมทดาตาเพื่อการสืบค้นฐานข้อมูลแบบจัดลำดับ
นักศึกษา นางสาวจารุณี แซ่หลี
รหัสประจำตัว 49062907
ปริญญา ปรัชญาดุษฎีบัณฑิต
สาขาวิชา วิทยาการคอมพิวเตอร์
พ.ศ. 2556
อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร. วีระ บุญจริง

บทคัดย่อ

เนื่องจากบริการต่างๆ บนอินเทอร์เน็ตและโปรแกรมคอมพิวเตอร์นิยมใช้ฐานข้อมูลในการจัดการข้อมูลต่างๆ ความต้องการในการสืบค้นข้อมูลจึงเพิ่มขึ้น ตัวอย่างการสืบค้นแบบง่ายๆ วิธีหนึ่งคือเทคนิคการค้นหาคำสำคัญบนอินเทอร์เน็ต ในขณะที่การค้นหาข้อมูลบนฐานข้อมูลผู้ใช้จำเป็นต้องมีความรู้ด้านโครงสร้างฐานข้อมูลและภาษาที่ใช้ในการสอบถาม การนำเทคนิคการค้นหาคำสำคัญบนอินเทอร์เน็ตมาใช้กับฐานข้อมูลก็ไม่สามารถทำได้โดยตรง เนื่องจากข้อมูลที่เก็บบนอินเทอร์เน็ตและฐานข้อมูลอยู่ในรูปแบบที่ต่างกัน การประยุกต์ใช้เทคนิคการค้นหาคำสำคัญเพื่อการสืบค้นข้อมูลบนฐานข้อมูลจึงต้องพิจารณาประเด็นสำคัญ ดังต่อไปนี้ การเตรียมข้อมูลหรือแบบจำลองข้อมูลที่นำมาใช้ในการค้นหา การค้นหาข้อมูลโดยที่ผลลัพธ์อาจเชื่อมต่อกันตามโครงสร้างฐานข้อมูล และการจัดลำดับผลลัพธ์ ซึ่งงานวิจัยนี้ได้นำเสนอวิธีการค้นหาคำสำคัญบนฐานข้อมูลโดยอาศัยแบบจำลองข้อมูลที่ประกอบด้วยข้อมูลฐานข้อมูล ค่าข้อมูล และคำพ้องความหมาย ซึ่งแบบจำลองที่นำเสนอสามารถนำมาพิจารณาเพื่อหากลุ่มคำสำคัญที่เกี่ยวข้องตามที่ผู้ใช้ต้องการได้ นอกจากนี้ขั้นตอนวิธีบรานช์แอนด์บาวด์ถูกนำมาใช้เพื่อพิจารณาหากลุ่มคำที่เหมาะสม และประยุกต์ใช้วิธีจัดเรียงการสืบค้นสารสนเทศในการจัดลำดับผลลัพธ์

คำสำคัญ: การค้นหาคำสำคัญ, การค้นหาโดยข้อมูลฐานข้อมูล, ฐานข้อมูลเชิงสัมพันธ์, การสืบค้นฐานข้อมูล

Thesis Title	A Metadata Search Approach to Ranked Keyword Database Query
Student	Miss Jarunee Saelee
Student ID.	49062907
Degree	Doctor of Philosophy
Programme	Computer Science
Year	2013
Thesis Advisor	Assoc.Prof.Dr.Veera Boonjing

ABSTRACT

Many services on the Web and applications widely use relational databases as structured information storages. So, the need for information retrieving is increasing. The simple way as search engines on the Web has popularized the keyword-based search paradigm, while searching in databases users need to know a database schema and a query language. However, keyword search techniques on the Web cannot directly be applied to databases because the data on the Internet and database are in different forms. Accordingly, to achieve the keyword query system over relational database, it brings some challenges that are: 1) how to prepare a data model used in a query model, 2) how to find the answers that may be a single tuple or joining tuples or joining sub-tuples via primary key-foreign key, and 3) how to rank the answers and return top-k results to users. Thus, this research aims to deal with these challenges by providing a natural capability for free-form keyword query in relational database. First of all, a database semantic representation is proposed as a semantic graph. It consists of metadata terms, value terms, and their preferred terms. This means that the approach can deal with relation-level, attribute-level, and value-level, and the answer graphs can be additional schema graphs. Second, this semantic graph is used to indicating what kinds of answer collections that users want, and branch-and-bound algorithm is used to achieve the optimal solutions based on the nature of the semantic model. Finally, existing IR ranking method is applied to evaluate scores between given query keywords and each tuple result.

Keywords: keyword search, metadata search, relational database, database query

ACKKNOWLEDGEMENTS

First and foremost I offer my sincerest gratitude to my thesis advisor, Assoc. Prof. Dr. Veera Boonjing, who has advised and supported me from the initial to the final level with his patience and knowledge. The author also gratefully acknowledges the helpful comments and suggestions of the committee.

I would like to thank Office of Academic Administration of King Mongkut's Institute of Technology Ladkrabang, Thai Higher Education Commission, Prince of Songkla University and National Centre of Excellence in Mathematics for supporting my Ph.D. study by the grant.

I would like thank Passamon Sittipinyo, Patthraporn Sornwiset, and Uraiwan Insoontorn for providing search query demo. I am also thankful my friends, Jirapond Tadrat, Weenawadee Moungon, and others for helping and encouraging me.

Finally, I heartily thankful my lovely family, my parents, my brother and sisters, for giving opportunities, encouraging me, pushing me, understanding me, and helping everything during the entire period of study.

Jarunee Saelee

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Statements

Many services on the Web and advanced applications widely use relational databases as structured information storages. Accordingly, the need for information retrieving is increasing. On the Web search engines, users simply search unstructured data by typing some keywords as a query and results are sorted relevant documents. In contrast, database query systems traditionally search structured data with structured query language or form-based interface. Moreover, the important feature of retrieving is that search results should be ranked based on some criteria. Traditional relational database search systems return all unranked tuples that satisfied the conditions in the query. Although 'ORDER-BY' operator is support ordered results, it do not order results in term of how well they match the query. So, if database users could query databases in the same way of Information Retrieval [15], database query would be simple without knowing database schema and database query languages. However, keyword search techniques on the Web cannot directly be applied to database because of the difference between data forms on the Internet and database. Accordingly, to achieve the keyword query system over relational database, it brings some challenges that are: 1) how to prepare data model used in a query model, 2) how to find the answers that may be a single tuple or joining tuples via primary key-foreign key, and 3) how to rank the answers and return top-k results to users.

Existing systems supporting keyword search in relational databases (e.g., [1, 6, 18, 21, 25]) 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 showed in Figure 1.1, 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. This means that 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

Figure 1.1 An example of movie database

Consequently, the contributions of this thesis are: 1) a database semantic representation is used to find the answers. It consists of the meaningfully connecting terms of database, 2) this semantic model of underlying database is used to indicate what kinds of answer collections that users want, and branch-and-bound algorithm is used to achieve the optimal solutions, and 3) a state-of-the-art IR ranking method is applied to evaluate scores between given query keywords and each tuple result.

1.2 Objectives

The objective of this thesis is to present an approach for free-form keyword query in a relational database by allowing users to query database using database value terms, metadata terms, and

their preferred terms. Moreover, a ranking function is proposed to rank results by applying IR ranking system.

1.3 Scope of the Study

The scopes of the study are:

1. The proposed system is based on keyword searching in a relational database.
2. A data model used in query process is based on a graph model.
3. A branch and bound algorithm is used to find the optimal answer graph.
4. A movie database from http://www.imdb.com/interfaces is used to be a data set in experiment.

1.4 Expected Benefits

The expected benefits of this thesis are:

1. A semantic-based model of a data for keyword-based search over relational database.
2. The simply process for keyword-based search over relational database.
3. A novel ranking function.

1.5 Process of the Study

1. Study and review literatures of keyword-based search in relational database and top-k ranking over these approaches.
2. Design an advantage data model used in the proposed approach.
3. Design an efficient and simple method to query a database by keywords.
4. Design a ranking function to rank results.
5. Implement a program for checking a proposed method.
6. Analyze experimental results.
7. Conclude the research.
8. Write a thesis document.

1.6 The Organization of the proposal

This chapter gives an overview of this thesis, including motivation, objectives, scope, expected benefits, and processes of the study. Chapter 2 briefly introduces related works. Chapter 3 describes a system design of the research approach, including how a relational database is modeled as a semantic graph and how to find the answers. The experiment results are showed in chapter 4. Chapter 5 concludes and recommends the thesis.

CHAPTER 2

LITERATURE REVIEWS

A keyword-based search or keyword query in relational databases is a convenient system, which allows users to searching in databases without having knowledge of database schema and query language but searching with keywords. A query result can be consists of one tuple from an entity or a combination of many tuples from many entities.

Keyword search systems over relational databases have been extensively proposed. The earlier survey [8, 20] overview systems such as BANKS-I [5], DBXplorer [18], DISCOVER [24], ObjectRank [1], and EASE [6] and briefly summarized the key techniques from several aspects.

DBXplorer, DISCOVER, and BANKS-I share a similar idea about database representation but differ from each other in their search algorithms and ranking functions. They return joining tuple trees as answers for a given keyword query. However, all of them just assume AND semantics for an answer whereas our approach supports both AND and OR semantics. Hristidis et al. [25] proposed the extension of DISCOVER that handles non-metadata queries with both AND and OR semantics. Thein et al. [13] proposed candidate network generation algorithms to generate a minimum number of joining tuples according to the maximum number of tuple set. Kacholia et al. [26] presented the bidirectional strategy to improve backward expanding search in BANKS-I by allowing forward search strategy. However, it still works by identifying Steiner trees from a whole graph. Furthermore, Ding et al. [3] employed a dynamic programming to improve efficiency of identifying Steiner trees.

DataSpot [21] 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. [27]. 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 shorthest 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 two solution paths. Dissimilarity to the other approaches, Wheeldon et al. [16] 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 [14] but EKSO provided offline indexing time to significantly reduce query time computation. Given a keyword query, the system in [16] 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 most similar in objective to our approach is in [10] and [23]. In [10], 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 1.1, 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 every tuples.

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. [29] proposed a new IR style method to join-tuple tree ranking. Yanwei et al. [28] 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. Moreover, Fakhraree et al. [22] proposed two factors, keyword proximity and keyword N-grams, to improve the search effectiveness in ranking function.

From the earlier approaches, this chapter is summarized about key techniques of keyword-based search system over relational databases. Table 2.1 illustrates the keyword search systems in relational databases in a common technique.

Table 2.1 Representative keyword search systems in relational database

Approach	Data model	Ranking measure	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 weights, 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

2.1 Database Representation

When a keyword-based search system receives a query, it needs to determine what this query means in term of related data. If the query consists of a single keyword, it can occur in many locations in database. If the query contains more than one keyword, it also needs to determine the relation between the keywords in underlying database. So a data model is necessary information to query processing. The keyword-based search systems design a data model as graph or virtual documents. A graph-based model consists of two types, schema graph and data graph, which can be directed or undirected. The other is virtual document, obtained from database tuples. Each virtual document can be consists of one tuple or many related tuples, such as [14, 16]. This thesis focuses on graph-based search system overviewed in the following.

2.1.1 Schema Graph-Based Model

Consider a relational database schema as a directed graph $G_S(V, E)$, called a *schema graph*, where V represents the set of entities $\{R_1, R_2, ..., R_n\}$ and E represents the set of edges between two entities. Given two entities R_i and R_j , there exists an edge in the schema graph from R_i to R_j, denoted $R_i \rightarrow R_j$, if the primary key defined on R_i is referenced by the foreign key defined on R_j. If there are different foreign keys defined on R_j referencing the primary key defined on R_i, there are multiple edges from R_i to R_j in G_S. In such a case, $R_i \xrightarrow{X} R_j$ is used, where X is the foreign key attribute names. A set of tuples on entity R_i is an instances of the entity, denoted $r(R_i)$. So, a relational database can be viewed as a data graph $G_D(V, E)$ on the schema graph G_S. DBXplorer and Discover are examples of this schema graph-based model.

$$R_1 \xrightarrow{DirectorId} R_2 \xleftarrow{MovieId} R_3 \xrightarrow{MovieId} R_4 \xleftarrow{ActorId} R_5$$

Figure 2.1 A schema graph of movie database

Figure 2.1 is a schema graph obtained from Figure 1.1 of movie database. R_1, R_2, R_3, R_4, and R_5 are entities *Directors*, *Movies2Directors*, *Movies*, *Movies2Actors*, and *Actors* respectively. *DirectorId*, *MovieId*, and *ActorId* are foreign key attribute names.

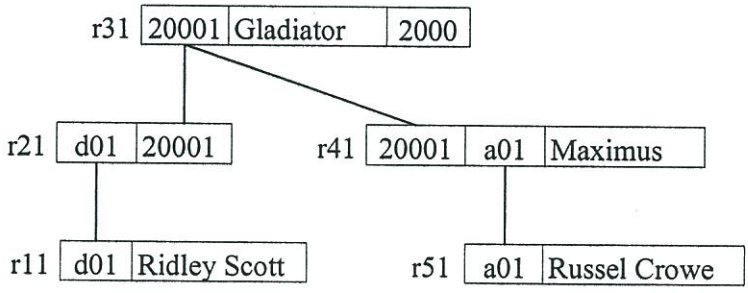


Figure 2.2 A fragment of a data graph-based model of movie database

2.1.2 Data Graph-Based Model

For a data graph-based model, a relational database is viewed as a data graph $G_D(V, E)$, where V represents a set of tuples, and E represents a set of edges between tuples. It can be directed or undirected graph. An edge exists between two tuples if there is a foreign key reference from one to the other. A tuple consists of attribute values. BANKS and EASE [6] are systems developed in this area. Figure 2.2 shows a fragment of a data graph achieved from Figure 1.1.

2.2 Query Processing of Keyword-Based Search System

Given a keyword query Q is a set of keywords with size l, denoted as $Q = \{k_1, k_2, \ldots, k_l\}$, almost all the existing approaches primarily return tuple trees as answers. A tuple tree is a joining tree of tuples and each tuple connects to the others via foreign key relationships. So an answer is a tuple tree which each keyword is found in at least one tuple node. However, the query processing of how to get the answer trees is based on a graph model mentioned in previous section.

2.2.1 Schema Graph-Based Keyword Searching

The two main steps in the schema graph-based approach are generating a set of SQL queries that can find all tuples from R that contain each keyword in a database, and finding all minimal total joining networks of tuples (MTJNT) [3, 25, 29].

Given an l-keyword query and a relational database with schema graph G_S, a joining network of tuples (JNT) is a connected tree of tuples where every two adjacent tuples, $r_{im} \in r(R_i)$ and $r_{jn} \in r(R_j)$ can be joined based on the foreign key relationship defined on relational schema R_i and R_j in G_S (either $R_i \longrightarrow R_j$ or $R_j \longrightarrow R_i$). An MTJNT is a joining network of tuples that satisfy the two conditions, total and minimal. By total, each query keyword must be contained in at least one tuple of the joining network. By minimal, a joining network of tuples is not total if any tuple is removed.

For example, consider a schema graph in Figure 2.3 (a) over five entities. Let a keyword query Q be $\{k_1, k_2, k_3\}$, assume that R_2 contains all three keywords, R_4 contains k_2, and R_5 contains k_3. There are four sub-graphs based on a schema shown on Figure 2.3 (b). The answers in term of a tuple tree or MTJNT are obtained from minimal total joining trees via SQL. Moreover, ranking issues are explained in the next section.

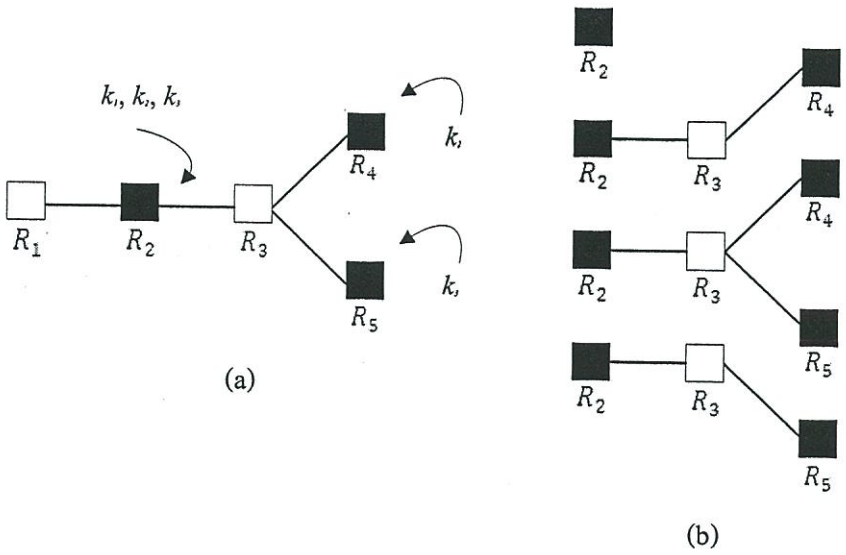


Figure 2.3 An example of schema graph (a) and minimal total joining trees (b)

2.2.2 Data Graph-Based Keyword Searching

In Section focuses on supporting keyword search based on a directed graph G_D. Unlike an undirected graph, the fact that a tuple v can reach to another tuple u in a directed graph does not necessarily mean that the tuple v is reachable from u. Such direction handling provides users with more information on how the tuples are interconnected.

BANKS-I [5] enumerates the answers as tuple trees using a backward search algorithm over a stiener-tree. Given a set of l keywords, it first find a set of nodes that contain keywords, S_i, for each keyword term k_i. This step can be accomplished efficiently using an inverted list index. Let $S = \bigcup_{i=1}^{l} S_i$. Then, the backward search algorithm concurrently runs $|S|$ copies of Dijkstra's single source shortest path algorithm, one for each keyword node v in S with node v as the source. The $|S|$ copies Dijkstra's algorithm run concurrently using iterators. All the Dijkstra's single source shortest path algorithms traverse graph G_D in reverse direction. When an iterator for keyword node v visits a node u, it finds a shortest path from u to the keyword node v. The idea of concurrent backward search is to find a common node from which there exists a shortest path to at least one node in each set S_i. Such paths will define a rooted directed tree with the common node as the root and the corresponding keyword nodes as the leaves.

Although BANKS-I can find an l-approximation of the optimal tuple trees as the answers. The non-first results returned by these algorithms cannot guarantee their quality and the

delay between consecutive results can be very large. In [2, 11], the authors show three algorithms to enumerate tuple tree in increasing weight order with polynomial delay.

The algorithm of BANKS-I can be directly applied to the distinct root semantics. It would explore an unnecessarily large number of nodes in the following two scenarios. First, the query contains a frequently occurring keyword. The algorithm would generate a large number of iterators if a keyword matches a large number of nodes. Second, an iterator reaches a node with large incoming edges. BANKS-II [26] proposes the strategy to overcome the drawbacks of BANKS-I. The main idea of bidirectional search is to start forward searches from potential roots. The main differences of bidirectional search from BANKS-I are as follows. First, all the single source shortest path iterators from the BANKS-I algorithm are merged into a single iterator, called the *incoming iterator*. Second, an *outgoing iterator* runs concurrently, which follows forwarding edges starting from all the nodes explored by the *incoming iterator*. Third, *spreading activation* is proposed to prioritize the search, which chooses *incoming iterator* or *outgoing iterator* to be called next. It also chooses the next node to be visited in the *incoming iterator* or *outgoing iterator*.

BLINKS [7] proposes a bi-level index to speed up BANKS-II, as no index except the keyword-node index. A naive index precomputes and indexes all the distances from the nodes to keywords, but this will incur very large index size, as the number of distinct keywords is in the order of the size of the data graph G_D. A bi-level index can be built by first partitioning graph, and then building intra-block index and block index. Two node-based partitioning methods are proposed to partition a graph into blocks, namely, BFS-Based Partitioning, and METIS-Based Partitioning. In a node-based partitioning of a graph, a node separator is called a *portal* node. A block consists of all nodes in a partition as well as all portals incident to the partition. For a block, a portal can be either "in-portal", "out-portal", or both. A portal is called in-portal if it has at least one incoming edge from another block and at least one outgoing edge in this block. And a portal is called out-portal if it has at least one outgoing edge to another block and at least one incoming edge from this block.

2.3 Ranking Model

The important thing of searching and data retrieving is the top-k ranking. This section explains the IR ranking system that used to apply to ranking in searching over relational database.

2.3.1 IR Ranking System

Generally, IR ranking system assigns a score for each document as an estimation of the document relevance to the given query. The widely used model to compute such a score is the vector space model [15]. Each text (both documents and queries) is represented as a vector of terms, each of which may be an individual keyword or a multi-word phrase. The vocabulary of terms makes up a term space. Each term occupies a dimension in the space. Each text (a document or a query) is represented as a vector on this term space, and each item in the vector of a text has a non-negative weight, which measures the importance of the corresponding term k in the text. Thus, a similarity value between a document vector D and a query vector Q can be computed as the ranking score.

$$Sim(Q, D) = \sum_{k \in Q,D} weight(k, Q) * weight(k, D) \tag{2.1}$$

$$weight(k, D) = \frac{ntf}{ndl} * idf \tag{2.2}$$

$$ntf = 1 + \ln(1 + \ln(tf)) \tag{2.3}$$

$$idf = \ln \frac{N}{df + 1} \tag{2.4}$$

$$ndl = (1 - s) + s * \frac{dl}{avgdl} \tag{2.5}$$

Equation 2.1 shows the inner product (dot product) function to compute the similarity. Weighting a term in a document (the *weight(k, D)* component in Formula 1) is the most critical problem in computing similarity values. Equation 2.2 shows the pivoted normalization weighting method [15], which is one of the most widely used weighting methods in IR. Note that, conceptually, we put the *idf* component into Equation 2.2 but not into Equation 2.1. The weight of a term in a document is determined by the following three factors.

1. Term Frequency (*tf* in Equation 2.3): the number of occurrences of a term in a document. Intuitively, the more a term occurs in a document, the higher the weight of the term should be. However, the same term may occur many times in a long document, and the importance of a term should not be linearly dependent on the raw *tf* when *tf* is rather

large. It has been accepted in IR that the raw *tf* should be dampened. Equation 2.3 applies the log function twice to normalize the raw *tf* to get *ntf*.

2. Document Frequency (*df* in Equation 2.4): the number of documents that a term occurs in a collection. By intuition, in a collection, the more documents a term appears in, the worse discriminator it is, and it should be assigned a smaller weight. Equation 2.4 shows the inverse document frequency (*idf*) weighting method to normalize *df*: dividing the total number of documents (*N* in Equation 2.4) by (*df*+1) and then applying the *log* function.
3. Document Length (*dl* in Equation 2.5): the length of a document in bytes or in number of terms contained in the document. Because longer documents contain more terms and higher term frequencies, longer documents tend to have higher inner product values for a given query. Equation 2.5 provides a normalization to reduce the term weights in long documents, where *avgdl* is the average document length in the collection, and *s* is a constant and is usually set to 0.2.

Weighting a term in a query (the *weight(k, Q)* component in Formula 1) is rather simple: a raw term frequency (*qtf*) is used in the query. Note that normalization on the above three factors has significantly improved search effectiveness in IR than the simple *tf*idf* weighting methods [15].

2.3.2 Keyword-Based Search Ranking in Relational Database

In keyword-based search over relational database, a good ranking model for answer graphs should satisfy the following consideration:

1. Confident answers: the answers containing facts with high extraction confidence should be ranked higher.
2. Informative answers: the informative answers should be ranked higher. For example from Figure 1.1, a query with "movie that Steven directed", the answer should be a list of movies that Steven is a director (it's not mean to an actor Steven).
3. Compact answers: structural information, e.g., the size of the answer-tuple tree is consideration. The idea is that the more of number of joining between entities the more difficult to understood.

Many approaches (e.g., [17, 19, 21, 24]) rank answers by the number of joins involved. However, there are some approaches (e.g., [3, 15, 29]) aim to designing effective ranking functions that capture both the textual information, e.g., IR-Styled ranking, and compact answer. There are two categories of ranking functions, namely, the attribute level ranking function and the tree level ranking function. Given an MTJNT T and a keyword query Q, the attribute level ranking function first assigns each text attribute for tuples in T and then combines them together to get the final score [3, 15]. Tree level ranking functions consider the whole MTJNT as a virtual document rather than each individual text attribute [29].

[3] and [5] incorporate a technique that assigns weights to tuples and edges between tuples. A combination of tuple weights and edge weights in a tuple tree is calculated to ranking.

Liu et al. [4] propose IR ranking methods based on TF-IDF. A similarity value between a given query Q and a tuple tree T needs to be computed to rank tuple trees. Let T be a tuple tree and $D = \{D_1, D_2, ..., D_m\}$ be all text column values in T. Each text column value D_i is defined as a document and T as a super-document. Then a similarity value between the query Q and the super-document T is computed as shown in Equation (2.6). The similarity is the dot product of the query vector and the super-document vector. The method of weighting a term in the query, *weight(k, Q)*, still uses the term's raw *qtf* (term frequency in the query). [4] focuses is on *weight(k, T)*,the weight of a term k in a super-document T.

$$Sim(Q,T) = \sum_{k \in Q,T} weight(k,Q) * weight(k,T) \tag{2.6}$$

$$weight(k,T) = \frac{ntf}{ndl} * idf \tag{2.7}$$

$$ntf\,(k_i, G) = 1 + \ln(1 + \ln(1 + tf\,(k_i, G))) \tag{2.8}$$

$$idf_{k_i} = \ln \frac{N+1}{N_{k_i}+1} \tag{2.9}$$

$$ndl = (1-s) + s * \frac{tl_G}{avg_{dl}} \tag{2.10}$$

where $tf(k_i, G)$ in Equation 8 denotes the term frequency of keyword k_i in the data graph G_D. N and N_{ki} in Equation 9 denote the number of tuple trees and the number of those tuple trees containing keyword k_i. In Equation 10, tl_G denotes the total number of terms in G_D and avg_{tl} is the average number of terms among all tuple trees. These parameters are consequently used to compute a ranking weight between a keyword and a tuple tree T.

CHAPTER 3

SYSTEM DESIGN

Keyword search over relational database is not full-text search on individual text fields, but it is done over the content of whole objects in the database in form of joined tuples. Moreover, there are many ways that tuples can be joined in a database. To enable keyword search in relational database, additional functionalities are required, data model and query processing. This chapter explains a keyword search system over relational database with a metadata search approach.

3.1 Database Semantic Representation

First of query processing, keyword search system identifies the location where query keywords appear in the relational database. To avoid the linear scanning of the whole content in the database, an indexing structure or a data model identifies the database schema and the instance that related to the query keywords, which is similar to an inverted file. This approach, a relational database is considered as a semantic model including metadata terms, database value terms, and user terms. Metadata and database value terms intuitively known as entity names, attribute names, and attribute values. User terms are abbreviations, words or phrases that users use to refer to objects in the model. Query results should be connected semantic sub-graphs 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.

A formalized semantic-based 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.

Informally, a semantic model is viewed as a graph with nodes representing instances of three classes, entity, attribute, and value. Edges represent connections between corresponding instances, entity to entity, entity to its attribute, and attribute to its value. An example of a semantic-based graph defined in Definition 1 for a movie database is illustrated in Figure 3.1.

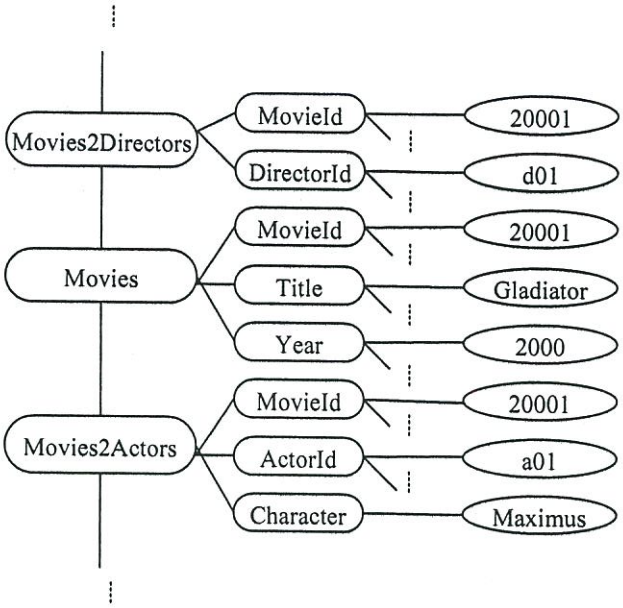


Figure 3.1 A semantic subgraph of movie database

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.

Table 3.1 Example of IMDB user-terms

User-term	Reference	Description
Film	R: Movies	A synonym of "Movies" relation
Cinema	R: Movies	Another synonym of "Movies" relation
direct	R: Movies2Directors	Word explains "Movies2Directors" relation, means who is a director of that movie
directed	R: Movies2Directors	Word also explain "Movies2Directors" relation
role	A: Character	A synonym of "Character" attribute
Director name	A: DName	Attribute "DName" can interpret to "director name"
US	V: United State Of America	An abbreviation of database value, "United State of America"
USA	V: United State Of America	A short form of "United State of America"
United State Of America	V: US	Full word of database value "US"
United State Of America	V: USA	Full word of "USA"

A user term in Definition 2 is referred to object or node in a semantic graph by defined as (class: object), where class consists of entity, attribute, and value, and object is instances of these classes. An example of IMDB user-terms in Definition 2 is showed in Table 3.1.

Definition 3. A query keyword K is a set of $\{k_1, k_2, ..., k_n\}$, 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.

A query keyword (Definition 3) is considered from user keyword query that matching with objects in a semantic model and a keyword node (Definition 4) is some objects in a semantic model corresponding to its keyword. For example, given a user query "What movies did Steven direct?", query keywords and their keyword nodes are showed in Table 3.2.

Table 3.2 Query keywords and keyword nodes for query "What films did Steven direct?"

Query Keyword	Keyword Node
film	R: Movies
Steven	V(DName): Steven Spielberg V(EName): Steven Seagal
direct	R: Movies2Directs

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.

Query image defined by Definition 5 is a keyword node set that each query keyword must matched with one of keyword node in its image. The number of query images for one query is the product of the amount of keyword nodes in each keyword. From Table 3.2, the amount of query images equals to 2. Examples of query images and basic paths (Definition 6) are showed in Figure 3.2 and 3.3 respectively.

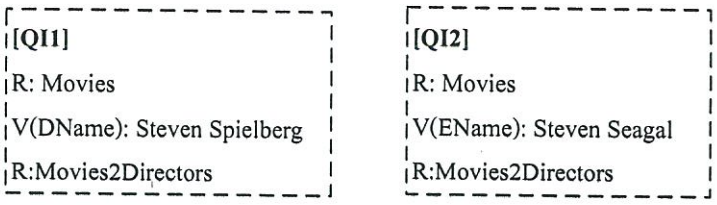


Figure 3.2 Example of query image for query keywords {film, Steven, direct}

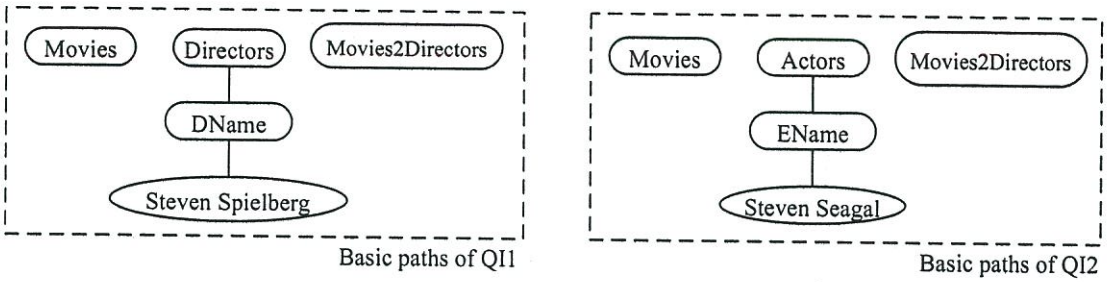


Figure 3.3 Example of basic path for query keywords {film, Steven, direct}

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 $i = 1, 2, \ldots, n$.

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.

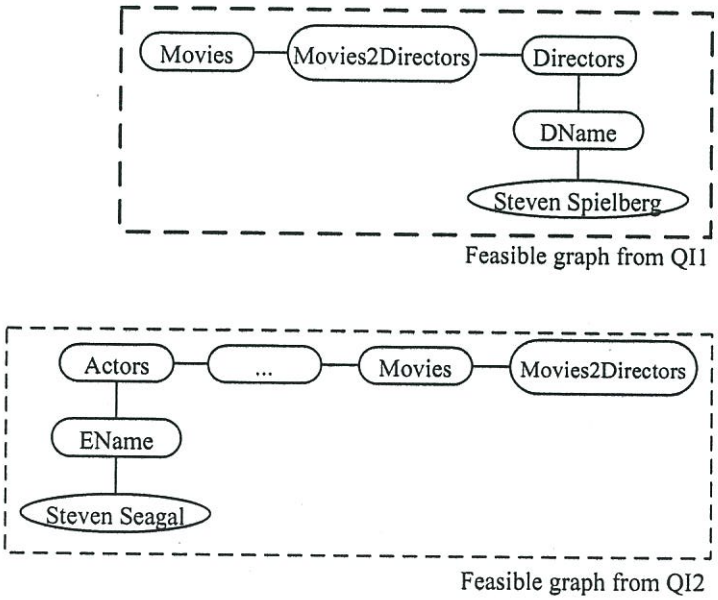


Figure 3.4 Example of feasible graph for query keywords {film, Steven, direct}

All subgraphs of a semantic graph that can connect basic paths of each query image are feasible graphs (Definition 7) and an answer graph (Definition 8) is a feasible graph with the minimum connections. Figure 3.4 shows feasible graphs derived from Figure 3.3 and answer graph is a feasible graph from *QI1*.

3.2 System Architecture

This section generally explains the architecture and strategies of the proposed approach based on database semantic representation. Figure 3.5 shows the architecture of the system. It consists of two components, preprocessing and query processing. A database semantic representation is explained for the preprocessing module in previous section. It automatically indexes resources except user terms, which updated by administrators. In the query processing, keyword matching process finds query keywords that corresponding to the instances in the semantic-based graph *G*. 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. A detail of query processing is presented in the following.

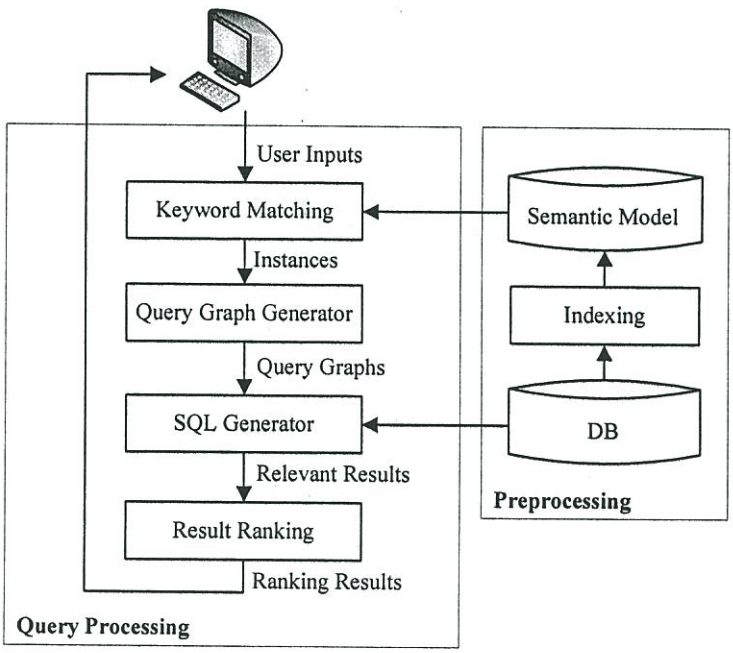


Figure 3.5 The architecture

3.2.1 Keyword Matching

The purpose of keyword matching is to find corresponding objects in semantic model for each query keyword. In other word, 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 1.1 in chapter 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}. Generally, keyword matching associates each query term with senses under the database semantic representation. Therefore, after keyword matching process, a set of resources for each keyword are indicating what kinds of elements that user wants.

3.2.2 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 3.6 gives an algorithm to generate optimal answer graphs and is explained in the following.

For a given query *Q*, the query processing is enabled through the following steps.

Step 1: Identify query keywords and their nodes in graph *G* (Definition 1). Therefore, after this step, a set of resources for each keyword are indicating what kinds of elements that users want.

Step 2: Determine all possible query images (Definition 5) as a minimal set of a collection of keyword nodes.

Step 3: Branch and bound algorithm is used to find optimal solutions with have a minimum weight based on the number of nodes in a query graph. Because of a query graph is a tree, its weight is $|V| - 1$ where V is a set of nodes in it.

Therefore, the best informative answer graphs which indicated what kinds of collections that users want will be determined first.

Step 4: Generate SQL statements from answer graphs.

Step 5: Rank obtained result tuples and returned to users.

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

   $K = \{k_1, k_2, ..., k_n\}$, where n is a number of query keywords

   $V_{k_i} = \{ v_{k_{i1}}, v_{k_{i2}}, ..., v_{k_{ij}} \}$, where $i = 1, 2, ..., n$ and j is a number of corresponding nodes in G of k_i

2. Determine a set of query images QI

   $QI_m = \{v_{k_{1j}} \in V_{k_1}, v_{k_{2j}} \in V_{k_2}, ..., v_{k_{nj}} \in V_{k_n}\}$,

   where $m = |V_{k_1}||V_{k_2}|...|V_{k_n}|$

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 $p_{k_i} = (v_{k_i}^R, v_{k_i}^A, v_{k_i})$ where $v_{k_i}^R$, $v_{k_i}^A$ 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 $p_{k_i} = (v_{k_i}^R, v_{k_i})$ where $v_{k_i}^R$ 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, $min_{i=1-m}(cost(FG_i))$ where m is a number of feasible graphs and $cost(FG) = |V| - 1$ where $|V|$ is a number of nodes in FG.

Figure 3.6 An algorithm for answer graph generation

Branch and bound algorithm in Figure 3.7 is used to find the optimal solutions from all candidate query graphs. We define the objective function as the compactness of 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. Thus, the lower bound $LB(g)$ is estimated by the number of nodes in a query image or a feasible graph. That is $LB(g)$ is the number of nodes in QI with its basic paths minus one where g is a query image and if g is a feasible graph, $LB(g)$ is the number of nodes in it minus one.

```
Input: possible query graphs
Output: the best answer graphs and the lowest score

01. initial configuration: node = {root}, lowest-cost = ∞
02. while node is not empty do
03.     b-node = nodes with the lowest cost //considering from node
04.     node = {}
05.     for each b-node do
06.         branch b-node with its possible graph nodes (g-node)
07.         for each g-node do
08.             cost(g-node) = LB(g-node)
09.             if cost(g-node) ≤ lowest-cost then
10.                 add g-node to node
11.                 if cost(g-node) < lowest-cost then
12.                     set new lowest-cost
13.                 end if
14.             end if
15.         end  for
16.     end for
17. end while
18. return node and lowest-cost
```

Figure 3.7 Branch and bound algorithm

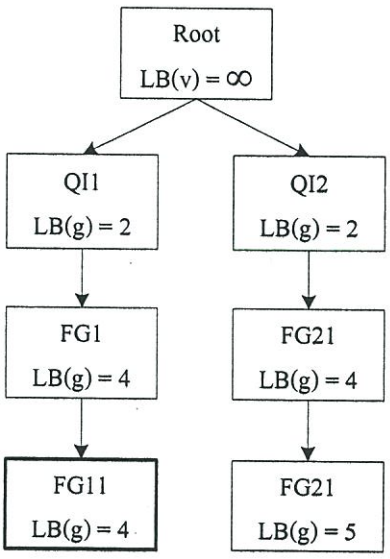


Figure 3.8 An example of search graph from branch and bound algorithm

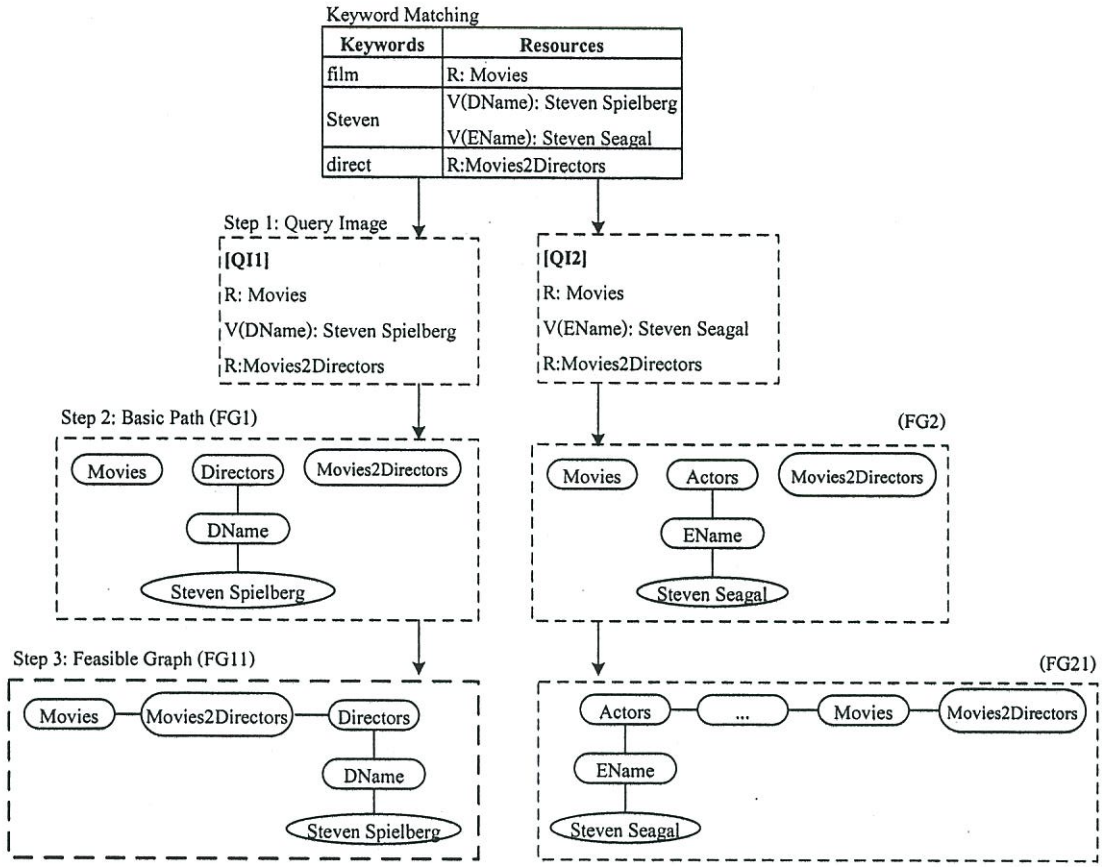


Figure 3.9 An example of query graph generator process

To comprehend the algorithms, Figure 3.8 shows the search graph derived from executed keyword query Q = {film, Steven, direct}, and Figure 3.9 illustrates steps of query graph generator. The search graph gives *FG11* derived from *QI1* is the best answer graph with the lowest cost 4. The answer graph *FG11* is the optimal solution.

Moreover, an example of queries with their description is showed in Table 3.3. Each description can be converted to form of semantic subgraph as showed in Figure 3.10 to Figure 3.14 respectively. It can be seen that a semantic graph can describe data from user keyword query.

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 user wants. Therefore, after the query graph generator process, the best informative answer graphs are converted to SQL in the next process.

Table 3.3 Example of queries for IMDB collection

#	Query	Description
1	film, Steven, direct	What movies did Steven direct?
2	Jaws, Steven, direct	Movie with title "Jaws", directed by Steven Spielberg
3	movie, Steven	What movies did Steven direct or produce or act or write or edit?
4	movie, Leonardo, Tom Hanks	Movies that "Leonardo" featuring "Tom Hanks"
5	James Bond, Daniel Craig, act	Movies with have part of title is "James Bond", acted by Daniel Craig

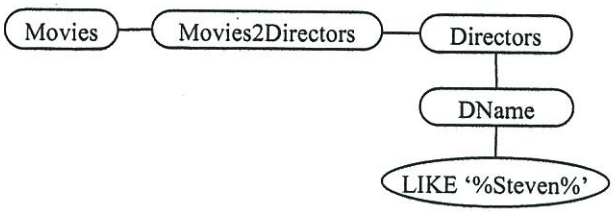


Figure 3.10 A semantic subgraph is described a keyword query {film, Steven, direct}

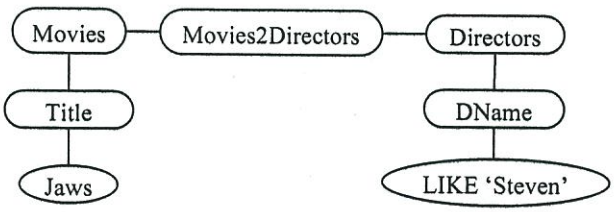


Figure 3.11 A semantic subgraph is described a keyword query {Jaws, Steven, direct}

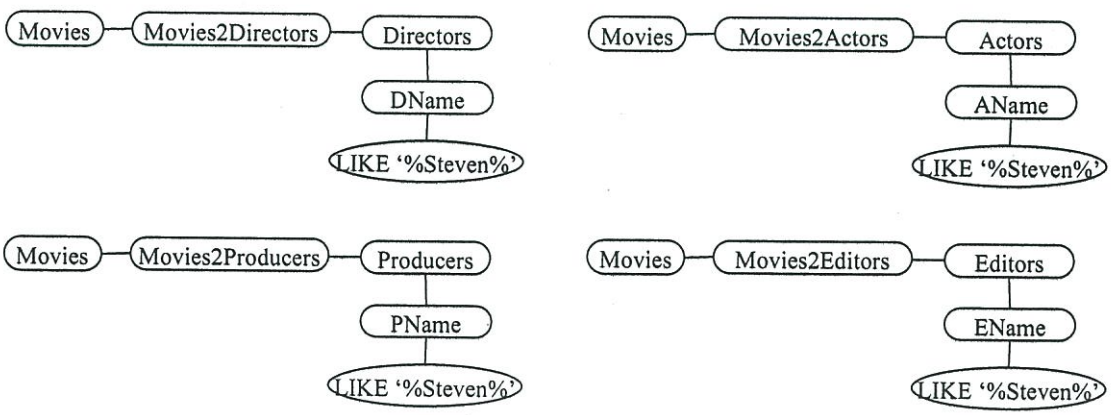


Figure 3.12 Examples of semantic subgraphs are described a keyword query {movie, Steven}

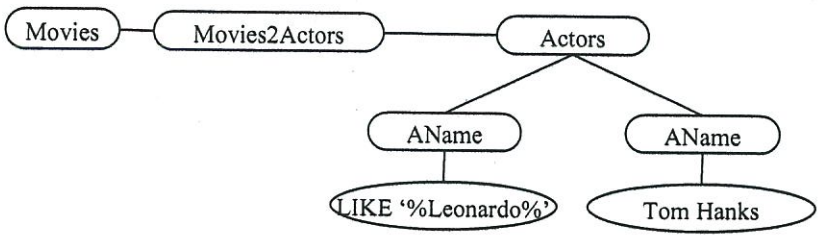


Figure 3.13 A semantic subgraph is described a keyword query {movie, Leonardo, Tom Hanks}

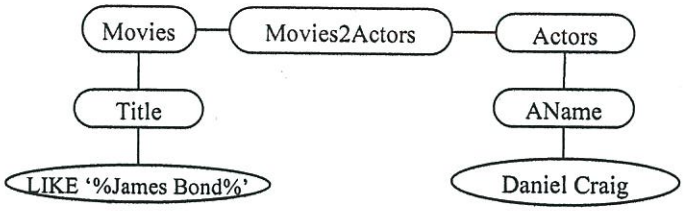


Figure 3.14 A semantic subgraph is described a keyword query {James Bond, Daniel Craig, act}

3.2.3 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 3.15 shows the nature 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.

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

SELECT [*attribute list*]

FROM [*relation list*]

WHERE [*selection condition*] [AND] [*join condition*]

From an example in Figure 3.9, an answer graph *FG11* is transformed to SQL statement as follow:

```
SELECT  Movies.*, Directors.DName
FROM    Movies, Movies2Directors, Directors
WHERE   Directors.DName = 'Steven Spielberg'
        AND Movies.MovieId = Movies2Directors.MovieId
        AND Movies2Directors.DirectorId=Directors.DirectorId
```

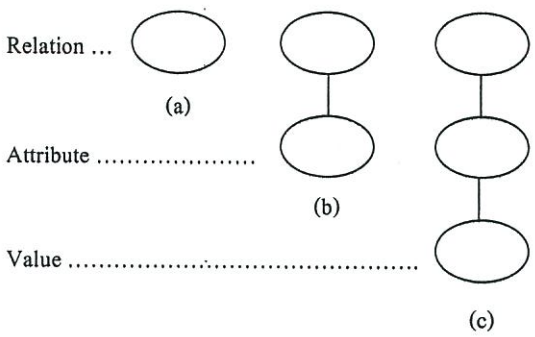


Figure 3.15 The nature of semantic model

3.2.4 Result Ranking

After SQL generator process, multiple consequence tuples are produced, that cause the question which top-k tuples are the most likely answers for the end users. In this section, result ranking process is used to solve the problem. We apply IR-style ranking function as in [25] by considering each tuple from previous section is a document. Moreover, the compactness of an answer graph is also used to rank result tuples because of it represents the concise relationship among objects in a relational database.

For IR-Style ranking function, we assume that values for all attributes are considered not limit to textual attributes. Thus, given a query Q, the ranking function is assigned to a tuple answer T is:

$$Score(T,Q) = \frac{\sum_{t \in T} Score(t,Q)}{size(T)} \tag{3.1}$$

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.

$$Score(t,Q) = \sum_{k \in Q \cap t} \frac{1 + \ln(1 + \ln(tf))}{(1-s) + s\frac{dl}{avdl}} \cdot \ln\frac{N+1}{df} \quad (3.2)$$

where t is an attribute value or attribute name or relation name that relevant to query Q, k is a single keyword in Q, tf is the frequency of a keyword k appears in t, df is the number of tuples that k appears, dl is the size of t in characters, $avdl$ is average length of t, N is the number of T, and s is a constant usually be 0.2.

For example, given query keywords Q = {film, Steven, direct}. The answer graph *FG11* from Figure 3.9 gives 87 result tuples, 51 tuples derived from "Steven Spielberg" and 36 tuples derived from "Steven Soderbergh". Table 3.4 shows two result tuples and their *Score* from Q.

Table 3.4 An example of result tuples of query keywords {film, Steven, direct}

MovieId	Title	Year	DName	*Score(T, Q)*
2000019	Erin Brockovich	2000	Steven Soderbergh	0.23
1993568	Jurassic Park	1993	Steven Spielberg	0.14

CHAPTER 4

EXPERIMENTAL EVALUATION

Keyword search evaluation over relational databases with a metadata search approach is becoming more challenging because of the existing techniques have different assumptions and goals. It makes some unfair any direct comparison among them. One consideration is that most of these techniques assume the capability for the instance-keyword search.

To evaluate the search effectiveness of this approach, the fraction of real data sets, the Internet Movie Database (IMDB)[1] is used in this experiment. A subset of original files is converted to relational tables as showed in Table 4.1 and used MySQL v5.0.24a with their default configuration and JDBC connections. All experiments were run on PC with a 1.66GHz CPU and 1G RAM. The database server and the client were run on the same PC. 29 real users were asked to provide a set of keyword queries and natural language explanation of what they were looking for. The users had no idea about the database schema information. Only the overview of the database contents were about had been provide to them. A database expert translated each natural language explanation to SQL query. The expected answer for each keyword query that expert had created is used as reference to evaluate the results returned by this approach.

Table 4.1 Internet movie dataset statistics

Relation Schema	#Tuples	Relation Schema	#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	movies2producers	13,768
movies2writers	11,680		
Total number of tuples			**141,012**

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

In preliminary experiments, the influence of the system does with metadata terms additionally is measured. Ten example queries were run with at least one metadata term in two systems, a metadata approach and non-metadata approach. Table 4.2 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). Consequently, a metadata search 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 4.2 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 4.3 The number of top-k results

Query	Top-10		Top-20		Top-30		Top-40		Top-50	
	#w/q	#w/o	#w/q	#w/o	#w/q	#w/o	#w/q	#w/o	#w/q	#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, an experiment shows a metadata search approach have the influence on top-k results. Top-k results are compared between two systems. Top-k columns of Table 4.3 (where k = 10, 20, 30, 40, and 50) show a number of top-k results obtained from the best answer graph of metadata search (#w/q) and non-metadata search (#w/o). It shows that most of all top-k results derive from the best answer graph of a metadata search, 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 3.9, 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.

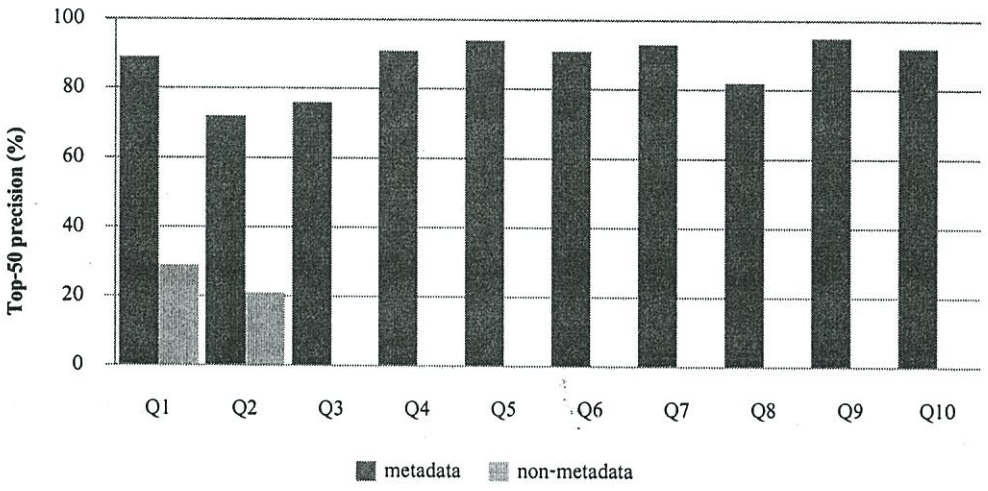


Figure 4.1 Top-50 precision on various queries

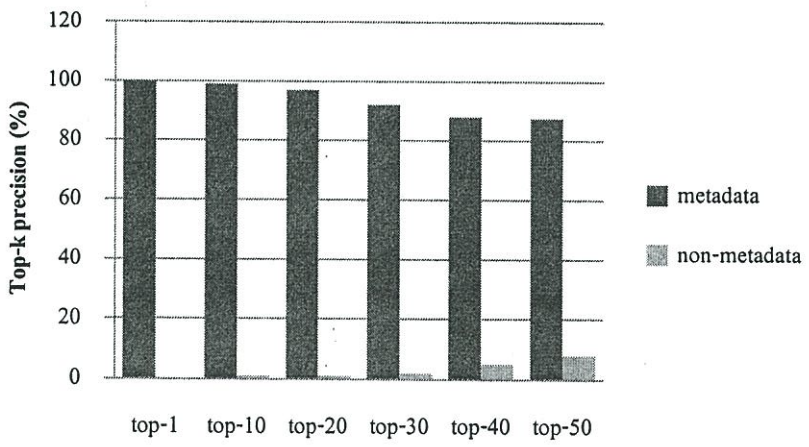


Figure 4.2 Top-k precision with different values of k

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 4.1 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 4.2. 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.

To analyze the best semantic answer graph, sets of query results are considered as shown in Figure 4.3. It shows that all of top-k results are in R(w/q). Moreover, all answer graphs of each query were showed to 20 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.

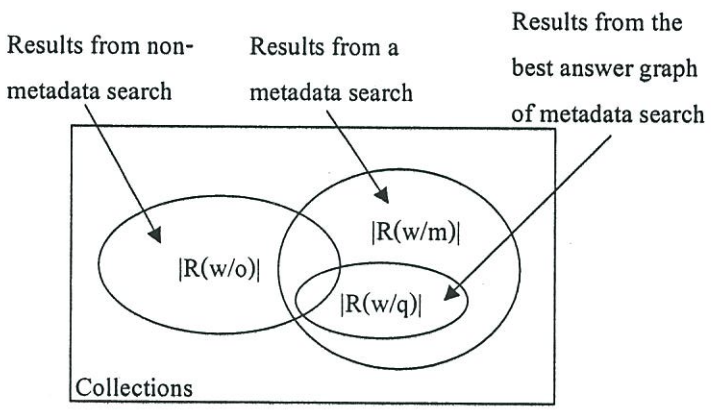


Figure 4.3 The sets of query results

CHAPTER 5

CONCLUSIONS AND RECOMMENDATION

5.1 Conclusions

This thesis presents a ranked keyword search system in relational databases with considering metadata terms. Users could query database by typing some keywords without knowing database schema and database query language. The query results are meaningful answers to query consisting of joining tuples and sometime could be single tuple. Moreover, the answers are ranked by using IR-style ranking function instead of using traditional style.

To achieve the purpose, a semantic-based graph is proposed as a data model and branch-and-bound strategy is used to find the optimal solutions. This semantic model is used to indicate what kinds of answer collections that users want. Additionally, IR-style ranking function is used to rank result tuples from answer graphs.

The experiments confirmed that metadata in a semantic representation is useful for giving precise answers in both attribute-level and relation-level. Moreover, preferred terms can solve ambiguity problem of querying. Answer graphs are optimal solutions and suitable for mapping to corresponding SQL statements.

5.1 Recommendation

There are many remaining challenges to make progressive advances in keyword-based query over relational database. First of all is ranking function. This approach ranks answer graphs based on the graph size. An answer with a smaller size should be ranked in higher because it represents more compactness and concise relationship among query keywords. However, the number of object relationships sometimes cannot directly estimate of interest in property. To improve the effective ranking, estimating the probability of relevance is one of factors to achieve. Moreover, efficient query processing is a crucial requirement for searching system that involves massive amounts of data. The efficient top-k processing is one of the key factors to improve the performance of keyword search in relational databases. Exceptionally, a query processing must efficiently generate only a few answers with high relevance score. A metadata search approach generates result tuples from semantic subgraph with high relevance score. However, it is possible

to an answer graph may not be the answer that users want. Thus, the directions of future work should be considering ranking function along with top-k processing.

REFERENCES

[1] Andrey B., Vagelis H. and Yannis P., "ObjectRank: Authority-Based Keyword Search in Databases", **International Conference on Very Large Data Bases**, vol. 30, Toronto, Canada, 2004, pp. 564-575.

[2] Benny K. and Yehoshua S., "Finding and Approximating Top-k Answers in Keyword Proximity Search", **ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, New York, USA,** 2006, pp. 173-182.

[3] Bolin D., Jeffrey X. Y., Shan W., Lu Q., Xiao Z. and Xuemin L., "Finding Top-k-Min-Cost Connected trees in Databases", **International Conference on Data Engineering**, 2007, pp. 836-845.

[4] Fang L., Clement Y., Weiyi M. and Abdur C., "Effective Keyword Search in Relational Databases", **ACM SIGMOD** International Conference on Management of Data, New York, USA, 2006, pp. 563-574.

[5] Gaurav B., Arvind H., Charuta N., Soumen C. and Sudarshan S., "Keyword Searching and Browsing in Databases using BANKS", **International Conference on Data Engineering,** Washington, DC, USA, 2002, pp. 431-440.

[6] Guoliang L., Beng C. O., Jianhua F., Jianyong W. and Lizhu Z., "EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data", **ACM SIGMOD International Conference on Management of Data**, New York, USA, 2008.

[7] Hao H., Haixun W., Jun Y. and Philip S. Y., "BLINKS: Ranked Keyword Searches on Graphs", **ACM SIGMOD International Conference on Management of Data**, New York, USA, 2007, pp. 305-316.

[8] Jaehui P. and Sang-goo L., "Keyword Search in Relational Databases", **Journal of Knowledge Information System**, vol. 26, no. 2, 2010, pp. 175-193.

[9] Jarunee S. and Veera B., "A Metadata Search Approach to Keyword Search in Relational Databases", **International Conference on Convergence and Hybrid Information Technology**, Nov. 2008, pp. 571-576.

[10] Jiajun G. and Hiroyuki K., "Extending Keyword Search to Metadata on Relational Databases", **International Workshop on Information-Explosion and Next Generation Search**, April 2008.

[11] Konstanin G., Benny K. and Yehoshua S., "Keyword Proximity Search in Complex Data Graphs", **International Conference on Management of Data,** New York USA, 2008, pp. 927-940.

[12] Lu Q., Jeffrey X. Y., Lijun C. and Yufei T., "Querying Communities in Relational Databases", **International Conference on Data Engineering**, Shanghai, 2009, pp. 724-735.

[13] Myint M. T. and Mie M. S. T., "Efficient Schema Based Keyword Search in Relational Databases", **International Journal of Computer Science, Engineering and Information Technology**, vol. 2, no. 6, Dec. 2012, pp. 13-32.

[14] Qi S. and Jennifer W., "Indexing Relational Database Content Offline for Efficient Keyword-Based Search", **International Database Engineering & Application Symposium**, Washington, DC, USA, 2005, pp. 297-306.

[15] Ricardo B. Y. and Berthier R. N., **Modern Information Retrieval**, ACM Press Series/Addison Wesley, New York, 1999.

[16] Richard W., Mark L. and Kevin K., "DbSurfer: A Search and Navigation Tool for Relational Databases", **Lecture Notes in Computer Science**, vol. 3112, Springer Berlin, Heidelberg, 2004, pp. 144-149.

[17] Roy G., Narayanan S., Suresh V. and Hector G. M., "Proximity Search in Databases", **International Conference on Very Large Data Bases**, Morgan Kaufmann Publishers Inc, San Francisco, 1998, pp. 26-37.

[18] Sanjay A., Surajit C. and Gautam D., "DBXplorer: A System for Keyword-Based Search over Relational Databases", **International Conference on Data Engineering**, Washington, DC, USA, 2002, pp. 5-16.

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

[20] Shan W. and Kun-Long Z., "Searching Databases with Keywords", **Journal of Computer Science and Technology**, vol. 20, no. 1, Jan. 2005, pp. 55-62.

[21] Shaul D., Gadi E., Shai G. and Eran P., "DTL's DataSpot: Database Exploration Using Plain Language", **International Conference on Very Large Data Bases**, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1998, pp. 645-649.

[22] Sina F. and Farshad F., "Effective Keyword Search over Relational Databases Considering Keywords Proximity and Keywords N-grams", **International Workshop on Database and Expert Systems Applications**, 2011, pp. 190-194.

[23] Sonia B., Elton D., Francesco G., Raquel T. L. and Yannis V., "Keyword Search over Relational Databases: A Metadata Approach", **ACM SIGMOD International Conference on Management of Data**, New York, USA, 2011, pp. 565-576.

[24] Vagelis H. and Yannis P., "Discover: Keyword Search in Relational Databases", **International Conference on Very Large Data Bases**, 2002, pp. 670-681.

[25] Vagelis H., Luis G. and Yannis P., "Efficient IR-Style Keyword Search over Relational Databases", **International Conference on Very Large Data Bases**, vol. 29, 2003, pp. 850-861.

[26] Varun K., Shashank P., Soumen C., Sudarhan S., Rushi D. and Hrishikesh K., "Biderectional Expansion for Keyword Search on Graph Databases", **International Conference on Very Large Data Bases**, 2005, pp. 505-516.

[27] Xiaoxin Y., Jiawei H. and Jiong Y., "Searching for Related Objects in Relational Databases", **International Conference on Scientific and Statistical Database Management**, 2005, pp. 227-236.

[28] Yanwei X., Yoshiharu I. and Jihong G., "Efficient Continual Top-k Keyword Search in Relational Databases", **Journal of Information Processing**, vol. 20, no. 1, Jan. 2012, pp. 114-127.

[29] Yi L., Xuemin L., Wei W. and Xiaofang Z., "SPARK: Top-k Keyword Query in Relational Databases", **The 2007 ACM SIGMOD International Conference on Management of Data**, New York, USA, 2007, pp. 115-126.

APPENDIX

APPENDIX A

PUBLICATIONS

International Journal

1. Jarunee Saelee and Veera Boonjing, "A Metadata Search Approach to Keyword Query in Relational Databases", **International Journal of Computer Applications**, vol. 69, no. 3, May 2013.

International Conference Proceedings

1. Jarunee Saelee and Veera Boonjing, "A Metadata Search Approach with Branch and Bound Algorithm to Keyword Query in Relational Databases", **International Conference on Computer Sciences and Convergence Information Technology,** Seoul, Korea, November 24 – 26, 2009, pages 653 – 658.
2. Jarunee Saelee and Veera Boonjing, "A Metadata Search Approach to Keyword Search in Relational Databases", **International Conference on Convergence and Hybrid Information Technology**, Busan, Korea, November 11-13, 2008, pages 571-576.

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:

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

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:

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

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:

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

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

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 {(MovieId: 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.

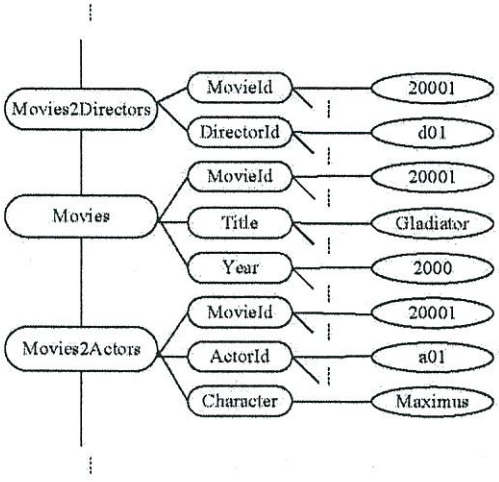


Fig 2: The semantic sub-graph

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 $\{k_1, k_2, \ldots, k_n\}$, 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 $i = 1, 2, \ldots, n$ 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 $i = 1, 2, \ldots, n$.

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.

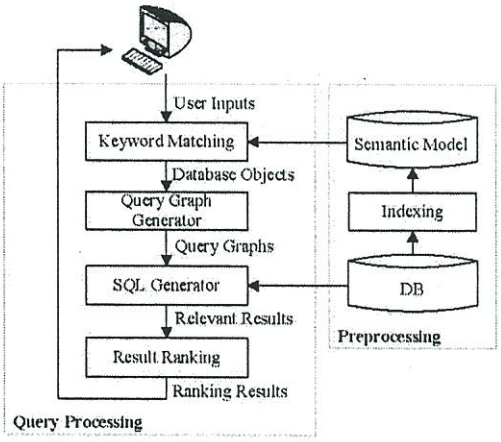


Fig 3: The architecture of a metadata search approach

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
   $K = \{k_1, k_2, ..., k_n\}$, where n is a number of query keywords
   $V_{k_i} = \{ v_{k_{i1}}, v_{k_{i2}}, ..., v_{k_{ij}} \}$, where $i = 1, 2, ..., n$ and j is a number of corresponding nodes in G of k_i
2. Determine a set of query images QI
   $QI_m = \{v_{k_{1j}} \in V_{k_1}, v_{k_{2j}} \in V_{k_2}, ..., v_{k_{nj}} \in V_{k_n}\}$.
   where $m = |V_{k_1}||V_{k_2}|...|V_{k_n}|$
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 $p_{k_i} = (v_{k_i}^R, v_{k_i}^A, v_{k_i})$ where $v_{k_i}^R$, $v_{k_i}^A$ 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 $p_{k_i} = (v_{k_i}^R, v_{k_i})$ where $v_{k_i}^R$ 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, $min_{i=1,m}\{cost(FG_i)\}$ 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.

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

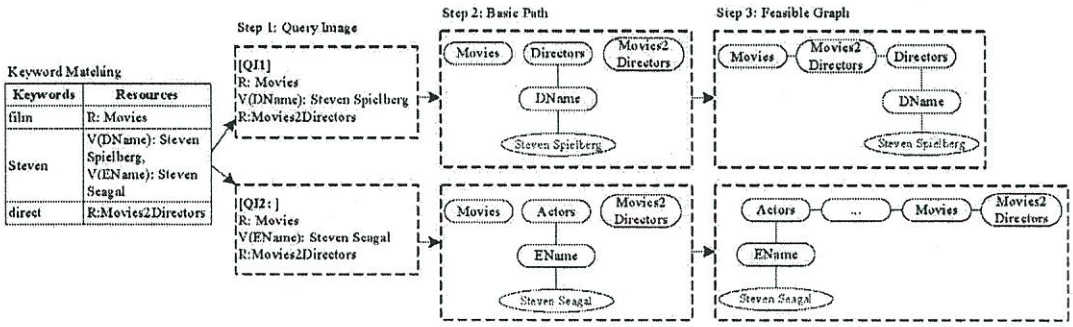


Fig 5: An example of query graph generator

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* $a = v_n$).

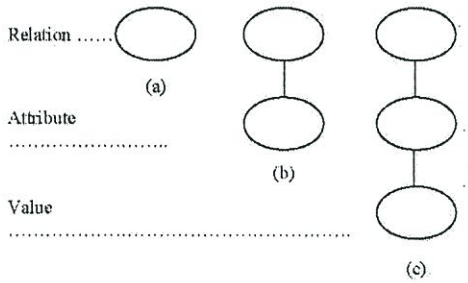


Fig 6: The nature of the semantic model

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:

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

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.

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

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.

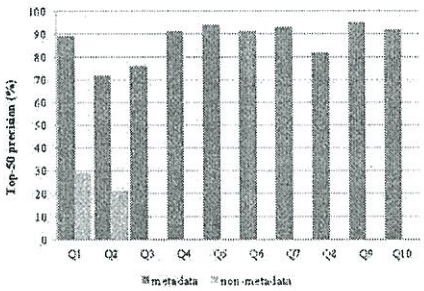


Fig 7: Top-50 precision on various queries

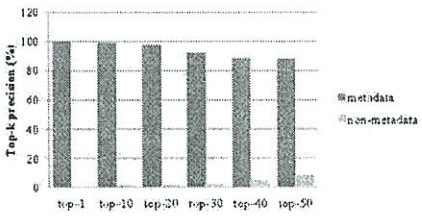


Fig 8: top-k precision with different values of k

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.

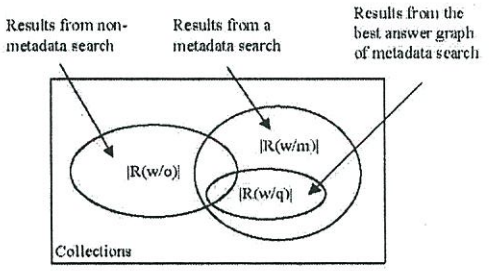


Fig 9: The sets of query results

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.

2009 Fourth International Conference on Computer Sciences and Convergence Information Technology

A Metadata Search Approach with Branch and Bound Algorithm to Keyword Query in Relational Databases

Jarunee Saelee[1,2]
[1]Software Systems Engineering Laboratory,
Department of Computer Science, Faculty of Science,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok, Thailand 10520
[2]National Centre of Excellence in Mathematics,
PERDO, Bangkok, Thailand 10400
s9062907@kmitl.ac.th

Veera Boonjing[1,2]
[1]Software Systems Engineering Laboratory,
Department of Computer Science, Faculty of Science,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok, Thailand 10520
[2]National Centre of Excellence in Mathematics,
PERDO, Bangkok, Thailand 10400
kbveera@kmitl.ac.th

Abstract— **Search engines on the Web have popularized the keyword-based search paradigm, while searching in databases users need to know a database schema and a query language. Keyword search techniques on the Web cannot directly be applied to databases because the data on the Internet and database are in different forms. So keyword search systems in relational databases have recently proposed. However existing systems limit type of keywords to database value terms, and generally assumed that answers are in instance level. Thus, this research aims to propose an effective approach for free-form keyword searching in databases which allows users to search either with database value terms, metadata terms or user terms. The metadata model accommodates these terms as well as underlying database semantics. Moreover, we present a branch and bound algorithm used for finding the optimal answer graphs. Our preliminary experiment results confirm precision of our approach.**

Keywords-component; Keyword search, Metadata search, Relational database, Database query, Branch and bound

I. INTRODUCTION

On the Web search engines, information retrieval systems [18] have popularized for searching unstructured data by keywords, while database query systems search structured data with structured query language or form-based interface. Internet users usually search information by typing some keywords as a query and results are sorted relevant documents. If database users could query databases in the same way, database query would be simple without knowing database schema and database query languages. 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., [1, 2, 8, 12, 17]) 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 Fig. 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. This means that the answer graph is a joining tuple tree which consists of all attributes of each tuple.

Our paper proposes to provide a natural capability to users by allowing them to query a database using database value terms, metadata terms, and their preferred terms. It employs a metadata search approach [13]. This approach 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 deal with relation-level, attribute-level, and value-level in vertical line. It means an answer graph can be an additional schema graph. We also present branch and bound algorithm to find the optimal solutions based on this approach. Moreover, we apply a state-of-the-art IR ranking method to evaluate scores between given query keywords and each tuple result.

The rest of this paper is organized as follows. Section 2 briefly introduces related works. Section 3 describes the background of a metadata search approach. Section 4 presents the query model. We give experiment results in Section 5. Section 6 concludes the paper and outlines future works.

978-0-7695-3896-9/09 $26.00 © 2009 IEEE
DOI 10.1109/ICCIT.2009.145

II. Related Work

Keyword search systems over relational databases have been extensively proposed. The earlier survey [16] overviewed systems such as BANKS [3], DBXplorer [1], DISCOVER [8], and ObjectRank [2], 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. 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. [9] proposed the extension of DISCOVER that handles non-metadata queries with both AND and OR semantics. Kacholia et al. [10] 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. [5] employed a dynamic programming to improve efficiency of identifying Steiner trees.

DataSpot [4] 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. [6] 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. [19]. 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 shorthest join paths to measure the strengths of their relationships. Mragyati [14] 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. Dissimilarly to the other approaches, Wheeldon et al. [17] 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 [15] but EKSO provided offline indexing time to significantly reduce query time computation. Given a keyword query, the system in [17] calculates a ranked set of virtual web pages with at least one keyword matching. 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 most similar in objective to our approach is in [7]. 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 Fig. 1, 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 every tuples.

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. [12] proposed a new IR style method to join-tuple tree ranking.

Directors

	DirectorId	DName
r11	d01	Ridley Scott
r12	d02	Steven Spielberg

Movies2Directors

	DirectorId	MovieId
r21	d01	20001
r22	d02	20021

Movies

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

Movies2Actors

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

Actors

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

Figure 1. Example of movie database instances.

III. Database Semantic Representation

We consider a database as a 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, a 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 a semantic graph for a movie database is illustrated in Fig. 2. Query results 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.

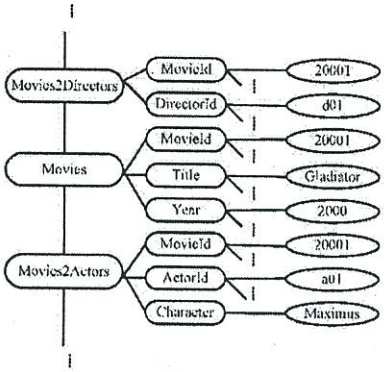


Figure 2. A semantic sub-graph of a movie database.

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 (relation-relation, relation-attribute, attribute-value).

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

Definition 3. A query keyword K is a set of $\{k_1, k_2, \ldots, k_n\}$, 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 $i = 1, 2, \ldots, n$ 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 $i = 1, 2, \ldots, n$.

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.

IV. Query Model

In this section, we describe the overview of the metadata search approach with branch and bound to generate the best answer graphs for given free-form queries, how answer graphs translate to SQL, and ranking function.

A. The Algorithms

For a given query Q, the query processing is enabled through the following steps.

Step 1: Identify query keywords and their nodes in graph G (definition 1). Therefore, after this step, a set of resources for each keyword are indicating what kinds of elements that users want.

Step 2: Determine all possible query images (definition 5) as a minimal set of a collection of keyword nodes.

Step 3: Branch and bound algorithm is used to find optimal solutions with have a minimum weight based on the number of nodes in a query graph. Because of a query graph is a tree, its weight is $|V| - 1$ where V is a set of nodes in it. Therefore, the best informative answer graphs which indicated what kinds of collections that users want will be determined first.

Step 4: Generate SQL statements from answer graphs.

Step 5: Rank obtained result tuples and returned to users.

```
Input: possible query graphs
Output: the best answer graphs and the lowest score
01. initial configuration: node = {root}, lowest-cost = ∞
02. while node is not empty do
03.     b-node = nodes with the lowest cost //considering from node
04.     node = {}
05.     for each b-node do
06.         branch b-node with its possible graph nodes (g-node)
07.         for each g-node do
08.             cost(g-node) = LB(g-node)
09.             if cost(g-node) ≤ lowest-cost then
10.                 add g-node to node
11.                 if cost(g-node) < lowest-cost then
12.                     set new lowest-cost
13.                 end if
14.             end if
15.         end for
16.     end for
17. end while
18. return node and lowest-cost
```

Figure 3. Branch and bound algorithm.

Branch and bound algorithm in Fig. 3 is used to find the optimal solutions from all candidate query graphs. We define the objective function as the compactness of 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. Thus, the lower bound $LB(g)$ is estimated by the number of nodes in a query image or a feasible graph. That is $LB(g)$ is the number of nodes in QI with its basic paths minus one where g is a query image and if g is a feasible graph, $LB(g)$ is the number of nodes in it minus one.

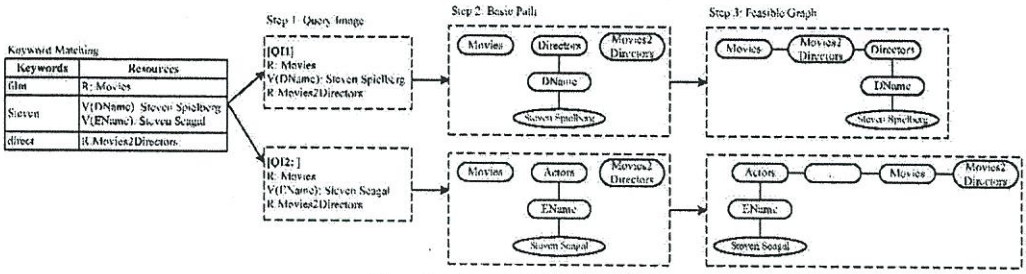


Figure 4. An example of query graphs.

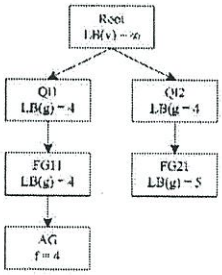


Figure 5. Search graph from branch and bound algorithm.

To comprehend the algorithms, Fig. 4 illustrates steps 1-3 with query keywords {film, Steven, direct} based on query graph generator process, and Fig. 3 shows branch and bound algorithm used in step 3 and Fig. 5 shows the search graph derived from executed these keywords. The search graph gives *FG11* derived from *Q11* is the best answer graph with the lowest cost 4. The answer graph *FG11* is the optimal solution. Moreover, we also give the details of step 4 and 5 in sections *B* and *C* respectively.

B. SQL Generator

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

- The *relation list* consists of all relation nodes in answer graph and indicates that what relations that tuple results originated from.
- The *attribute list* is a selection list based on the nature of a semantic model. Fig. 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.
- The *joining condition* extracts from foreign-key constraints of a database schema.
- 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 $v_1, v_2, \ldots, v_n$ belong to the same attribute a, the selection condition is *OR* semantics ($a = v_1$ *OR* $a = v_2$ *OR* ... *OR* $a = v_n$).

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

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

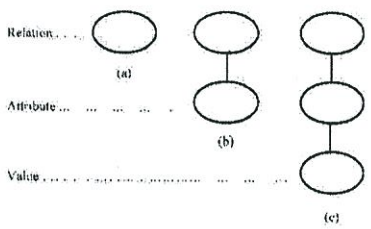


Figure 6. The nature of the semantic model.

C. Result Ranking

After SQL generator process, multiple consequence result tuples are produced, that causes the question which top-k tuples are the most likely answers for the end users. In this section, result ranking process is used to solve the problem. We apply IR-style ranking function as in [9] by considering each tuple from previous section is a document. Moreover, we assume that values for all attributes are considered not limit to textual attributes. Thus we have

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

where T_i is a result tuple that relevant to query Q, tf is the frequency of a keyword k appears in tuple T_i, df is the number of tuples that k appears, dl is the size of T_i in

characters, *avdl* is the average size of T, N is the number of T, and s is a constant usually be 0.2.

V. Experiment Results

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 1 and used MySQL v5.0.24a with their default configurations and JDBC connections. All experiments were run on PC with a 1.66GHz CPU and 1G RAM. The database server and the client were run on the same PC.

TABLE I. Internet Movie Dataset Statistics

Relation Schema	#Tuples	Relation Schema	#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	movies2producers	13,768
movies2writers	11,680		
Total number of tuples			**141,012**

In preliminary experiments, we measured the influence of the system does with metadata terms additionally. We run ten queries with at least one metadata term in two systems, a metadata approach and non-metadata approach. Table 2 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). We found that a metadata search 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 II. 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 III. The Number of Top-K Results

Query	Top-10		Top-20		Top-30		Top-40		Top-50	
	# w/q	# w/o	# w/q	# w/o	# w/q	# w/o	# w/q	# w/o	# w/q	# 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

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

Moreover, we show a metadata search approach have the influence on top-k results. We compared top-k results between two systems. Top-k columns of Table 3 (where k = 10, 20, 30, 40, and 50) show a number of top-k results obtained from the best answer graph of metadata search and non-metadata search. It shows that most of all top-k results derive from the best answer graph of a metadata search, 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 Fig. 4, 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 analyze the best semantic answer graph, we considered sets of query results as shown in Fig. 7. We found that all of top-k results are in R(w/q). Moreover, all answer graphs of each query were shown to 20 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.

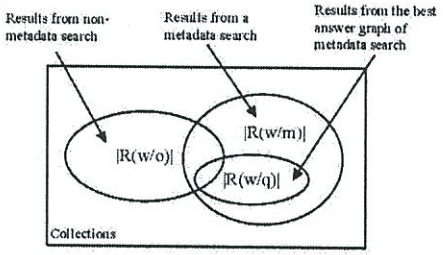


Figure 7. The sets of query results.

VI. Conclusions and Future Works

In this paper we propose a metadata search approach to ranked keyword search in relational databases. We also propose a semantic graph as a data model and strategies to find the optimal solutions. Additionally, we applied IR-style ranking function 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.

In the future, we plan to make the experiments further and will design a novel ranking function used to return top-k results in other aspects to users.

Acknowledgment

We are deeply grateful CEM research grant, Thailand. We thank Passamon Sittipinyo, Patthraporn Sornwiset, and Uraiwan Insoontorn for providing web search query demo.

REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases", *ICDE*, pages 5-16, 2002.

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

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti, "Keyword Searching and Browsing in Databases using BANKS", *ICDE*, pages 431-440, 2002.

[4] S. Dar, G. Entin, S. Geva, and E. Palmon, " DTL's DataSpot: Database Exploration Using Plain Language", *VLDB*, pages 645-649, 1998.

[5] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-k-Min-Cost Connected trees in Databases", *ICDE*, pages 836-845, 2007.

[6] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H.G. Molina, "Proximity Search in Databases", *VLDB*, pages 26-37, Morgan Kaufmann Publishers Inc, San Francisco, 1998.

[7] J. Gu, and H. Kitagawa, "Extending Keyword Search to Metadata on Relational Databases", *INGS*, 2008.

[8] V. Hristidis, and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases", *VLDB*, pages 670-681, 2002.

[9] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search Over Relational Databases", *VLDB*, pages 850-861, 2003.

[10] V. Kacholia, S. Pandit, A. Chakrabarti, S. Sudarhan, R. Desai, and H. Karambelkar, "Biderectional Expansion for Keyword Search on Graph Databases", *VLDB*, pages 505-516, 2005.

[11] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases", *SIGMOD*, ACM, New York, pages 563-574, 2006.

[12] Y. Luo, C. Yu, W. Wang, and X. Zhou, "SPARK: Top-k Keyword Query in Relational Databases", *SIGMOD*, ACM, New York, pages 115-126, 2007.

[13] J. Saelee, and V. Boonjing, "A Metadata Search Approach to Keyword Search in Relational Databases", *ICCIT*, pages 571-576, 2008.

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

[15] Q. Su, and J. Widom, "Indexing Relational Database Content Offline for Efficient Keyword-Based Search", *IDEAS*, pages 297-306, 2005.

[16] S. Wang, and K. Zhang, "Searching Databases with Keywords", *J. of Computer Science and Technology*, pages 55-62, 2005.

[17] R. Wheeldon, M. Levene, and K. Keenoy, "DbSurfer: A Search and Navigation Tool for Relational Databases", *LNCS*, pages 144-149, Springer Berlin, Heidelberg, 2004.

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

[19] X. Yin, J. Han, and J. Yang, "Searching for Related Objects in Relational Databases", *SSDBM*, pages 227 - 236, 2005.

Third 2008 International Conference on Convergence and Hybrid Information Technology

A Metadata Search Approach to Keyword Search in Relational Databases

Jarunee Saelee
Software Systems Engineering Laboratory, Department of Mathematics and Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand 10520
s9062907@kmitl.ac.th,

Veera Boonjing
Software Systems Engineering Laboratory, Department of Mathematics and Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand 10520
kbveera@kmitl.ac.th

Abstract

Search engines on the Web have popularized the keyword-based search paradigm, while searching in databases users need to know a database schema and a query language. Keyword search techniques on the Web cannot directly be applied to databases because the data on the Internet and database are in different forms. Moreover, existing systems for keyword searching limit type of keywords to database value terms. Thus, this research aims to propose a new method for keyword searching in databases which allows users to search either with database value terms, metadata terms or user terms. The metadata model accommodates these terms as well as underlying database semantics.

1. Introduction

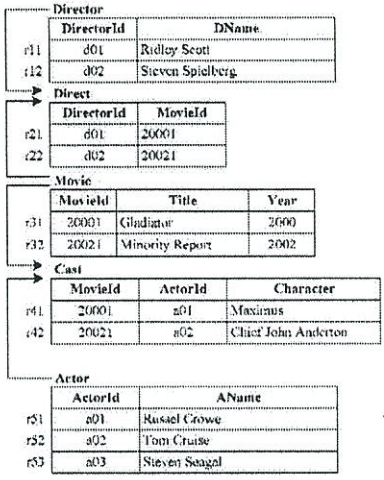
On the Web search engines, information retrieval systems [18] have popularized for searching unstructured data by keywords, while database query systems search structured data with structured query language or form-based interfaces. Internet users usually search information by typing some keywords as a query and results are sorted relevant documents. If databases users could query databases in the same way, database query would be simple without knowing database schema and database query languages. However, keyword search techniques on the Web cannot directly be applied to databases because the 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 the sorted relevant results in databases.

Existing systems supporting keyword search in relational databases (e.g., [1, 4, 9, 10, 13, 17]) limits 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. Moreover, users often query using their 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}.

Our paper proposes to provide a natural capability to users by allowing them to query a database using database value terms, metadata terms, and their preferred terms. It employs a metadata search approach. The new approach uses a semantic graph of underlying database to accommodate these terms and database semantics. The graph consists of database metadata, database values, user terms, and their semantic connections.

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 a query model. We give an illustrative example in Section 5. Section 6 concludes the paper.

978-0-7695-3407-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCIT.2008.70



Director

	DirectorId	DName
r11	d01	Ridley Scott
r12	d02	Steven Spielberg

Direct

	DirectorId	MovieId
r21	d01	20001
r22	d02	20021

Movie

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

Cast

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

Actor

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

Figure 1. An example of movie database instances

2. Related works

Keyword search systems over relational databases have been extensively proposed. The earlier survey is found in 2005 [12]. It overviewed systems such as BANKS [4], DBXplorer [10], DISCOVER [13], and ObjectRank [1], 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 query keywords. DBXplorer and Discover generate connected tuple trees through primary-key-foreign-key relationships that contain all query keywords. 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 which differ from our approach at our proposition supports both AND and OR semantics. Hristidis et al. [14] proposed the extension of DISCOVER that handles non-metadata queries with both AND and OR semantics. Kacholia et al. [15] 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. [2] employed a dynamic programming to improve efficiency of identifying Steiner trees.

Former times, DataSpot [11] is a database search system using free-form queries which similar to our approach of supporting free-form queries. 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. [8] 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. [16]. 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 shorthest join paths to measure the strengths of their relationships. Mragyati [6] is the system to keyword searching and browsing on relational databases. The system maps query keywords to a database schema using metadata as a four-level tree and translates an answer tree 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 paths greater than two. Dissimilarly to the other approaches, Wheeldon et al. [9] proposed a system to keyword search over relational databases which indexed a relational database as virtual documents to querying and navigation. Their approach indexes a 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 [7] but EKSO provided offline indexing time to significantly reduce query time computation. Given a keyword query, the system in [9] 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 is not support numerical queries.

More recent approaches have been attentively proposed ranking methods. ObjectRank [1] 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. [17] proposed a new ranking method that adapts the state of the art IR ranking method into the ranking join-tuple trees.

The most similar in objective to our approach is in [5]. It extended 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, this tuple is represented in form of {(MoviedId: 20001), (Title: Gladiator), (Year: 2000), (N4: Movie)} where N4 is a new metadata attribute. Considerably, there is too redundant information in every tuples.

3. Database semantic representation

We consider a database as a semantic model including metadata, database values, and user terms. Metadata and database values 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. In other word, it means to synonyms of objects that users type for querying. A user term is defined as (class, object), where classes consist of relation, attribute, and value, and objects are instances of these classes.

Informally, that a semantic model is viewed as a graph with nodes representing objects of three classes: a relation class, an attribute class, and a value class. Edges represent connections between corresponding objects: relation to relation, relation to its attribute, and attribute to attribute value. In other word, the object in the lower class is indicated to upper class. An example of a semantic graph for a movie database is illustrated in Figure 2. Query results should be connecting semantic subgraphs containing query keywords.

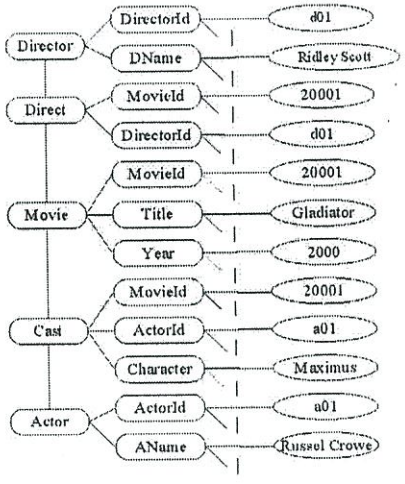


Figure 2. A semantic subgraph of a movie database

A formalized semantic graph is showed in the following. Moreover, we define necessary definitions used to describe a query model in the next section.

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 (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 be a set of $\{k_1, k_2, ..., k_n\}$, 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 containing v_i, and/or its attribute node v_i^A, and/or its relation node v_i^R, where A and R are used to indicate a class of v_i.

Definition 7. A feasible graph FG of QI is a collection of basic paths p_i in QI and shortest paths e connecting each p_i in graph G, where $i = 1, 2, ..., n$ and n is a number of basic paths in QI.

Definition 8. An answer graph is a 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. Query model

In this section, we describe the algorithm with branch and bound to generate results for given free-form queries and how answer graphs translate to SQL.

4.1 The algorithm

For a given query Q, Figure 3 gives an algorithm to generate optimal answer graphs. Step 1 examines a query to find query keywords and their nodes in graph G. Next determines all possible query images containing query keyword nodes. Step 3 finds distinct basic paths of each QI. In the last step, a branch and bound algorithm is used to finds optimal solutions with have a minimum weight based on the number of nodes in an answer graph. Because of an answer graph is a tree, its weight is $|V| - 1$ where V is a set of nodes in it.

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 $p_{k_i} = (v_{k_i}^R, v_{k_i}^A, v_{k_i})$ where $v_{k_i}^R$, $v_{k_i}^A$ 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 $p_{k_i} = (v_{k_i}^R, v_{k_i})$ where $v_{k_i}^R$ 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, $min_{i=1..m}(cost(FG_i))$ where m is a number of feasible graphs and $cost(FG) = |V| - 1$ where $|V|$ is a number of nodes in FG.

Figure 3. An algorithm for answer graph generation

For an optimization problem in the last step, objective function *f(x)* where *x* is an answer graph is to minimize a function *f(x)* over feasible answer graphs. The lower bound *LB(v)* is estimated by the number of nodes in a query image or a feasible graph. That is *LB(v)* is the number of distinct nodes in all basic paths of *QI* minus one where *v* is a query image and if *v* is a feasible graph, *LB(v)* is the number of nodes in it minus one.

4.2 SQL generation

To generate SQL from an answer graph of a query, objects, relationships, and their joining conditions are determined as follows. First, a *relation list* consists of every relation nodes in an answer graph. Second, an *attribute list* is a set of attributes which based on attribute nodes and relation nodes in a graph. An attribute list is considered by three cases. Initially, if an answer graph contains terminal-attribute nodes, an attribute list is a set of these terminal-attribute nodes, all attributes of terminal-relation nodes, and all attributes of value nodes. Next, if an answer graph does not contain any terminal-attribute nodes but contains terminal-relation nodes, an attribute list is a set of all attributes of terminal-relation nodes and all attributes of value nodes. Finally, if an answer graph does not contain any terminal-attribute nodes and terminal-relational nodes, an attribute list consists of all attributes of relation nodes in an answer graph. Third, the *joining condition* extracts from foreign-key constraints of a database schema. The last, the *selection condition* is determined from value nodes in an answer graph. In an 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 $v_1, v_2, \ldots, v_n$ belong to the same attribute a, the selection condition is *OR* semantics ($a = v_1$ *OR* $a = v_2$ *OR* ... *OR* $a = v_n$). Consequently, we have SQL for each answer graph as follow:

SELECT [*attribute list*]
FROM [*relation list*]
WHERE [*selection condition*] [AND]
[*joining condition*]

5. An example

To comprehend a metadata search approach, we give an example with the query "What are films Steven directed?" based on a database in Figure 1. Query keywords and their keyword node sets are shown in Table 1. To consider a keyword {film}, it's not directly matched with node {(Movie)^R} (*R* is indicated that Movie is a relation class) but corresponded to a user term with referred to {(Movie)^R}. A keyword {Steven} matches with two nodes containing {(Steven Spielberg)$^{V:DName}$, (Steven Seagal)$^{V:EName}$}, which are value nodes of attributes *DName* and *EName* respectively.

Table 1. Query keywords and keyword node sets

Keyword	Node
Film	(Movie)^R
Steven	(Steven Spielberg)$^{V:DName}$ (Steven Seagal)$^{V:EName}$
directed	(Direct)^E

Query images *QI1* and *QI2* are generated from Table 1 as in the following and their basic paths as in Figure 4 (a) and (b) respectively.

QI1: {(Movie)^R, (Steven Spielberg)$^{V:DName}$, (Direct)^E}
QI2: {(Movie)^R, (Steven Seagal)$^{V:EName}$, (Direct)^E}

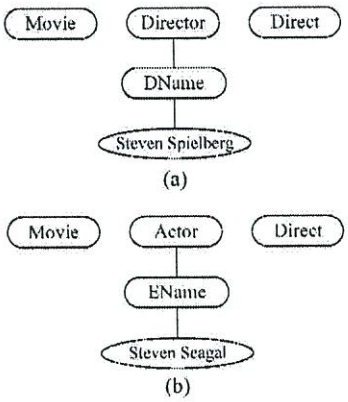


Figure 4. Basic paths of QI1 (a) and QI2 (b)

The branch and bound search graph is shown in Figure 5. It gives the optimal answer graph as a feasible graph *FG11* with cost = 4. To show the effectiveness of branch and bound search algorithm, we compare feasible graphs *FG11* and *FG21* in Figure 6. A feasible graph *FG11* intuitively shows the tighter and more optimal solution to a query than *FG21*.

The answer graph *FG11* in Figure 6 (a) is translated to the following SQL.

SELECT *Movie.*, Director.DName*
FROM *Movie, Direct, Director*
WHERE *Director.DName = 'Steven Spielberg'*
AND *Movie.MovieId = Direct.MovieId*
AND Direct.DirectorId=irector.DirectorId

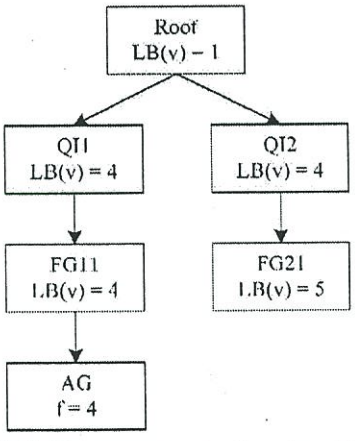


Figure 5. The search graph of branch and bound

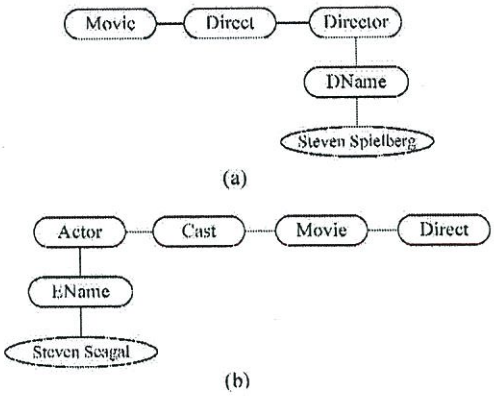


Figure 6. Feasible graphs FG11 of QI1 (a) and FG21 of QI2 (b)

6. Conclusion

In this paper we have proposed a metadata search approach to the problem of keyword search in relational database. We have also proposed a semantic graph as a data model and branch and bound algorithm to find the optimal solutions. Metadata in a semantic representation are useful for giving precise answers. Moreover, user terms can solve ambiguity problem of querying. Answer graphs obtained from branch and bound algorithm are optimal solutions and suitable for mapping to corresponding SQL statements.

Acknowledgements. We are deeply grateful Faculty of Science and Technology, Prince of Songkla University, Pattani Campus. We also thank Commission on Higher Education grant, Thailand.

7. References

[1] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "ObjectRank: Authority-Based Keyword Search in Databases", *VLDB*, VLDB Endowment, 2004, pp. 564-575.

[2] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-k-Min-Cost Connected trees in Databases", *ICDE*, 15-20 April, 2007, pp. 836-845.

[3] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases", *SIGMOD*, ACM, New York, 2006, pp. 563-574.

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

[5] J. Gu, and H. Kitagawa, "Extending Keyword Search to Metadata in Relational Database", *DEWS*, 2008.

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

[7] Q. Su, and J. Widom, "Indexing Relational Database Content Offline for Efficient Keyword-Based Search", *IDEAS*, 25-27 July, 2005, pp. 297-306.

[8] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H.G. Molina, "Proximity Search in Databases", *VLDB*, Morgan Kaufmann Publishers Inc, San Francisco, 1998, pp. 26-37.

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

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

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

[12] S. Wang, and K. Zhang, "Searching Databases with Keywords", *J. of Computer Science and Technology*, 2005, pp. 55-62.

[13] V. Hristidis, and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases", *VLDB*, VLDB Endowment, 2002, pp. 670-681.

[14] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search Over Relational Databases", *VLDB*, VLDB Endowment, 2003, pp. 850-861.

[15] V. Kacholia, S. Pandit, A. Chakrabarti, S. Sudarhan, R. Desai, and H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases", *VLDB*, VLDB Endowment, 2005, pp. 505-516.

[16] X. Yin, J. Han, and J. Yang, "Searching for Related Objects in Relational Databases", *SSDBM*, Lawrence Berkeley Laboratory, Berkeley, 2005, pp. 227 - 236.

[17] Y. Luo, C. Yu, W. Wang, and X. Zhou, "Spark: Top-k Keyword Query in Relational Databases", *SIGMOD*, ACM, New York, 2007, pp. 115-126.

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

AUTHOR BIOGRAPHY

PERSONNEL INFORMATION

Thai Name: นางสาว จารุณี แซ่หลี

English Name: Miss. Jarunee Saelee

Date of Birth: 20 June 1980

Permanent Address: 7/12 Moo 1, Sabayoi Sub-District, Sabayoi District, Songkhla Province, Thailand, 90210

Telephone: (+66) 0896670023

E-mail: s9062907@kmitl.ac.th

EDUCATION

2002 - 2005 Master Degree in Computer Science
Department of Computer Science, Faculty of Science,
King Mongkut's Institute of Technology Ladkrabang, Bangkok,
Thailand, 10520

1998 - 2002 Bechelor Degree in Applied Mathematics
Department of Mathematics and Computer Science,
Faculty of Science and Technology,
Prince of Songkla University, Pattani, Thailand, 94000

1996 - 1998 Hi-school in Science-Math Programme
Distance Education Institute, Songkhla, Thailand, 90000