Keyword Search in Databases: The Power of RDBMS

Lu Qin, Jeffrey Xu Yu, Lijun Chang
The Chinese University of Hong Kong, Hong Kong, China
{lqin,yu,ljchang}@se.cuhk.edu.hk

ABSTRACT
Keyword search in relational databases (RDBs) has been extensively studied recently. A keyword search (or a keyword query) in RDBs is specified by a set of keywords to explore the interconnected tuple structures in an RDB that cannot be easily identified using SQL on RDBMSs. In brief, it finds how the tuples containing the given keywords are connected via sequences of connections (foreign key references) among tuples in an RDB. Such interconnected tuple structures can be found as connected trees up to a certain size, sets of tuples that are reachable from a root tuple within a radius, or even multi-center subgraphs within a radius. In the literature, there are two main approaches. One is to generate a set of relational algebra expressions and evaluate every such expression using SQL on an RDBMS directly or in a middleware on top of an RDBMS indirectly. Due to a large number of relational algebra expressions needed to process, most of the existing works take a middleware approach without fully utilizing RDBMSs. The other is to materialize an RDB as a graph and find the interconnected tuple structures using graph-based algorithms in memory.
In this paper we focus on using SQL to compute all the interconnected tuple structures for a given keyword query. We use three types of interconnected tuple structures to achieve that and we control the size of the structures. We show that the current commercial RDBMSs are powerful enough to support such keyword queries in RDBs efficiently without any additional new indexing to be built and maintained. The main idea behind our approach is tuple reduction. In our approach, in the first reduction step, we prune tuples that do not participate in any results using SQL, and in the second join step, we process the relational algebra expressions using SQL over the reduced relations. We conducted extensive experimental studies using two commercial RDBMSs and two large real datasets, and we report the efficiency of our approaches in this paper.

Categories and Subject Descriptors
H.2.4 [Systems]: Query processing

General Terms
Algorithms, Performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '09, June 29–July 2, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-551-2/09/06 ...$5.00.

1. INTRODUCTION
A conventional RDBMS provides users with a query language SQL to query information maintained in large RDBs. It requires users to understand how the information is stored in an RDB on a relational schema, and know how to specify their requests using SQL precisely. In other words, all the information that can be possibly found in an RDB is the information that can be expressed using SQL. Due to the rapid information growth in the information era, many real applications require integrating both DB and IR technologies in one system. On one hand, the sophisticated DB techniques provide users with effective and efficient ways to access structured data managed by RDBMSs, and on the other hand, the advanced IR techniques allow users to use keywords to access unstructured data with scoring and ranking. Chaudhuri et al. discussed the issues on integrating DB and IR technologies in [6]. In supporting IR-styled search, commercial RDBMSs (such as DB2, ORACLE, SQL-SERVER) support full-text keyword search using a new SQL predicate of contain(A,k) where A is an attribute name and k is a user-given keyword. With the new predicate, a built-in full-text search engine in the RDBMS builds full-text indexes over text attributes in relations, and is used to retrieve the tuples that contain keywords in text attributes in relations efficiently.
In addition to full-text keyword search, another type of keyword search is to find how tuples that contain keywords in an RDB are interconnected [1, 16, 14, 23, 24, 4, 17, 19, 9, 11, 13, 26]. We call it structural keyword search. Consider a bibliography database that maintains publication records in several relations in an RDB. It is highly desirable to find out how certain research topics and/or authors are interrelated via sequences of co-authorship/citations. For example, given three keywords, “Keyword”, “DB”, and “Yannis”, the structural keyword search may find that “Yannis” possibly found in an RDB. It provides users with a query language SQL to query information maintained in large RDBs. It requires users to understand how the information is stored in an RDB on a relational schema, and know how to specify their requests using SQL precisely. In other words, all the information that can be possibly found in an RDB is the information that can be expressed using SQL. Due to the rapid information growth in the information era, many real applications require integrating both DB and IR technologies in one system. On one hand, the sophisticated DB techniques provide users with effective and efficient ways to access structured data managed by RDBMSs, and on the other hand, the advanced IR techniques allow users to use keywords to access unstructured data with scoring and ranking. Chaudhuri et al. discussed the issues on integrating DB and IR technologies in [6]. In supporting IR-styled search, commercial RDBMSs (such as DB2, ORACLE, SQL-SERVER) support full-text keyword search using a new SQL predicate of contain(A,k) where A is an attribute name and k is a user-given keyword. With the new predicate, a built-in full-text search engine in the RDBMS builds full-text indexes over text attributes in relations, and is used to retrieve the tuples that contain keywords in text attributes in relations efficiently.
In addition to full-text keyword search, another type of keyword search is to find how tuples that contain keywords in an RDB are interconnected [1, 16, 14, 23, 24, 4, 17, 19, 9, 11, 13, 26]. We call it structural keyword search. Consider a bibliography database that maintains publication records in several relations in an RDB. It is highly desirable to find out how certain research topics and/or authors are interrelated via sequences of co-authorship/citations. For example, given three keywords, “Keyword”, “DB”, and “Yannis”, the structural keyword search may find that “Yannis” possibly found in a tuple in an author relation, the three papers are three tuples in a paper relation, all are connected by foreign key references among tuples in other author-paper relation and paper-citation relation. The structural keyword search is completely different from full-text search. The former focuses on the interconnected tuple structures, whereas the latter focuses on the tuple content. In this paper, we concentrate ourselves on structural keyword search, and simply call it keyword search.
In the literature, the existing work are categorized into schema-based approaches and schema-free approaches for (structural) keyword search. The schema-based approaches [1, 16, 14, 23, 24] process a keyword query in two steps, namely, candidate network (CN) generation and CN evaluation. In the CN generation step, it generates all needed CNs (relational algebra expressions) up to a size, because it does not make sense if two tuples are far away in an interconnected tuple structure. In the CN evaluation step, it evalu-
ates all CNs using SQL. The schema-free approaches [4, 17, 19, 9, 11, 13, 26] support keyword search using graph-based in-memory algorithms by materializing an RDB as a graph. Almost all the existing work take a middleware approach [14, 23, 24, 4, 17, 19, 9, 11, 13, 26] except for two early work [1, 16] that evaluate CNs using SQL on an RDBMS directly. The middleware approach does not fully utilize the functionality of RDBMSs, and only uses SQL to retrieve data. In terms of the interconnected tuple structures, the majority of the work focus on connected trees [1, 16, 23, 24, 4, 17, 19, 9, 11]. Recently [13] studies finding sets of interconnected tuples which can be uniquely identified by a root within a user-given radius, and [26] studies finding sets of multi-center communities within a radius. All the three interconnected tuple structures are needed for different applications. But all are dealt in different ways and are not unified in the same framework.

A key issue we are studying in this work is how to support the three interconnected tuple structures (all connected trees up to certain size, all sets of tuples that are reachable from a root tuple within a radius, and all sets of multi-center subgraphs within a radius) in the same framework on RDBMS while not fully utilize the functionality of RDBMSs, and only uses SQL to retrieve data. In terms of the interconnected tuple structures, the majority of the work focus on connected trees [1, 16, 23, 24, 4, 17, 19, 9, 11]. Recently [13] studies finding sets of interconnected tuples which can be uniquely identified by a root within a user-given radius, and [26] studies finding sets of multi-center communities within a radius. All the three interconnected tuple structures are needed for different applications. But all are dealt in different ways and are not unified in the same framework.

The remainder of the paper is organized as follows. Section 2 discusses preliminaries. Section 3 discusses keyword search based on three different semantics, namely, connected tree semantics, distinct root semantics, and distinct core semantics. In Section 4 we discuss how to support the connected tree semantics using SQL on RDBMSs, and in Section 5 we discuss how to support the distinct core/root semantics using SQL on RDBMSs. The related work is given in Section 6. We conducted extensive performance studies and report our findings in Section 7. Finally, we conclude our work in Section 8.

2. PRELIMINARY

We consider a relational database schema as a directed graph $G_S(V,E)$, called a schema graph, where $V$ represents the set of relation schemas $\{R_1, R_2, \ldots \}$ and $E$ represents the set of edges between two relation schemas. Given two relation schemas, $R_i$ 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 $R_j$. There may exist multiple edges from $R_i$ to $R_j$ in $G_S$ if there are different foreign keys defined on $R_j$ referencing to the primary key defined on $R_i$. In such a case, we use $R_i \overset{n}{\rightarrow} R_j$, where $X$ is the foreign key attribute names. We use $V(G_S)$ and $E(G_S)$ to denote the set of nodes and the set of edges of $G_S$, respectively. In a relation schema $R_i$, we call an attribute, defined on strings or full-text, a text attribute, to which keyword search is allowed.

Given an RDB on the schema graph, $G_S$, we say two tuples $t_i$ and $t_j$ in an RDB are connected if there exists at least one foreign key reference from $t_i$ to $t_j$ or vice versa, and we say two tuples $t_i$ and $t_j$ in an RDB are reachable if there exists at least a sequence of connections between $t_i$ and $t_j$. The distance between two tuples, $t_i$ and $t_j$, denoted as $\text{dist}(t_i, t_j)$, is defined as the minimum number of connections between $t_i$ and $t_j$. An RDB can be viewed as a database graph $G_D(V,E)$ on the schema graph $G_S$. Here, $V$ represents a set of tuples, and $E$ represents a set of connections between tuples. There is a connection between two tuples $t_i$ and $t_j$ in $G_D$, if there exists at least one foreign key reference from $t_i$ to $t_j$ or vice versa (undirected) in the RDB. In general, two tuples, $t_i$ and $t_j$ are reachable if there exists a sequence of connections between $t_i$ and $t_j$ in $G_D$. The distance $\text{dist}(t_i, t_j)$ between two tuples $t_i$ and $t_j$ is defined as the same over the RDB. It is worth noting that we use $G_D$ to explain the semantics of keyword search but do not materialize $G_S$ over RDB.

Example 2.1: A simple DBLP database schema, $G_S$, is shown in Fig. 1. It consists of four relation schemas: Author, Write, Paper, and Cite. Each relation has a primary key TID. Author
has a text attribute Name. Paper has a text attribute Title. 
Write has two foreign key references: AID (refer to the primary
key defined on Author) and PID (refer to the primary key
defined on Paper). Cite specifies a citation relationship between
two papers using two foreign key references, namely, PID1 and
PID2 (PID2 is cited by paper PID1), and both refer to the
primary key defined on Paper. A simple DBLP database is shown
in Fig. 2. Fig. 2(a)-(d) show the four relations, where xi means a
primary key (or TID) value for the tuple identified with number i
in relation x (a, p, c, and w refer to Author, Paper, Cite, and
Write, respectively). Fig. 2(e) illustrates the database graph GD
for the simple DBLP database. The distance between a and p1,
dis(a1, p1), is 2.

3. KEYWORD SEARCH SEMANTICS

An m-keyword query is given as a set of keywords of size m,
\{k1, k2, \ldots, km\}, and is to search interconnected tuples that con-
tain the given keywords, where a tuple contains a keyword if a text
attribute of the tuple contains the keyword. To select all tuples from
a relation R that contain a keyword ki, a predicate contain(A, k1)
is supported in SQL in IBM DB2, ORACLE, and Microsoft SQL-
SERVER, where A is a text attribute in R. With the following SQL,
select * from R where contain(A1, k1) or contain(A2, k1)

it finds all tuples in R containing k1 provided that the attributes A1
and A2 are all and the only text attributes in relation R. Below, for
simplicity, we use \(\sigma_{\text{contain}(k_1)} R\) to indicate the selection of all tu-
uples in R that contains a keyword k1 in any possible text attributes.
We say a tuple contains a keyword, for example k1, if the tuple is
includid in the result of such selection.

In the literature, there are three types of semantics, which we call
called tree semantics, distinct root semantics, and distinct core
semantics. We introduce them below in brief.

Connected Tree Semantics: An m-keyword query finds connected
tuple trees [16, 4, 17, 9, 24, 19]. A result of such a query is a
minimal total joining network of tuples, denoted as MTINT.
First, a joining network of tuples (JNT) is a connected tree of tu-
ples where every two adjacent tuples, \(t_i \in r(R_a)\) and \(t_j \in r(R_b)\)
can be joined based on the foreign key reference defined on re-
tentional schemas \(R_a\) and \(R_b\) in \(G_S\) (either \(R_a \rightarrow R_b\) or \(R_b \rightarrow R_a\)).
Second, a joining network of tuples must contain all the m
keywords (by total). Third, a joining network of tuples is not total if
any tuple is removed (by minimal). The size of an MTINT is the
number of tuples in the tree. Because it is not meaningful if an
MTINT is too large in size, a user-given parameter Tmax specifies
the maximum number of tuples allowed in MTINTs.

Consider the DBLP database in Example 2.1 with a 2-keyword
query \(K = \{\text{Michelle}, \text{XML}\}\) and Tmax = 5. There are 7
MTINTs shown in Fig. 3(a). For example, the first connected tree
means that paper p1 is cited by paper p2 as specified by tuple c1.
Here p1 contains Michelle and p2 contains XML.

Distinct Root Semantics: An m-keyword query finds a collec-
tion of tuples, that contain all the keywords, reachable from a root
tuple (center) within a user-given distance (Dmax). The distinct
root semantics implies that the same root tuple determines the tu-
ples uniquely [13, 21, 15, 8]. In brief, suppose that there is a result
rooted at tuple \(t_r\). For any of the m-keyword, say \(k_1\), there is a tuple
\(t\) in the result that satisfies the following conditions. (1) \(t\) contains
the keyword \(k_1\). (2) Among all tuples that contain \(k_1\), the distance
between \(t\) and \(t_r\) is minimum. (3) The minimum distance between
\(t\) and \(t_r\) must be less than or equal to a user given parameter Dmax.

Figure 4: Join vs Semijoin/Join

Reconsider the DBLP database in Example 2.1 with the same
2-keyword query \(K = \{\text{Michelle}, \text{XML}\}\) (Dmax = 2). The
10 results are shown in Fig. 3(b). The root nodes are the nodes
shown in the top, and all root nodes are distinct. For example, the
rightmost result in Fig. 3(b) shows that two nodes, \(a_3\) (containing
Michelle) and \(p_2\) (containing XML), are reachable from the root
node \(p_4\) within Dmax = 2. Under the distinct root semantics, the
rightmost result can be output as a set \((p_4, a_3, p_2)\), where the
connections from the root node \(p_4\) to the two nodes can be ignored
as discussed in BLINKS [13].

Distinct Core Semantics: An m-keyword query finds multi-center
subgraphs, called communities [26]. A community, \(C(V, E)\), is
specified as follows. \(V\) is a union of three subsets of tuples, \(V =
V_c \cup V_b \cup V_e\). Here, \(V_c\) represents a set of keyword-tuples where
a keyword-tuple \(v_k \in V_c\) contains at least a keyword and all m
keywords in the given m-keyword query must appear in at least
one keyword-tuple in \(V_c\). \(V_b\) represents a set of center-tuples where
there exists at least a sequence of connections between \(v_c \in V_c\) and
every \(v_k \in V_b\) such that \(\text{dis}(v_c, v_k) \leq \text{Dmax}\), and \(V_c\) represen-
t a set of path-tuples which appear on a shortest sequence of con-
nections from a center-tuple \(v_c \in V_c\) to a keyword-tuple \(v_k \in V_b\)
if \(\text{dis}(v_c, v_k) \leq \text{Dmax}\). Note that a tuple may serve several roles
as keyword/center/path tuples in a community. \(E\) is a set of con-
nections for every pair of tuples in \(V\) if they are connected over
shortest paths from nodes in \(V_c\) to nodes in \(V_b\). A community, \(C(V, E)\),
is uniquely determined by the set of keyword tuples, \(V_c\), which is
called the core of the community, and denoted as \(\text{core}(C(V, E))\)
in [26].

Reconsider the DBLP database in Example 2.1 with the same
2-keyword query \(K = \{\text{Michelle}, \text{XML}\}\) and Dmax = 2. The
4 communities are shown in Fig. 3(c), and the 4 unique cores are
\((a_3, p_2)\), \((a_3, p_3)\), \((p_1, p_2)\), and \((p_1, p_3)\), for the 4 communities
from left to right, respectively. The multi-centers for each of the
communities are shown in the top. For example, for the rightmost
community, the two centers are \(p_2\) and \(c_2\).

It is important to note that the parameter Dmax used in the dis-

tinct core/root semantics is different from the parameter Tmax used
in the connected tree semantics. Dmax specifies a range from a
center (root tuple) in which a tuple containing a keyword can be
possibly included in a result, and Tmax specifies the maximum
number of nodes to be included in a result.

4. CONNECTED TREE IN RDBMS

In DISCOVER [16], the two main steps are candidate network
generation and candidate network evaluation, for processing an m-
keyword query over a schema graph \(G_S\), under the connected tree
semantics. In the first candidate network generation step, DIS-
COVER generates a set of candidate networks over \( G_S \). A candidate network \((CN)\) corresponds to a relational algebra that joins a sequence of relations to obtain \( MTJINTs \) over the relations involved. The set of \( CNs \) is proved to be sound/complete and duplication-free in [16]. In the second candidate network evaluation step, all \( CNs \) generated are translated into SQL queries, and each is evaluated on an RDBMS to obtain the final results.

In the candidate network generation step, over the schema graph \( G_S \), all \( CNs \) are generated. A \( CN \) is a sequence of joins, where the number of nodes is less than or equal to \( T_{max} \), and the union of the keywords represented in a \( CN \) is ensured to include all the \( m \)-keywords. An example of \( CN \) is shown in Fig. 4(a). Here, \( R_1 \{ K' \} \) represents a project of \( r(R_1) \) containing tuples that only contain all keywords in \( K' \) \( (\subseteq K) \) and no other keywords, as defined in Eq. (1) [16].

\[
R_1 \{ K' \} = \{ t | t \in r(R_1) \land \forall k \in K', t \text{ contains } k \land \\
\forall k \in (K - K'), t \text{ does not contain } k \}
\]  

(1)

where \( K \) is the set of \( m \) keywords, \( \{k_1, k_2, \cdots, k_m\} \). A connection between two nodes in a \( CN \) means a join using a foreign key reference. In detail, in Fig. 4(a), all \( P, C, W, \) and \( A \) represent the four relations, Paper, Cite, Write, and Author, in DBLP (Fig. 1). \( P\{XML\} \) means \( \sigma_{\text{contain}(XML)}(\sigma_{\text{contain}(Michelle)}P) \), or equivalently the following SQL:

\[
\text{select * from Paper as P where contain(Title, XML) and not contain(Title, Michelle)}
\]

Note that there is only one text-attribute \text{Title} in the \text{Paper} relation. In the similar fashion, \( P\{\} \) means

\[
\text{select * from Paper as P where not contain(Title, XML) and not contain(Title, Michelle)}
\]

All \( CNs \) ensure to find all possible connected trees. Several effective \( CN \) pruning rules are given in [16], and an efficient \( CNs \) generation algorithm is discussed in [24].

All \( CNs \) are computed using SQL. An operator tree (join plan) is shown in Fig. 4(b) to process the \( CN \) in Fig. 4(a) using 5 projects and 4 joins. The resulting relation, the output of the join (\( j_1 \)), is a temporal relation with 5 TIDs from the 5 projected relations, where a resulting tuple represents an \( MTJINT \). The rightmost two connected trees in Fig. 3(a), are the two results of the operator tree Fig. 4(b), \( (p_2, c_5, p_4, w_5, a_3) \) and \( (p_4, c_4, p_4, w_5, a_3) \).

In this paper, we propose to use semijoin/join sequences to compute a \( CN \). A semijoin between \( R \) and \( S \) is defined in Eq. (2), which is to project (II) the tuples from \( R \) that can possibly join at least a tuple in \( S \).

\[
R \Join S = \Pi_R (R \bowtie S)
\]

(2)

Based on semijoin, a join \( R \bowtie S \) can be supported by a semijoin and a join as given in Eq. (3).

\[
R \bowtie S = (R \Join S) \Join S
\]

(3)

Recall that semijoin/joins were proposed to join relations in a distributed RDBMS, in order to reduce high communication cost at the expense of I/O cost and CPU cost. But, there is no communication in a centralized RDBMS. In other words, there is no obvious reason to use \( (R \Join S) \bowtie S \) to process a single join \( R \bowtie S \), since the former needs to access the same relation \( S \) twice. Below, we address the significant cost saving of semijoin/joins over joins when the number of joins is large, in a centralized RDBMS.

We implemented \( CN \) evaluation using SQL based on the similar strategies of the cost sharing discussed in [16, 24] by adapting the algorithm in [24]. In [24], in order to share the computational cost of evaluating all \( CNs \), Markowitz et al. constructed an operator mash. In a mash, there are \( n \cdot 2^{m-1} \) clusters, where \( n \) is the number of relations in the schema graph \( G_S \) and \( m \) is the number of keywords. A cluster consists of a set of operator trees (left-deep trees) that share common expressions. A left-deep tree is shown in Fig. 4(b), where the leaf nodes are the projects, and the non-leaf nodes are joins. The output of a node (project/join) can be shared by other left-deep trees as input. Given a large number of joins, it is extremely difficult to obtain an optimal query processing plan. It is because one best plan for an operator tree may make others slow down, if its nodes are shared by other operator trees. In practice, it is to select a projected relation with the smallest number of tuples to start and to join. However, the temporal tuples generated can be very large and the majority of the generated temporal tuples does not appear in any \( MTJINTs \).

We also implemented our semijoin/join \( CN \) evaluation strategy. To compute \( R \Join (S \bowtie T) \), it is done as \( S' \leftarrow S \bowtie T, R' \leftarrow R \bowtie S' \), with semijoins, in the reduction phase, followed by \( T \bowtie (S' \bowtie R') \) in the join phase. For the same \( CN \) Fig. 4(a), in the reduction phase
Figure 5: # of Temporal Tuples (Default Tmax = 5, m = 3)

(Fig. 4(c)), \( C' \leftarrow C \times P \{ X M L \}, W' \leftarrow W \times A \{ M ichelle \}, P' \leftarrow P \times C', \) and \( P' \leftarrow P' \times W' \), and in the join phase (Fig. 4(d)), it joins \( P' \times C' \) first, because \( P' \) is fully reduced, such that every tuple in \( P' \) must appear at an MTJINT. The join order is shown in Fig. 4(d).

Fig. 5 shows the number of temporal tuples generated using a real database DBLP on IBM DB2. We randomly selected 5 3-keyword queries with different keyword selectivity (the probability that a tuple contains a keyword in DBLP) with \( Tmax = 5 \). The number of generated temporal tuples are shown in Fig. 5(a). The number of tuples generated by the semijoin-approach is significantly less than that by the join approach. In the similar fashion, the number of temporal tuples generated by the semijoin-approach is significantly less than that generated by the join approach when \( Tmax \) increases (Fig. 5(b)) for a 3-keyword query. It leads to the significant cost saving as confirmed in our extensive testings.

**Remark 4.1:** Based on our findings, when processing a large number of joins for keyword search on RDBMSs, it is the best in practice to process a large number of small joins to avoid intermediate join results to be very large and domiinative, if it is difficult to find an optimal query processing plan or the cost of finding an optimal query processing plan is large.

Note that the algorithms that compute MTJINTs using SQL on RDBMSs cannot efficiently support the distinct core/root semantics. The main reason is that it needs to use a large \( Tmax \) to find all MTJINTs that can possibly contain all the distinct core/root results whose size is controlled by a small radius (Dmax).

### 5. DISTINCT CORE/ROOT IN RDBMS

We outline our approach to process \( m \)-keyword queries with a radius (Dmax) based on the distinct core/root semantics. In the first step, for each keyword \( k_i \), we compute a temporal relation, \( Pair_i(tid_i, dis_i, TID) \), with three attributes, where both TID and \( tid_i \) are TIDs and \( dis_i \) is the shortest distance between TID and \( tid_i \), \( dis_i(TID, tid_i) \), which is less than or equal to Dmax. A tuple in \( Pair_i \) indicates that the \( TID \) tuple is in the shortest distance of \( dis_i \) with the \( tid_i \) tuple which contains the keyword \( k_i \). In the second step, we join all temporal relations, \( Pair_i \), for \( 1 \leq i \leq m \), on the attribute TID (center)

\[
S \leftarrow Pair_1 \cup Pair_2 \cdots \cup Pair_m \quad \text{where} \quad Pair_i \leftarrow \text{Pair}_i(\text{TID}=\text{Pair}_i \text{TID})
\]

Here, \( S \) is a \( 2m + 1 \) attribute relation, \( S(TID, tid_1, dis_1, \ldots, tid_m, dis_m) \).

Over the temporal relation \( S \), we can obtain the multi-center communities (distinct core) by grouping tuples on \( m \) attributes, \( tid_1, tid_2, \ldots, tid_m \). Consider the query \( K = \{ M ichelle, X M L \} \) and Dmax = 2, against the simple DBLP database in Fig. 2. The rightmost community in Fig. 3(c) is as follows.

<table>
<thead>
<tr>
<th>TID</th>
<th>tid1</th>
<th>dis1</th>
<th>tid2</th>
<th>dis2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>( p_1 )</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( p_2 )</td>
<td>( p_1 )</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( p_3 )</td>
<td>( p_1 )</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( p_4 )</td>
<td>( p_1 )</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Here, the distinct core consists of \( p_1 \) and \( p_4 \), where \( p_1 \) contains keyword \( M ichelle (k_1) \) and \( p_4 \) contains keyword \( X M L (k_2) \), and the 4 centers, \( \{ p_1, p_2, p_3, c_2 \} \), are listed on the TID column. Any center can reach all tuples in the core, \( \{ p_1, p_3 \} \), within Dmax. The above does not explicitly include the two nodes, \( c_1 \) and \( c_3 \) in the rightmost community in Fig. 3(c), which can be maintained in an additional attribute by concatenating the TIDs, for example, \( p_2, c_1, p_1, p_3, c_3, p_3 \). Due to space limit, we omit discussions on how to maintain paths by concatenating TIDs using SQL, which is trivial.

In the similar fashion, over the same temporal relation \( S \), we can also obtain the distinct root results by grouping tuples on the attribute TID. Consider the query \( K = \{ M ichelle, X M L \} \) and Dmax = 2, the rightmost result in Fig. 3(b) is obtained as follows.

<table>
<thead>
<tr>
<th>TID</th>
<th>tid1</th>
<th>dis1</th>
<th>tid2</th>
<th>dis2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_4 )</td>
<td>( a_3 )</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( p_4 )</td>
<td>( a_3 )</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

The distinct root is represented by the TID, and the rightmost result in Fig. 3(b) is the first of the two tuples, where \( a_3 \) contains keyword \( M ichelle (k_1) \) and \( p_4 \) contains keyword \( X M L (k_2) \). Note that a distinct root means a result is uniquely determined by the root. As shown above, there are two tuples with the same root \( p_4 \). We select one of them using the aggregate function \( \min \), following the semantics defined in BLINKS [13]. (Due to space limit, we omit discussions on how to record paths by concatenating TIDs.)

The complete results for the distinct core/root results are given in Table 1 and Table 2, respectively, for the same 2-keyword query, \( K = \{ M ichelle, X M L \} \) with Dmax = 2, against the DBLP database in Fig. 2. Both tables have an attribute \( Gid \) which is for easy reference of the distinct core/root results. Table 2 shows the same content as Table 1 by grouping on TID in which the yellow colored tuples are removed using the SQL aggregate function \( \min \) to ensure the distinct root semantics. The details will be discussed later in this paper.

### 5.1 Naïve Algorithms

In this section, we first explain our naïve algorithms using an example followed by discussions on the naïve algorithms in detail. Fig. 6 outlines the two main steps for processing the distinct core/root 2-keyword query, \( K = \{ M ichelle, X M L \} \), with Dmax = 2 against the simple DBLP database. Its schema graph, \( G_2 \), is in Fig. 1, and the database is in Fig. 2. In Fig. 6, the left side is to compute \( Pair_1 \) and \( Pair_2 \) temporal relations, for keyword
distance of \( d \) from a tuple containing a certain keyword, and its two attributes, \( tid_i \) and \( dis_i \), explicitly indicate that it is about keyword \( k_i \). The details of computing \( P_{i,j} \) for \( R_j \), \( 1 \leq j \leq 4 \), are given below.

\[
P_{1,1} = \Pi_{TID=tid_1,0 \cdots ,dis_1}(\sigma_{\text{contain}(k_1)R_1})
\]

\[
P_{1,2} = \Pi_{TID=tid_1,0 \cdots ,dis_1}(\sigma_{\text{contain}(k_1)R_2})
\]

\[
P_{1,3} = \Pi_{TID=tid_1,0 \cdots ,dis_1}(\sigma_{\text{contain}(k_1)R_3})
\]

\[
P_{1,4} = \Pi_{TID=tid_1,0 \cdots ,dis_1}(\sigma_{\text{contain}(k_1)R_4})
\]

Here, each join/project corresponds to a foreign key reference – an edge in schema graph \( G_S \). The idea is to compute \( P_{d,j} \) based on \( P_{d-1,j} \) if there is an edge between \( R_i \) and \( R_j \) in \( G_S \). Consider \( P_{2,3} \) for \( R_3 \), it computes \( P_{2,3} \) by union of three joins \( (P_{0,2} \Join R_3 \Join P_{0,4} \Join R_3) \), because there is one foreign key reference between \( R_3 \) (Paper) and \( R_2 \) (Write), and two foreign key references between \( R_3 \) and \( R_4 \) (Cite). It ensures that all \( R_j \) tuples that are reachable from distance \( d \) from a tuple containing a keyword \( k_i \) can be computed. Continue the example, to compute \( P_{2,3} \) for \( R_j \), \( 1 \leq j \leq 4 \), regarding keyword \( k_i \), we replace every \( P_{d,j} \) in Eq. (6) with \( P_{d+1,j} \) and replace “1 \( \rightarrow \) dis_1” with “\( 2 \rightarrow \) dis_1”. The process repeats for \( D_{max} \) times.

Suppose that we have computed \( P_{d,j} \) for \( 0 \leq d \leq D_{max} \) and \( 1 \leq j \leq 4 \), regarding keyword \( k_1 = \text{Michelle} \). We further compute the shortest distance between a \( R_i \) tuple and a tuple containing \( k_1 \) using union, group-by, and SQL aggregate function min. First, we conduct project, \( P_{d,j} \leftarrow \Pi_{TID,tid_1,dis_1}P_{d,j} \). Therefore, every \( P_{d,j} \) relation has the same tree attributes. Second, for \( R_j \), we compute the shortest distance from a \( R_i \) tuple to a tuple containing keyword \( k_1 \) using group-by (\( \Gamma \)) and SQL aggregate function min.

\[
G_1 \leftarrow TID,tid_1 \Gamma_{\min(dis_1)}(P_{0,j} \cup P_{1,j} \cup P_{2,j})
\]

(7)

where the left side of group-by (\( \Gamma \)) is group-by attributes, and the right side is the SQL aggregate function. Finally,

\[
Pair_1 \leftarrow G_1 \cup G_2 \cup G_3 \cup G_4
\]

(8)

Here, \( Pair_1 \) records all tuples that are in the shortest distance of a tuple containing keyword \( k_1 \), within \( D_{max} \). Note that \( G_1 \cap G_2 = \emptyset \), because \( G_1 \) and \( G_2 \) are tuples identified with TIDs from \( R_1 \) and \( R_2 \) relations and TIDs are unique in database as assumed. We can compute \( Pair_2 \) for keyword \( k_3 = \text{XML} \) following the same procedure as indicated in Eq. (5)-Eq. (8). With all \( Pair_1 \) and \( Pair_2 \) computed, we can easily compute distinct core/root results based on the relation of \( S \leftarrow Pair_1 \bowtie Pair_2 \) (Eq. (4)).

### Computing group-by \( \Gamma \) with SQL aggregate function

Consider Eq. (7), the group-by \( \Gamma \) can be computed by virtually pushing \( \Gamma \). Recall that all \( P_{d,j} \) relations, for \( 1 \leq d \leq D_{max} \), have the same schema, and \( P_{d,j} \) maintains \( R_i \) tuples that are in distance of \( d \) from a tuple containing a keyword. We use two pruning rules to reduce the number of temporal tuples computed.

### Rule-1

If the same \( (tid_i, TID) \) value appears in two different \( P_{d,j} \) and \( P_{d',j} \), then the shortest distance between \( tid_i \) and \( TID \) must be in \( P_{d',j} \) but not \( P_{d,j} \), if \( d' < d \). Therefore, Eq. (7) can be
Algorithm 1 Pair(GS, k1, Dmax, R1, \ldots, Rn)

Input: Schema GS, keyword k1, Dmax, n relations R1, \ldots, Rn.
Output: Pair, with 3 attributes: TID, tid1, dis1.

1: for j = 1 to n do
2: \( P_{d,j} \leftarrow \prod_{R_j \in R'} \sigma_{\text{tid}_j \leq \text{dmax}}(R_j) \); \( G_j \leftarrow \prod_{\text{tid}_j \leq \text{dmax}}(P_{d,j}) \);
3: end for
4: \( \text{for } d = 1 \) to Dmax do
5: \( \text{for } j = 1 \) to n do
6: \( \text{for } \Delta = 1 \) to Dmax do
7: \( P_{d,j} \leftarrow \emptyset; \)
8: \( \text{for all } (R_j, R_k) \in E(G) \) do
9: \( \Delta \leftarrow \prod_{\text{tid}_j \leq \text{dmax}}(R_j) \); \( G_j \leftarrow \prod_{\text{tid}_j \leq \text{dmax}}(P_{d,j}) \);
10: end for
11: end for
12: end for
13: return Pair;

Algorithm 2 DC-Naive(R1, \ldots, Rn, GS, K, Dmax)

Input: n relations R1, \ldots, Rn, schema graph GS, and m-keyword, K = \{k1, k2, \ldots, km\}, and radius Dmax.
Output: Relation with 2m + 1 attributes named TID, tid1, dis1, \ldots, tidm, dism.

1: for i = 1 to m do
2: \( \text{Pair}_i \leftarrow \text{Pair}(G, k_i, Dmax, R_1, \ldots, R_n); \)
3: \( S \leftarrow \text{Pair}_1 \cup \text{Pair}_2 \cup \ldots \cup \text{Pair}_m; \)
4: \( \text{Sort } S \text{ by tid1, tid2,} \ldots, \text{tidm}; \)
5: return S;

computed as follows.

\[
\begin{align*}
G_1 & = P_{d,j} \\
G_j & = G_j \cup \sigma_{\text{tid}_j \leq \text{dmax}}(P_{d,j}) \\
G_j & = G_j \cup \sigma_{\text{tid}_j \leq \text{dmax}}(P_{d,j})
\end{align*}
\]

Here, \( \sigma_{\text{tid}_j \leq \text{dmax}}(P_{d,j}) \) means to select \( P_{d,j} \) tuples where their \( \text{tid}_j \) does not appear in \( G_j \) already, in other words, there does not exist a shortest path between \( \text{tid}_j \) and \( TID \) before.

Rule-2: If there exists a shortest path between \( \text{tid}_j \) and \( TID \) value pair, say, \( \text{dis}_n(\text{tid}_j, TID) = d \), then there is no need to compute any tuple connections between the \( \text{tid}_j \) and \( TID \) pair, because all those will be removed later by group-by and SQL aggregate function min. In Eq. (6), every \( P_{d,j} \), \( 1 \leq j \leq 4 \), can be further reduced as \( P_{d,j} \leftarrow \sigma_{\text{tid}_j \leq \text{dmax}}(P_{d,j}) \).

The algorithm Pair() is given in Algorithm 1, which computes Pair, for keyword \( k_i \). It first computes all the initial \( P_{d,j} \) relations (refer to Eq. (5)), and initializes \( G_j \) relations (refer to the first equation in Eq. (9)) line 1-3. Second, it computes \( P_{d,j} \) for every \( 1 \leq d \leq \text{Dmax} \) and every relation \( R_j, 1 \leq j \leq n \), in two for loops (line 4-5). In line 7-11, it computes \( P_{d,j} \) based on the foreign key references in the schema graph GS, referencing to Eq. (6) and Eq. (9), using the two rules, Rule-1 and Rule-2. In our example, to compute \( \text{Pair}_1 \), it calls \( \text{Pair}(G, k_1, Dmax, R_1, R_2, R_3, R_4) \), where \( k_1 = \text{Michelle}, \text{Dmax} = 2 \), and the 4 relations \( R_j, 1 \leq j \leq 4 \).

The naive algorithm DC-Naive() to compute distinct cores is outlined in Algorithm 2. And the naive algorithm DR-Naive() to compute distinct roots can be implemented in the same way as DC-Naive() by replacing line 4 in Algorithm 2 with 2 group-bys as follows:

\[
X = \prod_{\text{tid}_1 \leq \text{dmax}}(S) = \prod_{\text{tid}_1 \leq \text{dmax}}(\text{tid}_1, \text{dis}_1, \ldots, \text{dis}_m) \quad \text{and} \quad S = \prod_{\text{tid}_1 \leq \text{dmax}}(S) \quad \text{and} \quad X = \prod_{\text{tid}_1 \leq \text{dmax}}(S) \quad \text{and} \quad S = \prod_{\text{tid}_1 \leq \text{dmax}}(S)
\]

5.2 Three-Phase Database Reduction

In this section, we discuss a new novel three-phase reduction approach to project a relational database \( RDB' \) out of \( RDB \) with which we compute multi-center communities (distinct core semantics). In other words, in the three-phase reduction, we significantly prune the tuples from an \( RDB \) that do not participate in any communities. We will also show that we can fast compute distinct root results using the same subroutine used in the three-phase reduction.

Fig. 7 outlines our main ideas for processing an \( m \)-keyword query, \( K = \{k_1, k_2, \ldots, k_m\} \), with a user-given Dmax, against an \( RDB \) with a schema graph GS.

The first reduction phase (from keyword to center): We consider a keyword \( k_i \) as a virtual node, called a keyword-node, and we take a keyword-node, \( k_i \), as a center to compute all tuples in an \( RDB \) that are reachable from \( k_i \) within \( Dmax \). A tuple \( t \) within \( Dmax \) from a virtual keyword-node \( k_i \) means that the tuple \( t \) can reach at least a tuple containing \( k_i \) within \( Dmax \). Let \( G_i \) be the set of tuples in \( RDB \) that can reach at least a tuple containing \( k_i \) within \( Dmax \), for \( 1 \leq i \leq m \). Based on all \( G_i \), we can compute \( Y = G_1 \times G_2 \times \cdots \times G_m \) which is the set of center-nodes that can reach any keyword-node \( k_i, 1 \leq i \leq m, \) within \( Dmax \). \( Y \) is illustrated as the shaded area in Fig. 7(a) for \( m = 2 \). Obviously, a center appears in a multi-center community must appear in \( Y \).

The second reduction phase (from center to keyword): In the similar fashion, we consider a virtual center-node. A tuple \( t \) within \( Dmax \) from a virtual center-node means that the tuple \( t \) is reachable from a tuple in \( Y \) within \( Dmax \). We compute all tuples that are reachable from \( Y \) within \( Dmax \). Let \( W \) be the set of tuples in \( G_i \) that can be reached from a center in \( Y \) within \( Dmax \), for \( 1 \leq i \leq m \). Note that \( W_i \subseteq G_i \). When \( m = 2 \), \( W_1 \) and \( W_2 \) are illustrated as the shaded areas on left and right in Fig. 7(b), respectively. Obviously, only the tuples, that contain a keyword within \( Dmax \) from a center, are possibly to appear in the final result as keyword tuples.

The third reduction phase (project DB): We project an \( RDB' \) out of the \( RDB \), which is sufficient to compute all multi-center communities by join \( G_i \times W_i \), for \( 1 \leq i \leq m \). Consider a tuple in \( G_i \) which contains a \( TID \) \( t' \) with a distance to the virtual keyword-node \( k_i \), denoted as \( \text{dis}(t', k_i) \), and consider a tuple in \( W_i \) which contains a \( TID \) \( t' \) with a distance to the virtual center-node \( c \), denoted as \( \text{dis}(t', c) \). If \( \text{dis}(t', k_i) + \text{dis}(t', c) \leq Dmax \),
Algorithm 3 DC$(R_1, R_2, \ldots, R_n, G_S, K, \text{Dmax})$

Input: $n$ relations $R_1, R_2, \ldots, R_n$, with schema graph $G_S$, and an $m$-keyword query, $K = \{k_1, k_2, \ldots, k_m\}$, and radius Dmax.

Output: Relation with $2 \cdot m + 1$ attributes named $\text{TID, tid}_1, \text{dis}_1, \ldots, \text{tid}_m, \text{dis}_m$.

1: for $j = 1$ to $n$, do
2: {Gj1, i, 1, Gj2, i, \ldots, Gjm, i} $\rightarrow$ PairRoot$(Gj, k, \text{Dmax, Rj, Rj, }\text{Rj, Rj, }\text{Rj, Rj, Rj, Rj})$,
3: if $j = 1$ to $n$, do
4: $Rj1 \rightarrow Rj2 \rightarrow Rj3$; \hspace{1cm} $Rj1 \rightarrow Rj2 \rightarrow Rj3$;
5: $Rj1 \rightarrow Rj2 \rightarrow Rj3$; \hspace{1cm} $Rj1 \rightarrow Rj2 \rightarrow Rj3$;
6: $Yj \rightarrow Gj2 \rightarrow Gj3 \rightarrow \ldots \rightarrow Gj, m$;
7: $Xj \rightarrow Rj \rightarrow Yj$;
8: for $j = 1$ to $n$ do
9: {Wj1, i, \ldots, Wjm, i} $\rightarrow$ PairRoot$(Gj, k, \text{Dmax, Rj, Rj, Rj, Rj, Xj, Xj, \ldots, Xj})$;
10: if $j = 1$ to $n$, do
11: Pathj1 $\rightarrow$ Gj1 $\rightarrow$ Gj1, TID=$\text{Wj}_1$ $\rightarrow$ TID $\rightarrow$ Pathj1; \hspace{1cm} Pathj1 $\rightarrow$ Gj1 $\rightarrow$ Gj1, TID=$\text{Wj}_1$ $\rightarrow$ TID $\rightarrow$ Pathj1;
12: Pathj1 $\rightarrow$ $\Pi_\text{TID}(Gj1, \text{dis}_1-\delta_d, \text{Gj1, dis}_1-\delta_d, \text{Pathj1})$; \hspace{1cm} Pathj1 $\rightarrow$ $\Pi_\text{TID}(Gj1, \text{dis}_1-\delta_d, \text{Gj1, dis}_1-\delta_d, \text{Pathj1})$;
13: Pathj1 $\rightarrow$ $\sigma_{\text{dis}_1-\delta_d}(\text{Pathj1})$; \hspace{1cm} Pathj1 $\rightarrow$ $\sigma_{\text{dis}_1-\delta_d}(\text{Pathj1})$;
14: $Rj1 \rightarrow Rj2 \rightarrow Rj3$; \hspace{1cm} $Rj1 \rightarrow Rj2 \rightarrow Rj3$;
15: for $j = 1$ to $n$ do
16: Pair$\rightarrow$ $\Pi_\text{pair}(Rj1', Rj2', \ldots, Rjm, Gj3, k, \text{Dmax})$;
17: $S \rightarrow$ $\Pi_\text{pair}(Rj1', Rj2', \ldots, Rjm, Gj3, k, \text{Dmax})$;
18: Sort $S$ by $\text{tid}_1, \text{tid}_2, \ldots, \text{tid}_m$;
19: return $S$.

Proof Sketch: First, we show if $y \notin Y$ then $y$ is a center-tuple in a community. Suppose $y \notin Y$ is in relation $R_j$, in line 6 of Algorithm 3, we know that $y \notin \Pi_\text{TID}(Gj1)$ for all $1 \leq i \leq m$, where $\Pi_\text{TID}(Gj1)$ contains all $R_j$ relation tuples that can reach keyword $k_i$ within Dmax. Hence, $y$ can reach all keywords within Dmax, or in other words, there exists tuples $u_i$ such that $\text{dis}(u_i, y) \leq \text{Dmax}$ and $u_i$ contains keyword $k_i$ for all $1 \leq i \leq m$. So $y$ is a center-tuple of the community with the core of $[u_1, u_2, \ldots, u_m]$. Next, we show if $y$ is a center-tuple in a community then $y \in Y$. Suppose $y$ is in relation $R_j$, and is a center-tuple in a community with the core of $[u_1, u_2, \ldots, u_m]$. By definition, $\text{dis}(u_i, y) \leq \text{Dmax}$, for any $1 \leq i \leq m$. So $y \in \Pi_\text{TID}(Gj1)$. In line 6, we know $y \in \Pi_\text{TID}(Y_j)$, then $y \in Y$.

Lemma 5.2: Suppose $Y$ is the set of center-tuples in all communities, i.e., $Y = \bigcup_{j=1}^{m} \Pi_\text{TID}(Y_j)$. For any tuple $u$ that contains a keyword, if there exists a tuple $y \in Y$ with $\text{dis}(u, y) \leq \text{Dmax}$, then $(u, y)$ is a keyword-tuple and center-tuple pair in a community.

Proof Sketch: Without loss of generality, we suppose $u$ contains keyword $k_i$. As proved in Lemma 5.1, if $y \notin Y$, there exists a community with the core of $[u_1, u_2, \ldots, u_m]$ that includes $y$ as a center-tuple. As $\text{dis}(u, y) \leq \text{Dmax}$, $[u_1, u_2, \ldots, u_m]$ is a core of a community that contains $y$ as a center-tuple. So $(u, y)$ is a pair of keyword-tuple and center-tuple in a community.

Theorem 5.1: Given a keyword query under distinct core semantics with Dmax. For any tuple $t$ in an RDB, if $t$ is not on any path of length $\leq \text{Dmax}$ from a keyword-tuple to a center-tuple in a community, then $t \notin \bigcup_{j \leq 1 \leq j \leq \leq m} \Pi_\text{TID}(Rj')$, where $Rj'$ are the relations used to join in the final step in Algorithm 3.

Proof Sketch: Suppose for a certain $j$ and $i$, $1 \leq j \leq n, 1 \leq i \leq m$, if $t \in \Pi_\text{TID}(Rj')$, from line 14 we get $t \in \Pi_\text{TID}(Pathj1)$, and thus in line 13 we have $t.d_y + t.d_y \leq \text{Dmax}$, where $t.d_y$ is the minimal distance from tuple $t$ to any tuple that contains keyword $k_i$, and $t.d_y$ is the minimal distance from tuple $t$ to any center-tuple. In other words, there must exist a tuple $u$ that contains keyword $k_i$ and a center tuple $y$, such that $\text{dis}(t, u) = t.d_y$ and $\text{dis}(t, y) = t.d_y$. This implies a path $u \rightarrow t \rightarrow y$ of length $\leq \text{Dmax}$.

From Lemma 5.2, we know $(u, y)$ is a pair of keyword-tuple and center-tuple in a community. This contradicts with the assumption that $t$ is not on any path of length $\leq \text{Dmax}$ from a keyword-tuple to a center-tuple in any of the 4 communities.

Consider the 4 communities in Fig. 3(c), $a_2$ in Fig. 2 can be pruned because it is not on any path of length $\leq 2$ from a keyword-tuple to a center-tuple in any of the 4 communities.
Theorem 5.2: Given a keyword query under distinct core semantics with Dmax. For any community C, suppose the core of C is \( \{u_1, u_2, \ldots, u_m\} \) and the centers are \( \{x_1, x_2, \ldots, x_r\} \), then for any \( 1 \leq k \leq r \), \( \{x_1, u_1, u_2, \ldots, u_m\} \in \Pi_{ID, tid_1, tid_2, \ldots, tid_m}(S) \).

Proof Sketch: As proved in Lemma 5.1, since the minimal distance from \( w \leq 1 \leq p \) is \( \leq \text{Dmax} \). Also, in line 9, we get all tuples \( w \) that satisfy (1) \( d(x_k, w) \leq \text{Dmax} \), and (2) the minimal distance from \( w \) to any tuple that contains a keyword is \( \leq \text{Dmax} \). The condition (2) is satisfied in line 1-3. Suppose for any \( 1 \leq i \leq m \), \( x_i \) contain \( u_i \) and the tuples on any shortest path from \( u_i \) to \( x_i \), then all tuples in \( P_i \) satisfy the conditions (1) and (2). Furthermore, for every tuple \( q \) in \( P_i \), it also satisfies \( q.d_{x_k} + q.d_{x_i} \leq \text{Dmax} \), so \( P_i \subseteq \Pi_{ID}(R_{j+1}) \) (line 13-14). Then in line 16, we have \( \{u_i, x_k\} \in \Pi_{id, tid}(Pairr) \), because there exists a path in \( P_i \) of length \( \leq \text{Dmax} \) that connects \( u_i \) and \( x_k \). In consequence, for any \( 1 \leq i \leq m \), \( \{u_i, x_k\} \in \Pi_{id, tid}(Pairr) \), and in line 17, \( \{x_1, u_1, u_2, \ldots, u_m\} \in \Pi_{id, tid_1, tid_2, \ldots, tid_m}(S) \). Consider the rightmost community in Fig. 3(c), the core is \( [p_1, p_3] \), and the centers include \( p_2 \) and \( c_2 \). As a result, \( (p_2, p_1, p_3) \) and \( (c_2, p_1, p_3) \) will both appear in \( \Pi_{id, tid_1, tid_2}(S) \).

The new algorithm \( DRI \) to compute distinct roots is given in Algorithm 5.

Algorithm 5 \( DRI(R_1, R_2, \ldots, R_n, G_S, K, \text{Dmax}) \)

Input: \( n \) relations \( R_1, R_2, \ldots, R_n \) with schema graph \( G_S \), and an \( m \)-keyword query, \( K = \{k_1, k_2, \ldots, k_m\} \), and radius \( \text{Dmax} \).

Output: Relation with 2 \( \cdot m + 1 \) attributes named TID, tid_1, dis_1, \ldots, tid_m, dis_m.

1: for \( i = 1 \) to \( m \) do
2: \( \{G_{j,i}, \ldots, G_{n,i}\} \leftarrow \text{PairRoot}(G_{j, k_i}, \text{Dmax}, R_1, \ldots, R_n, \sigma_{\text{contain}(k_i)}R_1, \ldots, \sigma_{\text{contain}(k_i)}R_n) \);
3: for \( j = 1 \) to \( n \) do
4: \( S_j \leftarrow G_{j,1} \land G_{j,2} \land \ldots \land G_{j,m} \);
5: \( S \leftarrow S_1 \cup S_2 \cup \ldots \cup S_n \);
6: return \( S \).

6. RELATED WORK

Keyword search in \( \text{RDBMS} \)s provides users with flexibility to retrieve information. The techniques to answer keyword queries in \( \text{RDBMS} \)s are mainly in two categories: \( \text{CN} \)-based (schema-based) and graph based (schema-free) approaches.

In the \( \text{CN} \)-based approaches [1, 16, 14, 23, 24], it processes an \( m \)-keyword query in two steps, candidate network (\( \text{CN} \)) generation and \( \text{CN} \) evaluation. \( \text{CN} \) evaluation is done using SQL on \( \text{RDBMS} \). \( \text{DBXplorer} \) [1] and \( \text{DISCOVER} \) [16] focused on retrieving connected trees using SQL on \( \text{RDBMS} \). In [16], Hristidis and Papakonstantinou proved how to generate a complete set of \( \text{CN} \)s to find all \( \text{MTJNTs} \) when the size of \( \text{MTJNTs} \) is at most allowed by a user-given \( \text{Tmax} \) (the number of nodes in \( \text{MTJNTs} \)), and discussed several query processing strategies with possible query optimization. In [24], Markowitz et al. discussed how to efficiently generate all \( \text{CN} \)s and how to process \( m \)-keyword queries on an \( \text{RDB} \) stream based on a sliding window model. The focus of work in [24] is on-demand query processing techniques in a middleware on top of an \( \text{RDBMS} \). All the above work [1, 16, 24] focused on finding all \( \text{MTJNTs} \), whose sizes are \( \leq \text{Tmax} \), that contain all \( m \) keywords (\( \text{AND} \)-semantics), and there is no ranking involved.

Among \( \text{CN} \)-based approaches, in \( \text{DISCOVER-II} \) [14], Hristidis et al. incorporated IR-style ranking techniques to rank the connected trees. Two algorithms, sparse and global pipeline were proposed in [14] to stop the query execution as soon as the top-k results are returned. \( \text{DISCOVER-II} \) is built in a middleware on top of an \( \text{RDBMS} \) and issues SQL queries to access data. In \( \text{SPARK} \) [23], Luo et al. proposed a new ranking function by treating each connected tree as a virtual document, and unified the \( \text{AND} \)-or \( \text{OR} \) semantics in the score function using a parameter. A monotonic upper bound score function was proposed to handle a non-monotonic score function. Two sweeping schemes, skyline sweep and block pipeline in \( \text{SPARK} \) were demonstrated to outperform \( \text{DISCOVER-II} \). Both work [14, 23] focused on finding top-k \( \text{MTJNTs} \), whose sizes are controlled by \( \text{Tmax} \), that contain all or some of the \( m \) keywords (\( \text{AND} \)-or \( \text{OR} \)-semantics). The ranking issues were also discussed in [2, 15, 22].

Finding top-k interconnected structures have been extensively studied in graph based approaches in which an \( \text{RDB} \) is materialized as a weighted database graph \( G_D(V, E) \).

The representative work on finding top-k connected trees are [4, 17, 19, 9, 11]. In brief, finding the exact top-k connected trees is an instance of the group Steiner tree problem [10], which is NP-hard. To find top-k connected trees, Bhalotia et al. proposed backward search in \( \text{BANKS-I} \) [4], and Kachoria et al. proposed bidirectional search in \( \text{BANKS-II} \) [17]. Kimelfeld et al. [19] proposed a general framework to retrieval top-k connected trees with polynomial delay under data complexity, which is independent of the underline minimum Steiner tree algorithm. Ding et al. [9] also introduced a dynamic programming approach to find the minimum connected tree and approximate top-k connected trees. Golenberg et al. in [11] attempted to find an approximate result in polynomial time under the query and data complexity.

Top-k connected trees are hard to compute, in \( \text{BLINKS} \) [13], He et al. proposed the distinct root semantics. Note that \( \text{BLINKS} \) deals with a general weighted graph whereas we deal with an unweighted graph in this work. In \( \text{BLINKS} \), search strategies were proposed with a bi-level index built to fast compute the shortest distances. \( \text{BLINKS} \) is a memory-based algorithm, which performs best when the bi-level index is in memory. In order to deal with large scale graphs, when the entire index can not resident in memory, Dalvi et al. [8] conducted keyword search on external memory graphs under the distinct root semantics. The work [8] is not based on SQL over an \( \text{RDBMS} \). In our approach, we can effectively use the \( \text{RDBMS} \) functions to handle large scale data without materializing an \( \text{RDB} \) as a graph.

Li et al. in \( \text{EASE} \) [21] defined a \( r \)-radius Steiner graph, where each \( r \)-radius Steiner graph is a subpart of a maximal \( r \)-radius subgraph. All the maximal \( r \)-radius subgraphs are precomputed and stored on disk, using an extended inverted index with keywords as the entries. To answer a keyword query, \( \text{EASE} \) [21] retrieves the relevant maximal \( r \)-radius subgraphs from disk, and returns the retrieved maximal \( r \)-radius subgraphs by removing the irrelevant nodes from the retrieved maximal \( r \)-radius subgraphs. With the semantics of maximal \( r \)-radius, some high ranked subpart that is contained in another subpart cannot be reported. Also, \( \text{EASE} \) needs to maintain a large inverted index for a given \( r \)-radius, and a larger \( r \)-radius can not be used to answer a query with a smaller radius, for example \( r' \)-radius, \( r' < r \). Qin et al. studied multi-center communities under the distinct core semantics in [26], and proposed new polynomial delay algorithms to compute all or top-k communities.

It is worth noting that all the work in the graph based approach do not use SQL on \( \text{RDBMS} \), and attempt to solve the top-k problems for an \( m \)-keyword query, using graph-based algorithms. Most of the work is in-memory based. The only exception is [8], which stores the graph in external memory.

Query processing and query optimization have been extensively studied. Some surveys can be found in [25, 12, 5, 30]. Semijoint was studied in [3]. Bernstein et al. in [3] proposed a linear-time
7. PERFORMANCE STUDIES

We conducted extensive performance studies to test the algorithms under the three semantics. For the connected tree semantics, we compare our semijoin/join based algorithm, denoted Semijoin-Join, with the join based algorithm [16, 24], denoted Join, and the block pipeline algorithm (BP) in SPARK [23] to compute the top 10 answers. For the distinct core semantics, we compare our new DC algorithm (Algorithm 3) with the naive DC-Naive algorithm (Algorithm 2). For the distinct root semantics, we compare our new DR algorithm (Algorithm 5) with the naive DR-Naive algorithm which is discussed at the end of Section 5.1. We did not compare our DR algorithm with BLINKS [13] and our DC algorithm with [26], because they are all memory based algorithms.

Two real large datasets, DBLP\(^2\) and IMDB\(^3\) are used for testing. The DBLP schema includes 4 tables: Author(Aid, Name), Write(Aid, Pid), Paper(Pid, Title), and Cite(Pid1, Pid2) with sizes, 651,253, 2,709,393, 1,089,689, and 112,303, respectively. The total number of tuples in DBLP used is 4,562,638. The IMDB schema includes 8 tables: Movie(Mid, Name), Direct(Mid, Did), Director(Did, Name), ActorPlay(Atid, Mid, Charactor), Actor(Atid, Name), ActressPlay(Asid, Mid, Charactor), Actress(Asid, Name), and Genres(Mid, Genre) with sizes, 1,276,733, 853,306, 150,762, 6,687,821, 964,845, 3,854,329, 572,905, and 813,379, respectively.

All algorithms were implemented in Java using JDK 1.5 and JDBC to connect to RDBMS. All tests were conducted in both ORACLE 10g Express and IBM DB2 Express-C 9.5, on a 2.8GHz CPU and 2GB memory PC running Windows XP. We report all our testing results on DB2 except for the algorithms to find connected trees in the IMDB dataset on ORACLE in order to show whether it is efficient to compute all results followed by finding top-k (our approach) or finding top-k using ORACLE.

The parameters and default values used for testing DBLP are shown in Table 3. Here, \(m\) is the number of keywords used in a keyword query, \(\text{Tmax}\) is used in the connected tree algorithms to control the number of nodes in a connected tree, and \(\text{Dmax}\) is used in the distinct root/core algorithms to specify the radius. In addition to \(m\) and \(\text{Tmax}/\text{Dmax}\), we use another two parameters \(k\text{sel}\) (keyword selectivity) and \(c\) (query compactness).

A \(k\text{sel}\) value is the average keyword selectivity for all keywords used in a keyword query, where the keyword selectivity is the probability that a tuple contains the keyword. We select \(k\text{sel}\) values as follows. We choose the max \(k\text{sel}\) value after removing the keywords that appear frequently such as stop words. In DBLP, there are in total 165, 260 different keywords among them 165, 154 keywords have \(k\text{sel}\) below 2E-3, which covers more than 99.9% of the total number of keywords. We choose \(2E-3\) to be the \(k\text{sel}\) value. Let the max \(k\text{sel}\) value be \(\alpha\). We divide the \(k\text{sel}\) range between \(\alpha\) and zero into 5 partitions, namely, \(\alpha/5, 2\alpha/5, 3\alpha/5, 4\alpha/5, \) and \(\alpha\), where the default value is \(3\alpha/5\). To test a specific \(k\text{sel}\) value, \(\alpha'\), for a given number of keywords, we select keywords such that the average \(k\text{sel}\) of the selected keywords is equal to \(\alpha'\). We tested many combinations and showed the representatives. For each keyword selectivity, the queries used under different semantics are shown in Table 4.

It is worth noting that a \(k\text{sel}\) value for an \(m\)-keyword query is related to the selectivity of individual keywords in a query. In addition to \(k\text{sel}\), we use another parameter \(c\) to control the connectivity between tuples that contain different keywords in a query, which we call compactness of an \(m\)-keyword query. Given an \(m\)-keyword query \(K = \{k_1, k_2, ..., k_m\}\), the compactness of \(K\), denoted as \(c(K)\), is defined as: \(c(K) = \max_{1 \leq i < j \leq m} \{\text{dis}(k_i, k_j)\}\). Here, \(\text{dis}(k_i, k_j)\) is the minimal distance between any tuple that contain keyword \(k_i\) and any tuple that contain keyword \(k_j\) in the database graph \(G_D\). Intuitively, a smaller \(c(K)\) implies a tighter connectivity between keywords in the query \(K\). When testing using \(c\), \(m\) and \(\text{Tmax}/\text{Dmax}\) are fixed to be the default values. For the DBLP dataset, \(c\) varies from 0 to 4 under the connected tree semantics, whereas \(c\) varies from 0 to 6 under the distinct core/root semantics. For each \(c\) value, the queries used under different semantics are shown in Table 5. Since \(k\text{sel}\) and \(c\) are set for different purposes, we use either \(k\text{sel}\) or \(c\) in our testing, together with \(m\), \(\text{Tmax}/\text{Dmax}\).
used for DBLP. Here, ksel values are smaller than those used in DBLP, because IMDB has a smaller average keyword selectivity. The Dmax values are also smaller than those used in DBLP because the database graph (G_D) of IMDB is denser than that of DBLP. For each keyword selectivity, the queries used under different semantics are shown in Table 7. For each c value, the queries used under different semantics are shown in Table 8.

We report both time and the number of temporal tuples generated. The time is the elapsed time (sec), and the number of temporal tuples generated is the number of all tuples generated in total.

### 7.1 Exp-1: Selectivity Testing

**Connected Tree**: The experimental results on DBLP under the connected tree semantics are shown in Fig. 8. Fig. 8(a) and Fig. 8(b) show that when the keyword selectivity increases, the time and the number of temporal tuples increase for both Join and Semijoin-Join approaches. Join consumes 2 times more time and generate 10 times more temporal tuples than Semijoin-Join on average. In Fig. 8(c) and Fig. 8(d), when the number of keywords m increases, the time for both Join and Semijoin-Join increases, but the numbers of temporal tuples increase when m ≤ 4 but decrease when m > 4 for both Join and Semijoin-Join. This is because, for a fixed Tmax, when m increases, the number of CNs increases, but the cost to evaluate a CN decreases since more constraints are added. Fig. 8(e) and Fig. 8(f) show the curves of Join and Semijoin-Join when varying Tmax from 3 to 7. Semijoin-Join outperforms Join. When Tmax increases from 3 to 4 or from 5 to 6, the time and number of tuples generated for both do not change, because when the tree size is even, at least one of the Write or Cite tuple will be a leaf node, and such a tree is invalid because Write or Cite do not have text-attributes.

Fig. 9 shows the testing results for Join and Semijoin-Join on IMDB. As shown in Fig. 9(a) and Fig. 9(b), when increasing the keyword selectivity, both time and number of tuples generated for Join and Semijoin-Join algorithm increase. Semijoin-Join outperforms Join. Fig. 9(c) and Fig. 9(d) show the curves when varying m. Like in DBLP, a small m means a small number of CNs but the cost for evaluating each CN increases. As the database graph of IMDB is denser than that of DBLP, when m is small, the cost for evaluating each CN is the dominant cost for Join but is not dominant for Semijoin-Join. When m increases, the difference between

<table>
<thead>
<tr>
<th>Semantics</th>
<th>ksel</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>2.5E-5</td>
<td>wish forget memory</td>
</tr>
<tr>
<td></td>
<td>5E-5</td>
<td>highway drive volume</td>
</tr>
<tr>
<td></td>
<td>7E-5</td>
<td>human smart target xmpion crown</td>
</tr>
<tr>
<td></td>
<td>1E-4</td>
<td>easy chance hat</td>
</tr>
<tr>
<td></td>
<td>1.25E-4</td>
<td>golden British diamond</td>
</tr>
</tbody>
</table>

| Core/Root | 3E-5 | style gentlemen rather |
|           | 0E-5 | that grow tomato |
|           | 7.5E-5 | lucky height slight money practice |
|           | 1E-4  | town winners advantage |
|           | 1.25E-4 | home packet successful |

Table 7: Keywords and ksel Used in IMDB

<table>
<thead>
<tr>
<th>Semantics</th>
<th>c</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>0</td>
<td>story school play</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>beaten to trendy king</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>bold video manager</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>school floor million</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>remember natural Michele</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>west Prussia arriving</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>series pork dumpling</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>May gather Merlin</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>dance ability bon</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>special equipment labour</td>
</tr>
</tbody>
</table>

Table 8: Keywords and c Used in IMDB

Join and Semijoin-Join becomes small, because the additional constraints by adding more keywords make the number of temporal tuples small. In Fig. 9(e) and Fig. 9(f), when Tmax increases, the costs for Join and Semijoin-Join increase, Semijoin-Join outperforms Join.

In this testing, we also tested BP. It is important to note that BP is an algorithm to compute top-k connected trees by pushing the ranking connected trees into the CN evaluation with Tmax and the cost saving of finding top-k is at the expense of computing more SQL to randomly access an RDB. In our Semijoin-Join approach, we attempt to compute all connected trees with Tmax followed by computing ranking on those resulting connected trees. In Fig. 9, Semijoin-Join is to compute all but not the top-k; and the additional cost to compute top-k can be ignored since the number of resulting connected trees is small. BP may be unstable because the time for BP does not largely depend on the keyword selectivity but on the distribution of the result trees with large scores. The time for BP increases when m increases, and is not effected by increasing Tmax because the top-k results tend to have small sizes, e.g. ≤ 3. We cannot report the number of temporal tuples generated by the BP code we obtained from the authors.

**Distinct Core**: The testing results for DC-Naive and DC on DBLP are shown in Fig. 10. Fig. 10(a) and Fig. 10(b) show that the time and the number of temporal tuples generated for both DC-Naive and DC increase when ksel increases. In Fig. 10(c) and Fig. 10(d), when m becomes larger, the costs of DC-Naive and DC increase. DC outperforms DC-Naive with less time and less number of temporal tuples generated. As shown in Fig. 10(e) and Fig. 10(f), when Dmax increases, the advantage of DC becomes more obvious.

Fig. 11 shows the testing results for DC-Naive and DC on IMDB. When ksel increases, DC outperforms DC-Naive (Fig. 11(a) and 11(b)). The time for DC-Naive increases sharply when ksel ≥ 7.5E−5. In Fig. 11(c) and Fig. 11(d), the time and the number of temporal tuples generated for DC-Naive and DC increase when m increases. DC saves more cost when m becomes larger. Fig. 11(e) and 11(f) show the curves for DC-Naive and DC when Dmax increases. DC generates less number of temporal tuples in all cases, but when Dmax ≤ 2, DC is slower than DC-Naive. This is because, when Dmax is small, the number of intermediate results for
DC-Naive is not large. In such a case, the performance of more small joins is not as effective as the performance of joins.

**Distinct Root:** The results under distinct root semantics are shown in Fig. 12. Fig. 12(a)-(c) show the time for DBLP and Fig. 12(d)-(f) show the time for IMDB. In all the cases, DR outperforms DR-Naive. In particular, when \( m \geq 4 \), the time of DR-Naive increases significantly while DR increases marginally, and when \( D_{max} \geq 4 \), the time for DR-Naive increases sharply while DR remains stable.

### 7.2 Exp-2: Compactness Testing

The testing results for DBLP and IMDB are shown in Fig. 13. The naive methods under all semantics perform worse in most cases.

As shown in Fig. 13(a), for the algorithms under the connected tree semantics, when the compactness of the query increases from 0 to 4 for DBLP, the processing time for both Join and Semijoin-Join decreases. The reason is given below. When the compactness of a query is small, the relationship between tuples that contain different keywords in the query will be tight, the tuples that contain different keywords can be connected even for a small \( T_{max} \), and the number of connected trees generated will be large. It results in large processing time. Semijoin-Join outperforms Join, because the number of intermediate tuples generated by Semijoin-Join is much smaller. Fig. 13(c) shows the processing time for the algorithms under the distinct core semantics of DBLP. The impact of the compactness values of queries under the distinct core semantics is not as obvious as that under the connected tree semantics. For example, the processing time with \( c = 0 \) is smaller than that with \( c = 2 \) under the distinct core semantics. It is because in the first step of the algorithms under the distinct core semantics, all keywords are evaluated individually. As a result, the cost for the first step is independent with the compactness, and it is possible that the cost for the first step becomes the dominant factor when the number of tuples generated in the first step is large. In Fig. 13(e) the performance for the algorithms under the distinct root semantics for DBLP is similar to that under the distinct core semantics.

The testing results for IMDB (Fig. 13(b) for the connected tree semantics, Fig. 13(d) for the distinct core semantics, Fig. 13(f) for the distinct root semantics) are similar to the testing results for DBLP for the similar reasons.

### 8. CONCLUSION

In this paper, we studied three different semantics of \( m \)-keyword queries, namely, connect-tree semantics, distinct core semantics, and distinct root semantics. We proposed a middleware free approach to compute such \( m \)-keyword queries on RDBMSs using SQL only. The efficiency is achieved by new tuple reduction approaches that prune unnecessary tuples in relations effectively followed by processing the final results over the reduced relations. Our middleware free approach makes it possible to fully utilize the functionality of RDBMS to support keyword queries in the same framework of RDBMS.

As a future work, we are planning to further study SQL query optimization to process such a large number of SQL statements efficiently. Also, in addition to computing all the interconnected tuple
structures for the three different semantics, we will study how to extend our approach to compute top-k interconnected tuple structures that are based on scoring and ranking.

Acknowledgment: This work was supported by grants of the Research Grants Council of the Hong Kong SAR, China (418206, 419008).

9. REFERENCES