# 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 ($RDB$s) has been extensively studied recently. A keyword search (or a keyword query) in $RDB$s 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 $RDB$s 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


*SIGMOD'09,* June 29–July 2, 2009, Providence, Rhode Island, USA.


## 1. INTRODUCTION

A conventional RDBMS provides users with a query language SQL to query information maintained in large $RDB$s. 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 RDBMSs 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 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" writes a paper cited by two papers whose titles contain "Keyword" and "DB" respectively. Here, "Yannis" possibly appears 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 *CN*s (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-

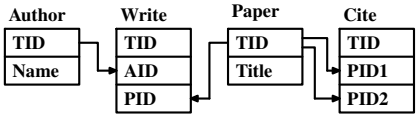



**Figure 1: *DBLP* Database Schema**

ates all *CN*s 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 *CN*s 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, 14, 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 RDBMSs without middleware.

The main contributions of this work are summarized below. We propose a middleware free approach, to support three types of keyword queries to find the three different interconnected tuple structures. We take a tuple reduction approach using SQL without additional new indexing to be built and maintained and without any precomputing required. To compute all connected trees, we propose a new approach to prune tuples that do not participate in any resulting connected trees followed by query processing over the reduced relations. To compute all multi-center subgraphs, we propose a new three-phase reduction approach to effectively prune tuples from relations followed by query processing over the reduced relations. We use the similar mechanism for computing all multi-center subgraphs to process sets of tuples that are reachable from a root tuple within a radius. We conducted extensive performance studies using two commercial RDBMSs and two large real datasets to confirm the efficiency of our proposed approaches.

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 $\{R_1, R_2, \cdots\}$ 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

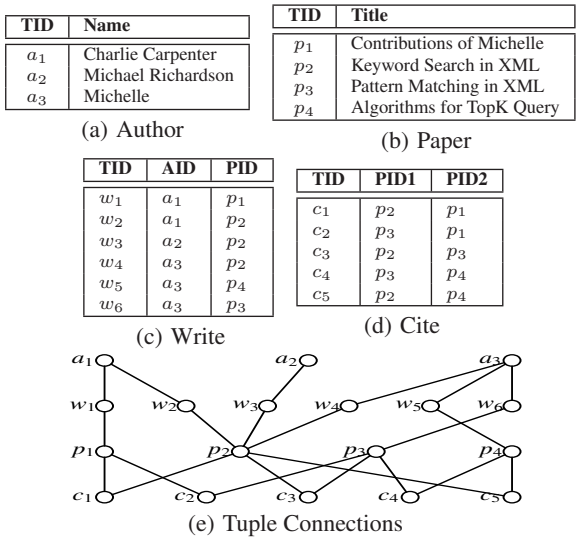



**Figure 2: *DBLP* Database**

use $R_i \xrightarrow{X} R_j$, 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.

A relation on relation schema $R_i$ is an instance of the relation schema (a set of tuples) conforming to the relation schema, denoted $r(R_i)$. We use $R_i$ to denote $r(R_i)$ if the context is obvious. A relational database ($RDB$) is a collection of relations. We assume, for a relation schema, $R_i$, there is an attribute called TID (Tuple ID), a tuple in $r(R_i)$ is uniquely identified by a TID value in the entire $RDB$. It is a reasonable assumption. In *ORACLE*, a hidden attribute called rowid in a relation can be used to identify a tuple in an $RDB$ uniquely. In addition, such a TID attribute can be easily supported as a composite attribute in a relation, $R_i$, using two attributes, namely, relation-identifier and tuple-identifier. The former keeps the unique relation schema identifier for $R_i$, and the latter keeps a unique tuple identifier in relation $r(R_i)$. Together with the two values, a tuple is uniquely identified in the entire $RDB$. For simplicity and without loss of generality, in the following discussions, we assume primary keys are TID, and use primary key and TID interchangeably.

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 $dis(t_i, t_j)$, 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 $dis(t_i, t_j)$ 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 $G_D$ 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 (paper 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 $x_i$ 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 $G_D$ for the simple *DBLP* database. The distance between $a_1$ and $p_1$, $dis(a_1, p_1)$, is 2. □

# 3. KEYWORD SEARCH SEMANTICS

An $m$-keyword query is given as a set of keywords of size $m$, $\{k_1, k_2, \cdots, k_m\}$, and is to search interconnected tuples that contain 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 $k_1$, a predicate $contain(A, k_1)$ 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** $contain(A_1, k_1)$ **or** $contain(A_2, k_1)$

it finds all tuples in $R$ containing $k_1$ provided that the attributes $A_1$ and $A_2$ are all and the only text attributes in relation $R$. Below, for simplicity, we use $\sigma_{contain(k_1)}R$ to indicate the selection of all tuples in $R$ that contains a keyword $k_1$ in any possible text attributes. We say a tuple contains a keyword, for example $k_1$, if the tuple is included in the result of such selection.

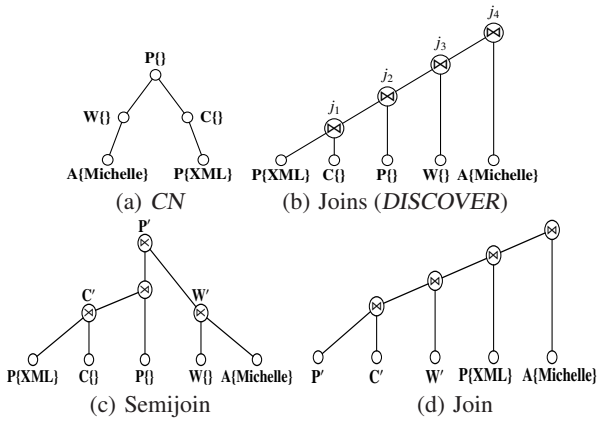
In the literature, there are three types of semantics, which we call *connected tree semantics*, *distinct root semantics*, and *distinct core semantics*. We introduce them below in brief.

**Connected Tree Semantics**: An $m$-keyword query finds connected tuple trees [16, 4, 17, 9, 24, 23, 19]. A result of such a query is a minimal total joining network of tuples, denoted as *MTJNT*. First, a joining network of tuples (*JNT*) is a connected tree of tuples where every two adjacent tuples, $t_i \in r(R_i)$ and $t_j \in r(R_j)$ can be joined based on the foreign key reference defined on relational schemas $R_i$ and $R_j$ in $G_S$ (either $R_i \rightarrow R_j$ or $R_j \rightarrow R_i$). 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 *MTJNT* is the total number of nodes in the tree. Because it is not meaningful if an *MTJNT* is too large in size, a user-given parameter Tmax specifies the maximum number of nodes allowed in *MTJNT*s.

Consider the *DBLP* database in Example 2.1 with a 2-keyword query $K = \{Michelle, XML\}$ and Tmax $= 5$. There are 7 *MTJNT*s shown in Fig. 3(a). For example, the first connected tree means that paper $p_1$ is cited by paper $p_2$ as specified by tuple $c_1$. Here $p_1$ contains $Michelle$ and $p_2$ contains $XML$.

**Distinct Root Semantics**: An $m$-keyword query finds a collection 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 tuples 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 $k_l$, there is a tuple $t$ in the result that satisfies the following conditions. (1) $t$ contains the keyword $k_l$. (2) Among all tuples that contain $k_l$, the distance between $t$ and $t_r$ is minimum[1]. (3) The minimum distance between $t$ and $t_r$ must be less than or equal to a user given parameter Dmax.

[1]If there is a tie, then a tuple is selected with a predefined order among tuples in practice.



(a) *CN* (b) Joins (*DISCOVER*)
(c) Semijoin (d) Join

**Figure 4: Join vs Semijoin/Join**

Reconsider the *DBLP* database in Example 2.1 with the same 2-keyword query $K = \{Michelle, XML\}$ (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].

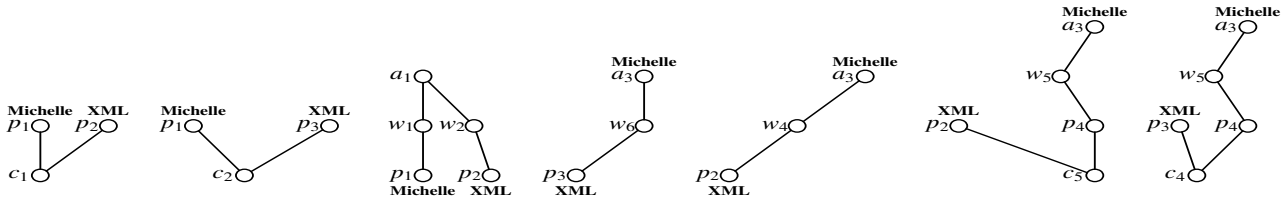
**Distinct Core Semantics**: An $m$-keyword query finds multi-center subgraphs, called communities [26]. A community, $C_i(V, E)$, is specified as follows. $V$ is a union of three subsets of tuples, $V = V_c \cup V_k \cup V_p$. Here, $V_k$ represents a set of keyword-tuples where a keyword-tuple $v_k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

Reconsider the *DBLP* database in Example 2.1 with the same 2-keyword query $K = \{Michelle, XML\}$ 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 $c_2$.

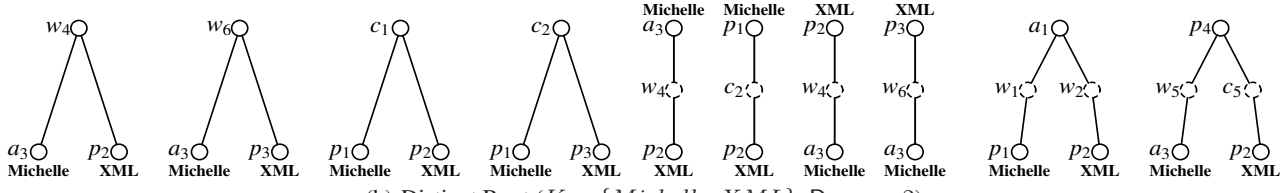
It is important to note that the parameter Dmax used in the distinct core/root semantics is different from the parameter Tmax used in the connected tree semantics. Dmax specifies a range from a center (root tuple) in which a tuple containing a keyword can be possibly included in a result, and Tmax specifies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

# 4. CONNECTED TREE IN RDBMS

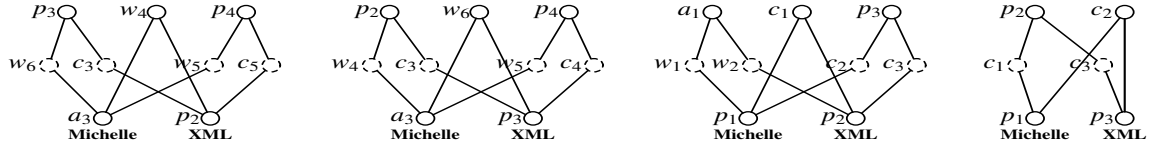
In *DISCOVER* [16], the two main steps are candidate network generation and candidate network evaluation, for processing an $m$-keyword query over a schema graph $G_S$, under the connected tree semantics. In the first candidate network generation step, *DIS-*




(a) Connected Tree ($K = \{Michelle, XML\}$, $\mathsf{Tmax} = 5$)




(b) Distinct Root ($K = \{Michelle, XML\}$, $\mathsf{Dmax} = 2$)




(c) Distinct Core ($K = \{Michelle, XML\}$, $\mathsf{Dmax} = 2$)

**Figure 3: Three Semantics**

*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 *MTJNT*s over the relations involved. The set of *CN*s is proved to be sound/complete and duplication-free in [16]. In the second candidate network evaluation step, all *CN*s 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 *CN*s are generated. A *CN* is a sequence of joins, where the number of nodes is less than or equal to $\mathsf{Tmax}$, 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, $R_i\{K'\}$ represents a project of $r(R_i)$ containing tuples that only contain all keywords in $K'$ ($\subseteq K$) and no other keywords, as defined in Eq. (1) [16].

$$R_i\{K'\} = \{t | t \in r(R_i) \wedge \forall k \in K', t \text{ contains } k \wedge \forall k \in (K - K'), t \text{ does not contain } k\} \quad (1)$$

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 $\sigma_{contain(XML)}(\sigma_{\neg contain(Michelle)}P)$, or equivalently the following SQL

**select** * **from** Paper **as** P
**where** contain(Title, XML) **and not** contain(Title, Michelle)

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

**select** * **from** Paper **as** P
**where not** contain(Title, XML) **and not** contain(Title, Michelle)

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

All *CN*s 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 ($j_4$), is a temporal relation with 5 TIDs from the 5 projected relations, where a resulting tuple represents an *MTJNT*. 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 $(p_3, c_4, p_4, w_5, a3)$.

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 ($\Pi$) the tuples from $R$ that can possibly join at least a tuple in $S$.

$$R \ltimes S = \Pi_R(R \bowtie S) \quad (2)$$

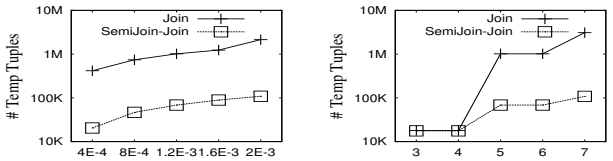
Based on semijoin, a join $R \bowtie S$ can be supported by a semijoin and a join as given in Eq. (3).

$$R \bowtie S = (R \ltimes S) \bowtie S \quad (3)$$

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 $(R \ltimes S) \bowtie S$ 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 *CN*s, Markowetz 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 *MTJNT*s.

We also implemented our semijoin/join *CN* evaluation strategy. To compute $R \bowtie (S \bowtie T)$, it is done as $S' \leftarrow S \ltimes T$, $R' \leftarrow R \ltimes S'$, 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




(a) Vary Keyword Selectivity (b) Vary Tmax

**Figure 5: # of Temporal Tuples** (Default Tmax $= 5$, $m = 3$)

(Fig. 4(c)), $C' \leftarrow C\{\} \ltimes P\{XML\}$, $W' \leftarrow W\{\} \ltimes A\{Michelle\}$, $P'' \leftarrow P\{\} \ltimes C'$, and $P' \leftarrow P'' \ltimes W'$, and in the join phase (Fig. 4(d)), it joins $P' \bowtie C'$ first, because $P'$ is fully reduced, such that every tuple in $P'$ must appear at an *MTJNT*. 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-join approach is significantly less than that by the join approach. In the similar fashion, the number of temporal tuples generated by the semijoin-join 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* RDBMS*s, it is the best in practice to process a large number of small joins to avoid intermediate join results to be very large and dominative, 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 *MTJNT*s 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 *MTJNT*s 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 $tid_i$ ($dis(TID, tid_i)$), 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)

$$S \leftarrow Pair_1 \underset{Pair_1.TID=Pair_2.TID}{\bowtie} Pair_2 \cdots Pair_{m-1} \underset{Pair_{m-1}.TID=Pair_m.TID}{\bowtie} Pair_m \quad (4)$$

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

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

| TID | tid1 | dis1 | tid2 | dis2 |
|---|---|---|---|---|
| $p_1$ | $p_1$ | 0 | $p_3$ | 2 |
| $p_2$ | $p_1$ | 2 | $p_3$ | 2 |
| $p_3$ | $p_1$ | 2 | $p_3$ | 0 |
| $c_2$ | $p_1$ | 1 | $p_3$ | 1 |

| Gid | TID | tid1 | dis1 | tid2 | dis2 |
|---|---|---|---|---|---|
| 1 | $a_3$ | $a_3$ | 0 | $p_2$ | 2 |
| 1 | $w_4$ | $a_3$ | 1 | $p_2$ | 1 |
| 1 | $p_2$ | $a_3$ | 2 | $p_2$ | 0 |
| 1 | $p_3$ | $a_3$ | 2 | $p_2$ | 2 |
| 1 | $p_4$ | $a_3$ | 2 | $p_2$ | 2 |
| 2 | $a_3$ | $a_3$ | 0 | $p_3$ | 2 |
| 2 | $w_6$ | $a_3$ | 1 | $p_3$ | 1 |
| 2 | $p_2$ | $a_3$ | 2 | $p_3$ | 2 |
| 2 | $p_3$ | $a_3$ | 2 | $p_3$ | 0 |
| 2 | $p_4$ | $a_3$ | 2 | $p_3$ | 2 |
| 3 | $a_1$ | $p_1$ | 2 | $p_2$ | 2 |
| 3 | $p_1$ | $p_1$ | 0 | $p_2$ | 2 |
| 3 | $p_2$ | $p_1$ | 2 | $p_2$ | 0 |
| 3 | $p_3$ | $p_1$ | 2 | $p_2$ | 2 |
| 3 | $c_1$ | $p_1$ | 1 | $p_2$ | 1 |
| 4 | $p_1$ | $p_1$ | 0 | $p_3$ | 2 |
| 4 | $p_2$ | $p_1$ | 2 | $p_3$ | 2 |
| 4 | $p_3$ | $p_1$ | 2 | $p_3$ | 0 |
| 4 | $c_2$ | $p_1$ | 1 | $p_3$ | 1 |

**Table 1: Distinct Core** ($K = \{Michelle, XML\}$, Dmax $= 2$)

Here, the distinct core consists of $p_1$ and $p_3$, where $p_1$ contains keyword $Michelle$ ($k_1$) and $p_3$ contains keyword $XML$ ($k_2$), 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, $p_2.c_1.p_1$ and $p_2.c_3.p_3$. 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 $K = \{Michelle, XML\}$ and Dmax $= 2$, the rightmost result in Fig. 3(b) is obtained as follows.

| TID | tid1 | dis1 | tid2 | dis2 |
|---|---|---|---|---|
| $p_4$ | $a_3$ | 2 | $p_2$ | 2 |
| $p_4$ | $a_3$ | 2 | $p_3$ | 2 |

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 $Michelle$ ($k_1$) and $p_2$ contains keyword $XML$ ($k_2$). 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, $K = \{Michelle, XML\}$ 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 Naive Algorithms

In this section, we first explain our naive algorithms using an example followed by discussions on the naive algorithms in detail. Fig. 6 outlines the two main steps for processing the distinct core/root 2-keyword query, $K = \{Michelle, XML\}$, with Dmax $= 2$ against the simple *DBLP* database. Its schema graph, $G_S$, 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

| Gid | TID | tid1 | dis1 | tid2 | dis2 |
|---|---|---|---|---|---|
| 1 | $w_4$ | $a_3$ | 1 | $p_2$ | 1 |
| 2 | $w_6$ | $a_3$ | 1 | $p_3$ | 1 |
| 3 | $c_1$ | $p_1$ | 1 | $p_2$ | 1 |
| 4 | $c_2$ | $p_1$ | 1 | $p_3$ | 1 |
| 5 | $a_3$ | $a_3$ | 0 | $p_2$ | 2 |
| 5 | $a_3$ | $a_3$ | 0 | $p_3$ | 2 |
| 6 | $p_1$ | $p_1$ | 0 | $p_2$ | 2 |
| 6 | $p_1$ | $p_1$ | 0 | $p_3$ | 2 |
| 7 | $p_2$ | $a_3$ | 2 | $p_2$ | 0 |
| 7 | $p_2$ | $a_3$ | 2 | $p_3$ | 2 |
| 7 | $p_2$ | $p_1$ | 2 | $p_2$ | 0 |
| 7 | $p_2$ | $p_1$ | 2 | $p_3$ | 2 |
| 8 | $p_3$ | $a_3$ | 2 | $p_3$ | 0 |
| 8 | $p_3$ | $a_3$ | 2 | $p_2$ | 2 |
| 8 | $p_3$ | $p_1$ | 2 | $p_2$ | 2 |
| 8 | $p_3$ | $p_1$ | 2 | $p_3$ | 0 |
| 9 | $a_1$ | $p_1$ | 2 | $p_2$ | 2 |
| 10 | $p_4$ | $a_3$ | 2 | $p_2$ | 2 |
| 10 | $p_4$ | $a_3$ | 2 | $p_3$ | 2 |

**Table 2: Distinct Root** ($K = \{Michelle, XML\}$, $\mathsf{Dmax} = 2$)

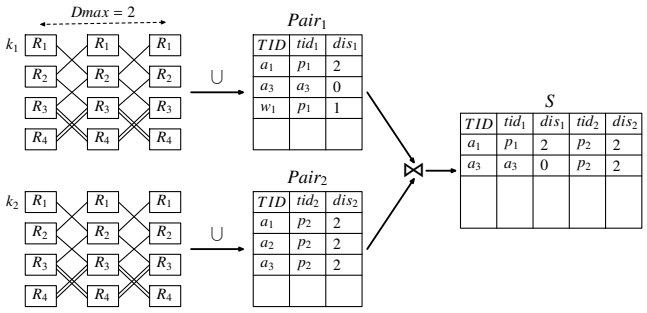



**Figure 6: An Overview** ($R_1$, $R_2$, $R_3$, and $R_4$ represent `Author`, `Write`, `Paper`, and `Cite` relations in Example 2.1)

$k_1 = Michelle$ and $k_2 = XML$, using projects, joins, unions, and group-by; and the right side is to join $Pair_1$ and $Pair_2$ to compute the $S$ relation (Eq. (4)).

Let $R_1$, $R_2$, $R_3$, and $R_4$ represent `Author`, `Write`, `Paper`, and `Cite` relations. The $Pair_1$ for the keyword $k_1$ is produced in the following steps.

$$\begin{aligned} P_{0,1} &\leftarrow \Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(\sigma_{contain(k_1)}R_1) \\ P_{0,2} &\leftarrow \Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(\sigma_{contain(k_1)}R_2) \\ P_{0,3} &\leftarrow \Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(\sigma_{contain(k_1)}R_3) \\ P_{0,4} &\leftarrow \Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(\sigma_{contain(k_1)}R_4) \end{aligned} \quad (5)$$

Here $\sigma_{contain(k_1)}R_j$ selects the tuples in $R_j$ that contain the keyword $k_1$. Let $R'_j \leftarrow \sigma_{contain(k_1)}R_j$, $\Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(R'_j)$ is to project tuples from $R'_j$ with all attributes ($*$) by further adding two attributes (renaming the attribute $TID$ to be $tid_1$, and adding a new attribute $dis_1$ with an initial value zero (supported in SQL)). For example, $\Pi_{TID\rightarrow tid_1,0\rightarrow dis_1,*}(\sigma_{contain(k_1)}R_1)$ is translated into the following SQL.

**select** TID **as** $tid_1$, 0 **as** $dis_1$, TID, Name
**from** Author **as** $R_1$ **where** contain(Title, Michelle)

The meaning of the temporal relation $P_{0,1}(tid_1, dis_1, TID, Name)$ is a set of $R_1$ relation tuples (identified by $TID$) that are in distance of $dis_1 = 0$ from the tuples (identified by $tid_1$) contain keyword $k_1 = Michelle$. It applies to other $P_{0,j}$ temporal relations. Following $P_{0,j}$ computed, $1 \le j \le 4$, we compute $P_{1,j}$ and then $P_{2,j}$ to obtain $R_j$ relation tuples that are in distance of 1 and distance of 2 from the tuples contain keyword $k_1 = Michelle$ ($\mathsf{Dmax} = 2$). Note that relation $P_{d,j}$ contains the set of tuples of $R_j$ that are in distance of $d$ from a tuple containing a certain keyword, and its two attributes, $tid_l$ and $dis_l$, explicitly indicate that it is about keyword $k_l$. The details of computing $P_{1,j}$ for $R_j$, $1 \le j \le 4$, are given below.

$$\begin{aligned} P_{1,1} &\leftarrow \Pi_{P_{0,2}.TID\rightarrow tid_1,1\rightarrow dis_1,R_1.*}(P_{0,2} \underset{P_{0,2}.AID=R_1.TID}{\bowtie} R_1) \\ P_{1,2} &\leftarrow \Pi_{P_{0,1}.TID\rightarrow tid_1,1\rightarrow dis_1,R_2.*}(P_{0,1} \underset{P_{0,1}.TID=R_2.AID}{\bowtie} R_2) \cup \\ &\quad \Pi_{P_{0,3}.TID\rightarrow tid_1,1\rightarrow dis_1,R_2.*}(P_{0,3} \underset{P_{0,3}.TID=R_2.PID}{\bowtie} R_2) \\ P_{1,3} &\leftarrow \Pi_{P_{0,2}.TID\rightarrow tid_1,1\rightarrow dis_1,R_3.*}(P_{0,2} \underset{P_{0,2}.PID=R_3.TID}{\bowtie} R_3) \cup \\ &\quad \Pi_{P_{0,4}.TID\rightarrow tid_1,1\rightarrow dis_1,R_3.*}(P_{0,4} \underset{P_{0,4}.PID1=R_3.TID}{\bowtie} R_3) \cup \\ &\quad \Pi_{P_{0,4}.TID\rightarrow tid_1,1\rightarrow dis_1,R_3.*}(P_{0,4} \underset{P_{0,4}.PID2=R_3.TID}{\bowtie} R_3) \\ P_{1,4} &\leftarrow \Pi_{P_{0,3}.TID\rightarrow tid_1,1\rightarrow dis_1,R_4.*}(P_{0,3} \underset{P_{0,3}.TID=R_4.PID1}{\bowtie} R_4) \cup \\ &\quad \Pi_{P_{0,3}.TID\rightarrow tid_1,1\rightarrow dis_1,R_4.*}(P_{0,3} \underset{P_{0,3}.TID=R_4.PID2}{\bowtie} R_4) \end{aligned} \quad (6)$$

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 $P_{d-1,i}$ if there is an edge between $R_j$ and $R_i$ in $G_S$. Consider $P_{1,3}$ for $R_3$, it computes $P_{1,3}$ by union of three joins ($P_{0,2} \bowtie R_3 \cup P_{0,4} \bowtie R_3 \cup P_{0,4} \bowtie R_3$), 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 $k_l$ can be computed. Continue the example, to compute $P_{2,j}$ for $R_j$, $1 \le j \le 4$, regarding keyword $k_1$, 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 $\mathsf{Dmax}$ times.

Suppose that we have computed $P_{d,j}$ for $0 \le d \le \mathsf{Dmax}$ and $1 \le j \le 4$, regarding keyword $k_1 = Michelle$. We further compute the shortest distance between a $R_j$ 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 $R_j$ tuple to a tuple containing keyword $k_1$ using group-by ($\Gamma$) and SQL aggregate function min.

$$G_j \leftarrow_{TID,tid_1} \Gamma_{\min(dis_1)}(P_{0,j} \cup P_{1,j} \cup P_{2,j}) \quad (7)$$

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

$$Pair_1 \leftarrow G_1 \cup G_2 \cup G_3 \cup G_4 \quad (8)$$

Here, $Pair_1$ records all tuples that are in the shortest distance of a tuple containing keyword $k_1$, within $\mathsf{Dmax}$. Note that $G_i \cap G_j = \emptyset$, because $G_i$ and $G_j$ are tuples identified with TIDs from $R_i$ and $R_j$ relations and TIDs are unique in database as assumed. We can compute $Pair_2$ for keyword $k_2 = XML$ 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** min: Consider Eq. (7), the group-by $\Gamma$ can be computed by virtually pushing $\Gamma$. Recall that all $P_{d,j}$ relations, for $1 \le d \le \mathsf{Dmax}$, have the same schema, and $P_{d,j}$ maintains $R_j$ 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*$(G_S, k_i, \mathsf{Dmax}, R_1, \cdots, R_n)$

**Input:** Schema $G_S$, keyword $k_i$, Dmax, $n$ relations $R_1, \cdots, R_n$.
**Output:** $Pair_i$ with 3 attributes: $TID$, $tid_i$, $dis_i$.

1: **for** $j = 1$ **to** $n$ **do**
2: $\quad P_{0,j} \leftarrow \prod_{R_j.TID \rightarrow tid_i,\ 0 \rightarrow dis_i,\ R_j.*}(\sigma_{contain(k_i)} R_j)$;
3: $\quad G_j \leftarrow \prod_{tid_i, dis_i, TID}(P_{0,j})$;
4: **for** $d = 1$ **to** Dmax **do**
5: $\quad$ **for** $j = 1$ **to** $n$ **do**
6: $\quad\quad P_{d,j} \leftarrow \emptyset$;
7: $\quad\quad$ **for all** $(R_j, R_l) \in E(G_S) \vee (R_l, R_j) \in E(G_S)$ **do**
8: $\quad\quad\quad \Delta \leftarrow \prod_{P_{d-1,l}.TID \rightarrow tid_i,\ d \rightarrow dis_i,\ R_j.*}(P_{d-1,l} \bowtie R_j)$;
9: $\quad\quad\quad \Delta \leftarrow \sigma_{(tid_i, TID) \notin \prod_{tid_i, TID}(G_j)}(\Delta)$;
10: $\quad\quad\quad P_{d,j} \leftarrow P_{d,j} \cup \Delta$;
11: $\quad\quad\quad G_j \leftarrow G_j \cup \prod_{tid_i, dis_i, TID}(\Delta)$;
12: $Pair_i \leftarrow G_1 \cup G_2 \cup \cdots \cup G_n$;
13: **return** $Pair_i$;

**Algorithm 2** *DC-Naive*$(R_1, \cdots, R_n, G_S, K, \mathsf{Dmax})$

**Input**: $n$ relations $R_1, R_2, \cdots, R_n$, schema graph $G_S$, and $m$-keyword, $K = \{k_1, k_2, \cdots, k_m\}$, and radius Dmax.
**Output**: Relation with $2m + 1$ attributes named $TID$, $tid_1$, $dis_1$, $\cdots$, $tid_m$, $dis_m$.

1: **for** $i = 1$ **to** $m$ **do**
2: $\quad Pair_i \leftarrow$ *Pair*$(G_S, k_i, \mathsf{Dmax}, R_1, \cdots, R_n)$;
3: $S \leftarrow Pair_1 \bowtie Pair_2 \bowtie \cdots \bowtie Pair_m$;
4: Sort $S$ by $tid_1, tid_2, \cdots, tid_m$;
5: **return** $S$;

computed as follows.

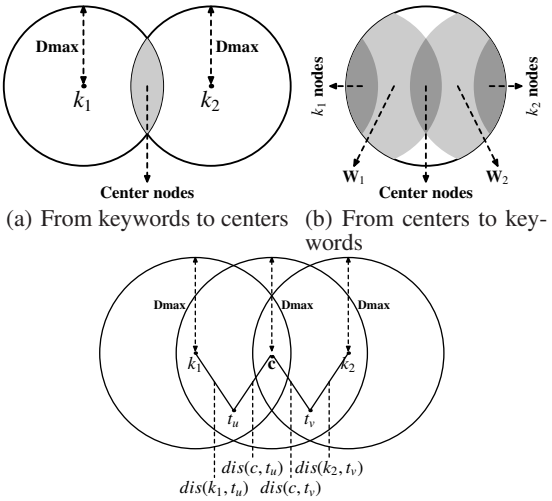
$$\begin{aligned} G_j &\leftarrow P_{0,j} \\ G_j &\leftarrow G_j \cup (\sigma_{(tid_1, TID) \notin \prod_{tid_1, TID}(G_j)} P_{1,j}) \\ G_j &\leftarrow G_j \cup (\sigma_{(tid_1, TID) \notin \prod_{tid_1, TID}(G_j)} P_{2,j}) \end{aligned} \quad (9)$$

Here, $\sigma_{(tid_1, TID) \notin \prod_{tid_1, TID}(G_j)} P_{2,j}$ means to select $P_{2,j}$ tuples where their $(tid_1, TID)$ does not appear in $G_j$ already, in other words, there does not exist a shortest path between $tid_1$ and $TID$ before.

**Rule-2**: If there exists a shortest path between $tid_i$ and $TID$ value pair, say, $dis_i(tid_i, TID) = d'$, then there is no need to compute any tuple connections between the $tid_i$ and $TID$ pair, because all those will be removed later by group-by and SQL aggregate function min. In Eq. (6), every $P_{1,j}$, $1 \le j \le 4$, can be further reduced as $P_{1,j} \leftarrow \sigma_{(tid_1, TID) \notin \prod_{tid_1, TID}(P_{0,j})} P_{1,j}$.

The algorithm *Pair*() is given in Algorithm 1, which computes $Pair_i$ for keyword $k_i$. It first computes all the initial $P_{0,j}$ 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 $1 \le d \le$ Dmax and every relation $R_j$, $1 \le j \le n$, 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 $G_S$, referencing to Eq. (6) and Eq. (9), using the two rules, Rule-1 and Rule-2. In our example, to compute $Pair_1$, it calls *Pair*$(G_S, k_1, \mathsf{Dmax}, R_1, R_2, R_3, R_4)$, where $k_1 = Michelle$, Dmax $= 2$, and the 4 relations $R_j$, $1 \le j \le 4$.

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: $X \leftarrow {}_{TID}\Gamma_{\min(dis_1) \rightarrow dis_1, \cdots, \min(dis_m) \rightarrow dis_m} S$, and $S \leftarrow {}_{TID, dis_1, \cdots, dis_m}\Gamma_{\min(tid_1) \rightarrow tid_1, \cdots, \min(tid_m) \rightarrow tid_m}(S \bowtie X)$.



(a) From keywords to centers (b) From centers to keywords
(c) Project Relations

**Figure 7: Three-Phase Reduction**

## 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, $K = \{k_1, k_2, \cdots, k_m\}$, with a user-given Dmax, against an $RDB$ with a schema graph $G_S$.

**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 $\mathbf{G}_i$ be the set of tuples in $RDB$ that can reach at least a tuple containing keyword $k_i$ within Dmax, for $1 \le i \le m$. Based on all $\mathbf{G}_i$, we can compute $\mathbf{Y} = \mathbf{G}_1 \bowtie \mathbf{G}_2 \bowtie \cdots \bowtie \mathbf{G}_m$ which is the set of center-nodes that can reach every keyword-node $k_i$, $1 \le i \le m$, within Dmax. $\mathbf{Y}$ 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 $\mathbf{Y}$.

**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 $\mathbf{Y}$ within Dmax. We compute all tuples that are reachable from $\mathbf{Y}$ within Dmax. Let $\mathbf{W}_i$ be the set of tuples in $\mathbf{G}_i$ that can be reached from a center in $\mathbf{Y}$ within Dmax, for $1 \le i \le m$. Note that $\mathbf{W}_i \subseteq \mathbf{G}_i$. When $m = 2$, $\mathbf{W}_1$ and $\mathbf{W}_2$ 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 $\mathbf{G}_i \bowtie \mathbf{W}_i$, for $1 \le i \le m$. Consider a tuple in $\mathbf{G}_i$ which contains a TID $t'$ with a distance to the virtual keyword-node $k_i$, denoted as $dis(t', k_i)$, and consider a tuple in $\mathbf{W}_i$ which contains a TID $t'$ with a distance to the virtual center-node $c$, denoted as $dis(t', c)$. If $dis(t', k_i) + dis(t', c) \le$ Dmax,

**Algorithm 3** *DC*($R_1, R_2, \cdots, R_n, G_S, K, \mathsf{Dmax}$)

**Input**: $n$ relations $R_1, R_2, \cdots, R_n$, with schema graph $G_S$, and an $m$-keyword query, $K = \{k_1, k_2, \cdots, k_m\}$, and radius Dmax.
**Output**: Relation with $2 \cdot m + 1$ attributes named TID, $tid_1, dis_1, \cdots, tid_m, dis_m$.

1: **for** $i = 1$ **to** $m$ **do**
2: $\{G_{1,i}, \cdots, G_{n,i}\} \leftarrow$ *PairRoot*($G_S, k_i, \mathsf{Dmax}, R_1, \cdots, R_n, \sigma_{contain(k_i)}R_1, \cdots, \sigma_{contain(k_i)}R_n$);
3: **for** $j = 1$ **to** $n$ **do**
4: $R_{j,i} \leftarrow R_j \ltimes G_{j,i}$;
5: **for** $j = 1$ **to** $n$ **do**
6: $Y_j \leftarrow G_{j,1} \bowtie G_{j,2} \bowtie \cdots \bowtie G_{j,m}$;
7: $X_j \leftarrow R_j \ltimes Y_j$;
8: **for** $i = 1$ **to** $m$ **do**
9: $\{W_{1,i}, \cdots, W_{n,i}\} \leftarrow$ *PairRoot*($G_S, k_i, \mathsf{Dmax}, R_{1,i}, \cdots, R_{n,i}, X_1, \cdots, X_n$);
10: **for** $j = 1$ **to** $n$ **do**
11: $Path_{j,i} \leftarrow G_{j,i} \ltimes_{G_{j,i}.TID=W_{j,i}.TID} W_{j,i}$;
12: $Path_{j,i} \leftarrow \prod_{TID,\ G_{j,i}.dis_i \rightarrow d_{k_i}, W_{j,i}.dis_i \rightarrow d_r}(Path_{j,i})$;
13: $Path_{j,i} \leftarrow \sigma_{d_{k_i}+d_r \leq \mathsf{Dmax}}(Path_{j,i})$;
14: $R'_{j,i} \leftarrow R_{j,i} \ltimes Path_{j,i}$;
15: **for** $i = 1$ **to** $m$ **do**
16: $Pair_i \leftarrow$ *Pair*($R'_{1,i}, R'_{2,i}, \cdots, R'_{n,i}, G_S, k_i, \mathsf{Dmax}$);
17: $S \leftarrow Pair_1 \bowtie Pair_2 \bowtie \cdots \bowtie Pair_m$;
18: Sort $S$ by $tid_1, tid_2, \cdots, tid_m$;
19: **return** $S$;

**Algorithm 4** *PairRoot*($G_S, k_i, \mathsf{Dmax}, R_1, \cdots, R_n, I_1, \cdots, I_n$)

**Input:** Schema graph $G_S$, keyword $k_i$, Dmax, $n$ relations $R_1, R_2, \cdots, R_n$, and $n$ initial relations $I_1, I_2, \cdots, I_n$.
**Output:** n relations $G_{1,i}, \cdots, G_{n,i}$ each has 3 attributes: TID, $tid_i, dis_i$.

1: **for** $j = 1$ **to** $n$ **do**
2: $P_{0,j} \leftarrow \prod_{I_j.TID \rightarrow tid_i, 0 \rightarrow dis_i, I_j.*}(I_j)$;
3: $G_{j,i} \leftarrow \prod_{tid_i, dis_i, TID}(P_{0,j})$;
4: **for** $d = 1$ **to** Dmax **do**
5: **for** $j = 1$ **to** $n$ **do**
6: $P_{d,j} \leftarrow \emptyset$;
7: **for all** $(R_j, R_l) \in E(G_S) \vee (R_l, R_j) \in E(G_S)$ **do**
8: $\Delta \leftarrow \prod_{P_{d-1,l}.TID \rightarrow tid_i, d \rightarrow dis_i, R_j.*}(P_{d-1,l} \bowtie R_j)$;
9: $\Delta \leftarrow_{R_j.*} \Gamma_{\min(tid_i), \min(dis_i)}(\Delta)$;
10: $\Delta \leftarrow \sigma_{TID \notin \prod_{TID}(G_{j,i})}(\Delta)$;
11: $P_{d,j} \leftarrow P_{d,j} \cup \Delta$;
12: $G_{j,i} \leftarrow G_{j,i} \cup \prod_{tid_i, dis_i, TID}(\Delta)$;
13: **return** $\{G_{1,i}, \cdots, G_{n,i}\}$;

the tuple $t'$ will be projected from the $RDB$. Here, both $dis(t', c)$ and $dis(t', k_i)$ are in the range of $[0, \mathsf{Dmax}]$. In this phase, all such tuples, $t'$, will be projected which are sufficient to compute all multi-center communities, because the set of such tuples contain every keyword-tuple, center-tuple, and path-tuple to compute all communities. It is illustrated in Fig. 7(c), when $m = 2$. We will prove the correctness of the three-phase reduction later in this paper.

The new *DC*() algorithm to compute communities under distinct core semantics is given in Algorithm 3. Suppose that there are $n$ relations in an $RDB$ for an $m$-keyword query. The first reduction phase is in line 1-7. The second/third reduction phases are done in a for-loop (line 8-14) in which the second reduction phase is line 9 and the third reduction phase is in line 10-14. Line 15-17 are similar as done in *DC-Naive*() to compute communities using $Pair_i$, $1 \leq i \leq m$, and $S$ relation. For the first reduction, it computes $G_{j,i}$ for every keyword $k_i$ and every relation $R_j$ separately by calling a procedure *PairRoot*() (Algorithm 4). *PairRoot*() is designed in the similar fashion like *Pair*(). The main difference is that *PairRoot*() computes tuples, $t$, that are in shortest distance to a virtual node (keyword or center) within Dmax. Take keyword-nodes as an example. The shortest distance to a tuple containing a keyword is more important than which tuple that contains a keyword. Therefore we only maintain the shortest distance (line 9 in Algorithm 4). *PairRoot*() returns a collection of $G_{j,i}$, for a given keyword $k_i$, for $1 \leq j \leq n$. Note that $\mathbf{G}_i = \bigcup_{j=1}^{n} G_{j,i}$. In line 3-4, it projects $R_j$ using semijoin $R_{j,i} \leftarrow R_j \ltimes G_{j,i}$. Here, $R_{j,i}$ ($\subseteq R_j$) is a set of tuples that are within Dmax from a virtual keyword-node $k_i$. Note that $\mathbf{Y} = \bigcup_{j=1}^{n} Y_j$. $X_j$ ($\subseteq R_j$) is a set of centers in relation $R_j$ (line 7). In line 9, starting from all center nodes $(X_1, \cdots, X_n)$, it computes $W_{j,i}$, for keyword $k_i$, for $1 \leq j \leq n$. Note that $\mathbf{W}_i = \bigcup_{j=1}^{n} W_{j,i}$. In line 10-14, it further projects $R'_{j,i}$ out of $R_{j,i}$, for a keyword $k_i$, for $1 \leq j \leq n$. In line 16, it computes $Pair_i$, using the projected relations, $R'_{1,i}, R'_{2,i}, \cdots, R'_{n,i}$. Below, we prove the correctness of Algorithm 3.

**Lemma 5.1:** *Suppose* $\mathbf{Y} = \bigcup_{j=1}^{n} \Pi_{TID}(Y_j)$*, then* $\mathbf{Y}$ *is the set of center-tuples in all communities.* □

**Proof Sketch:** First, we show if $y \in \mathbf{Y}$ then $y$ is a center-tuple in a community. Suppose $y \in \mathbf{Y}$ is in relation $R_j$, in line 6 of Algorithm 3, we know that $y \in \Pi_{TID}(G_{j,i})$ for all $1 \leq i \leq m$, where $\Pi_{TID}(G_{j,i})$ 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 $dis(u_i, y) \leq \mathsf{Dmax}$ 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 $[u_1, u_2, \cdots, u_m]$. Next, we show if $y$ is a center-tuple in a community then $y \in \mathbf{Y}$. Suppose $y$ is in relation $R_j$, and is a center-tuple in a community with the core of $[u_1, u_2, \cdots, u_m]$. By definition, $dis(u_i, y) \leq \mathsf{Dmax}$, for any $1 \leq i \leq m$. So $y \in \Pi_{TID}(G_{j,i})$. In line 6, we know $y \in \Pi_{TID}(Y_j)$, then $y \in \mathbf{Y}$. □

**Lemma 5.2:** *Suppose* $\mathbf{Y}$ *is the set of center-tuples in all communities, i.e.,* $\mathbf{Y} = \bigcup_{j=1}^{n} \Pi_{TID}(Y_j)$*. For any tuple* $u$ *that contain a keyword, if there exists a tuple* $y \in \mathbf{Y}$ *with* $dis(u, y) \leq \mathsf{Dmax}$*, 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 $k_1$. As proved in Lemma 5.1, if $y \in \mathbf{Y}$, there exists a community with the core of $[u_1, u_2, \cdots, u_m]$ that includes $y$ as a center-tuple. As $dis(u, y) \leq \mathsf{Dmax}$, $[u, u_2, \cdots, u_m]$ 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* $\leq \mathsf{Dmax}$ *from a keyword-tuple to a center-tuple in a community, then* $t \notin \bigcup_{1 \leq j \leq n, 1 \leq i \leq m} \Pi_{TID}(R'_{j,i})$*, where* $R'_{j,i}$ *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 $t \in \Pi_{TID}(R'_{j,i})$, from line 14 we get $t \in \Pi_{TID}(Path_{j,i})$, and thus in line 13 we have $t.d_{k_i} + t.d_r \leq \mathsf{Dmax}$, where $t.d_{k_i}$ is the minimal distance from tuple $t$ to any tuple that contains keyword $k_i$, and $t.d_r$ 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 $dis(t, u) = t.d_{k_i}$ and $dis(t, y) = t.d_r$. This implies a path $u \dashrightarrow t \dashrightarrow y$ of length $\leq \mathsf{Dmax}$. 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 $\leq \mathsf{Dmax}$ from a keyword-tuple to a center-tuple in a community. □

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 [$u_1$,$u_2$,$\cdots$,$u_m$] and the centers are $\{x_1, x_2, \cdots, x_r\}$, then for any $1 \leq k \leq r$, ($x_k$,$u_1$,$u_2$, $\cdots$, $u_m$) $\in \Pi_{TID,tid_1,tid_2,\cdots,tid_m}(S)$.* □

**Proof Sketch:** As proved in Lemma 5.1, since $x_k$ is a center-tuple in community, $C$, we have $x_k \in \bigcup_{j=1}^{n} \Pi_{TID}(X_j)$. Also, in line 9, we get all tuples $w$ that satisfy (1) $dis(x_k, w) \leq$ Dmax, and (2) the minimal distance from $w$ to any tuple that contains a keyword is $\leq$ Dmax. The condition (2) is satisfied in line 1-3. Suppose for any $1 \leq i \leq m$, let $P_i$ contain $u_i$ and the tuples on any shortest path from $u_i$ to $x_k$, then all tuples in $P_i$ satisfy the conditions (1) and (2). Furthermore, for every tuple $q$ in $P_i$, it also satisfies $q.d_{k_i} + q.d_r \leq$ Dmax, so $P_i \subseteq \bigcup_{j=1}^{n} \Pi_{TID}(R'_{j,i})$ (line 13-14). Then in line 16, we have $(u_i, x_k) \in \Pi_{tid_i,TID}(Pair_i)$, because there exists a path in $P_i$ of length $\leq$ Dmax that connects $u_i$ and $x_k$. In consequence, for any $1 \leq i \leq m$, $(u_i, x_k) \in \Pi_{tid_i,TID}(Pair_i)$, and in line 17, ($x_k$,$u_1$,$u_2$, $\cdots$, $u_m$) $\in \Pi_{TID,tid_1,tid_2,\ldots,tid_m}(S)$. □

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 $\Pi_{TID,tid_1,tid_2}(S)$.

The new algorithm *DR*() to compute distinct roots is given in Algorithm 5.

**Algorithm 5** *DR*$(R_1, R_2, \cdots, R_n, G_S, K,$ Dmax$)$

**Input**: $n$ relations $R_1, R_2, \cdots, R_n$, with schema graph $G_S$, and an $m$-keyword query, $K = \{k_1, k_2, \cdots, k_m\}$, and radius Dmax.
**Output**: Relation with $2 \cdot m + 1$ attributes named TID, $tid_1, dis_1, \cdots, tid_m, dis_m$.

1: **for** $i = 1$ **to** $m$ **do**
2: $\quad \{G_{1,i}, \cdots, G_{n,i}\} \leftarrow$ *PairRoot*$(G_S, k_i,$ Dmax$, R_1, \cdots, R_n, \sigma_{contain(k_i)}R_1, \cdots, \sigma_{contain(k_i)}R_n)$;
3: **for** $j = 1$ **to** $n$ **do**
4: $\quad S_j \leftarrow G_{j,1} \bowtie G_{j,2} \bowtie \cdots \bowtie G_{j,m}$;
5: $S \leftarrow S_1 \cup S_2 \cup \cdots \cup S_n$;
6: **return** $S$;

# 6. RELATED WORK

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

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

Among *CN*-based approaches, in *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. *DISCOVER-II* is built in a middleware on top of an RDBMS and issues SQL queries to access data. In *SPARK* [23], Luo et al. proposed a new ranking function by treating each connected tree as a virtual document, and unified the AND/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 *SPARK* were demonstrated to outperform *DISCOVER-II*. Both work [14, 23] focused on finding top-$k$ *MTJNT*s, whose sizes are controlled by Tmax, that contain all or some of the $m$ keywords (AND/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 $RDB$ 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 *BANKS-I* [4], and Kacholia et al. proposed bidirectional search in *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. in [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 *BLINKS* [13], He et al. proposed the distinct root semantics. Note that *BLINKS* deals with a general weighted graph whereas we deal with an unweighted graph in this work. In *BLINKS*, search strategies were proposed with a bi-level index built to fast compute the shortest distances. *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 RDBMS. In our approach, we can effectively use the RDBMS functions to handle large scale data without materializing an $RDB$ as a graph.

Li et al. in *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, *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, *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 RDBMSs, 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]. Semijoin was studied in [3]. Bernstein et al. in [3] proposed a linear-time

| Parameter | Range | Default |
|---|---|---|
| $m$ | 2, 3, 4, 5 | 3 |
| Tmax | 3, 4, 5, 6, 7 | 5 |
| Dmax | 2, 3, 4, 5, 6 | 4 |
| $ksel$ | $4E$-4, $8E$-4, $1.2E$-3, $1.6E$-3, $2E$-3 | $1.2E$-3 |
| $c$ | 0, 2, 4, 6 | - |

**Table 3: Parameters for DBLP Testing**

| Semantics | ksel | Keywords |
|---|---|---|
| Tree | $4E$-4 | theoretic uniform probability |
| | $8E$-4 | flexible task document |
| | $1.2E$-3 | vector extraction technique power average |
| | $1.6E$-3 | multimedia technology protocol |
| | $2E$-3 | fast memory theory |
| Core/Root | $4E$-4 | XML minimal struct |
| | $8E$-4 | machine area ipod |
| | $1.2E$-3 | scheme iterator flexible queries simple |
| | $1.6E$-3 | agent group lateness |
| | $2E$-3 | graphs coding mistake |

**Table 4: Keywords and $ksel$ Used in DBLP**

algorithm to check whether a full reduction of the original database can be obtained using a sequence of semijoins, and output the semijoins if exists. Semijoin was also used to reduce the processing load of regular joins, in order to avoid creating large number of intermediate results [29]. Chaudhuri and Shim studied pushing group-by in [7], which is to optimize SQL queries that contain group-by. In Précis, Kasneci et al. in [20], and Simitsis et al. in [28] investigated the issue of returning a multi-relation database, which is a subset of the original database, as the answer of keyword queries, the results consists of tuples that directly or implicitly related to the given keywords. Three-phase reductions were not studied in [20, 28]. Multi-queries optimization were also studied in [27, 31, 18] by sharing common subexpressions among queries or inside a query.

# 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. The total number of tuples in *IMDB* used is 15,174,080.

All algorithms were implemented in Java using JDK 1.5 and JDBC to connect to RDBMSs. 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

[2]http://www.informatik.uni-trier.de/~ley/db

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

| Semantics | c | Keywords |
|---|---|---|
| Tree | 0 | global simple generation |
| | 2 | business binary selection |
| | 4 | output mobility sensor |
| Core/Root | 0 | demand pause scheme |
| | 2 | impact Jim access |
| | 4 | intersect report decision |
| | 6 | heuristic submit multimedia |

**Table 5: Keywords and $c$ Used in DBLP**

| Parameter | Range | Default |
|---|---|---|
| $m$ | 2, 3, 4, 5 | 3 |
| Tmax | 3, 4, 5, 6, 7 | 5 |
| Dmax | 1, 2, 3, 4 | 3 |
| $ksel$ | $2.5E$-5, $5E$-5, $7.5E$-5, $1E$-4, $1.25E$-4 | $7.5E$-5 |
| $c$ | 0, 1, 2, 3, 4 | - |

**Table 6: Parameters for IMDB Testing**

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 SQL (BP algorithm in [23] was implemented on *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, Tmax is used in the connected tree algorithms to control the number of nodes in a connected tree, and Dmax is used in the distinct root/core algorithms to specify the radius. In addition to $m$ and Tmax/Dmax, we use another two parameters $ksel$ (keyword selectivity) and $c$ (query compactness).

A $ksel$ 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 $ksel$ values as follows. We choose the max $ksel$ 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 $ksel$ below 2E-3, which covers more than 99.9% of the total number of keywords. We choose 2E-3 to be the max $ksel$ value. Let the max $ksel$ value be $\alpha$. We divide the $ksel$ 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 $ksel$ value, $\alpha'$, for a given number of keywords, we select keywords such that the average $ksel$ 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 $ksel$ value for an $m$-keyword query is related to the selectivity of individual keywords in a query. In addition to $ksel$, 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: $c(K) = max_{1 \le i < j \le m}\{dis(k_i, k_j)\}$. Here, $dis(k_i, k_j)$ 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 Tmax/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 $ksel$ and $c$ are set for different purposes, we use either $ksel$ or $c$ in our testing, together with $m$, Tmax/Dmax.

The parameters and default values used for testing *IMDB* are shown in Table 6. All parameters have the same meanings as those

| Semantics | ksel | Keywords |
|---|---|---|
| Tree | $2.5E$-5 | wish forget memory |
| | $5E$-5 | highway drive volume |
| | $7.5E$-5 | human smart helper Simpson clown |
| | $1E$-4 | easy chance hat |
| | $1.25E$-4 | golden British diamond |
| Core/Root | $2.5E$-5 | style gentlemen rather |
| | $5E$-5 | fish grow formal |
| | $7.5E$-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**

| Semantics | c | Keywords |
|---|---|---|
| Tree | 0 | story school play |
| | 1 | beautiful Teddy king |
| | 2 | bold video manager |
| | 3 | school floor million |
| | 4 | remember natural Michelle |
| Core/Root | 0 | west Prussia arriving |
| | 1 | series pork dumpling |
| | 2 | May gather Merlin |
| | 3 | dance ability boil |
| | 4 | special equipment labour |

**Table 8: Keywords and $c$ Used in IMDB**

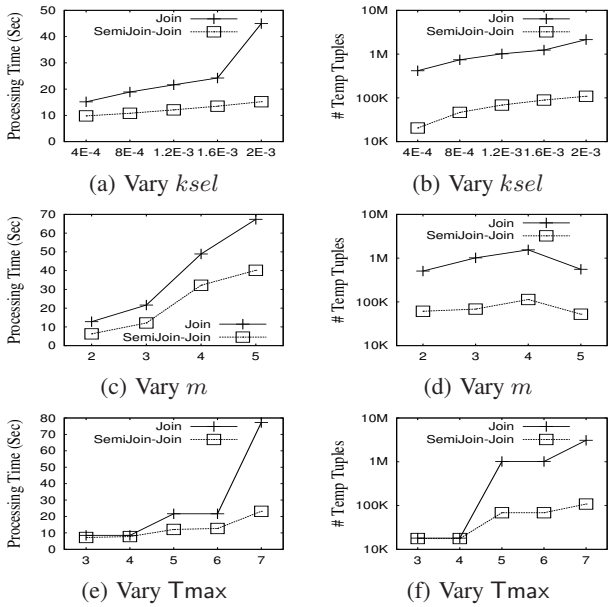
used for *DBLP*. Here, $ksel$ values are smaller than those used in *DBLP*, because *IMDB* has a smaller average keyword selectivity. The Dmax values are also smaller than those used in *DBLP* because the database graph ($G_D$) of *IMDB* is denser than that of *DBLP*. For each keyword selectivity, the 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 elapse 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 $m \leq 4$ but decrease when $m > 4$ for both Join and Semijoin-Join. This is because, for a fixed Tmax, when $m$ increases, the number of *CN*s 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 *CN*s 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



(a) Vary $ksel$ (b) Vary $ksel$

(c) Vary $m$ (d) Vary $m$

(e) Vary Tmax (f) Vary Tmax

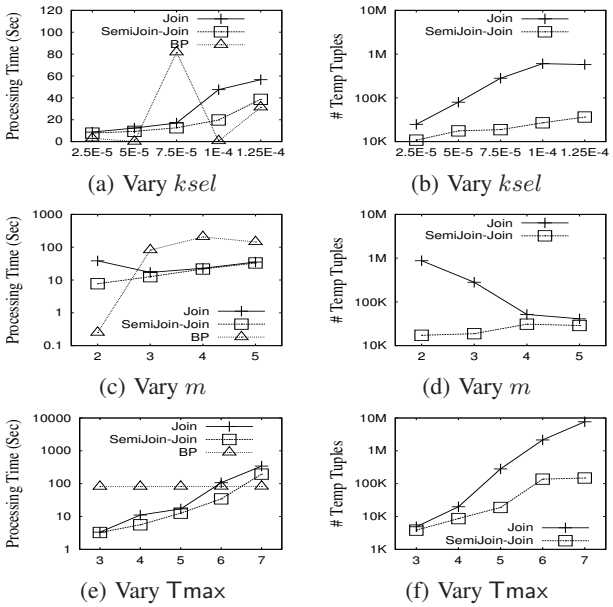
**Figure 8: Connected Tree (DBLP)**

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. $\leq 3$. 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 $ksel \geq 7.5E-5$. 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 Fig. 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 $\leq 2$, DC is slower than DC-Naive. This is because, when Dmax is small, the number of intermediate results for

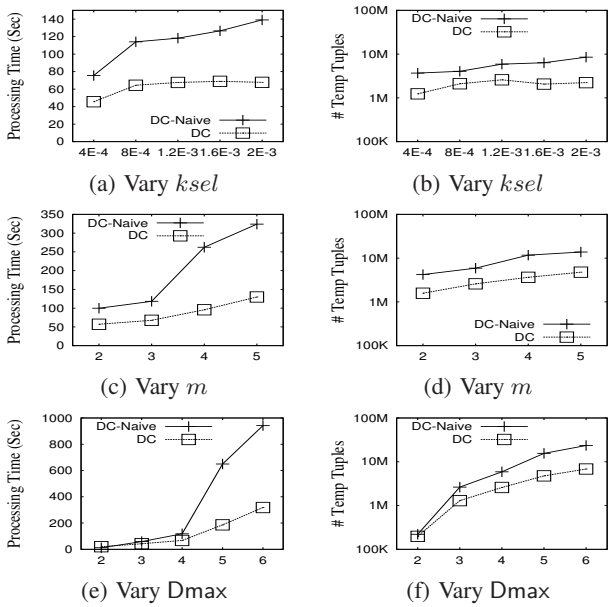



(a) Vary $ksel$ (b) Vary $ksel$

(c) Vary $m$ (d) Vary $m$

(e) Vary Tmax (f) Vary Tmax

**Figure 9: Connected Tree (IMDB)**




(a) Vary $ksel$ (b) Vary $ksel$

(c) Vary $m$ (d) Vary $m$

(e) Vary Dmax (f) Vary Dmax

**Figure 10: Distinct Core (DBLP)**

DC-Naive is not large. In such a case, the performance of more small joins is not as effective as the performance of joins.

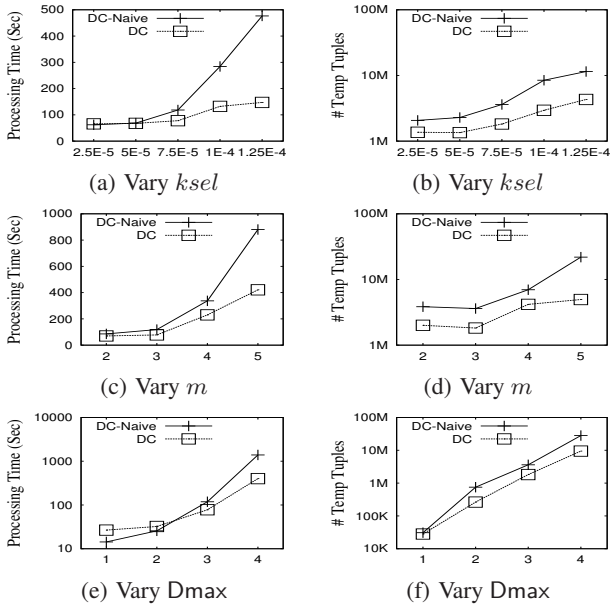
**Distinct Root**: The results under distinct root semantics are shown in Fig. 12. Fig. 12(a)-(c) show the time for *DBLP* and Fig. 12(d)-(f) show the time for *IMDB*. In all the cases, DR outperforms DR-Naive. In particular, when $m \geq 4$, the time of DR-Naive increases significantly while DR increases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




(a) Vary $ksel$ (b) Vary $ksel$

(c) Vary $m$ (d) Vary $m$

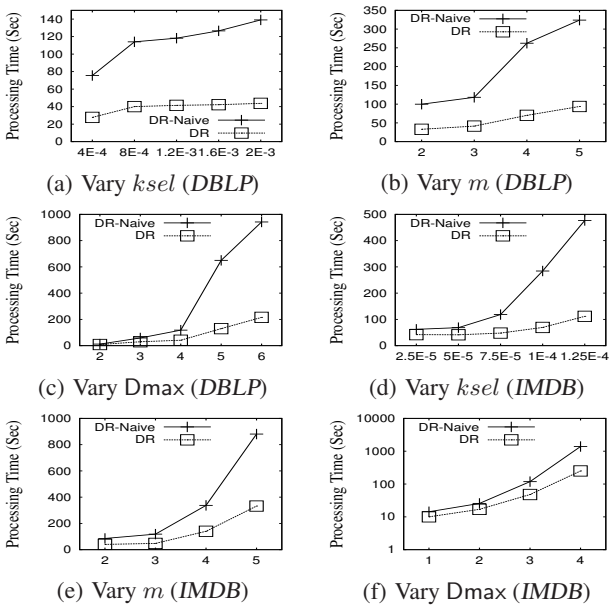
(e) Vary Dmax (f) Vary Dmax

**Figure 11: Distinct Core (IMDB)**

# 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 RDBMSs to support keyword queries in the same framework of RDBMSs.

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

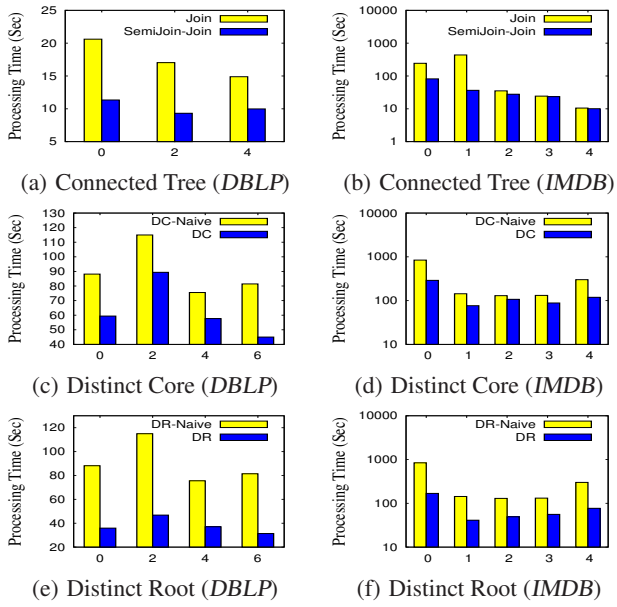



(a) Vary $ksel$ (*DBLP*) (b) Vary $m$ (*DBLP*)

(c) Vary Dmax (*DBLP*) (d) Vary $ksel$ (*IMDB*)

(e) Vary $m$ (*IMDB*) (f) Vary Dmax (*IMDB*)

**Figure 12: Distinct Root (DBLP and IMDB)**




(a) Connected Tree (*DBLP*) (b) Connected Tree (*IMDB*)

(c) Distinct Core (*DBLP*) (d) Distinct Core (*IMDB*)

(e) Distinct Root (*DBLP*) (f) Distinct Root (*IMDB*)

**Figure 13: Query Compactness**

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

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

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *Proc. of VLDB'04*, 2004.

[3] P. A. Bernstein and D.-M. W. Chiu. Using semi-joins to solve relational queries. *J. ACM*, 28(1), 1981.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE'02*, 2002.

[5] S. Chaudhuri. An overview of query optimization in relational systems. In *Proc. of PODS'98*, 1998.

[6] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *Proc. of CIDR*, 2005.

[7] S. Chaudhuri and K. Shim. Including group-by in query optimization. In *Proc. of VLDB'94*, 1994.

[8] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1), 2008.

[9] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *Proc. of ICDE'07*, 2007.

[10] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. In *Networks*, 1972.

[11] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *Proc. of SIGMOD'08*, 2008.

[12] G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2), 1993.

[13] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *Proc. of SIGMOD'07*, 2007.

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

[15] V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 33(1), 2008.

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

[17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB'05*, 2005.

[18] A. Kementsietsidis, F. Neven, D. V. de Craen, and S. Vansummeren. Scalable multi-query optimization for exploratory queries over federated scientific databases. *PVLDB*, 1(1), 2008.

[19] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proc. of PODS'06*, 2006.

[20] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Précis: The essence of a query answer. In *Proc. of ICDE'06*, 2006.

[21] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: Efficient and adaptive keyword search on unstructured, semi-structured and structured data. In *Proc. of SIGMOD'08*, 2008.

[22] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *Proc. of SIGMOD'06*, 2006.

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

[24] A. Markowetz, Y. Yang, and D. Papadias. Keyword search on relational data streams. In *Proc. of SIGMOD'07*, 2007.

[25] P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1), 1992.

[26] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *Proc. of ICDE'09*, 2009.

[27] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proc. of SIGMOD'00*, 2000.

[28] A. Simitsis, G. Koutrika, and Y. E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1), 2008.

[29] H. Yoo and S. Lafortune. An intelligent search method for query optimization by semijoins. *IEEE Transactions on Knowledge and Data Engineering*, 1(2), 1989.

[30] C. Yu and W. Meng. *Principles of Database Query Processing for Advanced Applications*. Morgan Kaufmann, 1998.

[31] J. Zhou, P.-A. Larson, J. C. Freytag, and W. Lehner. Efficient exploitation of similar subexpressions for query processing. In *Proc. of SIGMOD'07*, 2007.