Keyword search in databases: the power of RDBMS

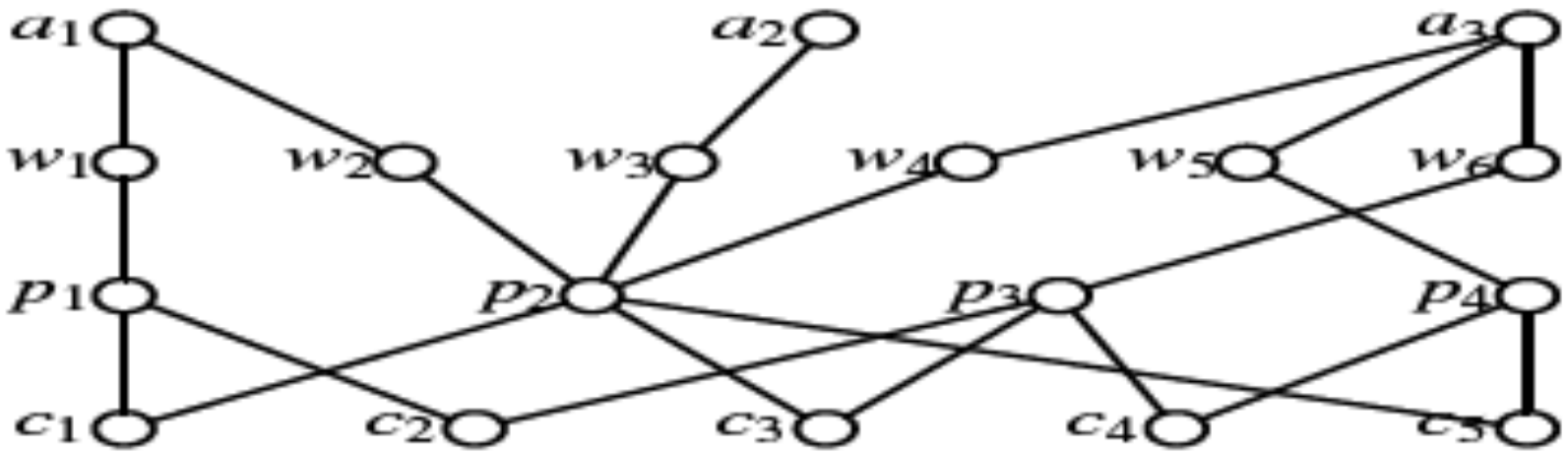
Introduction

- Two 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

Introduction (cont.)

- 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.
- The middleware(graph-based in-memory algorithms by materializing an RDB as a graph) approach does not fully utilize the functionality of RDBMSs, and only uses SQL to retrieve data.

Graph



(e) Tuple Connections

Preliminary

- GD(V,E) on the schema graph GS. Here, V represents a set of tuples, and E represents a set of connections between tuples.
- It is worth noting that we use GD to explain the semantics of keyword search but do not materialize GD over RDB.

DBLP Database schema

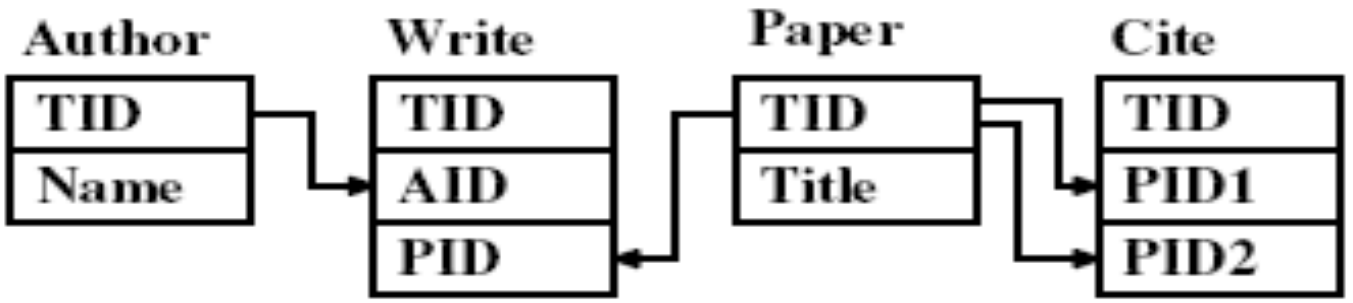


Figure 1: *DBLP* Database Schema

Three types of keyword queries

1. Connected tree semantics
2. Distinct root semantics
3. Distinct core semantics

Connected Tree Semantics

- result of such a query is a minimal total joining network of tuples, denoted as MTJNT
  - 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
  - a joining network of tuples must contain all the m keywords (by total).
  - a joining network of tuples is not total if any tuple is removed (by minimal).

Connected Tree Semantics

TID	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

(a) Author

TID	Title
p_1	Contributions of Michelle
p_2	Keyword Search in XML
p_3	Pattern Matching in XML
p_4	Algorithms for TopK Query

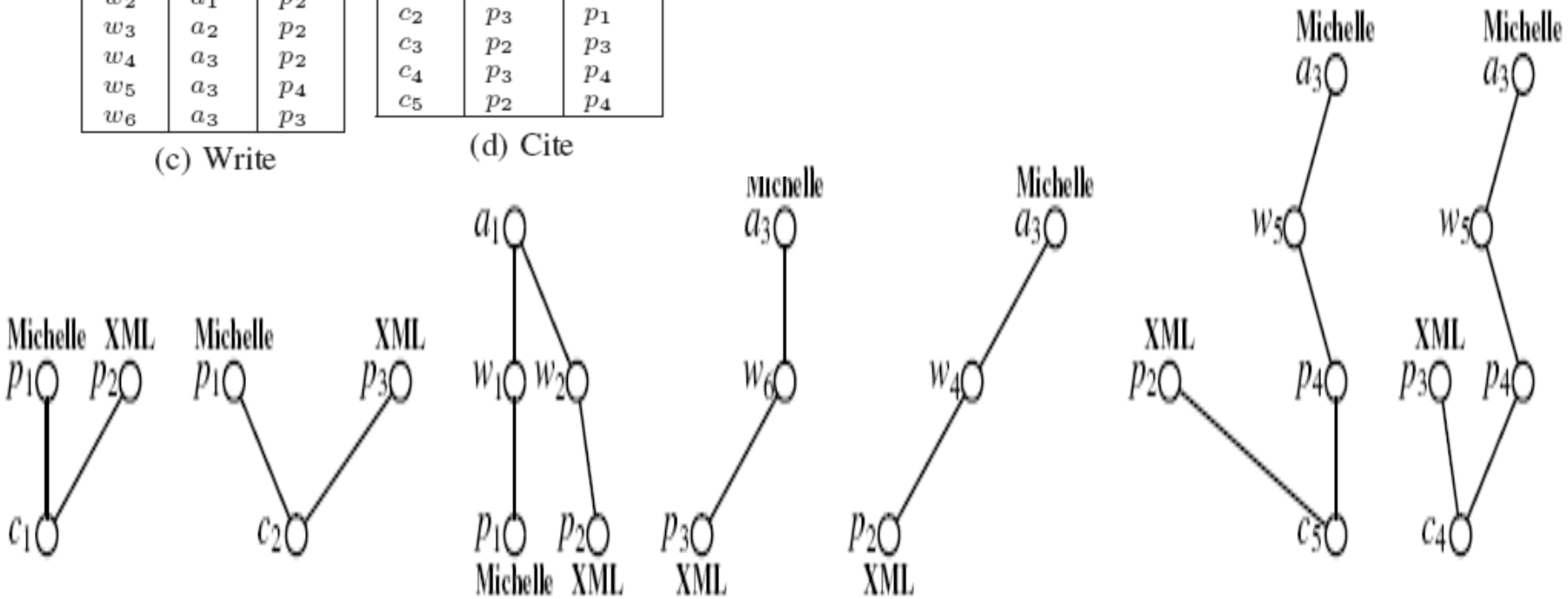
(b) Paper

TID	AID	PID
w_1	a_1	p_1
w_2	a_1	p_2
w_3	a_2	p_2
w_4	a_3	p_2
w_5	a_3	p_4
w_6	a_3	p_3

(c) Write

TID	PID1	PID2
c_1	p_2	p_1
c_2	p_3	p_1
c_3	p_2	p_3
c_4	p_3	p_4
c_5	p_2	p_4

(d) Cite



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

Connected Tree Semantics

- 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 MTJNTs
- with a 2-keyword query K = {Michelle,XML} and Tmax = 5. There are 7 MTJNTs shown in following figure.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

- Suppose that there is a result rooted at tuple tr. For any of them-keyword, say kl, there is a tuple t in the result that satisfies the following conditions.
  - (1) t contains the keyword kl.
  - (2) Among all tuples that contain kl, the distance between t and tr (tuple root) is minimum.
  - (3) The minimum distance between t and tr must be less than or equal to a user given parameter Dmax

Distinct Root Semantics

TID	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

(a) Author

TID	Title
p_1	Contributions of Michelle
p_2	Keyword Search in XML
p_3	Pattern Matching in XML
p_4	Algorithms for TopK Query

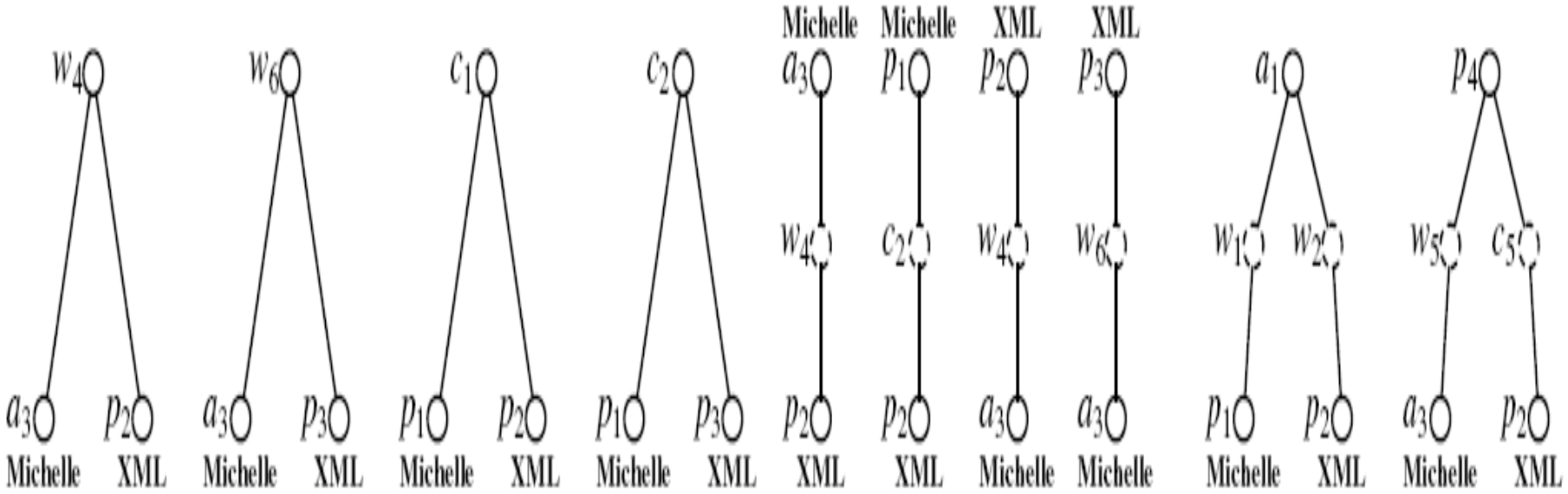
(b) Paper

TID	AID	PID
w_1	a_1	p_1
w_2	a_1	p_2
w_3	a_2	p_2
w_4	a_3	p_2
w_5	a_3	p_4
w_6	a_3	p_3

(c) Write

TID	PID1	PID2
c_1	p_2	p_1
c_2	p_3	p_1
c_3	p_2	p_3
c_4	p_3	p_4
c_5	p_2	p_4

(d) Cite



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

Distinct Core Semantics

- A community, Ci(V,E), is specified as follows. V is a union of three subsets of tuples, V = Vc union Vk union Vp.
  - Vk represents a set of keyword-tuples where a keyword-tuple vk element of Vk contains at least a keyword
  - Vc represents a set of center-tuples where there exists at least a sequence of connections between vc element of Vc and every vk element of Vk such that dis(vc, vk) Dmax, and
  - Vp represents a set of path-tuples which appear on a shortest sequence of connections from a center-tuple vc element of Vc to a keyword-tuple vk 2 Vk if dis(vc, vk) Dmax.
- Note that a tuple may serve several roles as keyword/ center/path tuples in a community

Distinct Core Semantics

TID	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

(a) Author

TID	Title
p_1	Contributions of Michelle
p_2	Keyword Search in XML
p_3	Pattern Matching in XML
p_4	Algorithms for TopK Query

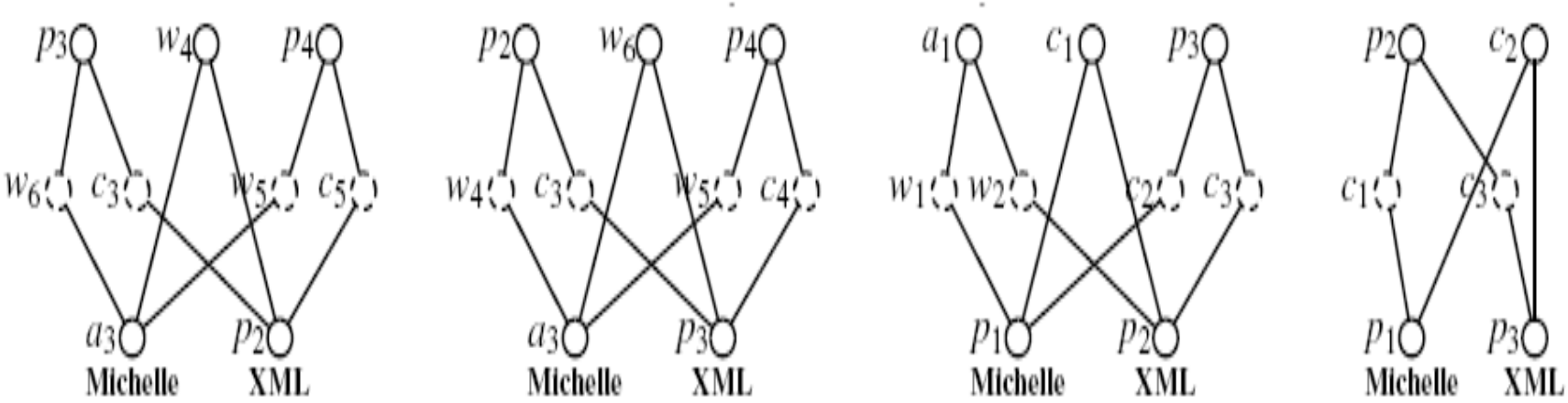
(b) Paper

TID	AID	PID
w_1	a_1	p_1
w_2	a_1	p_2
w_3	a_2	p_2
w_4	a_3	p_2
w_5	a_3	p_4
w_6	a_3	p_3

(c) Write

TID	PID1	PID2
c_1	p_2	p_1
c_2	p_3	p_1
c_3	p_2	p_3
c_4	p_3	p_4
c_5	p_2	p_4

(d) Cite



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

Figure 3: Three Semantics

CONNECTED TREE IN RDBMS

- A candidate network (CN) corresponds to a relational algebra that joins a sequence of relations to obtain MTJNTs over the relations involved. The set of CNs is proved to be sound/complete and duplication-free .
- 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.

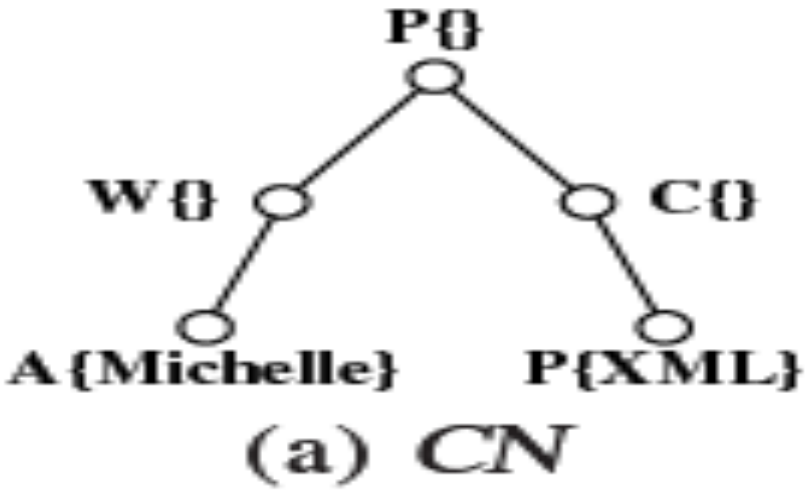
CONNECTED TREE IN RDBMS

- CN is a sequence of joins, where the number of nodes is less than or equal to Tmax, and the union of the keywords represented in a CN is ensured to include all the m- keywords.
- P{XML} means contain(XML)(¬contain(Michelle) P), or equivalently the following SQL:
  - select * from Paper as P

  where contain(Title, XML)

  and

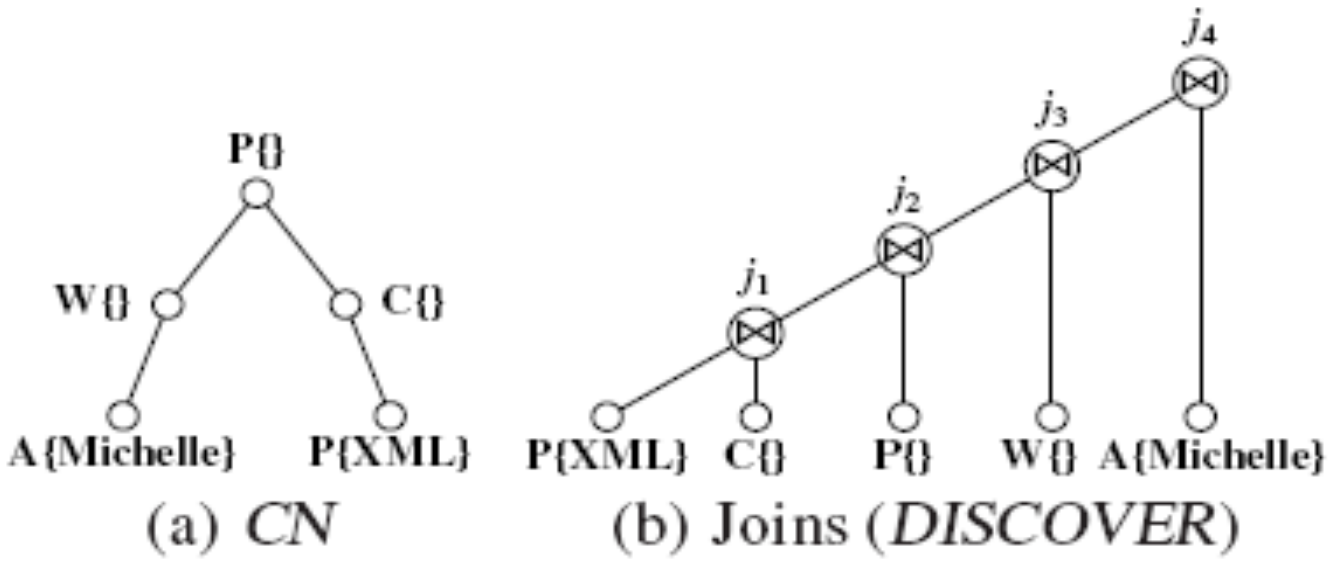
  not contain(Title, Michelle)



(a) CN

CONNECTED TREE IN RDBMS

- All CNs are computed using SQL. An operator tree (join plan) is shown in Figure below to process the CN in Fig. (a) using 5 projects and 4 joins.



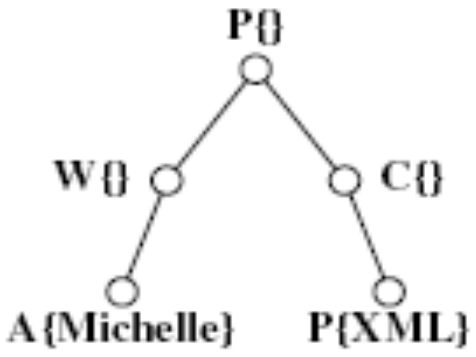
CONNECTED TREE IN RDBMS

- They 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 () the tuples from R that can possibly join at least a tuple in S.
  - $R \ltimes S$ = project(R)(R join S) (2)
- Based on semijoin, a join R 1 S can be supported by a semijoin and a join as given in Eq. (3).
  - R join S = $(R \ltimes S)$ join S (3)

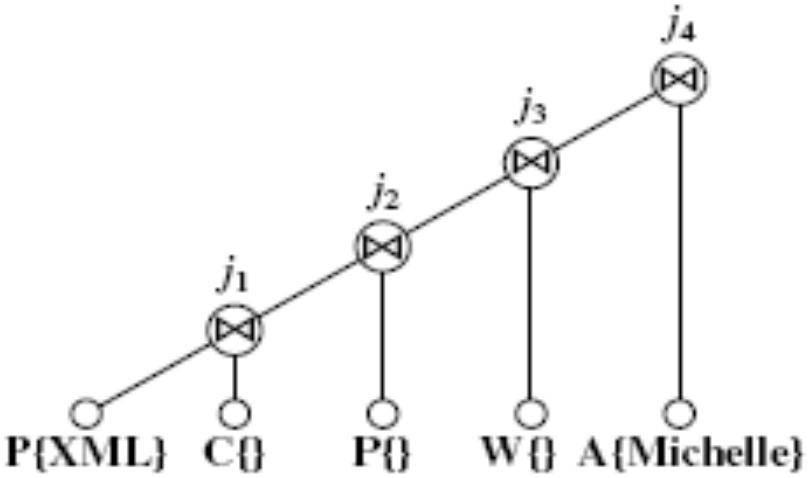
CONNECTED TREE IN RDBMS

- 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

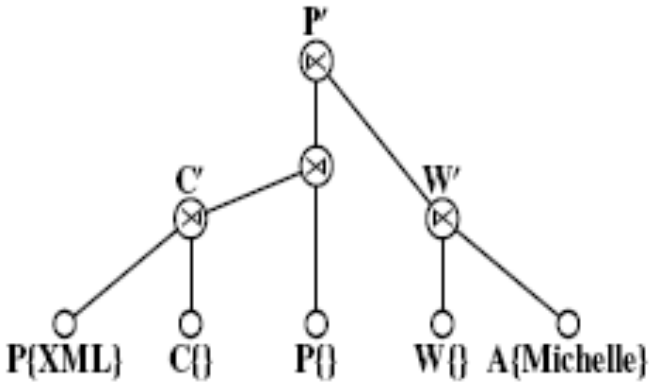
CONNECTED TREE IN RDBMS



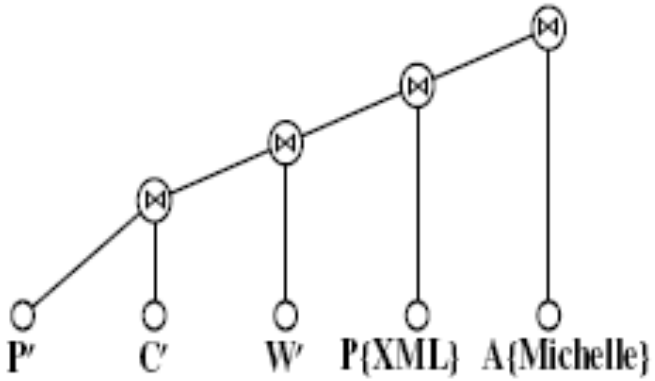
(a) *CN*



(b) Joins (*DISCOVER*)

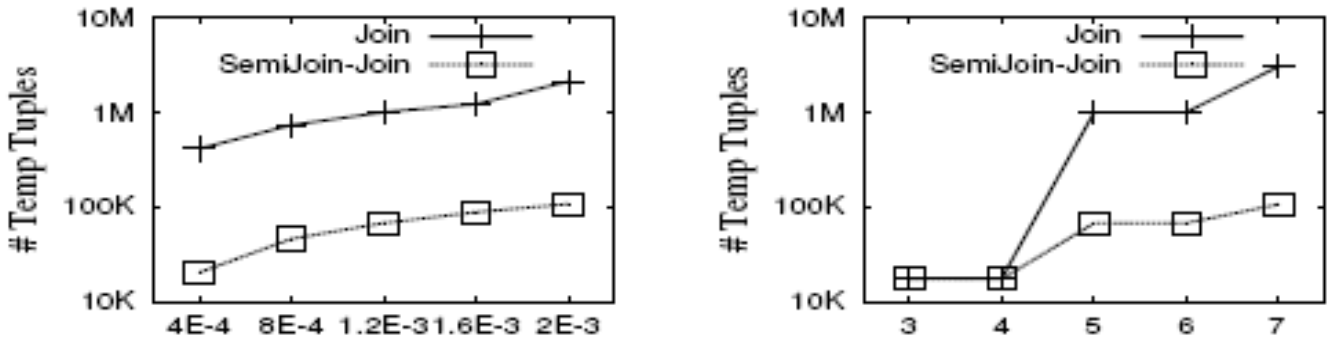


(c) Semijoin



(d) Join

Semi-join better than join operation



(a) Vary Keyword Selectivity (b) Vary Tmax

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

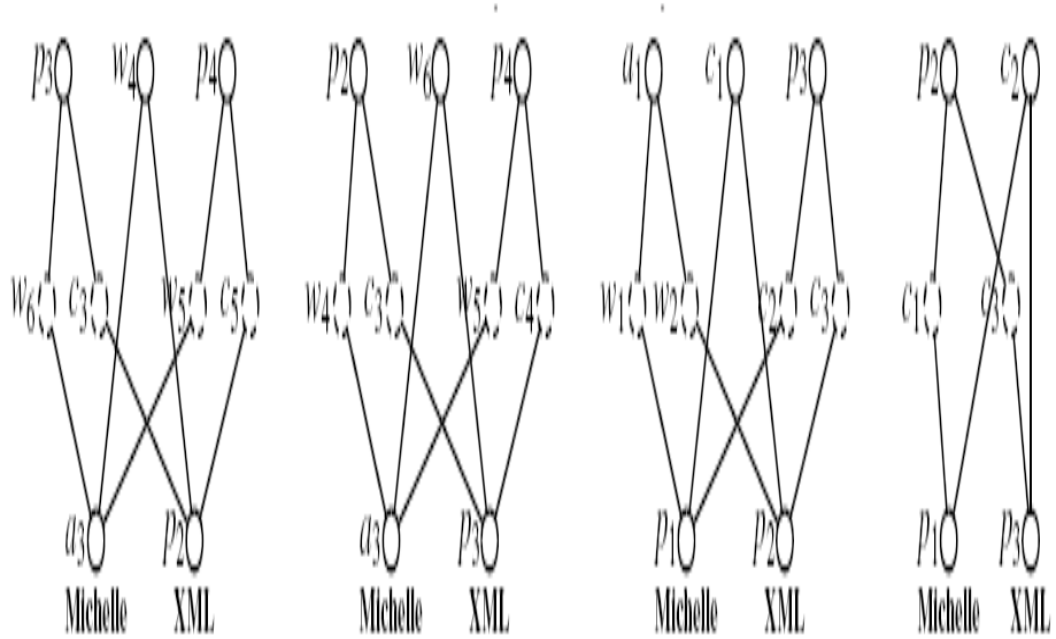
Remark

- Based on their 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 dominative, if it is difficult to find an optimal query processing plan or the cost of finding an optimal query processing plan is large.

DISTINCT CORE IN RDBMS

- In the first step, for each keyword ki, we compute a temporal relation, Pairi(tidi, disi, TID), with three attributes, where both TID and tidi are TIDs and disi is the shortest distance between TID and tidi (dis(TID, tidi)), which is less than or equal to Dmax

DISTINCT CORE IN RDBMS



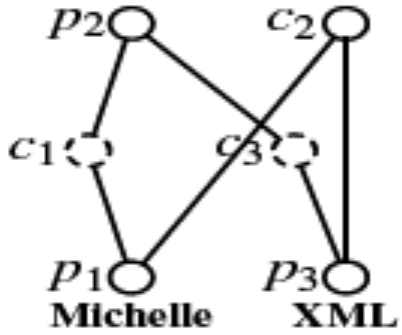
(c) Distinct Core ($K = \{Michelle, XML\}$, Dmax = 2)

Figure 3: Three Semantics

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)

DISTINCT CORE IN RDBMS

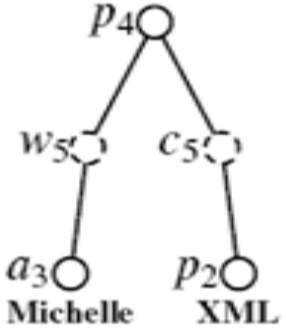


TID	tid1	dis1	tid2	dis2
p_1	p_1	0	p_3	2
p_2	p_1	2	p_3	2
p_3	p_1	2	p_3	0
c_2	p_1	1	p_3	1

- the distinct core,
  - p1 contains keyword Michelle (k1) and
  - p3 contains keyword XML (k2), and
  - the 4 centers, {p1, p2, p3, c2}, are listed on the TID column. Any center can reach all tuples in the core, {p1, p3}, within Dmax.

DISTINCT ROOT IN RDBMS

- 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. below 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 a3 contains keyword Michelle (k1) and p2 contains keyword XML (k2).
- 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 p4. We select one of them using the aggregate function min, following the

DISTINCT ROOT IN RDBMS

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 ($K = \{Michelle, XML\}$, Dmax $= 2$)

DISTINCT ROOT IN RDBMS

- 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.

Naive Algorithms

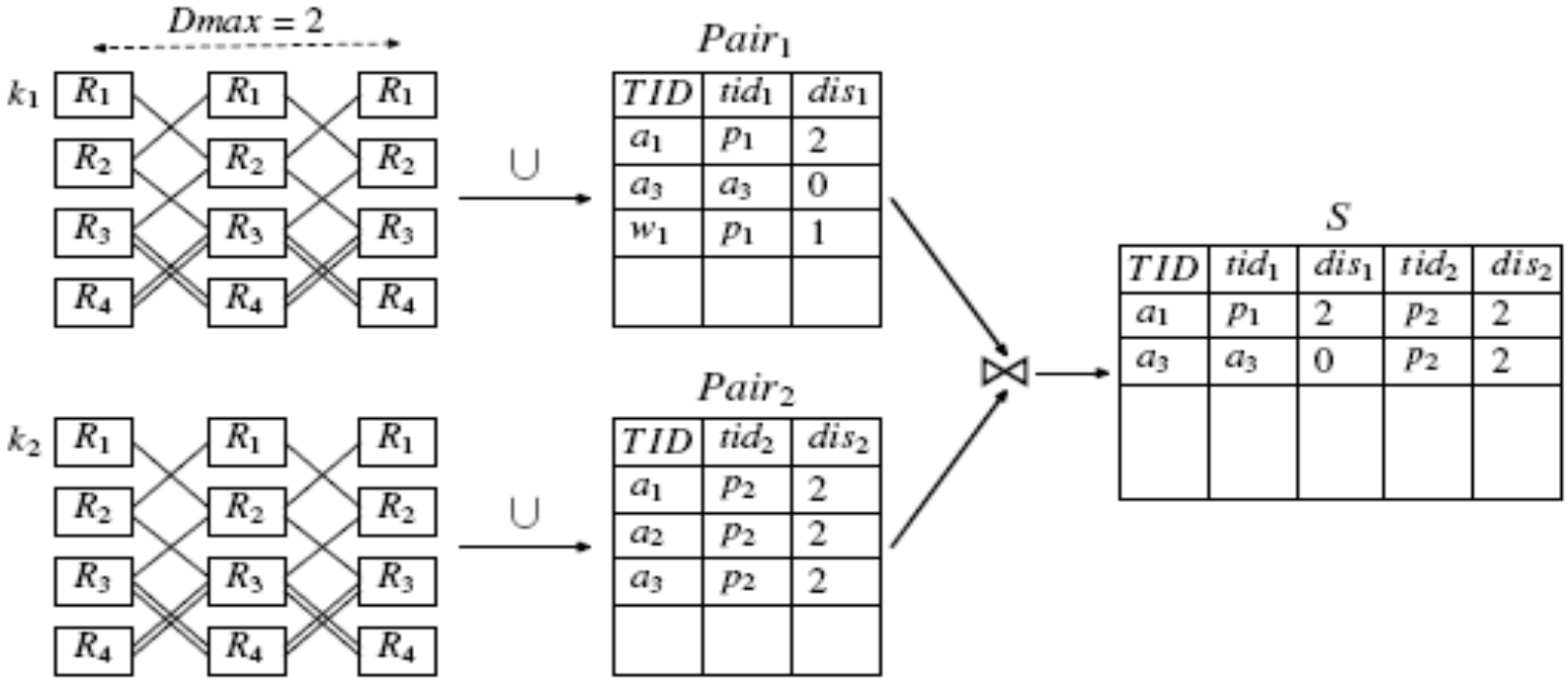


Figure 6: An Overview (R_1, R_2, R_3, and R_4 represent `Author`, `Write`, `Paper`, and `Cite` relations in Example 2.1)

Naive Algorithms

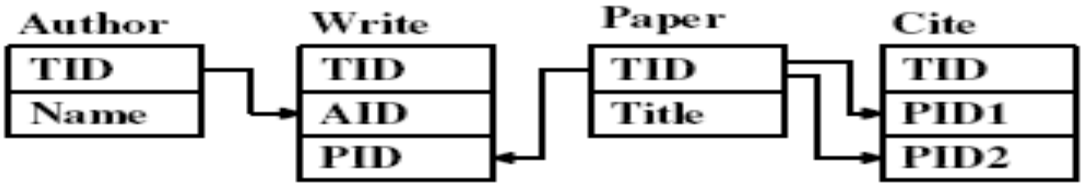


Figure 1: *DBLP* Database Schema

$$P_{0,1} \leftarrow \Pi_{TID \rightarrow tid_1, 0 \rightarrow dis_1, *}(\sigma_{contain(k_1)} R_1)$$

$$P_{0,2} \leftarrow \Pi_{TID \rightarrow tid_1, 0 \rightarrow dis_1, *}(\sigma_{contain(k_1)} R_2)$$

$$P_{0,3} \leftarrow \Pi_{TID \rightarrow tid_1, 0 \rightarrow dis_1, *}(\sigma_{contain(k_1)} R_3)$$

$$P_{0,4} \leftarrow \Pi_{TID \rightarrow tid_1, 0 \rightarrow dis_1, *}(\sigma_{contain(k_1)} R_4)$$

Naive Algorithms

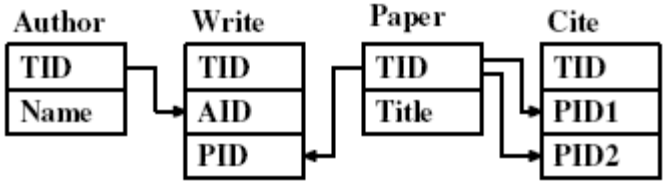


Figure 1: *DBLP* **Database Schema**

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

Naive Algorithms

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 (Γ) and SQL aggregate function min.

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

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

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

Naive Algorithms

- Rule-1: If the same (tidi, TID) value appears in two different Pd',j and Pd,j , then the shortest distance between tidi and TID must be in Pd',j but not Pd,j , if d' < d. Therefore, Eq. (7) can be computed as follows.

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

- There does not exist a shortest path between tid1 and TID before

Naive Algorithms

- Rule-2: If there exists a shortest path between tidi and TID value pair, say, disi(tidi, TID) = d', then there is no need to compute any tuple connections between the tidi and TID pair, because all those will be removed later by group-by and SQL aggregate function min. In Eq. (6), every P1,j , 1 < j < 4, can be further reduced as $P_{1,j} \leftarrow \sigma_{(tid_1,TID) \notin \prod_{tid_1,TID}(P_{0,j})} P_{1,j}.$

Naive Algorithms

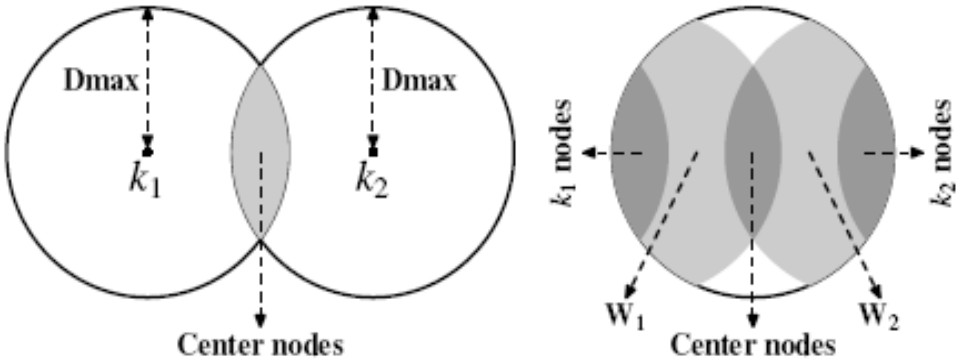
- The naive algorithm DR-Naive() to compute distinct roots can be implemented in the same way as DC Naive() 2 with 2 group-bys as follows:

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

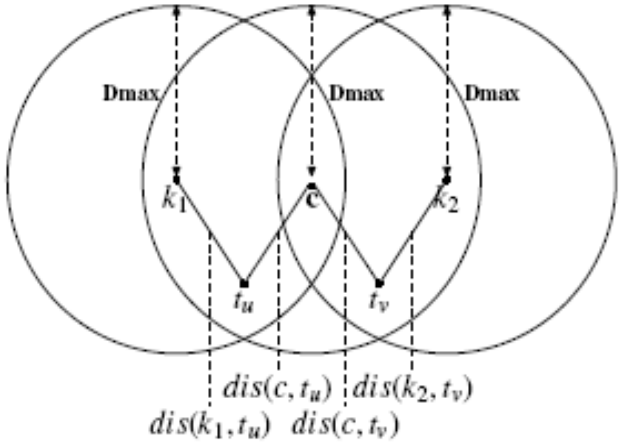
Three-phase reduction Algorithm

- in the three-phase reduction, we significantly prune the tuples from an RDB that do not participate in any communities.

Three-phase reduction



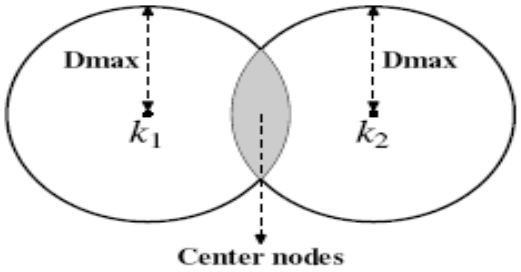
(a) From keywords to centers (b) From centers to keywords



(c) Project Relations

Figure 7: Three-Phase Reduction

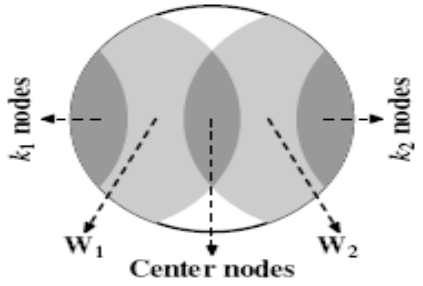
The first reduction phase (from keyword to center):



(a) From keywords to centers

- We consider a keyword ki as a virtual node, called a keyword-node, and we take a keyword-node, ki, as a center to compute all tuples in an RDB that are reachable from ki within Dmax.
- Let Gi be the set of tuples in RDB that can reach at least a tuple containing keyword ki within Dmax, for 1 <i< m. Based on all Gi, we can compute Y = G1 join G2 join · · · join Gm which is the set of center-nodes that can reach every keyword-node ki, 1 <i< m, within Dmax

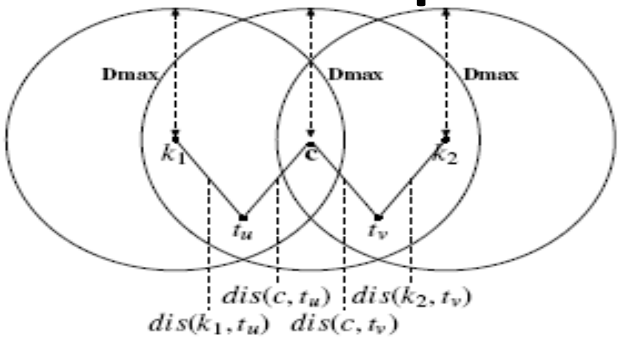
The second reduction phase (from center to keyword)



(b) From centers to key-words

- 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 Wi be the set of tuples in Gi that can be reached from a center in Y within Dmax, for 1 <i< m. Note that Wi subset Gi.
- 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)



(c) Project Relations

- We project an RDB′ out of the RDB, which is sufficient to compute all multi-center communities by join Gi join Wi, for 1 <i< m. Consider a tuple in Gi which contains a TID t′ with a distance to the virtual keyword-node ki, denoted as dis(t′, ki), and consider a tuple in Wi which contains a TID t′ with a distance to the virtual center node c, denoted as dis(t′, c).
- The sum of two distances range[0,Dmax]

Performance studies

- Conducted 3 algo :
  - semijoin/join based algorithm, denoted Semijoin-Join
  - the join based algorithm , denoted Join
  - the block pipeline algorithm (BP) to compute the top 10 answers.

Exp-1: Selectivity Testing

- Connected Tree
- Ksel :keyword SelectivityS
- M:number of Keywords
- Tmax: Tree size

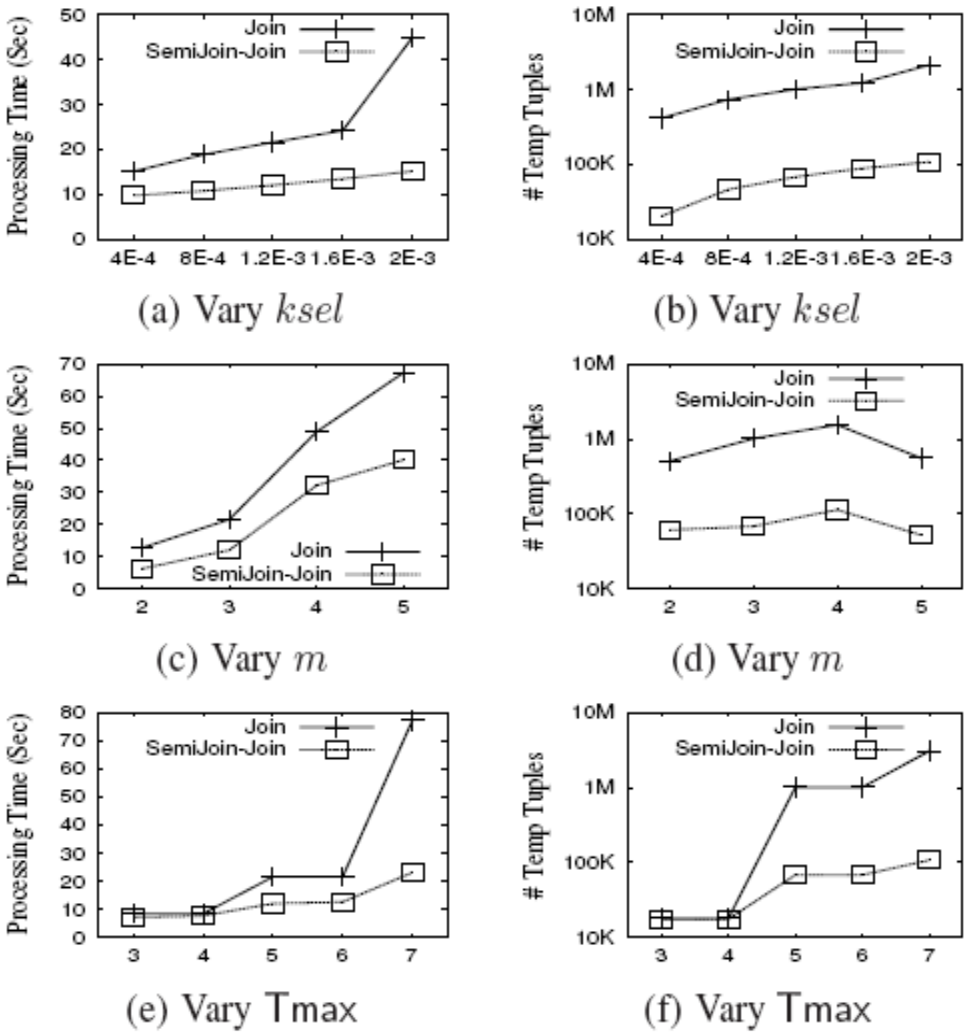


Figure 8: Connected Tree (DBLP)

Exp-1: Selectivity Testing

- Connected Tree
- 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.

Exp-1: Selectivity Testing

- 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.
- 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.

Exp-1: Selectivity Testing

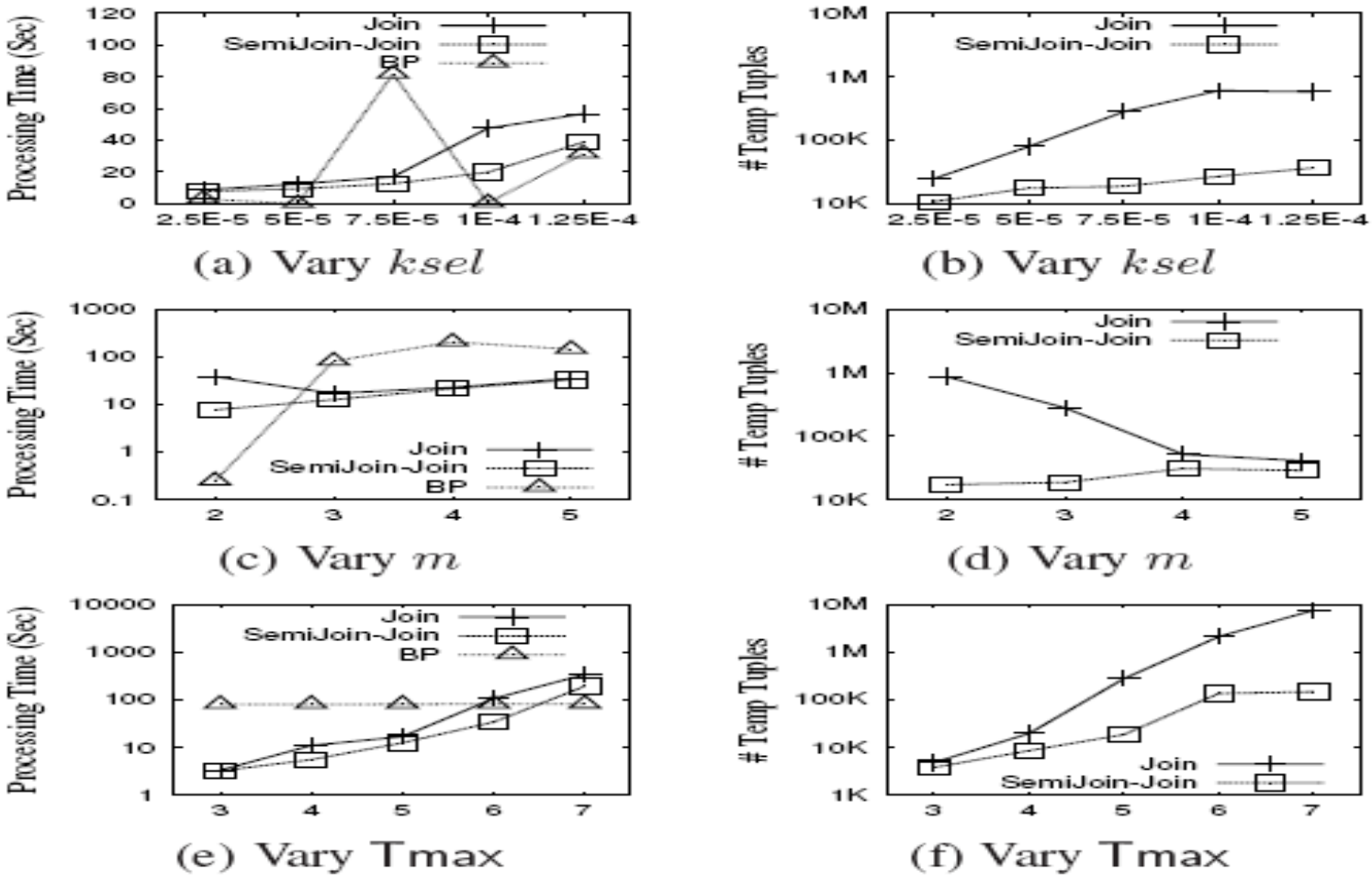


Figure 9: Connected Tree (IMDB)

Exp-1: Selectivity Testing

- 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

Exp-1: Selectivity Testing

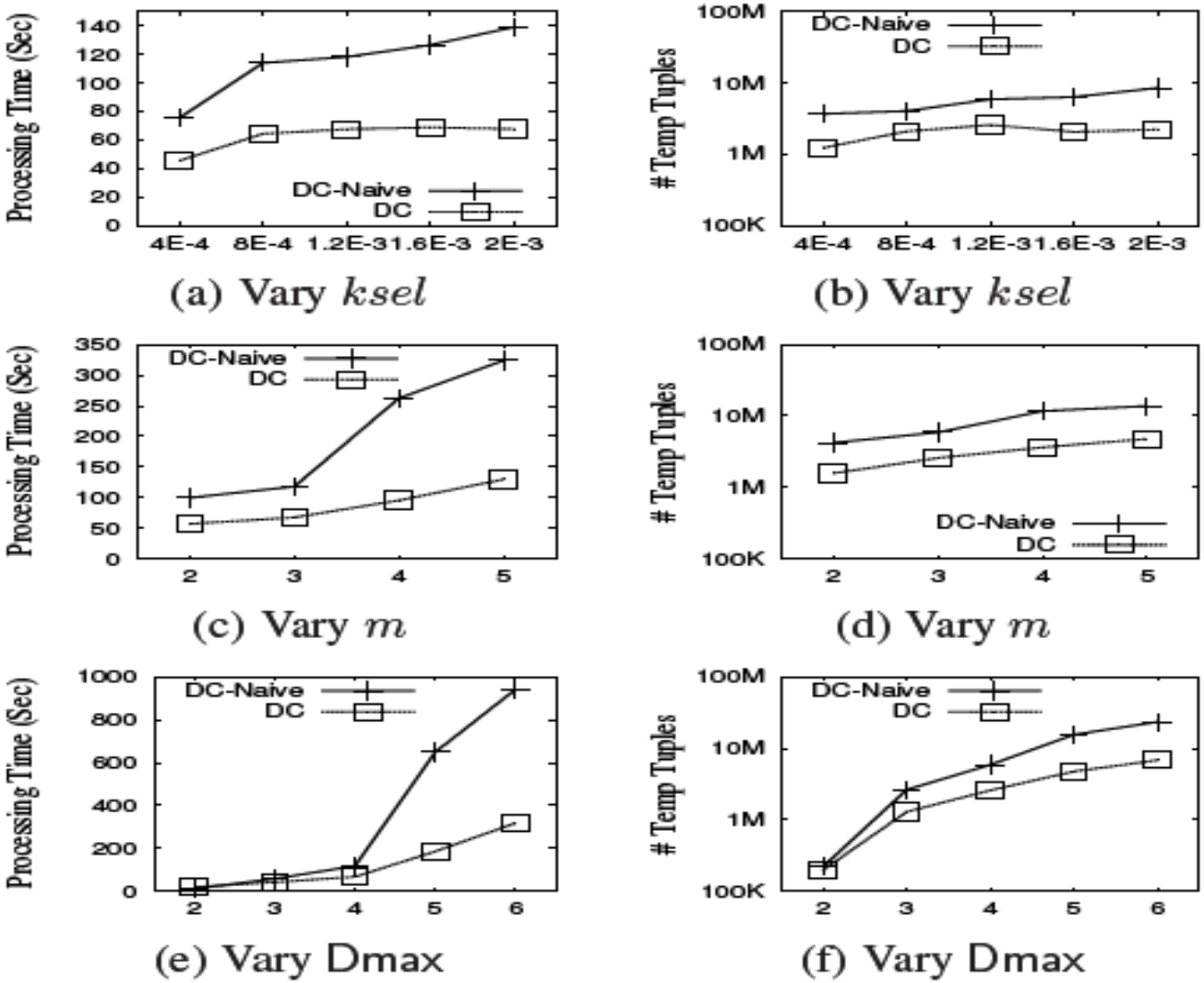
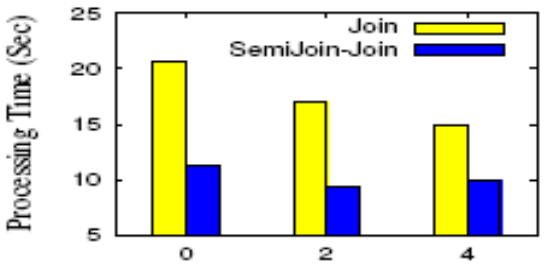
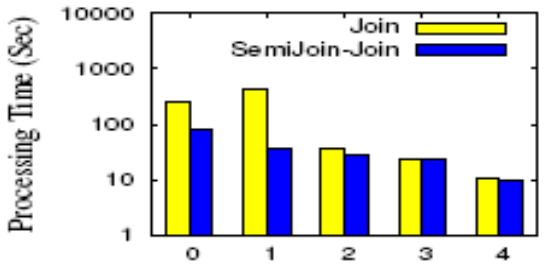


Figure 10: Distinct Core (DBLP)

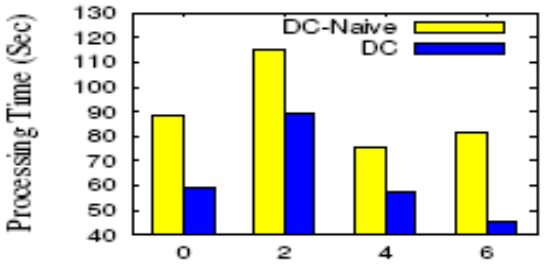
Exp-2: Compactness Testing



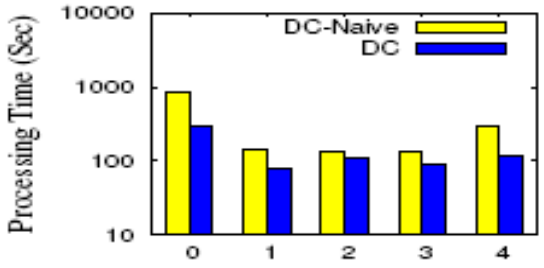
(a) Connected Tree (*DBLP*)



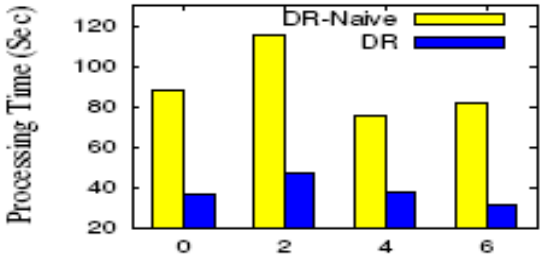
(b) Connected Tree (*IMDB*)



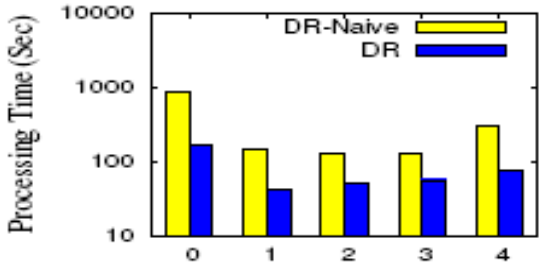
(c) Distinct Core (*DBLP*)



(d) Distinct Core (*IMDB*)



(e) Distinct Root (*DBLP*)



(f) Distinct Root (*IMDB*)

Figure 13: Query Compactness

Exp-2: Compactness Testing

- 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.

Exp-2: Compactness Testing

- 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

Contribution

- 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
- a new approach to prune tuples that do not participate in any resulting connected trees followed by query processing over the reduced relations.
- new three-phase reduction approach to effectively prune tuples from relations followed by query processing over the reduced relations