

Promise Winter School
Bridging between Information Retrieval and Databases

Bressanone, Italy 4 - 8 February 2013

**The Keyword Search on Relational
Databases**

Prof. Sonia Bergamaschi

Ing. Giovanni Simonini, Ing. Silvia Rota, PhD Francesco Guerra

Dipartimento di Ingegneria “Enzo Ferrari”
University of Modena and Reggio Emilia, Italy

Outline

- ▶ Motivation
- ▶ Keyword Search Goal
- ▶ Keyword Search Systems Overview
- ▶ Indexing
- ▶ Data Representation Model
- ▶ Schema-based Approach
- ▶ Graph-based Approach
- ▶ Comparison of the Systems

Keyword Queries

- ▶ popular in recent years thanks to web search engines and their easy query interface for any kind of user
- ▶ difficult to express complex queries
- ▶ Actually does not work for retrieving data from databases publicly available on the web; a keyword search interface might be useful
- ▶ No research prototypes have transitioned from proof-of-concept implementations into deployed systems [TKDE2012]


Main Issues

- ▶ Traditional query techniques based on structured query languages (SQL) are being day-by-day proven limited in highly heterogeneous environments:
 - ▶ they require a complete knowledge of the underlying data structures and instances.
- ▶ Solving a query requires addressing a number of critical tasks concerning structural and lexical aspects:
 1. the user selects the tables and attributes target of a query on the basis of their names, which may be misleading or not meaningful (lexical aspect);



Main Issues (2)


- the user expresses conditions on attributes without having accurate knowledge of the domain. Thus, s/he may define over selective or, vice-versa, too broad or illegal selection clauses (lexical aspect);



Professors in the area of **information retrieval**?

Name	Area	Email	Address

- the user does not know the relationships between the tables and, consequently, it is hard to pose multi-table queries (structural and lexical aspect).



Professors living in Rome?

City	Country
Rome	Italy
Milan	Italy
Paris	France
Berlin	Germany
Lion	France

Name
Watson
Date
Hunt
Bill
Tan

Keyword Search Approach

- ▶ A database can be viewed as a data graph $G(V, E)$, where V is a set of tuples, and E is a set of connections between tuples. A connection between two tuples exists if there is at least one reference from one of the tuples to the other one.
- ▶ A l -keyword query Q is a set of l keywords
 $Q = \{k_1, k_2, \dots, k_l\}$
- ▶ The result to be returned is a set of connected sub-graphs
 $R = \{R_1(V, E), R_2(V, E), \dots, R_m(V, E)\}$

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

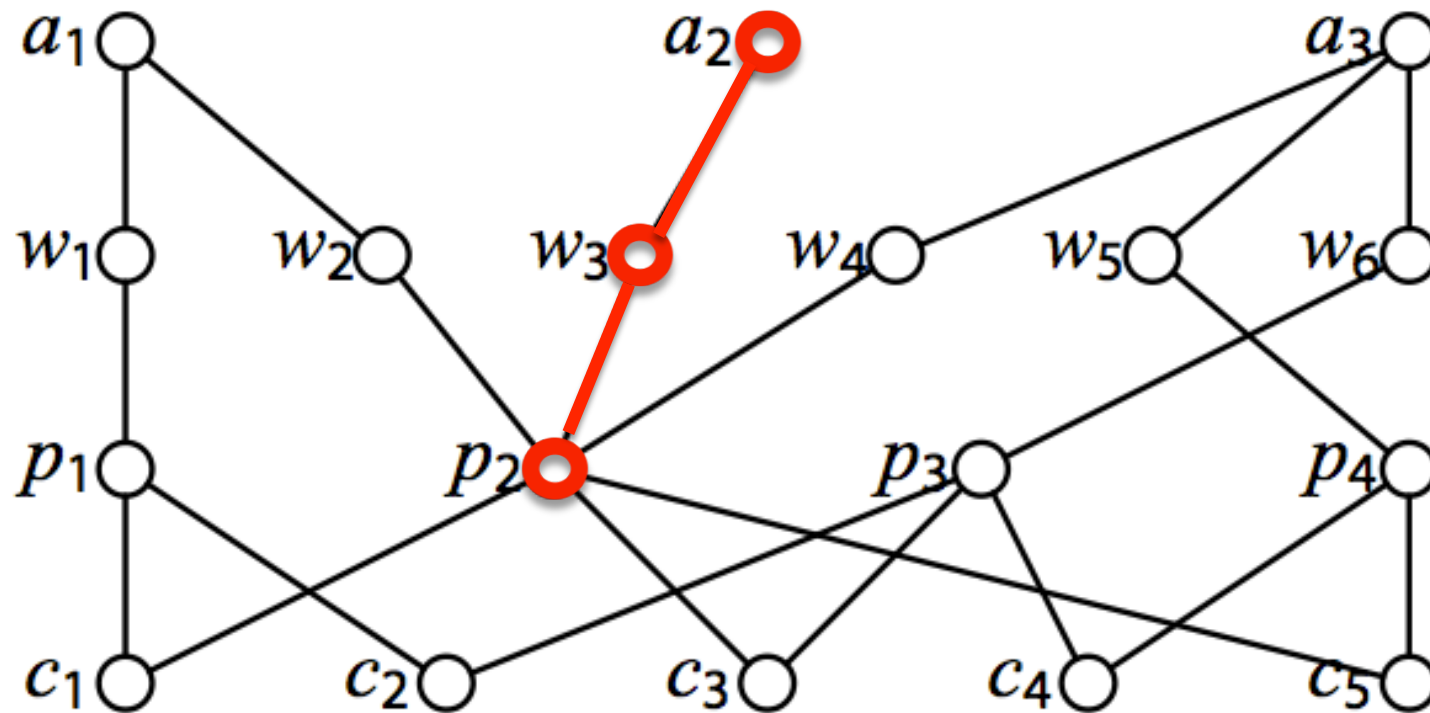
(b) Paper

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



$Q = \{\text{Michael, XML}\}$

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

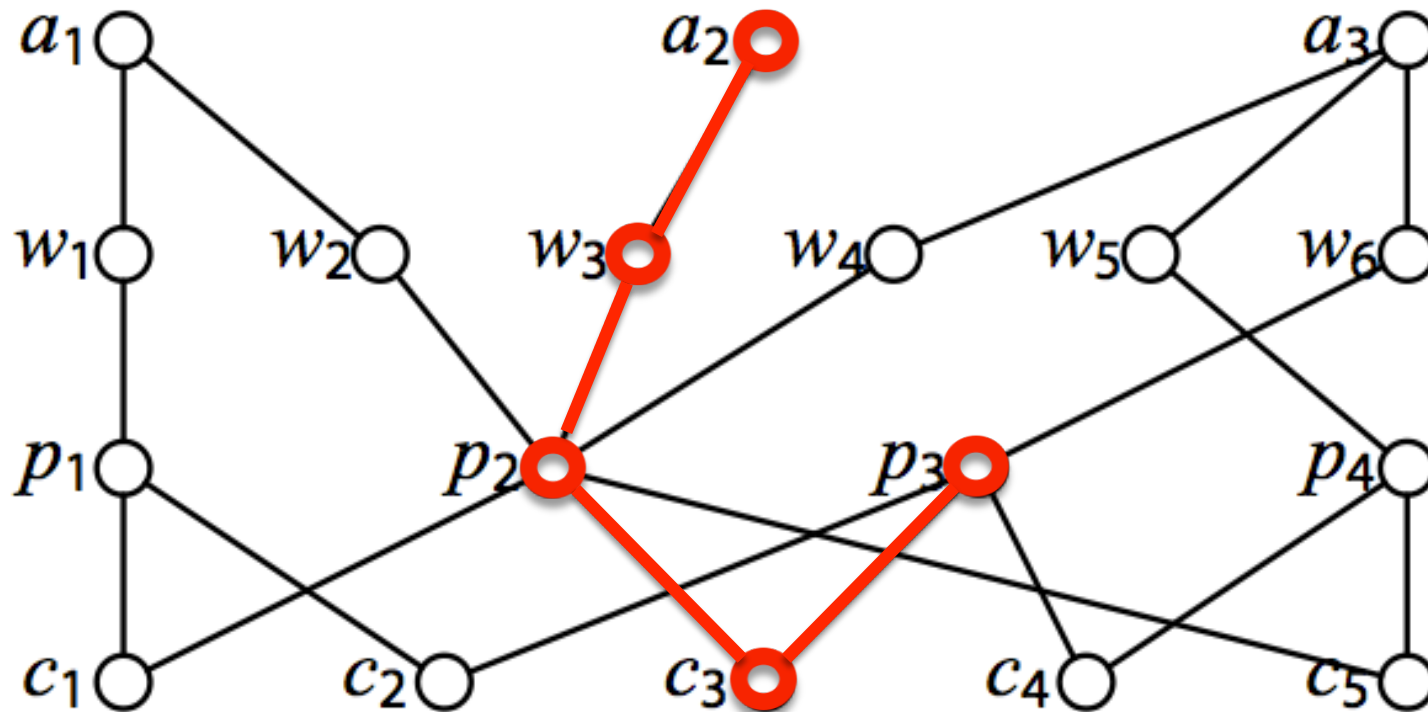
(b) Paper

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



$Q = \{\text{Michael, XML}\}$

Hidden Web

- ▶ The web is bigger than it looks
 - ▶ Beyond the billions of pages that populate the major search engines, lies an even vaster *hidden web* of data:
 - Classified ads
 - Library
 - Catalogs
 - Airlines reservation systems
 - Scientific databases
 - ...

generally, dynamic contents and their underlying databases

- ▶ *Hidden (or “Deep”) Web* size is estimated at up to 500 times larger than the *Surface Web* of static HTML pages [Alex Wright: Searching the deep web. Commun. ACM 51 (10):14-15 (2008)]
- ▶ The deep Web contains 7,500 TB of information compared to 19 TB of information in the surface Web [Bergman, K. T. (2001). The deep web: surfacing hidden value. The Journal of Electronic Publishing 7. <http://dx.doi.org/10.3998/3336451.0007.104>]

Searching the Hidden Web

Classical search engines are not able to query the hidden web (and RDBs on Web)

- ▶ Is it possible to integrate in the same platform both database and information retrieval techniques?
 - ▶ the sophisticated DB facilities provided by a DBMS assist users to query well-structured information using a structured query language
 - ▶ IR techniques allow users to search unstructured information using keywords with a simple language

Keyword Search Goal

- ▶ Querying publicly available RDBs on the Web with keyword queries
 - ▶ without knowing DBs schema

Web Pages vs Databases

- ▶ In the following table the most important differences between a web page and a database are listed:

	Web Page	Database
Type	Flat Document	Structured data
Access	Through URL	Through DBMS
Previous knowledge	None	Data structure
Relations between data	Hyperlinks	Foreign keys
Output	Flat document (partial/total)	Tuples

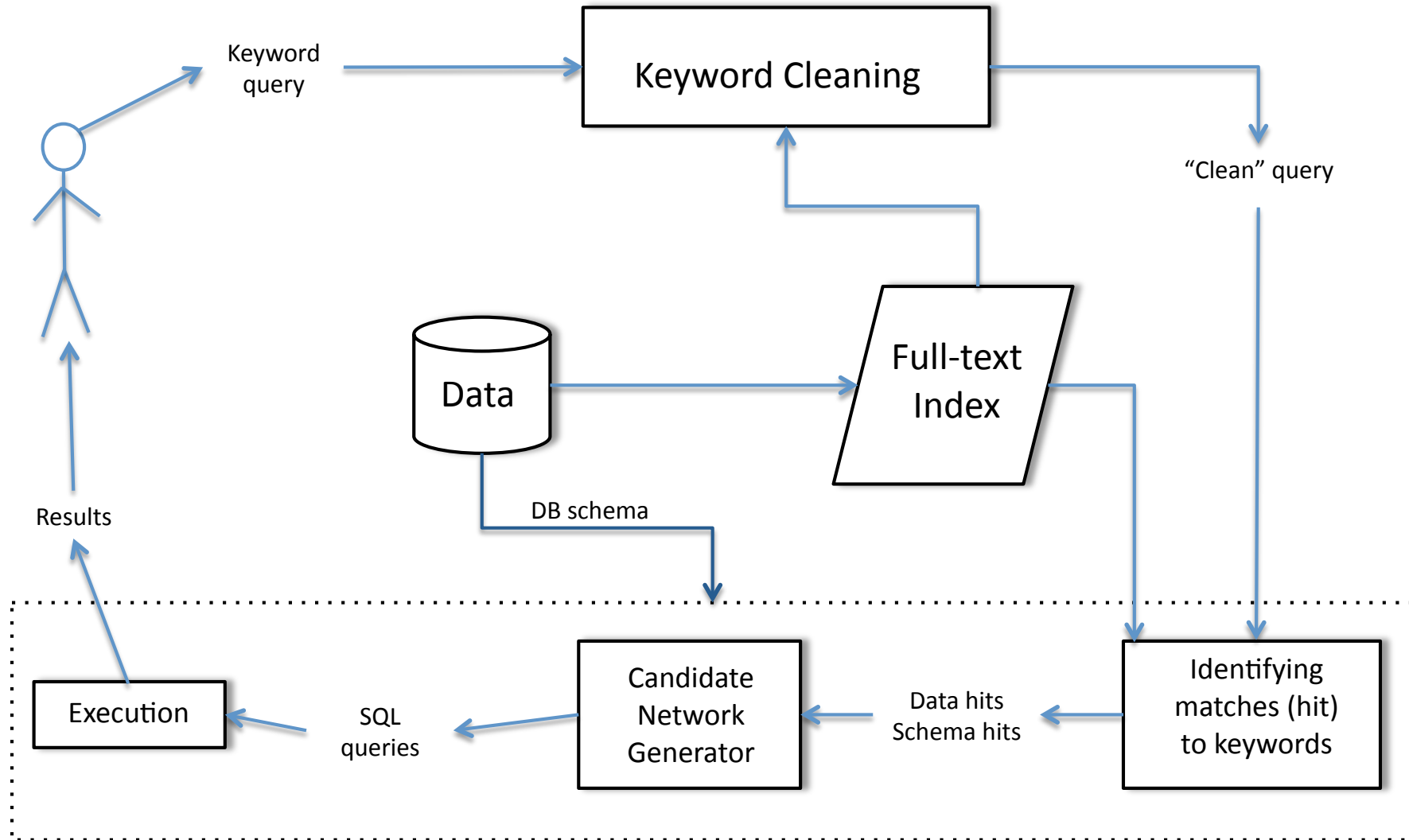
Keyword Search vs Classical Search Engine

- Traditional IR techniques consider textual documents as "**bags of words**": each keyword present in the documents is indexed separately and the semantics represented by the order of the words and phrases in the text is completely lost
- At query time the index is used to retrieve all documents that contain the keywords in the query

The difficulties in accessing structured data with classical search engines are:

- ▶ The crawling of structured data is not possible because querying databases requires to have knowledge of the schema in advance, in order to access any piece of data. Instead, all web resources are accessed using URL and no other information
- ▶ The structure embeds the relationships between data and this is itself an important part of the information

A general system overview



Keyword-based querying

- ▶ **Keyword cleaning:** for handling queries that do not match exactly the content of the database
 - ▶ Natural language queries
 - ▶ Spelling corrections
 - ▶ Keywords expansion
 - ▶ Keywords are expanded to similar words that exist in the database
 - ▶ (example) a keyword “Gorge” can be expanded to “George”, “Gerbo”, and “Georgia” (assuming that those three words exist in the database)
- ▶ Query segmentation
 - ▶ query is segmented into subsequences (keyword *segments*)

[FRISK in 2009]

Keyword-based Querying

- ▶ **Conjunctive keyword queries:** retrieval of the tuples that contain exactly all the keywords, i.e. it uses AND semantic

[DBXplorer, DISCOVER, BANKS (in 2002); ObjectRank (in 2004)]

- ▶ **Conjunctive or Disjunctive keyword queries:** the user may specify which semantic to use. All the keywords are in AND or in OR semantic

[SPARK in 2007]

Keyword-based querying (2)

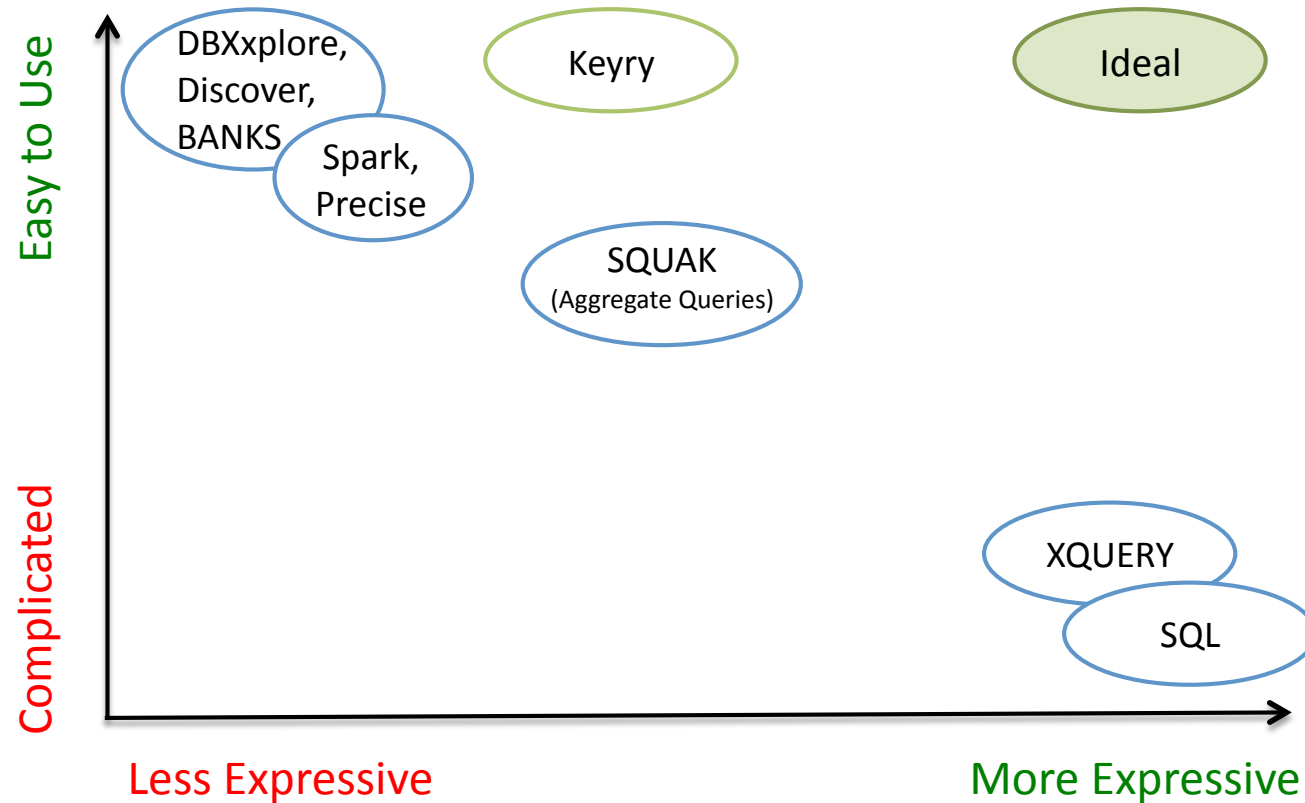
- ▶ **Boolean operators AND, OR, NOT:** a query may contain many keywords combined in a logical expression through AND, OR, NOT and parenthesis

[Precis in 2008]

- ▶ **Aggregate queries:** the user may specify queries that represent aggregate functions, such as “John number courses”

[SQAK in 2008]

Comparison of Existing Approches



Partially taken from [8]

Full-text Indexing on RDBs

- ▶ the data is indexed for a fast look up process
- ▶ Keyword search is heavily related on indexes built over the database instance:
 - + fast way to find the exact keyword inside the DB
 - hard to manage frequently updated DBs
 - “bag of words” approach that loses the DB semantics

Creating a Full-text index on an RDB

- ▶ Techniques commonly used:

- ▶ Symbol Table [DBXplorer in 2002]
- ▶ Master Index [DISCOVER in 2002]
- ▶ Double Index [BANKS in 2002]
- ▶ ...

- ▶ RDBMSs that support full-text indexing and searching:

IBM DB2, ORACLE, Microsoft SQL-Server, Postgres and MySQL

Indexing

- ▶ **Symbol Tables:** the equivalent of an inverted index for databases.
 - ▶ **Column granularity:** when the DB provides indexes on the columns, the symbol table can be more compact and contains for each keyword the list of matching columns (small amount of memory used)
 - ▶ **Row granularity:** for each keyword it keeps the list of rows that contains it

Both types enable to find only the DB instances that match the exact value of the keyword

Another Symbol Table is needed for long text attributes

- ▶ **Token-level:** necessary to search keyword in long textual attributes (e.g.: city name inside an attribute containing whole address)

Indexing (2)

- ▶ **Master index:** it takes as input a set of keywords k_1, \dots, k_m and outputs a set of basic tuple sets $R_i^{k_j}$ (i number of relations in the DB) and j number of keywords. The basic tuple set $R_i^{k_j}$ consists of all the tuples of R_i that contain the keyword k_j . The master index builds full-text indices on single attributes of relations. Then it inspects the index of each attribute and combines the results
- ▶ implemented using the Oracle8i interMedia Text

[DISCOVER in 2002]

Indexing (3)

- ▶ **Double indexing:** In this approach two indexes are built:
 - ▶ Inverted index to map keywords to RIDs (tuple identifiers) resident on the disk
 - ▶ Index that maps RIDs to the graph nodes stored in memory

The approach assumes that the graph fits in memory. The in-memory node representation stores the RID

[BANKS in 2002]

MySQL Full-text Search

- ▶ In MySQL full-text search is performed using `MATCH()...AGAINST` syntax:
 - ▶ `MATCH` takes a comma-separated list of the column names to be searched
 - ▶ `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform:
`AGAINST(expr, [search_modifier])`
- ▶ Full-text index initialization, on existing tables or at creation time:
 - ▶ `ALTER TABLE tableName ADD FULLTEXT(C1,C2,...,Cn);`
 - ▶ `CREATE TABLE tableName (
 key INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
 C1 VARCHAR(200),
 C2 TEXT,
 FULLTEXT (C1, C2));`
- ▶ MySQL uses Ranking with Vector Spaces for ordinary full-text queries [MySQL documentation: <http://dev.mysql.com/doc/internals/en/full-text-search.html>]

MySQL Full-text Search (2)

- ▶ There are 3 types of full-text searches (*search_modifier*):
 1. **Natural Language (default)**
The stopwords list applies. In additions, words that are present in 50% or more of the rows are considered common and do not match
 2. **Boolean (IN BOOLEAN MODE)**
Allows to use some boolean operator:
 - [*no operator*] stands for **OR** (default)
 - + stands for **AND**
 - - stands for **NOT**
 - ...
 3. **Query Expansion Search (WITH QUERY EXPANSION)**
Words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search

MySQL Query Expansion Search Example

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body) AGAINST ('database');
```

id	title	body
5	MySQL vs. YourSQL	In the following database comparison ...
1	MySQL Tutorial	DBMS stands for DataBase ...

2 rows in set (0.00 sec)


```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' WITH QUERY EXPANSION);
```

id	title	body
1	MySQL Tutorial	DBMS stands for DataBase ...
5	MySQL vs. YourSQL	In the following database comparison ...

3 rows in set (0.00 sec)

Postgres Full-text Search

- ▶ Postgres supports 2 kinds of indexes that can be used for full text searches (since version 8.3 released in 2008)

1. Generalized Search Tree (Gist)-based index

- ▶ *Lossy*, may produce false matches because each document is represented in the index by a *fixed-length signature*. The signature is generated by hashing each word into a single bit in an n-bit string, with all these bits OR-ed together to produce an n-bit document signature. When two words hash to the same bit position there will be a false match.

2. Generalized Inverted Index (GIN)-based index

- ▶ *Not Lossy*, but their performance depends logarithmically on the number of unique words

- ▶ Which one?

- ▶ GIN index lookups are about three times faster than GiST
- ▶ GIN indexes take about three times longer to build than GiST
- ▶ GIN indexes are about ten times slower to update than GiST indexes
- ▶ GIN indexes are two-to-three times larger than GiST indexes

Classification

- ▶ In literature there are many approaches, that differ for:
 - ▶ Data Model
 - ▶ Indexing technique
 - ▶ Query Optimization
 - ▶ Results Ranking

- ▶ We choose to classify the main approaches basing on a rough classification of their data model

Based on [Qin L. et al. 2010. Keyword search in databases. Morgan & Claypool Publishers]

Data Model

▶ **Schema-based** to generate SQL queries

- ▶ DBXplorer, DISCOVER (in 2002)
- ▶ SPARK (in 2007)
- ▶ SQAK (in 2008)
- ▶ KEYMANTIC (in 2010)
- ▶ KEYRY (in 2011)

▶ **Graph-based** to provide the result tuples

- ▶ BANKS (in 2002)
- ▶ STAR (in 2009)
- ▶ PRECIS (in 2008)

Schema-based

- ▶ Database schema information used to issue SQL queries (CNs generation)
- ▶ 2 main steps:
 - ▶ Generation of a set of SQL queries
 - ▶ ranking the generated set of SQL queries & executing

Graph-based

- ▶ DB is modelled as a directed graph where each tuple in the DB is a node
 - ▶ Each foreign-key to primary-key link is modelled as a directed edge between the corresponding tuples
 - ▶ Both the nodes and edges may have weights

Also called “**schema-free**”: it does not request any further database schema assistance

Relation and Attribute Graph

- ▶ A relational schema is denoted as $R = \{R_1, \dots, R_n\}$
- ▶ R_i has a set of attributes $A_i = \{A_{ij} : 1 \leq j \leq k_i\}$
- ▶ A Database schema graph $G(V, E)$ is a directed graph
- ▶ There are two types of nodes in V :
 - ▶ relational node R **for each relation** in the schema
 - ▶ attribute node A **for each attribute** of each relation in the schema

[PRECIS in 2008]

Relation and Attribute Graph (2)

- ▶ There are two types of edges in E:
 - ▶ projection edges “K”, one for each attribute node emanating from its container relation node and ending at the attribute node
 - ▶ join edges “J” emanating from a relational node and ending at another relational node
- ▶ A database schema is represented as:
 $G(V,E)$ where $V = R \cup A$ and $E = K \cup J$
- ▶ Each edge has a weight assigned
 - ▶ K edges are undirected and have only one weight associated
 - ▶ J edges are directed so they have two weights, one for each direction
 - weights are both syntactic (i.e. a bound on the number of relations)
 - more semantic may be added for improving relevance to the query

Schema-based Approach

Partially taken from [Qin L. et al. 2010. Keyword search in databases. Morgan & Claypool Publishers]

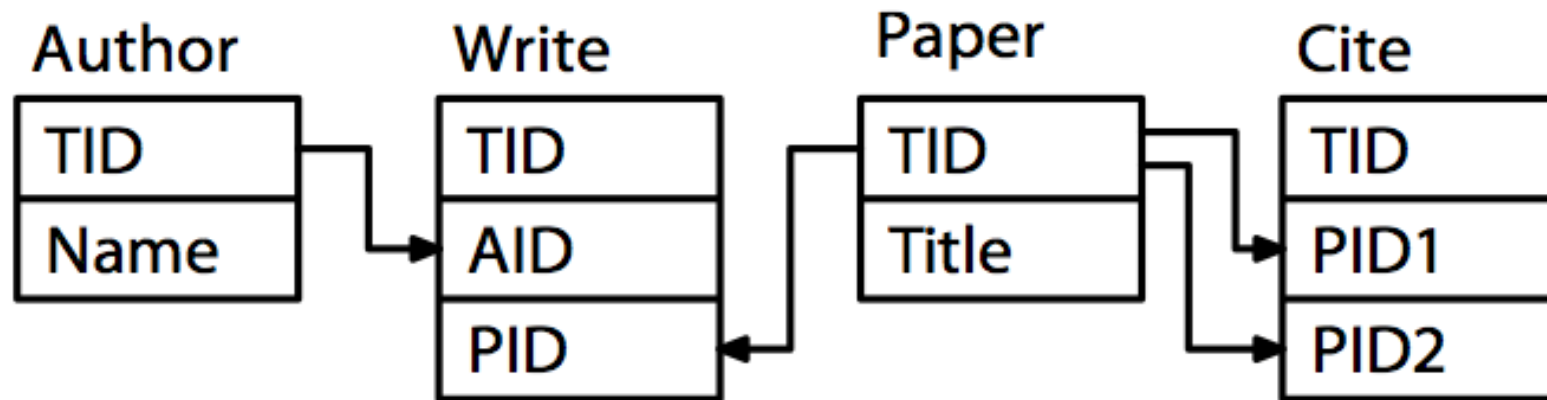


Useful definitions

► Schema Graph

- *A database schema can be represented using a directed graph $G_s(V,E)$, where V is the set of relation (table) schemas $\{R_1, R_2, \dots, R_n\}$ and E is the set of edges between the relation schemas*
- *An edge in the schema graph exists between two relation schemas R_i and R_j , if R_j has a foreign key referencing the primary key in R_i*
- *Note that multiple edges may exist between R_i and R_j in case R_j has multiple foreign keys referencing R_i*

Schema Graph



A portion of the DBLP schema graph

Useful definitions (2)

Keyword query (Q):

A set of keywords of size l $Q = \{k_1, k_2, \dots, k_l\}$

Keyword Relation:

given a keyword query Q and a schema graph G_S , a keyword relation $R_i\{K'\}$ is a subset of the relations r_i where the tuples contain a subset of the keywords in the query $K' \subseteq Q$, formally:

$$R_i\{K'\} = \{t \mid t \in r_i \wedge \forall k \in K', t \text{ contains } k \wedge \forall k \in (Q - K'), t \text{ does not contain } k\}$$

An empty keyword relation is a relation that does not contain any keyword, i.e., $K' = \emptyset$.

Useful definitions (3)

Candidate Network (CN):

given a keyword query Q and a schema graph G_S , a candidate network is a tree of keyword relations, i.e., for each two adjacent relations:

$R_i\{K_1\}$ and $R_j\{K_2\}$, $(R_i, R_j) \in E(G_S)$ or $(R_j, R_i) \in E(G_S)$.

A candidate network needs to satisfy the following conditions:

- ▶ **Total:** *a candidate network must contain each keyword in at least one keyword relation*
- ▶ **Minimal:** *the total condition is not satisfied anymore if any keyword relation is removed from the candidate network*

Example – CNs ($Q=\{Michelle, XML\}$, $T_{max}=5$)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

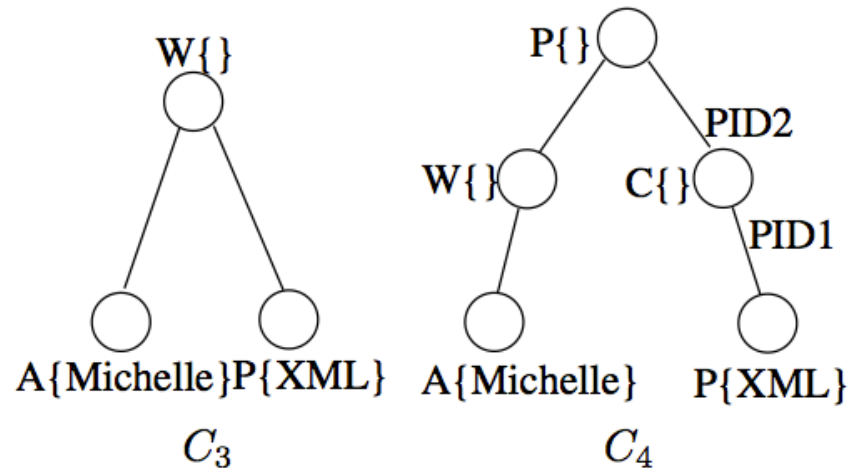
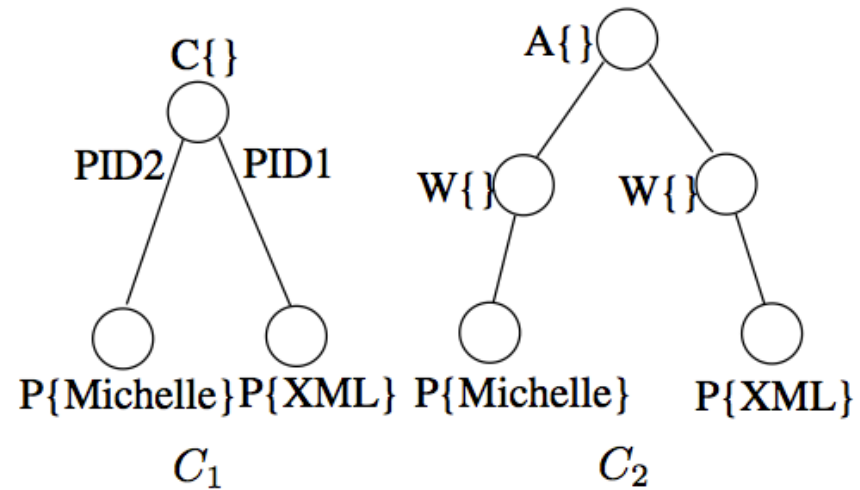
(a) Author

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



Useful definitions(4)

Minimal Total Joining Network of Tuples (MTJNT):

Given a keyword query and a relational database with schema graph G_s , a **joining network of tuples** (JNT) is a connected tree of tuples where every two adjacent tuples, $t_i \in r(R_i)$ and $t_j \in r(R_j)$ can be joined based on the foreign key reference defined on the relational schema R_i and R_j in G_s (either $R_i \rightarrow R_j$ or $R_j \rightarrow R_i$). An *MTJNT* is a joining network of tuples that satisfy the following two conditions:

- ▶ **Total:** *the joining network of tuples must contain each keyword in at least one tuple*
- ▶ **Minimal:** *the total condition is not satisfied anymore if any tuple is removed from the joining network of tuples*

T_{\max} : specify the maximum number T of tuples allowed in an MTJNT.

Example – MTJNTs ($Q=\{Michelle, XML\}$, $T_{max}=5$)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

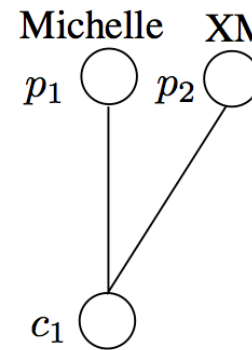
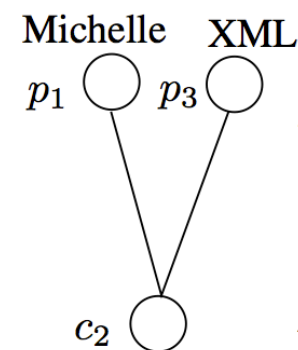
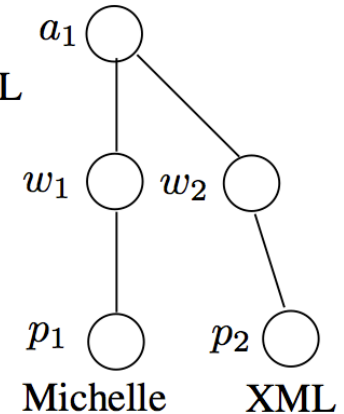
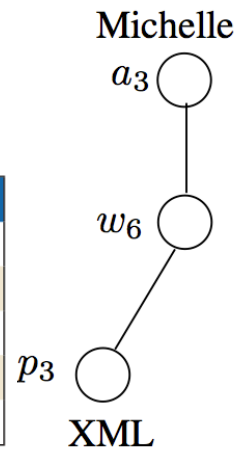
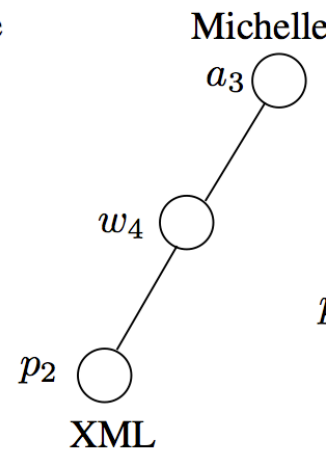
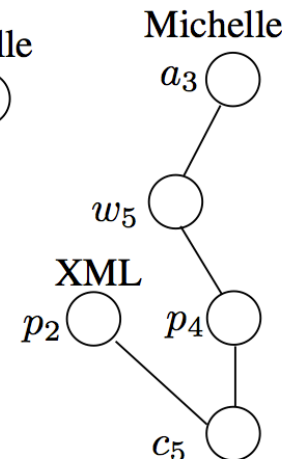
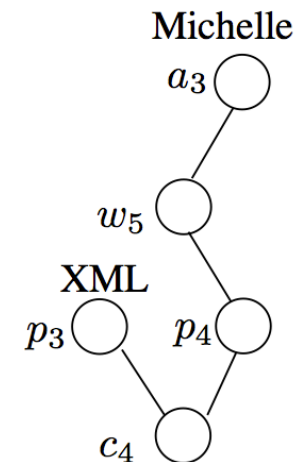
(a) Author

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

 T_1 (from C_1) T_2 (from C_1) T_3 (from C_2) T_4 (from C_3) T_5 (from C_3) T_6 (from C_4) T_7 (from C_4)

Candidate Network

- ▶ The two main steps of processing a keyword query are
 - ▶ **CN generation:** a set of candidate networks $C=\{C_1, C_2, \dots\}$ is generated over a graph schema G_s . The set of *CNs* shall be complete and duplication-free. The former ensures that all *MTJNTs* are found, and the latter is mainly for efficiency consideration
 - ▶ **CN evaluation:** all $C_i \in C$ are evaluated according to different semantics/environments for the keyword query (e.g., the AND/OR semantics, the all/top-k semantics, the static/stream environments)

Candidate Network Generation

- ▶ Given a keyword query Q over a relational database with schema graph G_s , algorithms are designed to generate candidate networks $C=\{C_1, C_2, \dots\}$ that satisfy the following two conditions:
 - ▶ **Complete:** For each solution T of the keyword query, there exists a candidate network $C_i \in C$ that can produce T
 - ▶ **Duplication-Free:** For every two CNs $C_i \in C$ and $C_j \in C$, C_i and C_j are not isomorphic to each other

CN Generation in DISCOVER

The algorithm **evaluates all the relations** in order to **discover which of them are keyword relations**. The keyword relations are partial CNs with just one node. Then, in order to generate all CNs, the **algorithm expands the partial CNs generated to larger partial CNs until all CNs are generated**.

There are 3 pruning rules for partial CNs:

- ▶ **Rule-1:** Duplicated CNs are pruned (based on tree isomorphism)
- ▶ **Rule-2:** A CN can be pruned if it contains all the keywords and there is a leaf node, $R_j \{K'\}$, where $K' = \emptyset$, because it will generate results that do not satisfy the condition of minimality
- ▶ **Rule-3:** When there only exists a single foreign key reference between two relation schemas (for example, $R_i \rightarrow R_j$), CNs including $R_i \{K_1\} \rightarrow R_j \{K_2\} \leftarrow R_i \{K_3\}$ will be pruned, where K_1, K_2 , and K_3 are three subsets of Q , and $R_i \{K_1\}$, $R_j \{K_2\}$, and $R_i \{K_3\}$ are keyword relations*

*The Rule-3 reflects the fact that the primary key defined on R_i and a tuple in the relation of $R_j \{K_2\}$ must refer to the same tuple appearing in both relations $R_i \{K_1\}$ and $R_i \{K_3\}$

Algorithm Discover-CNGen (Q, T_{\max}, G_S)

Input: an l -keyword query $Q = \{k_1, k_2, \dots, k_l\}$, the size control parameter T_{\max} , the schema graph G_S .

Output: the set of CNs $\mathcal{C} = \{C_1, C_2, \dots\}$.

```

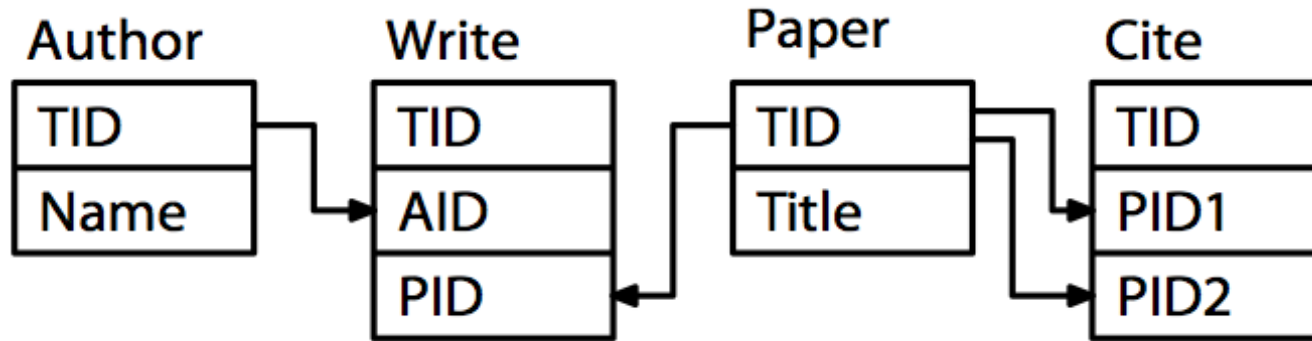
1:  $Q \leftarrow \emptyset; \mathcal{C} \leftarrow \emptyset$ 
2: for all  $R_i \in V(G_S), K' \subseteq Q$  do
3:    $Q.enqueue(R_i\{K'\})$ 
4: while  $Q \neq \emptyset$  do
5:    $T \leftarrow Q.dequeue()$ 
6:   if  $T$  is minimal and total and  $T$  does not satisfy Rule-1 then
7:      $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$ ; continue
8:   if the size of  $T < T_{\max}$  then
9:     for all  $R_i \in T$  do
10:      for all  $(R_i, R_j) \in E(G_S)$  or  $(R_j, R_i) \in E(G_S)$  do
11:         $T' \leftarrow T \cup (R_i, R_j)$ 
12:        if  $T'$  does not satisfy Rule-2 or Rule-3 then
13:           $Q.enqueue(T')$ 
14: return  $\mathcal{C}$ ;
```

Algorithm Discover-CNGen (Q, T_{\max}, G_S)

- ▶ Lines 1-3 initialize a queue Q that maintains all partial CN s to be a list of trees with size 1 that contain any subset of keywords
- ▶ From line 4, each partial tree T is dequeued from Q , iteratively
- ▶ Lines 6-7 check whether T is a valid CN that has not been generated before. If so, it is added to the result set C and there is no need to further expand T
- ▶ Lines 8-13 expand T by adding an edge from any relation R_i in T to another new relation R_j , and form a new partial CN T' . T' is enqueued for further expansion if it does not satisfy the pruning rules 2 or 3

Algorithm Discover-CNGen (example)

Let us Consider the 2-keyword query $Q=\{\text{Michelle, XML}\}$ DB with schema:



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

Algorithm Discover-CNGen (example 2)

$Q = \{\text{Michelle}, \text{XML}\}$

Note that we show only the generation of a CN

- ▶ First, we enqueue the set of partial CNs with size 1
 - ▶ $T_1 = A\{\text{Michelle}\}$
- ▶ Expanding T_1 , as there is an edge $A \rightarrow W$ in G_s , we can add the corresponding edge in T_1 and form another partial CN
 - ▶ $T_2 = A\{\text{Michelle}\} \text{ join } W\{\}$
- ▶ When expanding T_2 , we can add edges: $W\{\} \leftarrow A\{\text{XML}\}$, $W\{\} \leftarrow P\{\text{Michelle}\}$ and $W\{\} \leftarrow P\{\text{XML}\}$, and obtain three partial trees:
 - ▶ $T_3 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } A\{\text{XML}\}$
 - ▶ $T_4 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } P\{\text{Michelle}\}$
 - ▶ $T_5 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } P\{\text{XML}\}$

Algorithm Discover-CNGen (example 3)

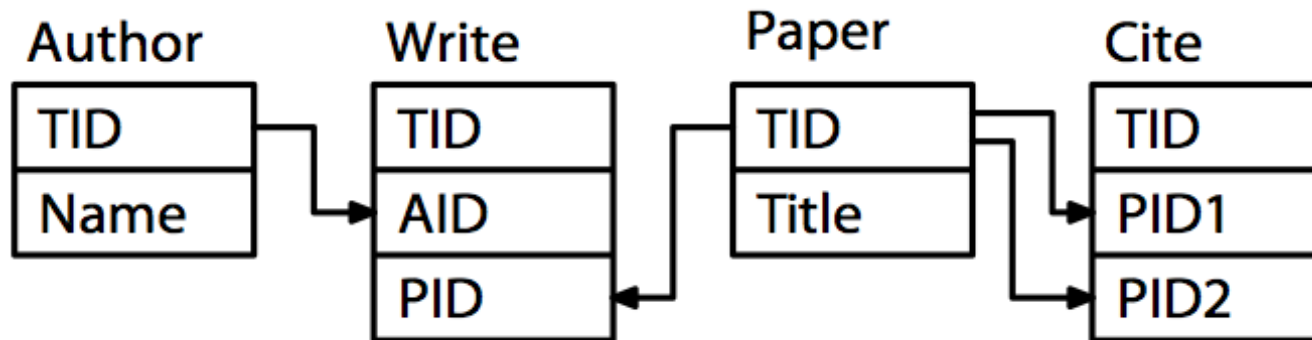
- ▶ $T_3 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } A\{\text{XML}\}$
 - ▶ W has only one foreign key referencing to A
 - ▶ satisfies Rule-3, thus it will be pruned (line 12)

- ▶ $T_4 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } P\{\text{Michelle}\}$
 - ▶ Not total and does not satisfy any pruning conditions
 - ▶ enqueued for further expansion (line 13)

- ▶ $T_5 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } P\{\text{XML}\}$
 - ▶ a valid CN added to the final result set (line 7)

Algorithm Discover-CNGen (example 4)

$$T_5 = A\{\text{Michelle}\} \text{ join } W\{\} \text{ join } P\{\text{XML}\}$$



TID	Name
<i>a</i> ₁	Charlie Carpenter
<i>a</i> ₂	Michael Richardson
<i>a</i> ₃	Michelle

(a) Author

TID	Title
<i>p</i> ₁	Contributions of Michelle
<i>p</i> ₂	Keyword Search in XML
<i>p</i> ₃	Pattern Matching in XML
<i>p</i> ₄	Algorithms for TopK Query

(b) Paper

TID	AID	PID
<i>w</i> ₁	<i>a</i> ₁	<i>p</i> ₁
<i>w</i> ₂	<i>a</i> ₁	<i>p</i> ₂
<i>w</i> ₃	<i>a</i> ₂	<i>p</i> ₂
<i>w</i> ₄	<i>a</i> ₃	<i>p</i> ₂
<i>w</i> ₅	<i>a</i> ₃	<i>p</i> ₁
<i>w</i> ₆	<i>a</i> ₃	<i>p</i> ₃

(c) Write

TID	PID1	PID2
<i>c</i> ₁	<i>p</i> ₂	<i>p</i> ₁
<i>c</i> ₂	<i>p</i> ₃	<i>p</i> ₁
<i>c</i> ₃	<i>p</i> ₂	<i>p</i> ₃
<i>c</i> ₄	<i>p</i> ₃	<i>p</i> ₄
<i>c</i> ₅	<i>p</i> ₂	<i>p</i> ₄

(d) Cite

Algorithm Discover-CNGen performance

The above algorithm can generate a complete and duplication-free set of CNs, but the **cost of generating the set of CNs is high**, due to:

- ▶ Given a I-keyword query and a large T_{\max} over a complex database schema G_s the number of CNs to be generated can be very large. The number of CNs increases exponentially while any of these factor increases
- ▶ Adding an arbitrary edge to an arbitrary position in a partial CN when expanding, makes the number of temporal results extremely large, while only few of them will contribute to the final results. Most of the results end up with a partial CN of size T_{\max} that does not contain all keywords, so is not total
- ▶ The algorithm needs a large number of tree isomorphism tests, which is costly. This is because the isomorphism test will only be performed when a valid MTJNT is generated. As a result, all isomorphisms of an MTJNT will be generated and checked

Rightmost algorithm

- ▶ In order to solve the above problems, the rightmost algorithm was proposed:
 - ▶ reduces the number of partial results generated by expanding the partial CNs
 - ▶ avoids isomorphism testing by assigning a proper expansion order

[Markowetz et al., Keyword search on relational data streams. In *Proc. 2007 ACM SIGMOD Int. Conf. On Management of Data*, pages 605–616, 2007]

CN evaluation

After generating all candidate networks, we have to evaluate them in order to obtain the final result. The approaches can be split in two categories:

- ▶ **Getting all MTJNTs in the DB:** all MTJNTs are evaluated upon the set of CN generated by specifying a proper execution plan
- ▶ **Getting top-k MTJNTs in the DB:** since it can be ineffective to present users a huge number of MTJNTs, only top-k MTJNTs are returned

Getting all MTJNTs

- ▶ **Greedy algorithm** based on the following observations:
 - ▶ Sub-expressions (intermediate CNs) that are shared by most CNs should be evaluated first
 - ▶ Sub-expressions that may generate the smallest number of results should be evaluated first

The size of results for any sub-expression e is denoted as $estimate(e)$ and the number of CNs that share the sub-expression e is denoted as $frequency(e)$. For any sub-expression e (a and b are constants):

$$score(e) = frequency(e)^a / (\log(estimate(e)))^b$$

Getting all MTJNTs (2)

Algorithm

1. First evaluates all sub-expressions of size 1
 2. Iteratively evaluates sub-expressions of size 2 (join) and selects the one with the highest score value until all CNs are evaluated
 3. Newly evaluated sub-expression must be a join with two sub-expressions already evaluated
- ▶ The MTJNTs are then ranked by measuring the number of joins involved
 - ▶ The smaller the number of joins, the higher the rank because it represents a closer association

[DISCOVER in 2002]

Greedy Algorithm (example)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

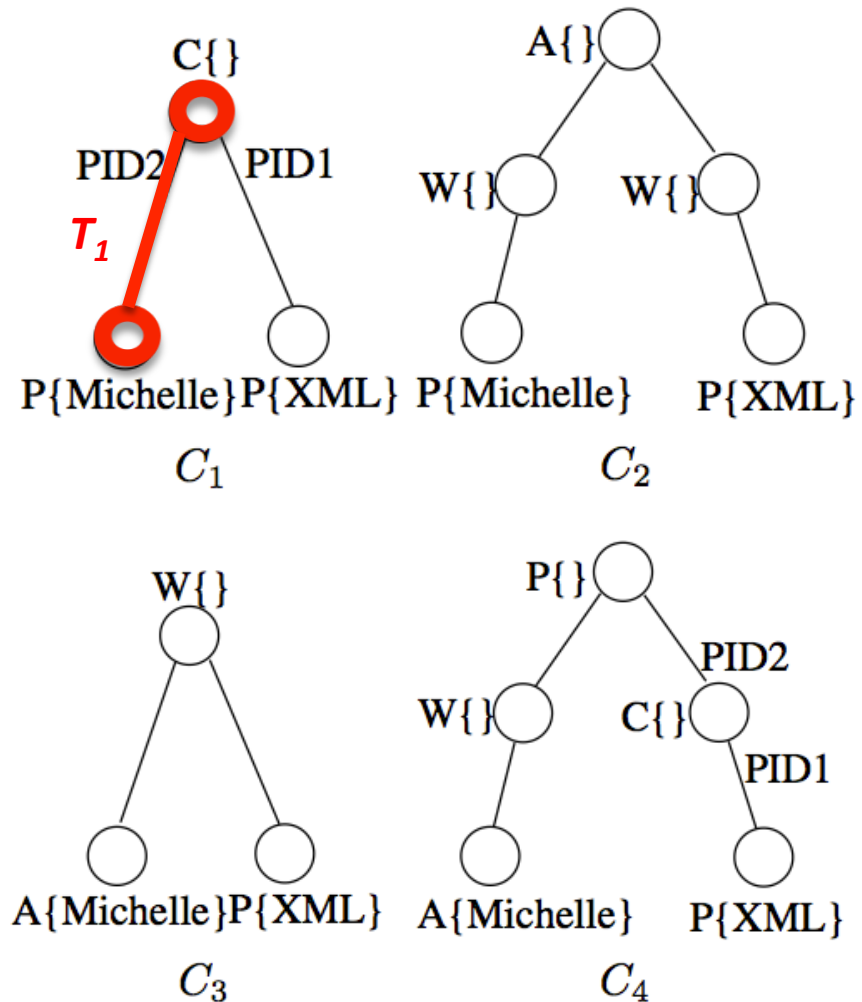
(a) Author

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



Greedy Algorithm (example)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

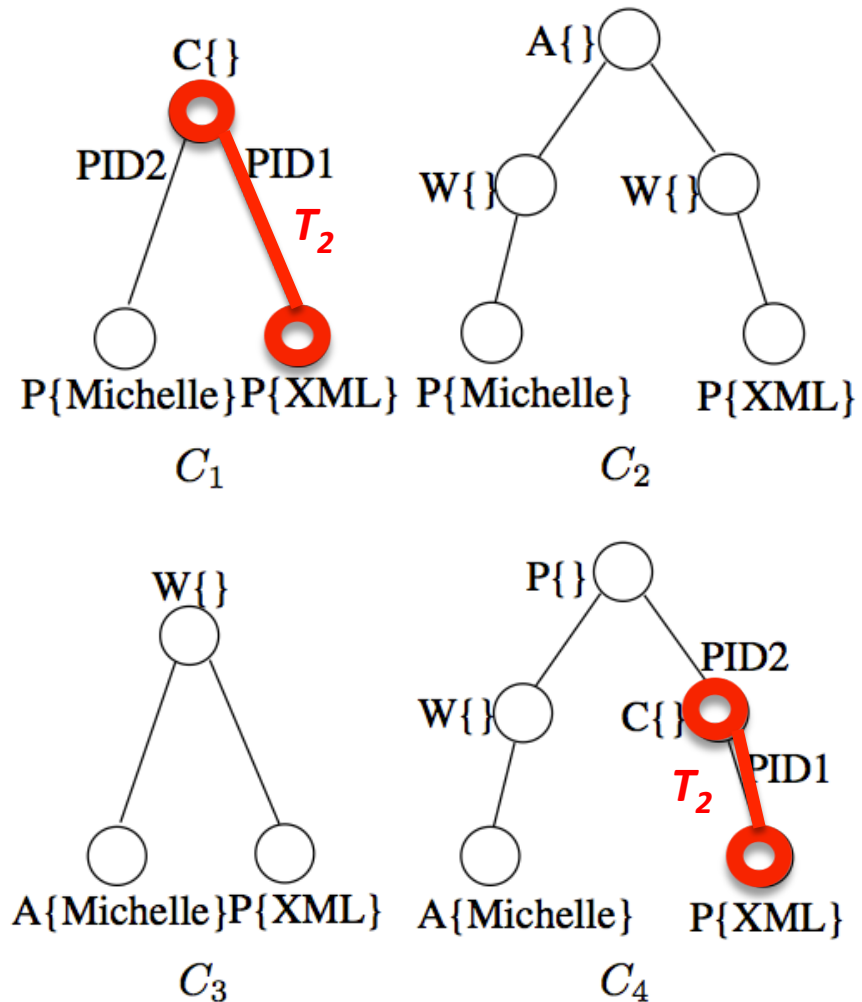
(a) Author

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



Greedy Algorithm (example)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

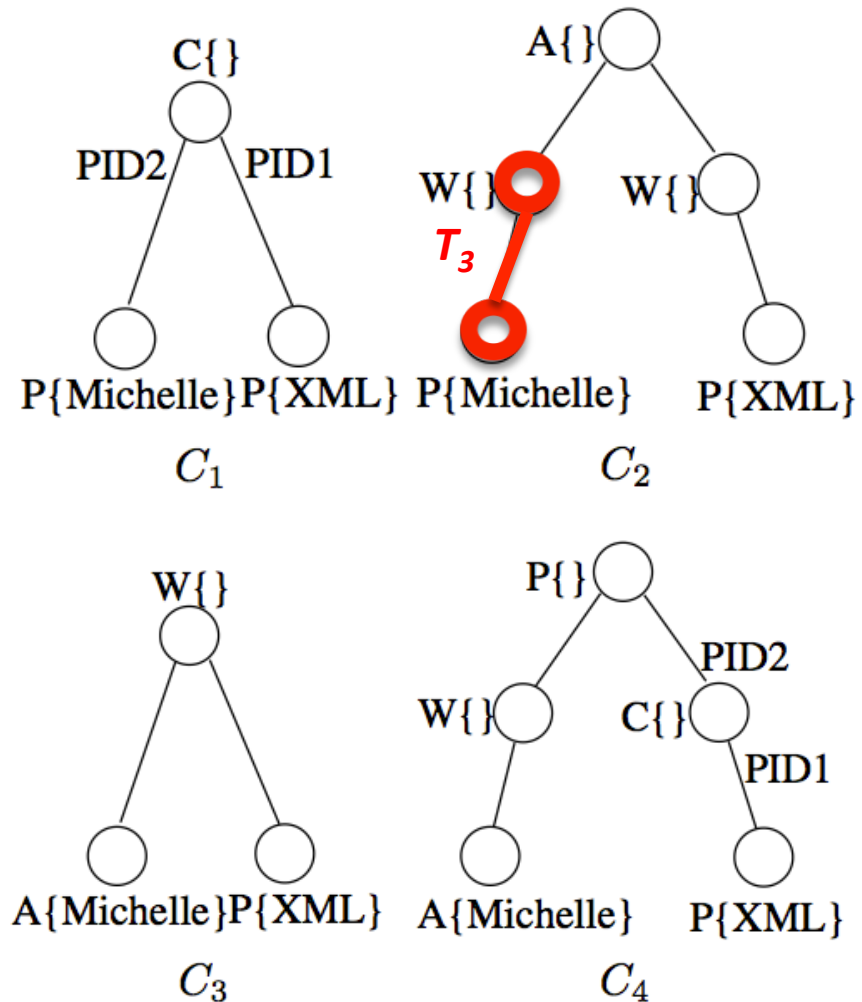
(a) Author

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



Greedy Algorithm (example)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

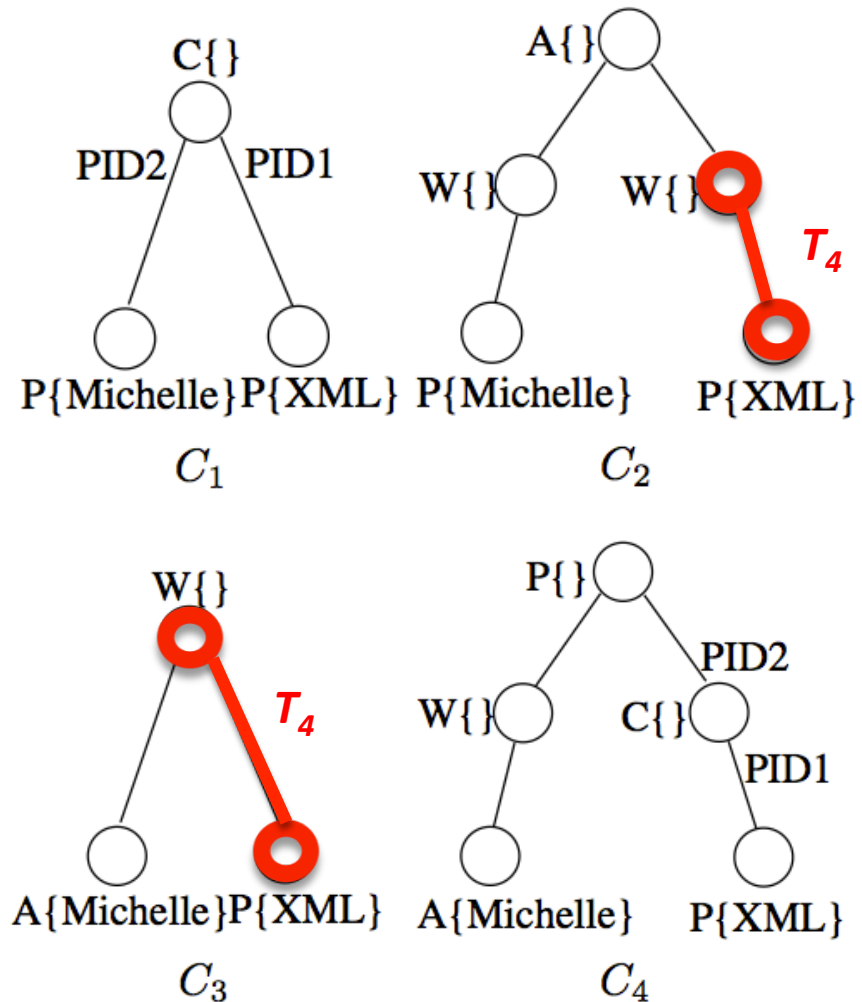
(a) Author

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



Greedy Algorithm (example)

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	Name
a_1	Charlie Carpenter
a_2	Michael Richardson
a_3	Michelle

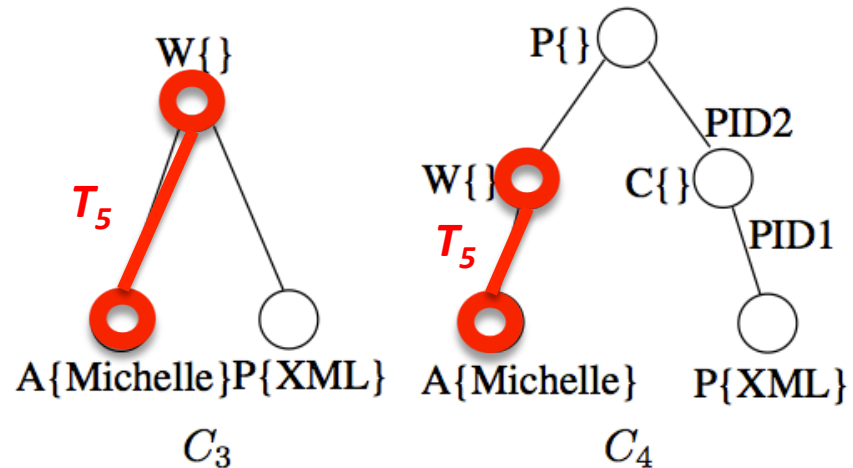
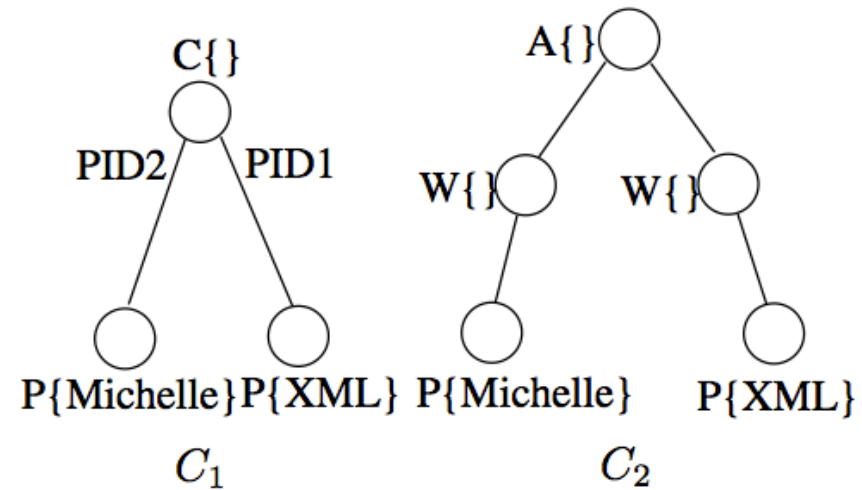
(a) Author

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



Greedy Algorithm (example)

	intermediate CN	N° of sharing	Estimate(e)	Score(e) (a=1,b=0.3)
$T_1)$	P{Michelle} – C{}	1	5/4	2
$T_2)$	P{XML} – C{}	2	5/2	2.6
$T_3)$	P{Michelle} – W{}	1	3/2	1.7
$T_4)$	P{XML} – W{}	2	3	2.5
$T_5)$	A{Michelle} – W{}	2	2	2.9

*Estimate(e):

- we can get the **sizes of $R_i\{K\}$ from the master index**
- we know the **selectivity** of the primary to foreign key joins, which can be calculated from the sizes of the relation

Greedy Algorithm (example)

▶ Execution plan:

- ▶ $C_1: P\{Michelle\} \text{ join } C\} \text{ join } P\{XML\}$
 - $T_1 \leftarrow P\{Michelle\} \text{ join } C\}$
 - $T_2 \leftarrow P\{XML\} \text{ join } C\}$
 - $\text{Score}(T_2) > \text{Score}(T_1) \rightarrow \text{execute } T_2$
 - ▶ $C_1 \leftarrow T_2 \text{ join } P\{Michelle\}$
- ▶ $C_3: A\{Michelle\} \text{ join } W\} \text{ join } P\{XML\}$
 - $T_5 \leftarrow A\{Michelle\} \text{ join } W\}$
 - $T_4 \leftarrow P\{XML\} \text{ join } W\}$
 - $\text{Score}(T_5) > \text{Score}(T_4) \rightarrow \text{execute } T_5$
 - ▶ $C_1 \leftarrow T_5 \text{ join } P\{XML\}$
- ▶ $C_2: P\{Michelle\} \text{ join } W\} \text{ join } A\} \text{ join } W\} \text{ join } P\{XML\}$
 - $T_3 \leftarrow P\{Michelle\} \text{ join } W\}$
 - $T_4 \leftarrow P\{XML\} \text{ join } W\}$
 - ▶ $C_2 \leftarrow T_3 \text{ join } A\} \text{ join } T_4$
- ▶ $C_4: A\{Michelle\} \text{ join } W\} \text{ join } P\} \text{ join } C\} \text{ join } P\{XML\}$
 - $T_5 \leftarrow A\{Michelle\} \text{ join } W\}$
 - $T_2 \leftarrow P\{XML\} \text{ join } C\}$
 - ▶ $C_4 \leftarrow T_5 \text{ join } P\} \text{ join } T_2$

Getting top-k MTJNTs

- ▶ The aim of all the algorithms is to find a proper ordering criteria of generating MTJNTs in order to stop early before all MTJNTs are generated
- ▶ The **Sparse** algorithm, the **Single-Pipelined** algorithm, and the **Global-Pipelined** algorithm are **based on attribute level ranking functions** [DISCOVER-II in 2003]
- ▶ The **Skyline-Sweeping** algorithm and the **Block-Pipelined** algorithm are **based on tree level ranking functions**, which are not tuple-monotonic [SPARK in 2007]

Attribute level ranking function

- ▶ Given an MTJNT T , $\text{size}(T)$ is the number of its tuples, and a keyword query Q , the tuple level ranking function first assigns each text attribute for tuples in T an individual score and then combines them together to get the final score:

$$\text{score}(T, Q) = \frac{\sum_{a \in T} \text{score}(a, Q)}{\text{size}(T)}$$

$\text{score}(a, Q)$ is the IR-style relevance score for the virtual document in the MTJNT T and Q that is defined as:

$$\text{score}(a, Q) = \sum_{k \in Q \cap a} \frac{1 + \ln(1 + \ln(\text{tf}(a, k)))}{(1 - s) + s \cdot \frac{\text{dl}(a)}{\text{avdl}(\text{Rel}(a))}} \cdot \text{idf}(a, k)$$

$$\text{idf}(a, k) = \ln\left(\frac{N(\text{Rel}(a))}{\text{df}(\text{Rel}(a), k) + 1}\right)$$

Tree level ranking function

- ▶ In the attribute level ranking functions, each text attribute of an MTJNT is considered as a virtual document
- ▶ Tree level ranking functions consider the whole MTJNT as a virtual document rather than each individual text attribute

$$\text{score}(T, Q) = \text{score}_a(T, Q) \cdot \text{score}_b(T, Q) \cdot \text{score}_c(T, Q)$$

where:

- ▶ $\text{score}_a(T, Q)$: the TF-IDF score
- ▶ $\text{score}_b(T, Q)$: the completeness score
- ▶ $\text{score}_c(T, Q)$: the size normalization score

Graph-based Approach

Partially taken from [Qin L. et al. 2010. Keyword search in databases. Morgan & Claypool Publishers]



Data Model

- ▶ DB modeled as a direct graph:
 - ▶ Each tuple is a node
 - ▶ Nodes has weight, also called “prestige” and it is a measure of the number of pointers to that node (similar to PageRank)
 - ▶ Each foreign-key to primary-key link is an edge
 - ▶ Forward edges: reflects the strength of the proximity relationship between two tuples
 - ▶ is set to 1 by default
 - ▶ can be set to any desired value to reflect the importance of the edge
 - ▶ Backward edges (v,u): the weight is proportional to the number of links to v from the nodes of the same type as u

[BANKS in 2002]

Problem Definition

Given a directed weighted data graph GD , an l -keyword query consists of a set of $l \geq 2$ keywords

► **Problem:**

find a set of subgraphs of G_D : $R(G_D, Q) = \{R_1(V, E), R_2(V, E), \dots\}$, where each $R_i(V, E)$ is a connected subgraph of G_D that contains all the l keywords

► **Solution:**

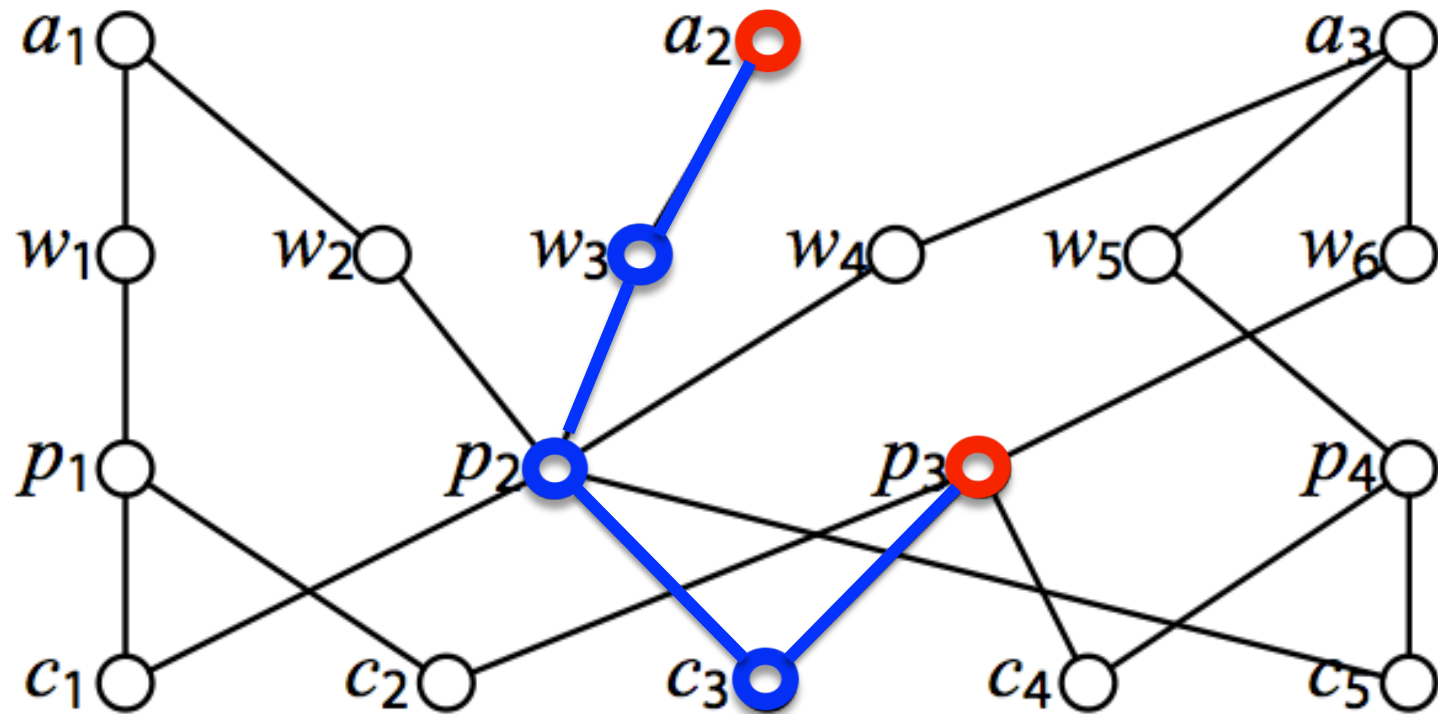
Find a reduced tree that contains all the keywords and satisfies one of this requirements (semantics):

- **Steiner Tree-Based Semantics** (optimal, but **NP-complete**)
- **Distinct Root-Based Semantics** (not optimal, but easier to compute)

Minimum Steiner Tree Problem

- ▶ Given a weighted graph $G = (V, E)$ and a set $R \subseteq V$, our goal is to determine the least cost connected subgraph spanning R . Vertices in R are called terminal nodes and those in $V \setminus R$ are called Steiner vertices
- ▶ we are free to use non terminal vertices of the graph (the Steiner nodes) in order to determine the Steiner tree
- ▶ NP-complete

Minimum Steiner Example



(e) Tuple Connections



Steiner Tree-based Keyword Search

- ▶ Answer to Q (called a Q-subtree) is defined as any subtree T of GD that is reduced with respect to Q

- ▶ Steiner Tree-Based Semantics

- ▶ Weight of a Q-subtree is defined as the total weight of the edges in the tree:

$$w(T) = \sum_{\langle u, v \rangle \in E(T)} w_e(\langle u, v \rangle)$$

- ▶ where $E(T)$ is the set of edges in T . The l-keyword query finds all (or top-k) Q-subtrees in weight increasing order, where the weight denotes the cost to connect the l keywords. Under this semantics, finding the Q-subtree with the smallest weight is the well-known *optimal steiner tree problem*, which is NP-complete

Steiner Tree-based Keyword Search

- ▶ Algorithms

- ▶ Backward Search

- ▶ the first tree returned is an l -approximation of the optimal steiner tree

- [BANKS in 2002]

- ▶ Dynamic Programming

- ▶ finds the optimal (top-1) steiner tree in time:

- $O(3l n + 2l ((l + \log n)n + m))$

- [14] in 2007

- ▶ Polynomial Delay

- [STAR in 2009]

- [15] SIGMOD in 2008

Empirical Performance Comparison

Characteristics of the evaluation data sets

Dataset	Size (MBs)	Relations	in <i>thousands</i>		
			$ V $	$ E $	$ T $
MONDIAL	16	28	17	56	12
IMDb	459	6	1673	6075	1748
Wikipedia	391	6	206	785	750

Legend

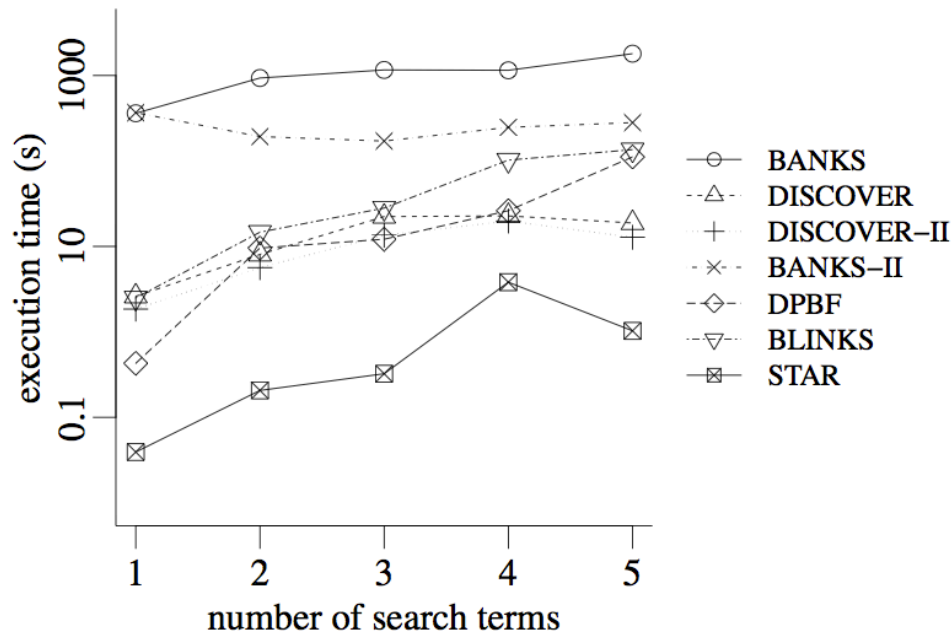
- $|V|$ number of nodes (tuples) in data graph
- $|E|$ number of edges (foreign keys) in data graph
- $|T|$ number of unique terms

[16] TDKE in 2012

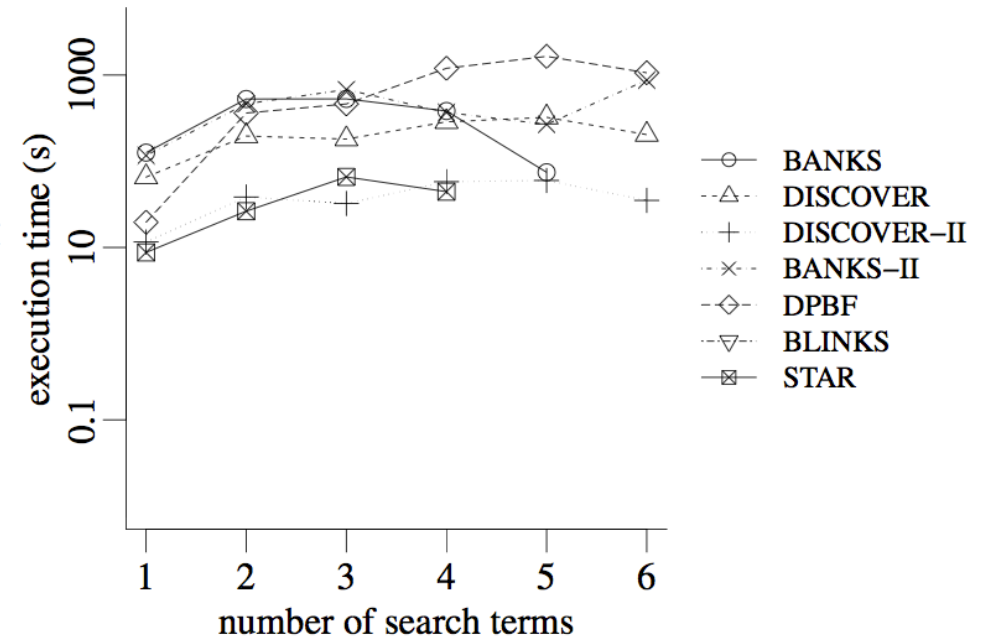
Empirical Performance Comparison

- Retrieval depth: 100 results
- y-axis is in a log scale

(a) MONDIAL



(b) Wikipedia



[16] TDKE in 2012

References

- [1] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2009. Keyword search in databases: the power of RDBMS. (SIGMOD '09), Carsten Binnig and Benoit Dageville (Eds.). ACM, New York, NY, USA, 681-694.
- [2] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2010. Keyword search in databases. Morgan & Claypool Publishers, 2010. ISBN 160845195X, 9781608451951
- [3] S. Agrawal, S. Chaudhuri, and G. Das, “DBXplorer: A system for keyword-based search over relational databases”, In *ICDE*, 2002.
- [4] Vagelis Hristidis and Yannis Papakonstantinou. 2002. Discover: keyword search in relational databases. (VLDB '02). VLDB Endowment 670-681.
- [5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, “Keyword Searching and Browsing in Databases using BANKS”, *ICDE*, 2002.
- [6] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. 2007. Spark: top-k keyword query in relational databases (SIGMOD '07). ACM, New York, NY, USA, 115-126.
- [7] Ken Q. Pu and Xiaohui Yu. 2009. FRISK: Keyword Query Cleaning and Processing in Action. (ICDE '09). IEEE Computer Society, Washington, DC, USA, 1531-1534. DOI=10.1109/ICDE.2009.139
- [8] Sandeep Tata and Guy M. Lohman. 2008. SQAK: doing more with keywords. (SIGMOD '08). ACM, New York, NY, USA, 889-902. DOI=10.1145/1376616.1376705
- [9] Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. 2008. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal* 17, 1 (January 2008), 117-149.

References

- [10] V. Hristidis, L. Gravano, and Y. Papakonstantinou, “Efficient ir-style keyword search over relational databases”, In *VLDB, 2003*.
 - [11] A. Balmin, V. Hristidis, Y. Papakonstantinou, “ObjectRank: Authority-Based Keyword Search in Databases”, *VLDB, 2004*.
 - [12] Markowetz et al., Keyword search on relational data streams. In *Proc. 2007 ACM SIGMOD Int. Conf. On Management of Data*, pages 605–616, 2007
 - [13] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. In *Networks*, 1972.
 - [14] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min cost connected trees in databases. In *Proc. 23rd Int. Conf. on Data Engineering*, pages 836–845, 2007
 - [15] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In *Proc. 2008 ACM SIGMOD Int. Conf. On Management of Data*, pages 927–940, 2008
 - [16] Coffman J, Weaver A C” An Empirical Performance Evaluation of Relational Keyword Search Techniques”. To appear in *IEEE TKDE*, ISSN : 1041-4347. 2012
 - [17] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, “STAR: Steiner-Tree Approximation in Relationship Graphs,” in *ICDE '09*, March 2009, pp. 868–879.
 - [18] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, Y. Velegrakis: Keyword search over relational databases: a metadata approach. *SIGMOD Conference 2011*: 565-576
 - [19] S. Bergamaschi, E. Domnori, F. Guerra, M. Orsini, R. Trillo Lado, Y. Velegrakis: Keymantic: Semantic Keyword-based Searching in Data Integration Systems. *PVLDB 3(2)*: 1637-1640 (2010)
-

References

- [20] Sonia Bergamaschi, Francesco Guerra, Silvia Rota, Yannis Velegrakis: A Hidden Markov Model Approach to Keyword-Based Search over Relational Databases. ER 2011: 411-420S.
- [21] Silvia Rota, Sonia Bergamaschi, Francesco Guerra: The list Viterbi training algorithm and its application to keyword search over databases. CIKM 2011: 1601-1606
- [22] Alex Wright: Searching the deep web. Commun. ACM 51(10):14-15 (2008)
- [23] <http://dev.mysql.com/doc/internals/en/full-text-search.html>
- [24] <http://www.postgresql.org/docs/8.3/static/textsearch-indexes.html>
- [25] Markowetz et al., Keyword search on relational data streams. In *Proc. 2007 ACM SIGMOD Int. Conf. On Management of Data*, pages 605–616, 2007