Keyword query interpretation over structured data

Advanced Methods of IR

Elena Demidova

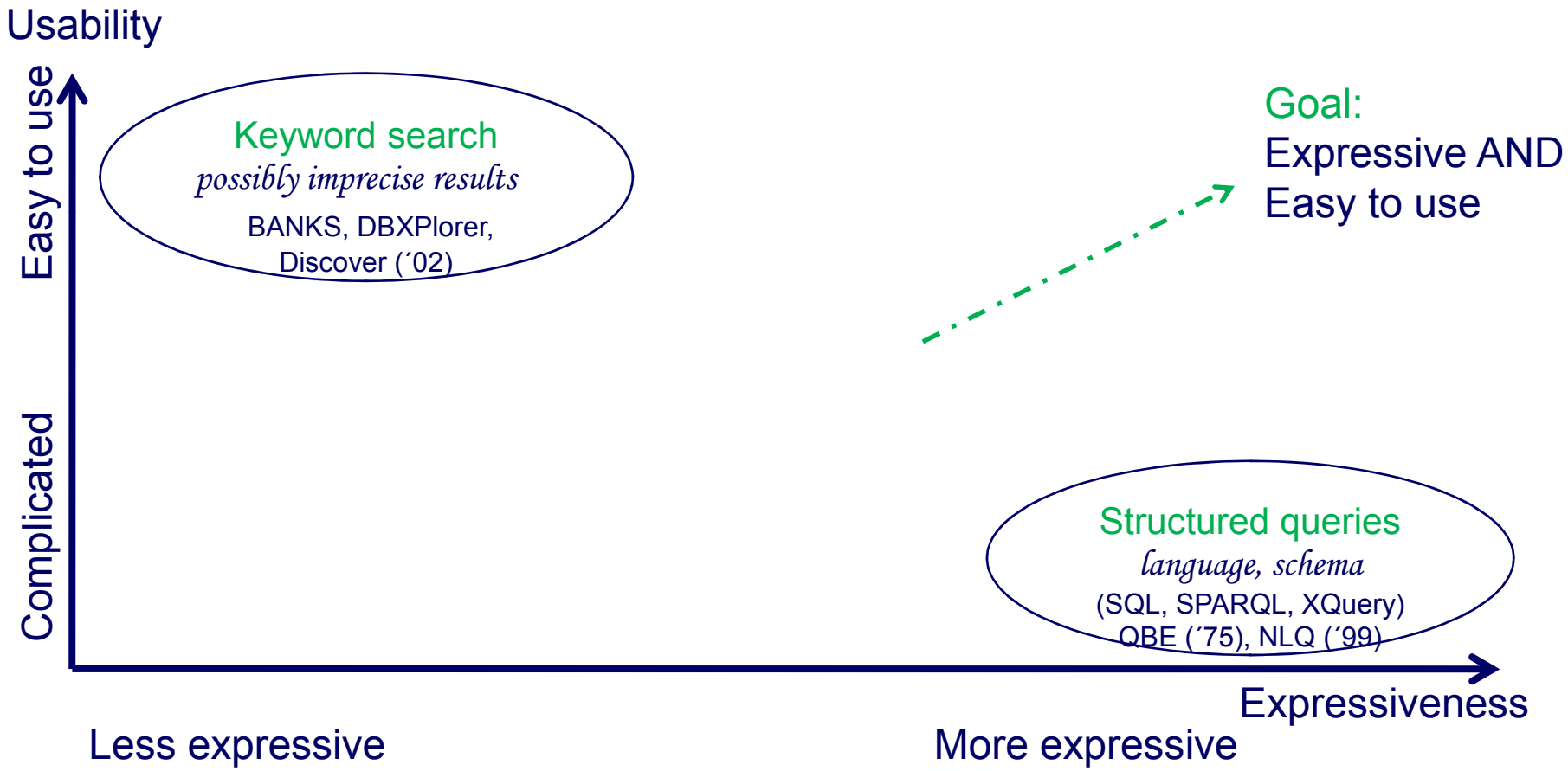
SS 2014

Materials used in the slides:

Jeffrey Xu Yu, Lu Qin, Lijun Chang. Keyword Search in Databases. Synthesis Lectures on Data Management. Morgan & Claypool Publishers. 2009.
Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In Proc. of the VLDB 2002.
Sandeep Tata and Guy M. Lohman. SQAK: doing more with keywords. In Proc. of the 2008 ACM SIGMOD.

adapted from: [Tata et. al 2008]

Database queries: expressiveness vs. usability



Database queries: expressiveness vs. usability

- **Database queries**:
  - knowledge of database schema
  - knowledge of query language syntax
- **Keyword search**:
  - Easy-to-use but imprecise
  - Ambiguous: unclear information need
- **Keyword query interpretation**:
  - Automatically translate keyword query in a (most likely) structured query (-ies)

DPLP example and definitions from: [Yu et. al 2009]

From keywords to structured queries: An example

K = {Michelle, XML}

1. Identify tuples / attributes containing keywords

$\sigma_{\text{michelle} \in \text{name}}(\text{Author})$: *michelle*

$\sigma_{\text{xml} \in \text{title}}(\text{Paper})$: *xml*

$\sigma_{\text{michelle} \in \text{title}}(\text{Paper})$: *michelle*

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

2. Identify join paths to connect all keywords in the query

$Q = \sigma_{\text{michelle} \in \text{name}}(\text{Author}) \bowtie \text{Write} \bowtie \sigma_{\text{xml} \in \text{title}}(\text{Paper})$

Other paths?

From keywords to structured queries: An example

K = {Michelle, XML}

$Q = {}_{\text{michelle} \in \text{name}}(\text{Author}) \bowtie \text{Write} \bowtie \sigma_{\text{xml} \in \text{title}}(\text{Paper})$

The translation *K* - > *Q* requires:

1. Knowledge of the **schema graph** (tables, attributes, join paths)
2. Knowledge of **keyword occurrences**
3. **Efficient algorithms**

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

Definitions and notations: The schema graph

Schema graph: a directed graph G_s (V,E)

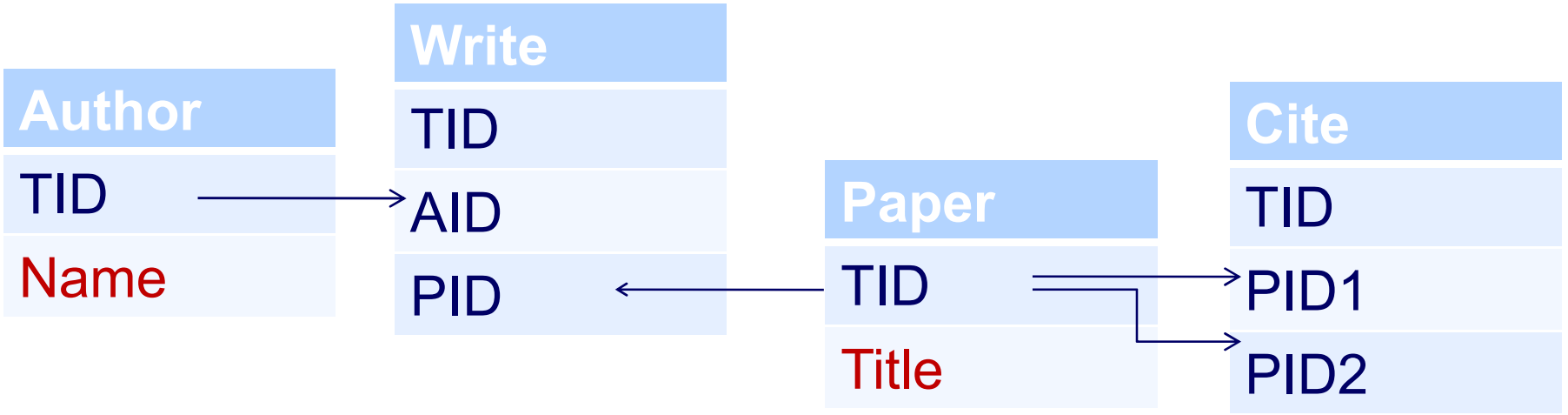
V – the set of relation schemas $\{R_1, R_2, \ldots, R_n\}$. An instance of a relation schema is a set of tuples (i.e. a database table).

E - the set of edges R_i -> R_j between two relation schemas. An edge is a primary key to foreign key relation.

TID – primary key attribute (i.e. tuple identifier)

Text attribute – an attribute allowing full-text search

An example: The DBLP schema graph



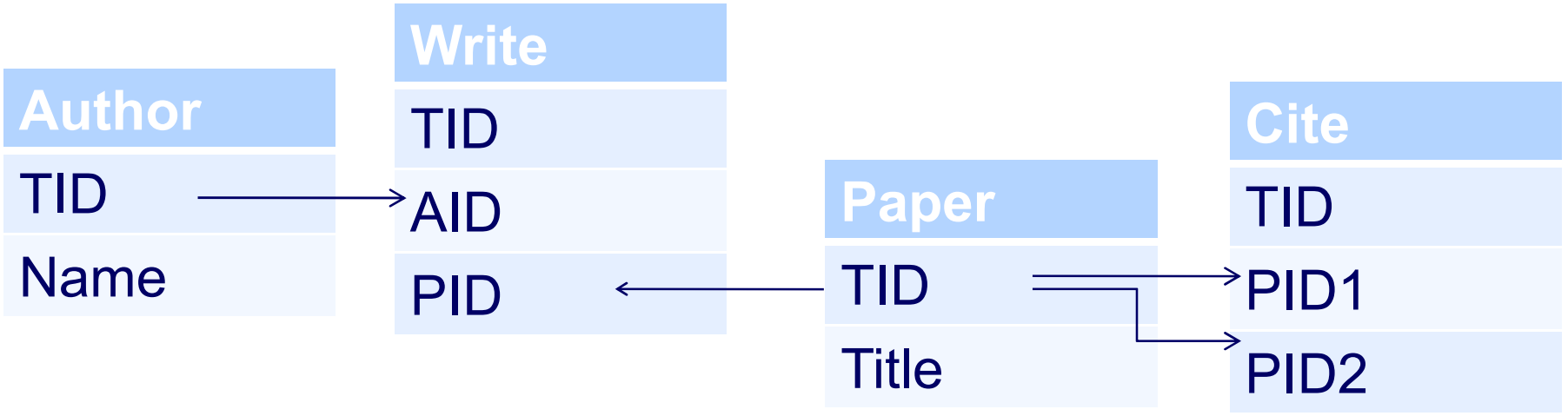
V = {Author, Write, Paper, Cite}

E = {Author.TID -> Write.AID, Paper.TID -> Write.PID,
Paper.TID -> Cite.PID1, Paper.TID -> Cite.PID2}

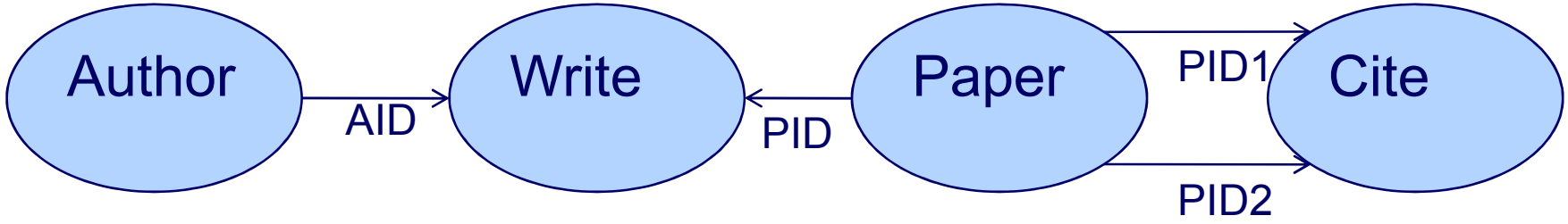
Primary keys: Author.TID, Write.TID. Paper.TID, Cite.TID

Text attributes: Author.Name, Paper.Title

An example: The DBLP schema graph



A simplified representation of the schema graph:



Definitions and notations: The database graph

The *database graph*: a directed graph G_D (V_t, E_t) on the schema graph *Gs*.

V_t – the set of tuples {t_1, t_2, … , t_n}.

E_t - the set of edges between tuples

Two tuples t_i and t_j are *connected* if there exists a foreign key (fk) reference t_i -> t_j or t_j -> t_i.

Two tuples t_i, t_j are *reachable* if there exists a sequence of connections between them, e.g. t_i -> t_1, …., t_n -> t_j.

The *distance* between two tuples dis(t_i, t_j) is the *minimum* number of connections between t_i, t_j.

An example: The DPLP database graph

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

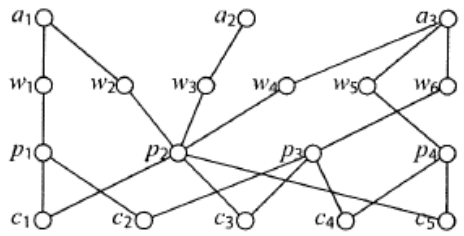
(b) Paper

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



(e) Tuple Connections

The *distance* between two tuples dis(t_i, t_j) is the *minimum* number of connections between t_i, t_j.

dis (a1, p4)?

Keyword query

A *l*-keyword query $K = \{k_1, k_2, \ldots, k_l\}$ –

a set of keywords of size *l*.

K semantics (typically): search for interconnected tuples that jointly contain $\{k_1, k_2, \ldots, k_l\}$.

How can we find the tuples containing $\{k_1, k_2, \ldots, k_l\}$ in a database?

Full-text search on a specific database attribute

Full-text search on specific attribute is supported by major databases, e.g. using **contains predicate**:

contains (*R.A*, k_i) – the predicate selecting all tuples from a relation *R* that contain keyword k_i in the text attribute *R.A*.

SELECT * FROM Author WHERE contains(Author.Name, „Michelle“);

String comparison operators (e.g. ***like***):

SELECT * FROM Author WHERE Author.Name LIKE '%michelle%';

Differences?

DB indexing

Inverted index using Lucene, Solr, …

Granularity:

Tuple level:

Dictionary		Postings		
Michelle	->	Author.a_3	Paper.p_1	…
XML	->	Paper.p_2	Paper.p_3	…

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

Attribute level:

Dictionary		Postings		
Michelle	->	Author.Name	Paper.Title	…
XML	->	Paper.Title	…	

SQL full-text search vs. indexing

Built-in full-text search capabilities are database dependent

Contains can use indexes but is neither flexible, nor not generally available

String comparison operators can require sequential scan (e.g. like operator if the prefix is undefined)

Each textual attribute needs to be queried separately

In the global full-text index, the list of attributes is immediately available

Index construction cost

Storage cost (depends on the index granularity)

Interconnecting keywords: MTJNT

An answer to a *l*-keyword query is a Minimal Total Joining Network of Tuples (MTJNT)

JNT (Joining Network of Tuples) – a connected tree of tuples. Every two adjacent tuples t_i, t_j in JNT an be joined based on the fk-reference in the schema i.e. either R_i -> R_j or R_j -> R_i (ignoring direction)

TJNT (Total JNT) w.r.t. a *l*-keyword query *K* if it contains all keywords of *K*

MTJNT (Minimal TJNT) if no tuple can be removed such that JNT remains total

T_{max} – a size control parameter to define the maximum number of tuples in MTJNT

Keyword query answers: MTJNT examples

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

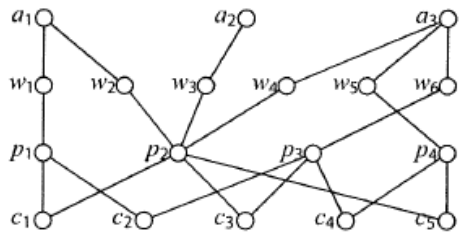
(b) Paper

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



(e) Tuple Connections

K = {Michele, XML}

T_{max} = 5

MTJNTs = {?}

Work in groups: 10 minutes

Keyword query answers: MTJNT examples

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

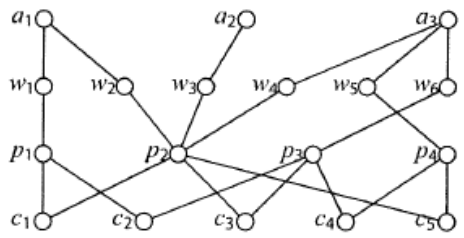
(b) Paper

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



(e) Tuple Connections

K = {Michele, XML}

T_{max} = 5

MTJNTs = {?}

contains (a_3, „Michelle“)

contains (p_1, „Michelle“)

contains (p_2, „XML“)

contains (p_3, „XML“)

Keyword query answers: MTJNT examples

K = {Michelle, XML}

T_{max} = 5

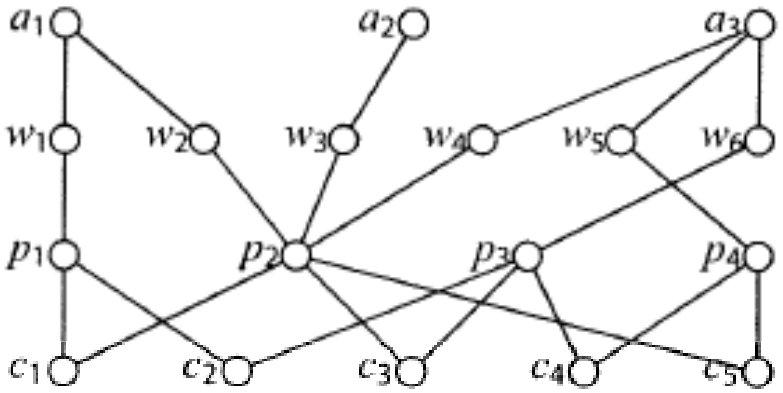
contains (a_3, „Michelle“)

contains (p_1, „Michelle“)

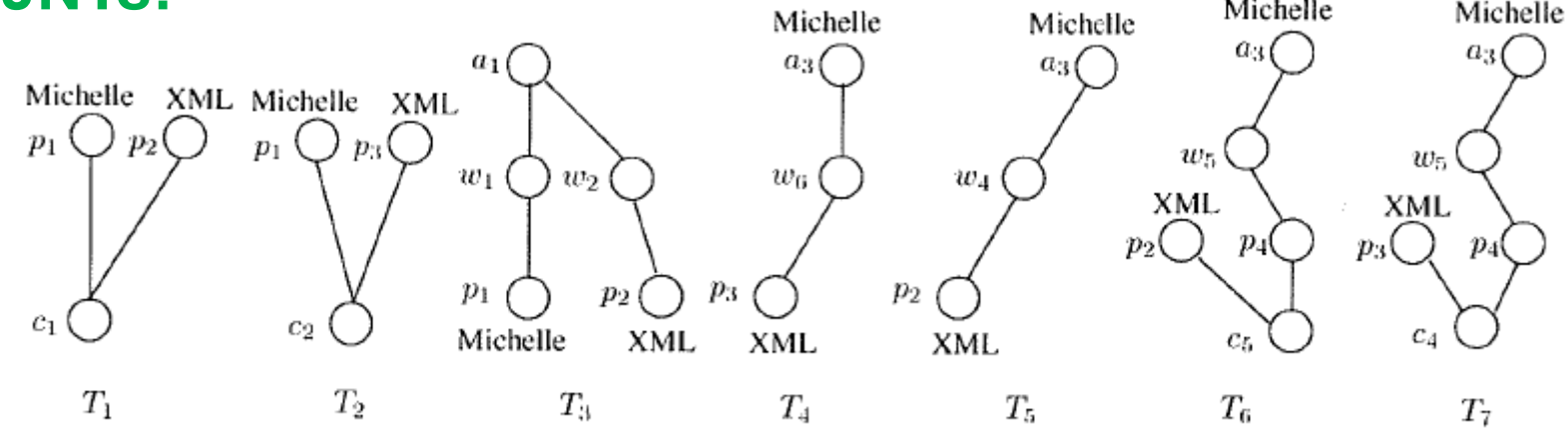
contains (p_2, „XML“)

contains (p_3, „XML“)

MTJNTs:



(c) Tuple Connections



MTJNT issues

Size and scalability:

The data graph is potentially very large, i.e. search is very costly

The search space increases exponentially by adding new data entries

Results semantics and presentation

The results are heterogeneous in terms of structure, i.e. difficult to present and understand

Aggregation / summarization is needed

The idea: Generate structured queries first

Schema graph is much smaller

Structured queries naturally aggregate MTJNTs

Structured queries: Candidate Network (CN)

A *keyword relation*: a subset R_i *{K'}* of relation R_i that contains a subset *K'* of keywords from *K (and no other keywords from K)*. The subset can be empty R_i *{ }*.

A *Candidate Network (CN)* is a connected tree of keyword relations. Every two adjacent keyword relations R_i, R_j in CN are joined based on the fk-reference in the schema G_s.

CN is total w.r.t. a *l*-keyword query *K* if its keyword relations jointly contain all keywords of *K.*

CN is minimal if no keyword relation can be removed such that CN remains total.

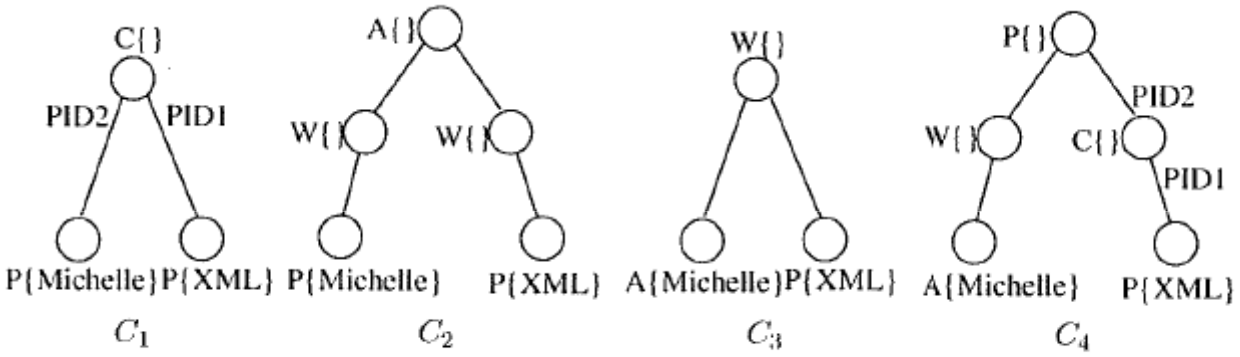
T_{max} – a size control parameter to define the maximum number of keyword relations in CN.

A CN can produce a set of possibly empty MTJNTs. One MTJNTs can be generated by exactly one CN.

CN examples

K = {Michelle, XML}, T_{max} = 5, P{Michelle}, P{XML}, A{Michelle}

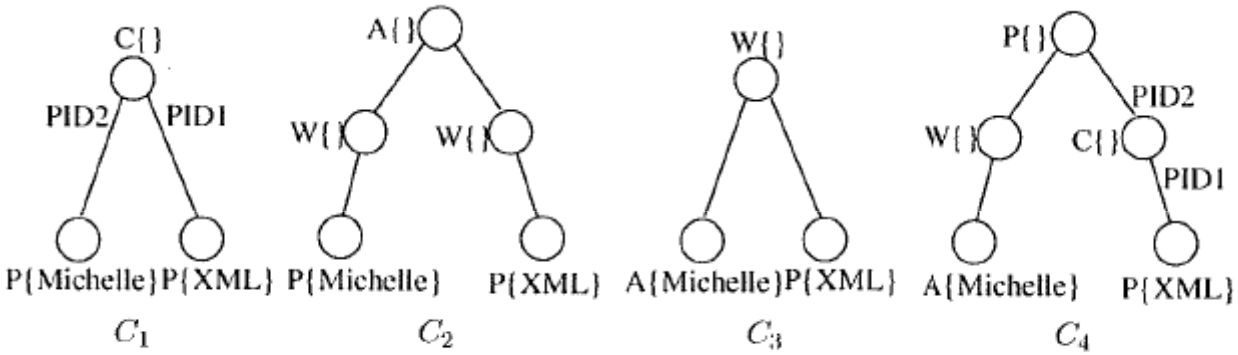
CNs:



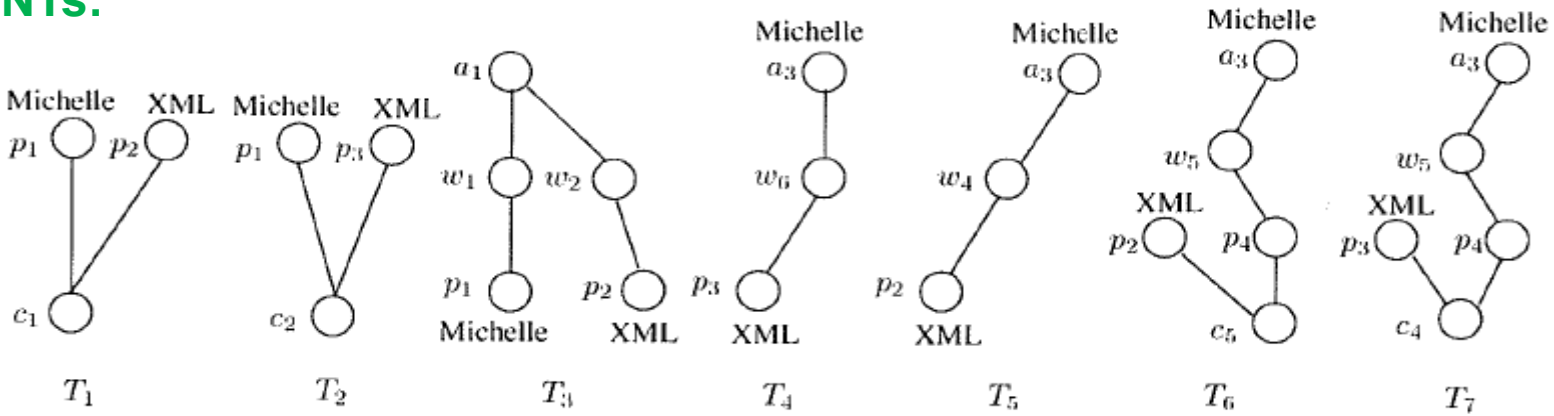
CN examples

K = {Michelle, XML}, T_{max} = 5, P{Michelle}, P{XML}, A{Michelle}

CNs:



MTJNTs:

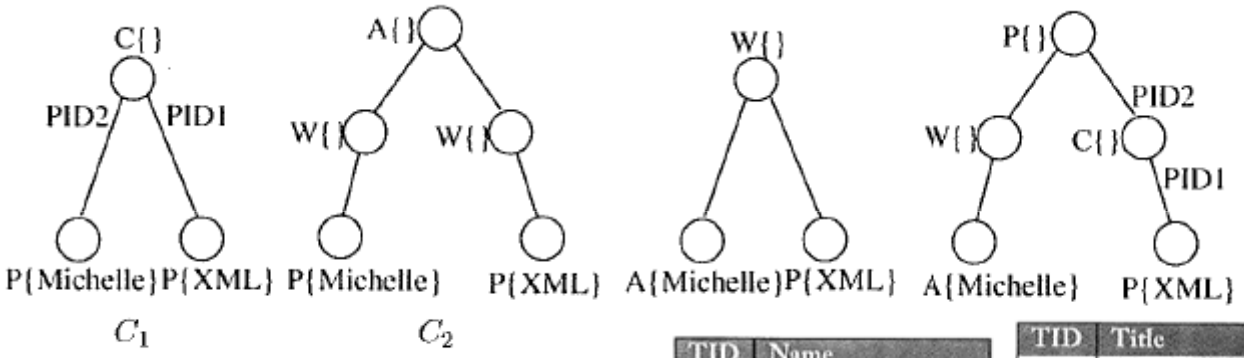


Which MTJNTs are generated by which CNs?

CNs in SQL: Work in groups

K = {Michelle, XML}, T_{max} = 5, P{Michelle}, P{XML}, A{Michelle}

CNs:



SQL:

Work in groups:

Write SQL query expressions to generate C_1 , …, C_5

Time: 10 minutes
1 SQL expert per group?

Tipp: use contains predicate

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

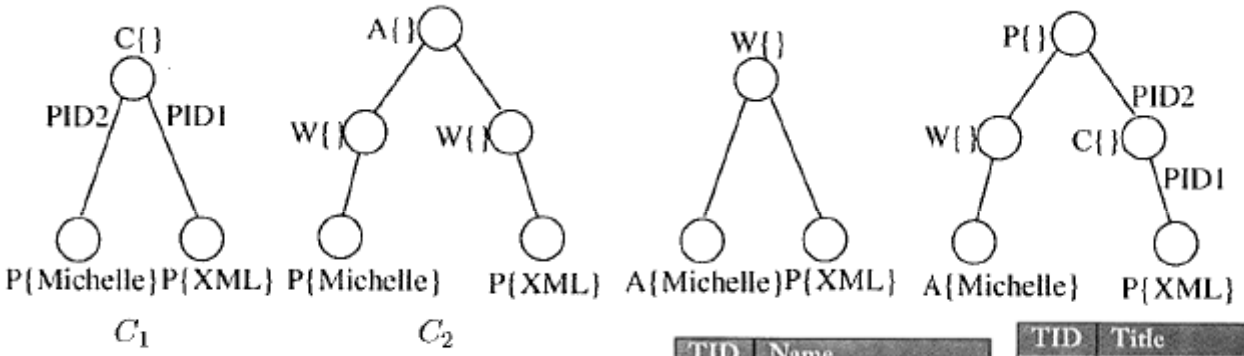
TID	PID1	PID2
c_1	p_2	p_1
c_2	p_3	p_1
c_3	p_2	p_3
c_4	p_3	p_4
c_5	p_2	p_4

(d) Cite

CNs in SQL: Work in groups

K = {Michelle, XML}, T_{max} = 5, P („Michelle“), P („XML“), A („Michelle“)

CNs:



SQL: (C1)

SELECT * from Paper as P1,
Cite as C, Paper as P2
WHERE contains (P1.Title, „Michelle“)
AND NOT contains (P1.Title, „XML“)
AND P1.TID = C.PID2
AND C.PID1 = P2.TID
AND contains (P2.Title, „XML“)
AND NOT contains (P2.Title, „Michelle“)

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

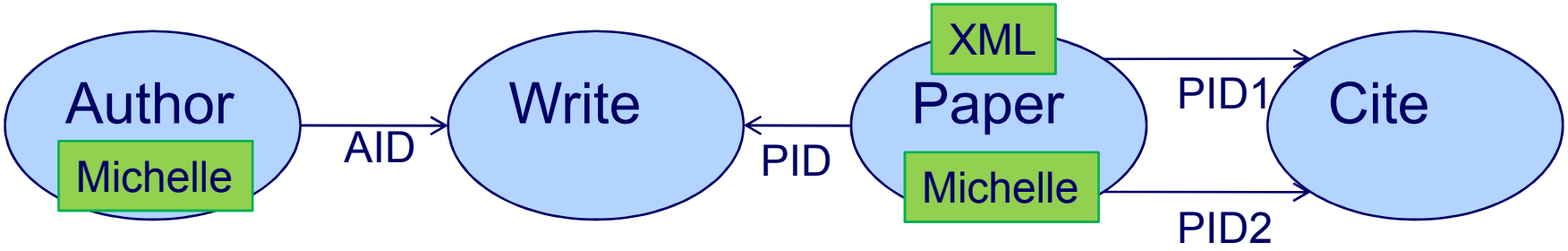
CN generation algorithms

Given are:

1. Keyword query $K = \{k_1, k_2, \ldots, k_l\}$
2. Schema graph G_s
3. The nodes of G_s containing each keyword k_i in K

The Problem: Find the path(s) connecting all $\{k_1, k_2, \ldots, k_l\}$ in G_s (i.e. the structured query(-ies))

Example: K = {Michelle, XML}



Complexity?

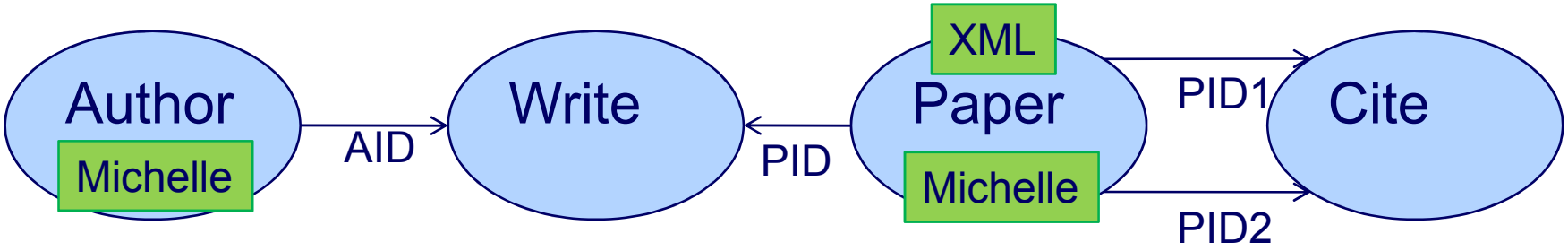
CN generation algorithms

Complexity: similar to the Steiner tree problem - find the shortest interconnect for a given set of objects: NP-complete.

Approximation algorithms:

Iteratively explore the schema graph to construct the paths

Algorithm ideas?

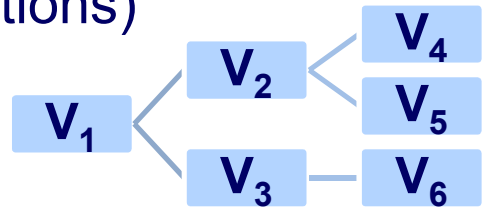


Data structures?

Search algorithms and data structures: BFS

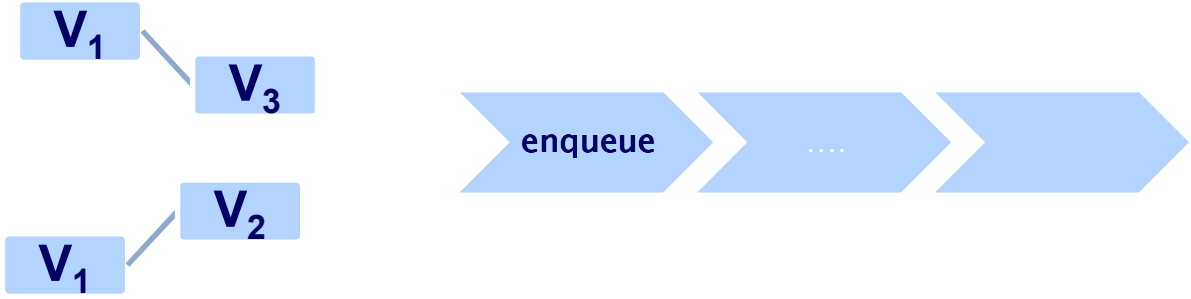
Search on the schema graph G_s (with keyword relations)

Breadth-First-Search (**BFS**): queue



Step i:

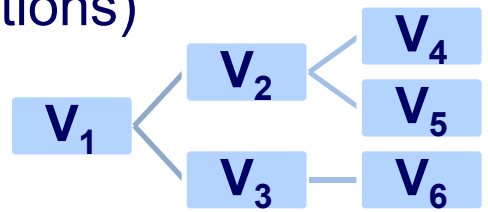
Step i+1:



Search algorithms and data structures: BFS

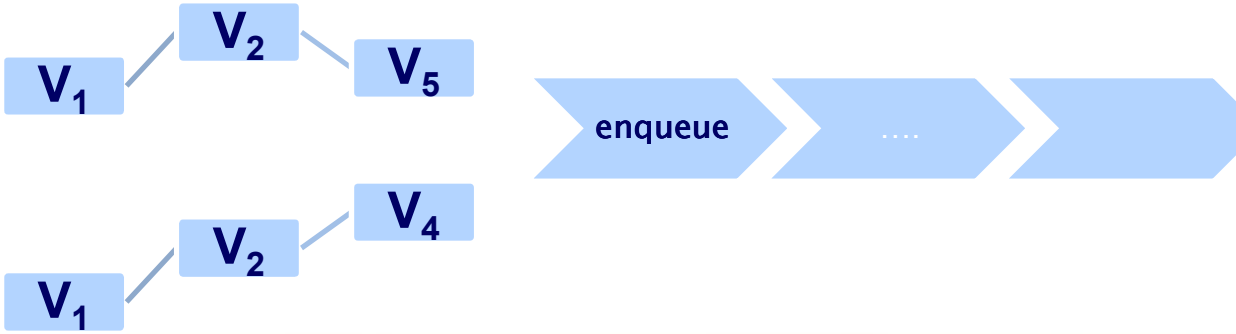
Search on the schema graph G_s (with keyword relations)

Breadth-First-Search (**BFS**): queue



Step j:

Step j+1:

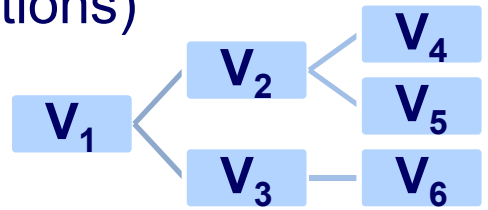


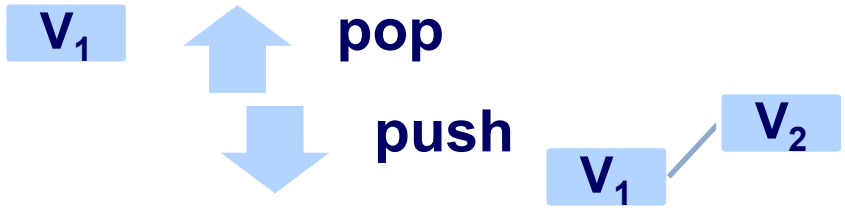
Search algorithms and data structures: DFS

Search on the schema graph G_s (with keyword relations)

Depth First Search (**DFS**) – for top-k generation:

Stack





CN generation: Pruning rules

Goal:

Generate total, minimal and non-duplicating CNs

Pruning rules:

Duplicate elimination (requires graph isomorphism checking)

Pruning total but not minimal CNs

Avoiding cycles (estimated based on pk - fk references)

algorithm from [Hristidis et. al. 2002]

CN generation algorithm (BFS-based): Discover

Algorithm 1 Discover-CNGen (Q, Tmax, G_S)

Notation: here Q is a keyword query!

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

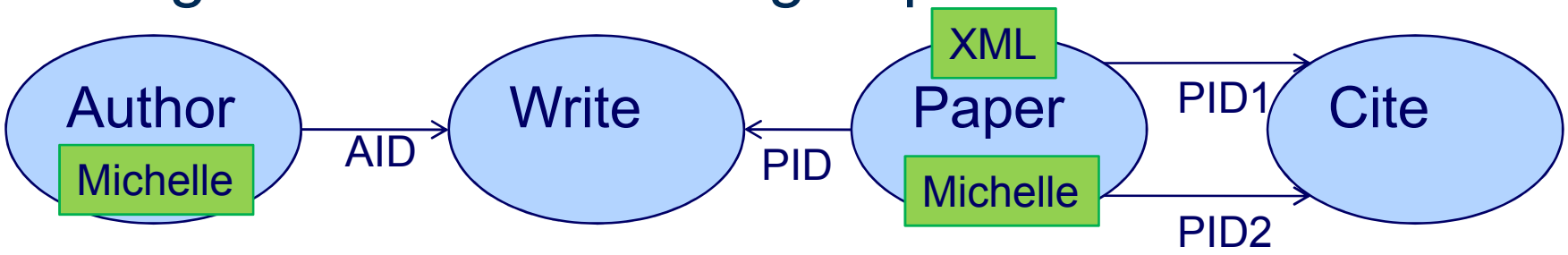
Output: the set of CNs $\mathcal{C} = \{C_1, C_2, \cdots\}$.

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

Rule 1: duplicate elim.

Rule 2: minimality
Rule 3: avoid cycles

CN generation: Work in groups



Keyword relations:
P{Michelle}, P{XML}, A{Michelle}
…

Algorithm 1 Discover-CNGen (Q, Tmax, G_S)

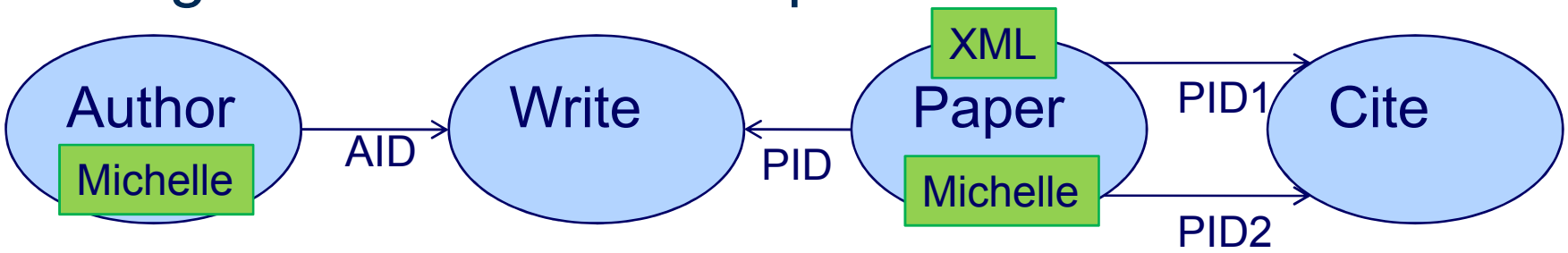
Input: an l-keyword query $Q = \{k_1, k_2, \cdots, k_l\}$, the size control parameter Tmax, the schema graph G_S.
Output: the set of CNs $\mathcal{C} = \{C_1, C_2, \cdots\}$.

```
1:  Q ← ∅; C ← ∅
2:  for all R_i ∈ V(G_S), K' ⊆ Q do
3:      Q.enqueue(R_i{K'})
4:  while Q ≠ ∅ do
5:      T ← Q.dequeue()
6:      if T is minimal and total and T does not satisfy Rule-1 then
7:          C ← C ⋃ {T}; continue
8:      if the size of T < Tmax then
9:          for all R_i ∈ T do
10:             for all (R_i, R_j) ∈ E(G_S) or (R_j, R_i) ∈ E(G_S) do
11:                 T' ← T ⋃ (R_i, R_j)
12:                 if T' does not satisfy Rule-2 or Rule-3 then
13:                     Q.enqueue(T')
14: return C;
```

Work in Groups (10 minutes):

Write down the essential steps of of the algorithm until the first valid (i.e. total and minimal) CN is generated

CN generation: An example



Keyword relations:
P{Michelle}, P{XML}, A{Michelle}
…

Algorithm 1 Discover-CNGen (Q, Tmax, G_S)

Input: an l-keyword query $Q = \{k_1, k_2, \cdots, k_l\}$, the size control parameter Tmax, the schema graph G_S.
Output: the set of CNs $\mathcal{C} = \{C_1, C_2, \cdots\}$.

```
1:  Q ← ∅; C ← ∅
2:  for all R_i ∈ V(G_S), K' ⊆ Q do
3:      Q.enqueue(R_i{K'})
4:  while Q ≠ ∅ do
5:      T ← Q.dequeue()
6:      if T is minimal and total and T does not satisfy Rule-1 then
7:          C ← C ⋃ {T}; continue
8:      if the size of T < Tmax then
9:          for all R_i ∈ T do
10:             for all (R_i, R_j) ∈ E(G_S) or (R_j, R_i) ∈ E(G_S) do
11:                 T' ← T ⋃ (R_i, R_j)
12:                 if T' does not satisfy Rule-2 or Rule-3 then
13:                     Q.enqueue(T')
14: return C;
```

enqueue: P{Michelle}, P{XML}, A{Michelle}
dequeue: T_1 <- A{Michelle}
expand: T_2 <- A{Michelle} ⋈ W{}
enqueue: T_2
…
dequeue: T_2 <- A{Michelle} ⋈ W{}
expand: T_3 <- A{Michelle} ⋈ W{} ⋈ P{XML}
enqueue: T_3
…
dequeue: T_3, check if T_3 is minimal and total, add T_3 to the result

CN generation: Complexity and optimizations

Complexity factors:

- Size of the schema graph G_s – the number of nodes and edges
- Maximum number of joins (T_{max})
- Size of the keyword query (l)

The number of CNs grows exponentially with these factors.

Algorithm optimizations:

- Avoid generation of duplicate CNs by defining the expansion order
- Generate only the top-k CNs
- …

CN and MTJNT ranking factors

Ranking can be performed at **CN and MTJNT levels**

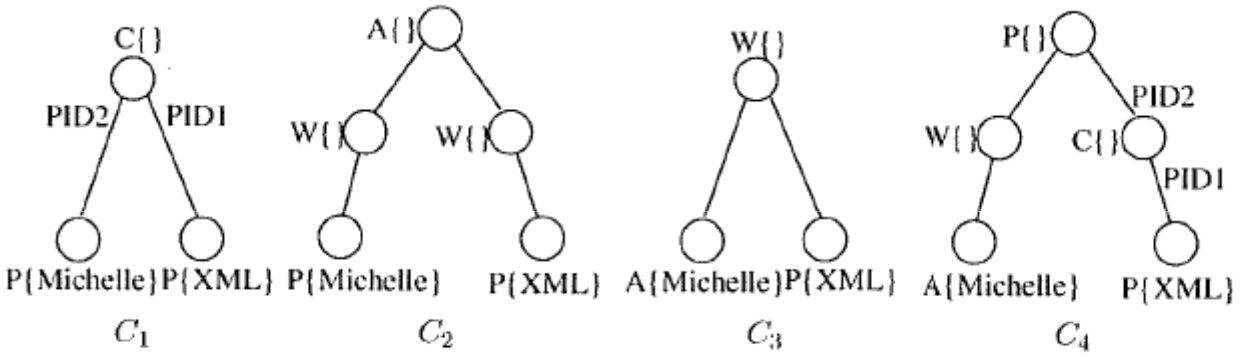
Typical ranking factors include:

- Size of the CN / tuple tree – preference to the short paths
- IR-Style factors
  - Frequency-based keyword weights
  - Keyword selectivity (IDF)
  - Length normalizations
- Global attribute weight in a database (PageRank / ObjectRank)

Typically, the factors are combined

Ranking query interpretations: An example

Rank the following CNs using the size factor:



References and further reading

References:

[Yu et. al 2009] Jeffrey Xu Yu, Lu Qin, Lijun Chang. Keyword Search in Databases. Synthesis Lectures on Data Management. Morgan & Claypool Publishers. 2009. (Chapter 2.)

[Qin et. al 2009] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Keyword search in databases: the power of RDBMS. In Proc. of the 2009 ACM SIGMOD http://dl.acm.org/citation.cfm?id=1559917

[Hristidis et. al 2002] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In Proc. of VLDB 2002. http://dl.acm.org/citation.cfm?id=1287427

Further reading:

[Tata et. al 2008] Sandeep Tata and Guy M. Lohman. SQAK: doing more with keywords. In Proc. of the 2008 ACM SIGMOD. http://doi.acm.org/10.1145/1376616.1376705

[Nandi et. al 2009] Nandi, A., Jagadish, H.V.: Qunits: queried units in database search. In CIDR (2009). http://arxiv.org/ftp/arxiv/papers/0909/0909.1765.pdf