iTrails: Pay-as-you-go Information Integration in Dataspaces

Marcos Antonio Vaz Salles Jens-Peter Dittrich Shant Kirakos Karakashian
Oliver Rene Girard Luras Blunschi

ETH Zurich
8092 Zurich, Switzerland

CSE718. Advanced Topics in Database Systems

Querying heteregenous data sources

1. Schema first approach(SFA)
   - Semantically integrated view over a set of data sources
   - Mappings between source schemas and mediated schema
   - Queries have clearly defined semantics
   - Expensive to construct and maintain

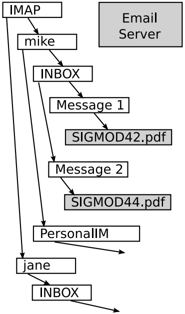
Querying heteregenous data sources

1. Schema first approach(SFA)
   - Semantically integrated view over a set of data sources
   - Mappings between source schemas and mediated schema
   - Queries have clearly defined semantics
   - Expensive to construct and maintain
2. No schema approach(NSA)
   - Keyword search
   - Requires good result ranking methods
   - Performs no integration

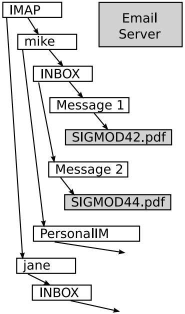
Querying heteregenous data sources

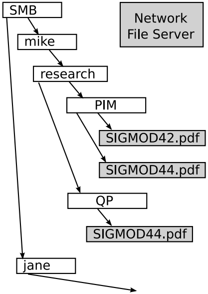
1. Schema first approach(SFA)
   - Semantically integrated view over a set of data sources
   - Mappings between source schemas and mediated schema
   - Queries have clearly defined semantics
   - Expensive to construct and maintain
2. No schema approach(NSA)
   - Keyword search
   - Requires good result ranking methods
   - Performs no integration
3. Dataspaces
   - Starts with NSA
   - Gradually approaches SFA by means of hints (trails)

Dataspaces. Motivation

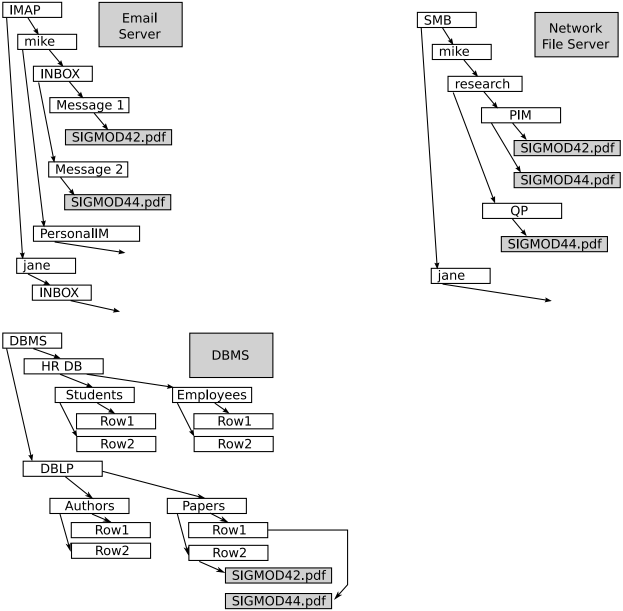


Dataspaces. Motivation

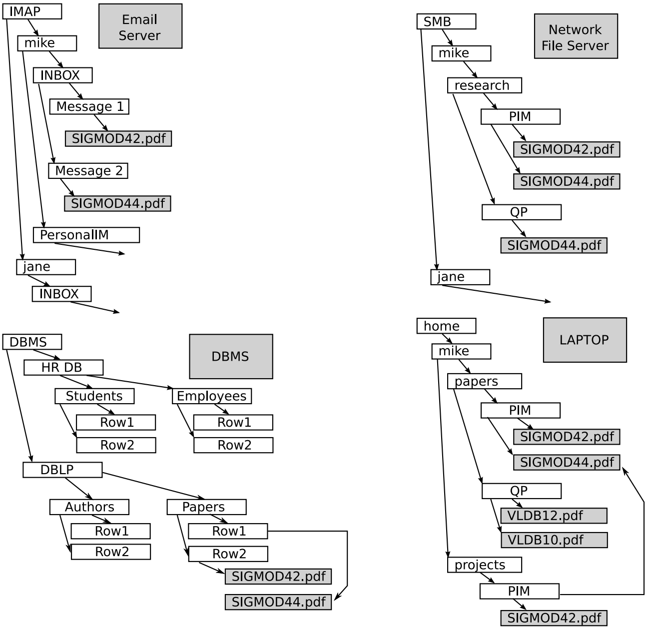




Dataspaces. Motivation



Dataspaces. Motivation



Motivation. Possible queries

Query 1

Retrieve all pdf documents that were added or modified yesterday

Motivation. Possible queries

Query 1

Retrieve all pdf documents that were added or modified yesterday

State-of-the-art

Select all pdf documents that

- **Email server** are attachements to emails with the attribute `received` set to yesterday;
- **DBMS** are pointed by rows whose value of the `lastmodified` column is set to yesterday
- **Net file-server, laptop** have an attribute `lastmodified` set to yesterday.

Motivation. Possible queries

Query 1

Retrieve all pdf documents that were added or modified yesterday

State-of-the-art

Select all pdf documents that

- **Email server** are attachements to emails with the attribute `received` set to yesterday;
- **DBMS** are pointed by rows whose value of the `lastmodified` column is set to yesterday

Net file-server, laptop have an attribute `lastmodified` set to yesterday.

Goal

Provide a method that allows to specify the same query by typing the keywords `pdf yesterday`. Exploit *hints* (trails) to provide partial schema knowledge

1. The `yesterday` keyword is mapped to a query for values of the `date` attribute equal to the date of yesterday
2. The `date` attribute is mapped to the `lastmodified` attribute
3. The `date` attribute is mapped to the `received` attribute
4. The `pdf` keyword is mapped to a query for elements whose names end in pdf.

Motivation. Possible queries

Query 2

Retrieve all information about the current work on project PIM

Motivation. Possible queries

Query 2

Retrieve all information about the current work on project PIM

State-of-the-art

Issue the following queries to the search engine

- Email server `//mike/personalIM`
- Laptop `//projects/PIM` (but not `//papers/PIM`)
- Net file-server `//mike/research/PIM`

Motivation. Possible queries

Query 2

Retrieve all information about the current work on project PIM

State-of-the-art

Issue the following queries to the search engine

Email server	`//mike/personalIM`
Laptop	`//projects/PIM` (but not `//papers/PIM`)
Net file-server	`//mike/research/PIM`

Goal

Provide a method of specifying the query by typing `//projects/PIM`

1. Queries for the path `//projects/PIM` should also consider the path `//mike/research/PIM`
2. Queries for the path `//projects/PIM` should also consider the path `//mike/personalIM`

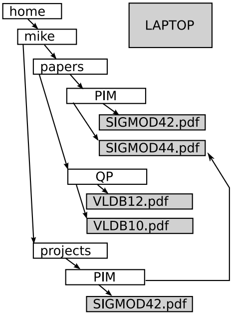
Data model

Definition

- All data is represented by a logical graph $G = (RV, E)$
- RV is the set of nodes $\{V_1, \ldots V_n\}$ each of which termed resource view
- E is a sequence of ordered pairs (V_i, V_j) of resource views representing directed edges from V_i to V_j
- $V_i \rightsquigarrow V_j$ denotes the fact that V_j is reachable from V_i by traversing the edges E
- A resource view V_i has three components: name, tuple, and content

Component of V_i	Definition
$V_i.name$	Name (string) of the resource view
$V_i.tuple$	Set of attribute value pairs $(\langle att_0, value_0\rangle, \langle att_1, value_1\rangle, \ldots)$
$V_i.content$	Finite by sequence of content (e.g. text)

Example



$$
\begin{aligned}
X1 = \{\ & .name = \text{‘}home\text{‘}, \\
& .tuple = \{.owner = \text{‘}root\text{‘}, \\
& \qquad .lastmodified = \text{‘05.01.2000‘}\}, \\
& .content = \text{‘‘}\} \\
X2 = \{\ & .name = \text{‘}mike\text{‘}, \\
& .tuple = \{.owner = \text{‘}root\text{‘}, \\
& \qquad .lastmodified = \text{‘04.17.2008‘}\}, \\
& .content = \text{‘‘}\} \\
\ldots & \\
X5 = \{\ & .name = \text{‘}SIGMOD42.pdf\text{‘}, \\
& .tuple = \{size = 10k, \\
& \qquad .owner = \text{‘}mike\text{‘}, \\
& \qquad .lastmodified = \text{‘04.01.2007‘}\}, \\
& .content = \text{‘@}PDF \ldots \text{‘}\}
\end{aligned}
$$

$$\ldots$$

Query model

Query expression

A query expresion Q selects a subset of nodes $R := Q(G) \subseteq G.RV$

Example: `//mike/papers`

Component projection

A component projection $C \in \{.name, .tuple.\langle att_i \rangle, .content\}$ obtains a projection of the set of resource views selected by a query expression Q, i.e. a set of components $R' := \{V_i.C | V_i \in Q(G)\}$

Example: `//mike//PIM/*.tuple.lastmodified`

Query language

Syntax of query expressions

```
QUERY_EXPRESSION ::= (PATH | KT_PREDICATE) (UNION QUERY_EXPRESSION)*
PATH             ::= (LOCATION_STEP)+
LOCATION_STEP    ::= LS_SEP NAME_PREDICATE ('[' KT_PREDICATE ']')?
LS_SEP           ::= '//' | '/'
NAME_PREDICATE   ::= '*' | ('*')? VALUE ('*')?
KT_PREDICATE     ::= (KEYWORD | TUPLE) (LOGOP KT_PREDICATE)*
KEYWORD          ::= '"' VALUE (WHITESPACE VALUE)* '"'
                               | VALUE (WHITESPACE KEYWORD)*
TUPLE            ::= ATTRIBUTE_IDENTIFIER OPERATOR VALUE
OPERATOR         ::= '=' | '<' | '>'
LOGOP            ::= 'AND' | 'OR'
```

Semantics of query expressions

Semantics

Query expression	Semantics
`//*`	$\{V \mid V \in G.RV\}$
`a`	$\{V \mid V \in G.RV \wedge \text{'}a\text{'} \subseteq V.content\}$
`a b`	$\{V \mid V \in G.RV \wedge \text{'}a\text{'} \subseteq V.content \wedge \text{'}b\text{'} \subseteq V.content\}$
`//A`	$\{V \mid V \in G.RV \wedge V.name = \text{'}A\text{'}\}$
`//A/B`	$\{V \mid V \in G.RV \wedge V.name = \text{'}B\text{'} \wedge \exists (W, V) \in G.E : W.name = \text{'}A\text{'}\}$
`//A//B`	$\{V \mid V \in G.RV \wedge V.name = \text{'}B\text{'} \wedge \exists (W, Z_1), (Z_1, ..), \ldots, (.., Z_n), (Z_n, V) \in G.E : W.name = \text{'}A\text{'}\}$
`b=42`	$\{V \mid V \in G.RV \wedge \exists V.tuple.b : V.tuple.b = 42\}$
`b=42 a`	:= `b=42` $\cap$ `a`
`//A/B[b=42]`	:= `//A/B` $\cap$ `b=42`

Logical algebra for query expressions

Logical algebra

Operator	Name	Semantics
G	All resource views	$\{V \| V \in G.RV\}$
$\sigma_P(I)$	Selection	$\{V \| V \in I \wedge P(V)\}$
$\mu(I)$	Shallow unnest	$\{W \| (V, W) \in G.E \wedge V \in I\}$
$\omega(I)$	Deep unnest	$\{V \| V \rightsquigarrow W \wedge V \in I\}$
$I_1 \cap I_2$	Intersection	$\{V \| V \in I_1 \wedge V \in I_2\}$
$I_1 \cup I_2$	Union	$\{V \| V \in I_1 \vee V \in I_2\}$

Canonical form

Definition

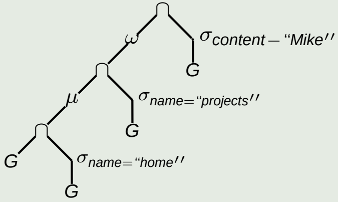
The canonical form $\Gamma(Q)$ of a query Q is obtained by decomposing Q into location step separators and predicates (P) according to the **grammar**. $\Gamma(Q)$ is constructed by the following recursion:

$$\text{tree} = \begin{cases} G & \text{if tree is empty} \\ \omega(\textit{tree}) & \text{if LS\_SEP} = // \text{ and not first location step,} \\ \mu(\textit{tree}) & \text{if LS\_SEP} = / \text{ and not first location step,} \\ \textit{tree} \cap \sigma_P(G) & \text{otherwise} \end{cases}$$

Finally, $\Gamma(Q) := \textit{tree}$ is returned.

Example

Q := `//home/projects//*["Mike"]`



Definition

A unidirectional trail is denoted as

$$\psi_i := Q_L[.C_L] \longrightarrow Q_R[.C_R].$$

This means that the query (resp. component projection) on the left $Q_L[.C_L]$ induces the query (resp. component projection) on the right $Q_R[.C_R]$, i.e. whenever we query for $Q_L[.C_L]$, we should also query for $Q_R[.C_R]$.

A bidirectional trail is denoted as

$$\psi_i := Q_L[.C_L] \longleftrightarrow Q_R[.C_R].$$

The latter also means that the query on the right $Q_R[.C_R]$ induces the query on the left $Q_L[.C_L]$. The component projections C_L and C_R should either appear on both sides of the trail or on none.

Use cases

Functional equivalence

$$\psi_1 := \quad //*.\mathit{tuple}.\mathit{date} \longrightarrow //*.\mathit{tuple}.\mathit{modif}$$
$$\psi_2 := \quad //*.\mathit{tuple}.\mathit{date} \longrightarrow //*.\mathit{tuple}.\mathit{recd}$$
$$\psi_3 := \quad \mathit{yesterday} \longrightarrow \mathit{date} = \mathit{yesterday}()$$

Q: `yesterday`

Q': `yesterday` $\cup$ `//*[date=yesterday() OR modif=yesterday() OR recd=yesterday()]`

Use cases

Functional equivalence

$\psi_1 :=$ $//*.tuple.date \longrightarrow //*.tuple.modif$

$\psi_2 :=$ $//*.tuple.date \longrightarrow //*.tuple.recd$

$\psi_3 :=$ $yesterday \longrightarrow date = yesterday()$

Q: `yesterday`

Q': `yesterday` $\cup$ `//*[date=yesterday() OR modif=yesterday() OR recd=yesterday()]`

Type restriction

$\psi_5 :=$ $email \longrightarrow class = email$

Use cases

Functional equivalence

$$\psi_1 := \quad //*.tuple.date \longrightarrow //*.tuple.modif$$
$$\psi_2 := \quad //*.tuple.date \longrightarrow //*.tuple.recd$$
$$\psi_3 := \quad yesterday \longrightarrow date = yesterday()$$

Q: `yesterday`

Q': `yesterday` ∪ `//*[date=yesterday() OR modif=yesterday() OR recd=yesterday()]`

Type restriction

$$\psi_5 := \quad email \longrightarrow class = email$$

Semantic search

$$\psi_{20} := \quad car \longrightarrow auto$$

Trail variants

Probabilistic trail

A probabilistic trail assigns a probability value $0 \leq p \leq 1$ to a trail definition

$$\psi := Q_L[.C_L] \longrightarrow_p Q_R[.C_R]$$

Scored trail

A scored trail assigns a scoring factor $sf \geq 1$ to a trail definition

$$\psi := Q_L[.C_L] \longrightarrow_{sf} Q_R[.C_R]$$

iTrail query processing

1. Matching
2. Transformation
3. Merging

Trail matching

Definition

$$\psi_i := \psi_i^{L.Q}[.\psi_i^{L.C}] \longrightarrow \psi_i^{R.Q}[.\psi_i^{R.C}]$$

Matching A trail ψ_i matches a query Q whenever its left side query $\psi_i^{L.Q}$ is contained in a query subtree Q_S of the canonical form $\Gamma(Q)$. We denote this as $\psi_i^{L.Q} \subseteq Q_S$. Furthermore, Q_S must be maximal.

Query

$$\psi_i := \psi_i^{L.Q} \longrightarrow \psi_i^{R.Q}$$

For such ψ_i , we require Q_S not to contain $\psi_i^{R.Q}$, i.e. $\psi_i^{R.Q} \not\subseteq Q_S$. We then take $Q_{\psi_i}^M := Q_S$.

Comp projection

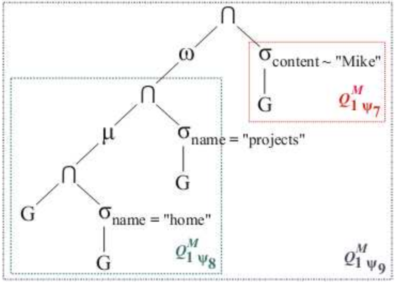
$$\psi_i := \psi_i^{L.Q}.\psi_i^{L.C} \longrightarrow \psi_i^{R.Q}.\psi_i^{R.C}$$

For such ψ_i, we require that the component projection $\psi_i^{L.C}$ be referenced in the query in a selection by an operator immediately after Q_S in $\Gamma(Q)$. The matching subtree $Q_{\psi_i}^M$ is then obrained by extending Q_S by the portion of the query referencing the component projection $\psi_i^{L.C}$.

Trail matching

Example

ψ_7 := `Mike` $\longrightarrow$ `Carey`
ψ_8 := `//home/*.name` $\longrightarrow$ `//calendar//*.tuple.category`
ψ_9 := `//home/projectios/OLAP//*[“Mike”]` $\longrightarrow$
`//imap//*[“OLAP” “Mike”]`



Trail transformation

Definition

Query Given a query expression Q and a trail ψ_i without component projections, we compute the transformation $Q^T_{\psi_i}$ by setting $Q^T_{\psi_i} := \psi_i^{R.Q}$.

Comp projection For a trail ψ_j with component projections, we take $Q^T_{\psi_j} := \psi_j^{R.Q} \cap \sigma_P(G)$. The predicate P is obtained by taking the predicate at the last location step of $Q^M_{\psi_j}$ and replacing all occurences of $\psi_j^{L.C}$ for $\psi_j^{R.C}$.

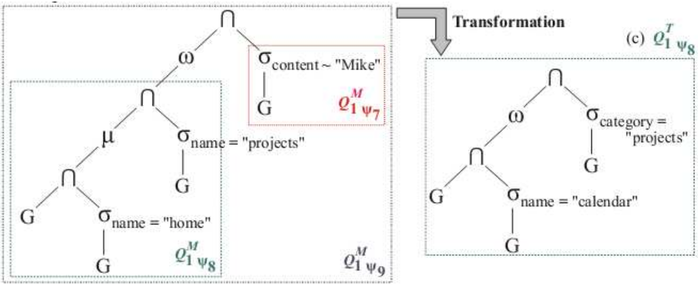
Trail transformation

Example

$\psi_7 := \texttt{Mike} \longrightarrow \texttt{Carey}$

$\psi_8 := \texttt{//home/*.name} \longrightarrow \texttt{//calendar//*.tuple.category}$

$\psi_9 := \texttt{//home/projectios/OLAP//*[“Mike”]} \longrightarrow$
$\texttt{//imap//*[“OLAP” “Mike”]}$



Trail merging

Definition

Given a query Q and a trail ψ_i, the merging $Q^*_{\{\psi_i\}}$ is given by substituting $Q^M_{\psi_i}$ for $Q^M_{\psi_i} \cup Q^T_{\psi_i}$ in $\Gamma(Q)$.

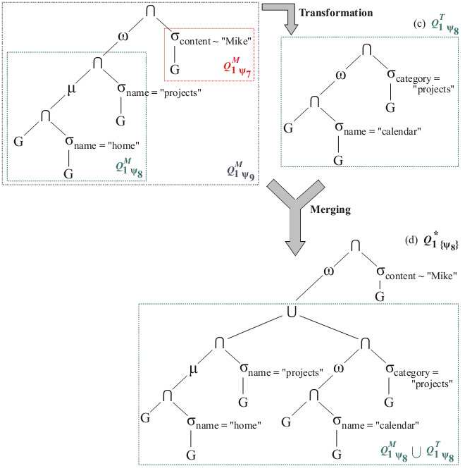
Trail merging

Example

ψ_7 := Mike $\longrightarrow$ Carey

ψ_8 := //home/*.name $\longrightarrow$ //calendar//*.tuple.category

ψ_9 := //home/projectios/OLAP//*[“Mike”] $\longrightarrow$
//imap//*[“OLAP” “Mike”]



Multiple trails

Issues

- Possibility of reapplication of trails
- Order of application
- Termination in the event of reapplication

Multiple trails

Issues

- Possibility of reapplication of trails
- Order of application
- Termination in the event of reapplication

Solution

Keep the *history* of all trails matched or introduced for any query node. Use the *Multiple Match Colouring Algorithm (MMCA)*.

MMCA Algorithm

Algorithm 1: Multiple Match Colouring Algorithm (MMCA)

Input: Set of Trails $\Psi = \{\psi_1, \psi_2, \ldots, \psi_n\}$
Canonical form of query $\Gamma(Q)$
Maximum number of levels *maxL*
Output: Rewritten query tree Q_R

Set $mergeSet \leftarrow$ <>
Query $Q_R \leftarrow \Gamma(Q)$
Query $previousQ_R \leftarrow nil$
$currentL \leftarrow 1$
// (1) Loop until maximum allowed level is reached:
while $(currentL \leq maxL \wedge Q_R \neq previousQ_R)$ **do**
    // (2) Perform matching on snapshot of input query Q_R:
    for $\psi_i \in \Psi$ **do**
        if $(Q^M_{R\,\psi_i}$ *exists* $\wedge$ *root node of* $Q^M_{R\,\psi_i}$ *is not colored by* $\psi_i)$ **then**
            Calculate $Q^T_{R\,\psi_i}$
            Color root node of $Q^T_{R\,\psi_i}$ with color i of ψ_i
            Node $annotatedNode \leftarrow$ root node of $Q^M_{R\,\psi_i}$
            Entry $entry \leftarrow mergeSet$.getEntry($annotatedNode$)
            $entry.transformationList$.append($Q^T_{R\,\psi_i}$)
        end
    end

MMCA Algorithm

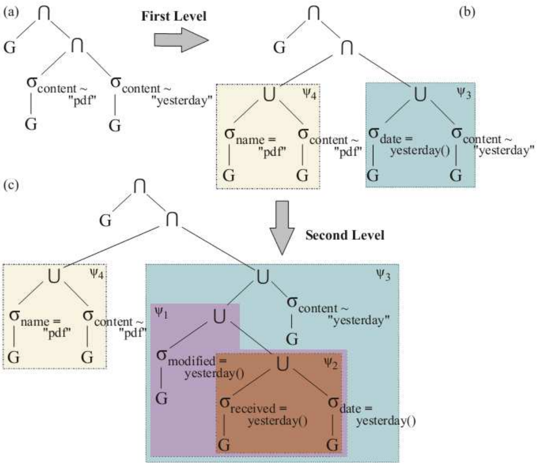
// (3) Create new query based on {node, transformationList} entries:
$previousQ_R \leftarrow Q_R$
for $e \in mergeSet$ **do**
    ColorSet $CS \leftarrow$ (all colors in $e.annotatedNode$) $\cup$
        (all colors in root nodes of $e.transformationList$)
    Node $mergedNode \leftarrow$
        $e.annotatedNode \cup Q^T_{R\,\psi_{i_1}} \cup \ldots \cup Q^T_{R\,\psi_{i_k}}$,
        for all colors $\{i_1, \ldots, i_k\}$ in $e.transformationList$
    Color all nodes in $mergedNode$ with all colors in color set CS
    Calculate $Q^*_{R\,\{\psi_{i_1},\ldots,\psi_{i_k}\}}$ by replacing $e.annotatedNode$ by
        $mergedNode$ in Q_R
    $Q_R \leftarrow Q^*_{R\,\{\psi_{i_1},\ldots,\psi_{i_k}\}}$
end
// (4) Increase counter for next level:
$currentL \leftarrow currentL + 1$
$mergeSet \leftarrow <>$
end
return Q_R

MMCA Algorithm

Example

$$\psi_1 := \; //*.tuple.date \longrightarrow //*.tuple.modif$$
$$\psi_2 := \; //*.tuple.date \longrightarrow //*.tuple.recd$$
$$\psi_3 := \; yesterday \longrightarrow date = yesterday()$$
$$\psi_4 := \; pdf \longrightarrow //*.pdf$$

Q: `pdf yesterday`



MMCA

Algorithm runtime

- L - total number of leaves in the query Q
- M - maximum number of leaves in the query plans introduced by a trail ψ_i
- N - total number of trails
- $d \in \{1, \ldots, N\}$ be the number of levels

The maximum number of trail applications performed by MMCA and the maximum number of leaves in the merged query tree are both bounded by

$$O(L \bullet M^d)$$

Pruning

Trail ranking

- Use a probabilistic weighting function to weight $Q^T_{\psi_i}$
- Use a scoring function to find a scoring factor of $Q^T_{\psi_i}$

Pruning strategies

1. Prune by level - punish recursive rewrites
2. Prune by Top-K Ranked Matched Trails - use weighting/scoring functions
3. Other - timeout, progressively compute query results

Experiments

Approaches to compare

- **Semi-structured Search Baseline** No schema, keyword and XPath-like queries
- **Perfect query** Schema first, keyword and XPath-like queries
- **Pay-as-you-go** iTrails, keyword and XPath-like queries

Dataset for experiments, MB

	Desktop	Wiki4V	Enron	DBLP	Σ
Gross Data size	44,459	26,392	111	713	71,675
Net Data size	1,230	26,392	111	713	28,446

Experiments

Trails

```
pics ⟶ //photos//* ∪ //pictures//*
//*.tuple.date ⟷ //*.tuple.lastmodified
//*.tuple.date ⟷ //*.tuple.sent
pdf ⟶ //*.pdf
yesterday ⟶ date=yesterday()
publication ⟶ //*.pdf ∪ //dblp//*
//*.tuple.address ⟷ //*.tuple.to
//*.tuple.address ⟷ //*.tuple.from
excel ⟷ //*.xls ∪ *.ods
//*.xls ⟷ //*.ods
//imemex/workspace ⟶
  //ethz/testworkspace ∪ //ethz/workspace
//ethz/testworkspace ⟷ //ethz/workspace
music ⟶ //*.mp3 ∪ //*.wma
working ⟶ //vldb//* ∪ vldb07//*
paper ⟶ //*.tex
//vldb ⟶ //ethz/workspace/VLDB07
email ⟶ class=email
mimeType=image ⟶ mimeType=image/jpeg
```

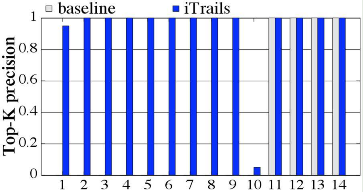
Experiments

Queries

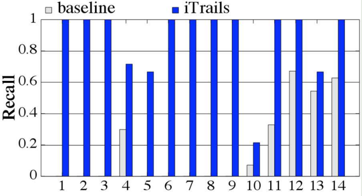
No.	Query Expression	Original Tree Size	Final Tree Size	# Trails Applied
1	`//bern//*["pics"]`	6	14	1
2	`date > 22.10.2006`	2	8	2
3	`pdf yesterday`	5	44	4
4	`Halevy publication`	5	12	1
5	`address=raimund.grube@enron.com`	2	8	2
6	`excel`	2	8	2
7	`//imemex/workspace/VLDB07/*.tex`	14	35	2
8	`//*Aznavour*["music"]`	5	11	1
9	`working paper`	5	41	5
10	`family email`	5	8	1
11	`lastmodified > 16.06.2000`	2	8	2
12	`sent < 16.06.2000`	2	8	2
13	`to=raimund.grube@enron.com`	2	8	2
14	`//*.xls`	2	5	1

Experiments. Quality and Completeness

Precision



Recall



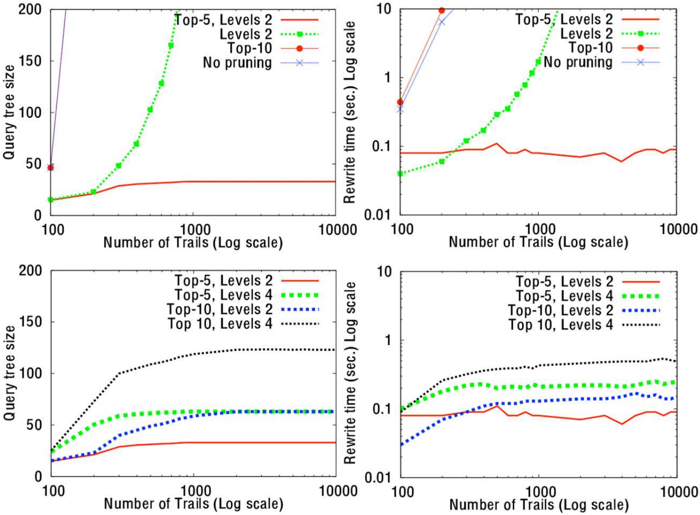
$$K = 20$$

Experiments. Query performance

Q. No.	Perfect Query with Basic Indexes	iTrails with Basic Indexes	iTrails with Trail Mat.
1	0.99	2.18	0.21
2	1.10	0.74	0.52
3	4.33	10.72	0.39
4	0.39	1.86	0.07
5	0.29	0.56	0.44
6	0.14	0.32	0.05
7	0.63	1.73	0.67
8	1.55	5.27	0.48
9	186.39	179.02	1.50
10	0.65	10.14	0.29
11	0.68	0.60	0.60
12	0.67	0.60	0.60
13	0.28	0.49	0.44
14	0.14	0.14	0.14

Execution times [sec]

Experiments. Query performance



References

1. Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier Rene Girard, Lukas Blunschi: iTrails: Pay-as-you-go Information Integration in Dataspaces. *VLDB* 2007: 663-674
2. M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27-33, 2005
3. A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). *In INEX Workshop*, 2004.