Tree Signatures and Unordered XML Pattern Matching

Pavel Zezula
Masaryk University of Brno, Czech Republic

and
Federica Mandreoli, Riccardo Martoglia
University of Modena and Reggio Emilia, Italy

SOFSEM 2004, Měřín

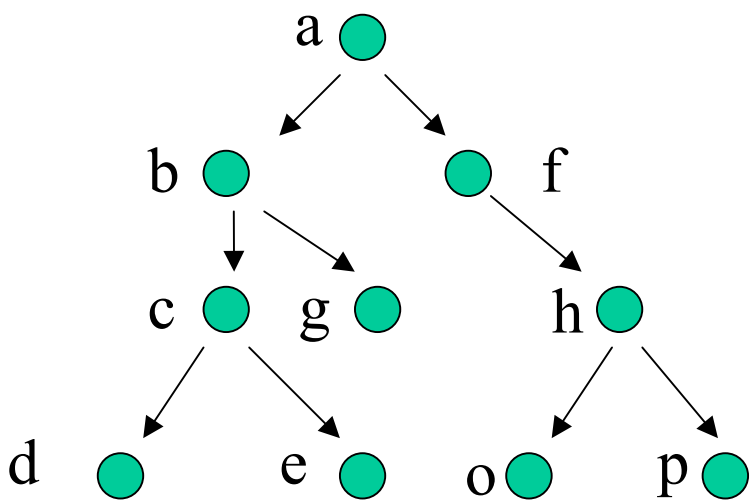
Overview of the talk

- objectives and the approach
- trees and sequences
- tree signatures and their properties
- experimental query evaluation
- future research directions

Objectives and the approach

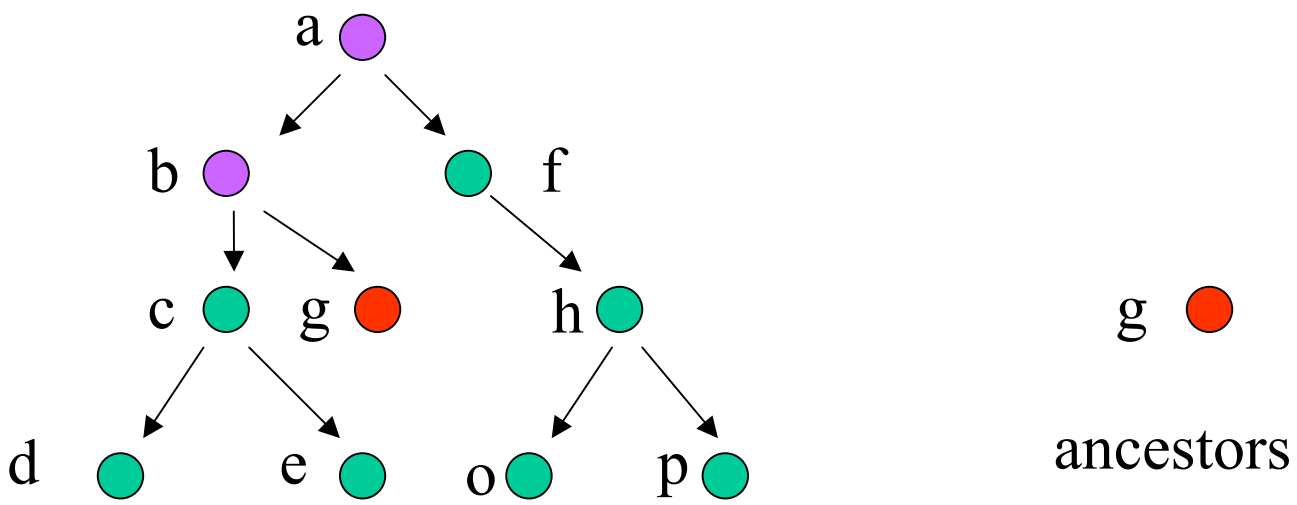
- Accelerate execution of queries on XML data, formulated by languages such as XPath and XQuery.
- Signature file approach.
- Create bases for other types of structured data processing.

XML data abstraction

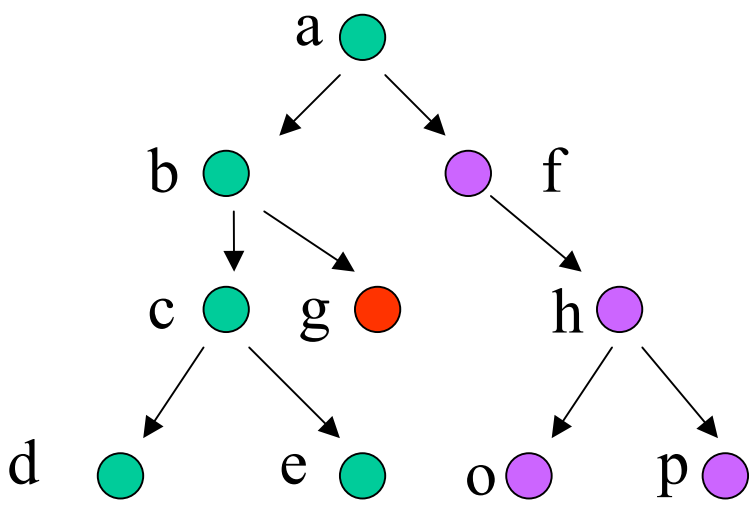


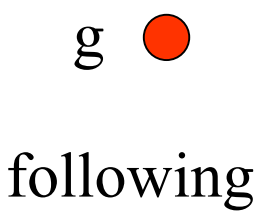
Ordered labeled trees

Search problems -navigation

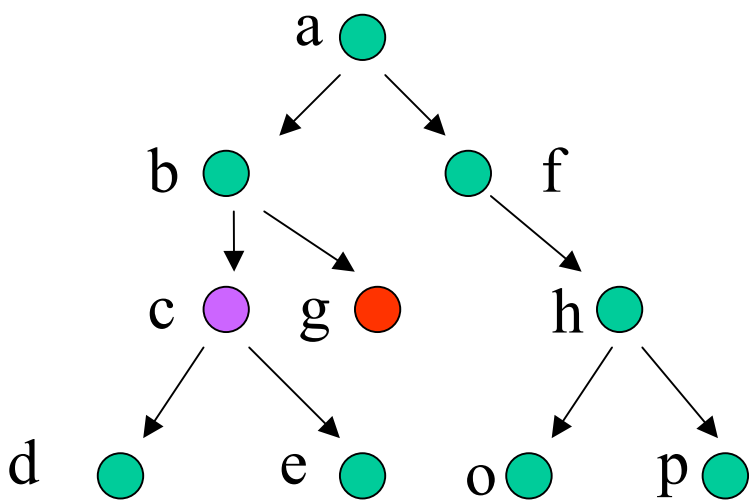


Search problems - navigation



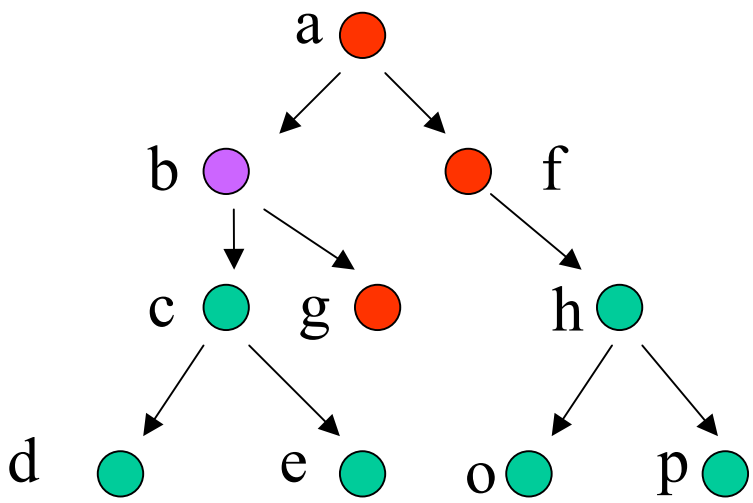


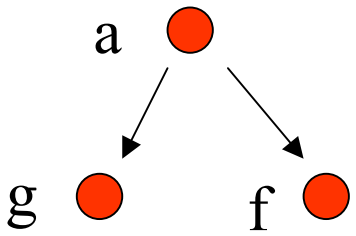
Search problems - navigation



left sibling

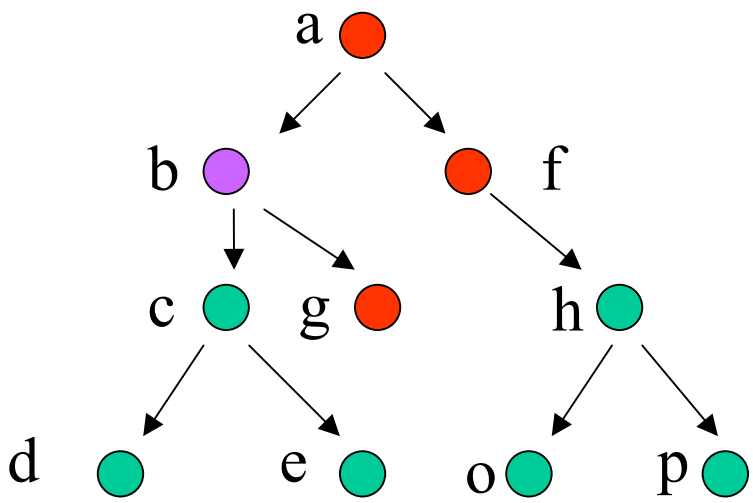
Search problems – tree pattern matching

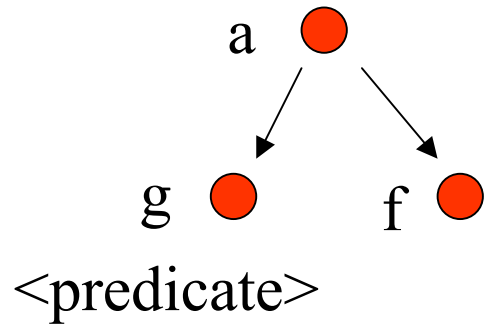




Query tree

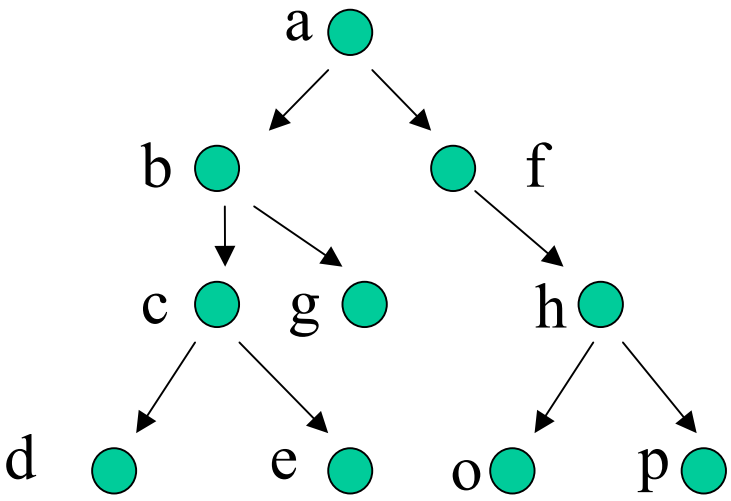
Search problems – tree pattern matching





Query tree

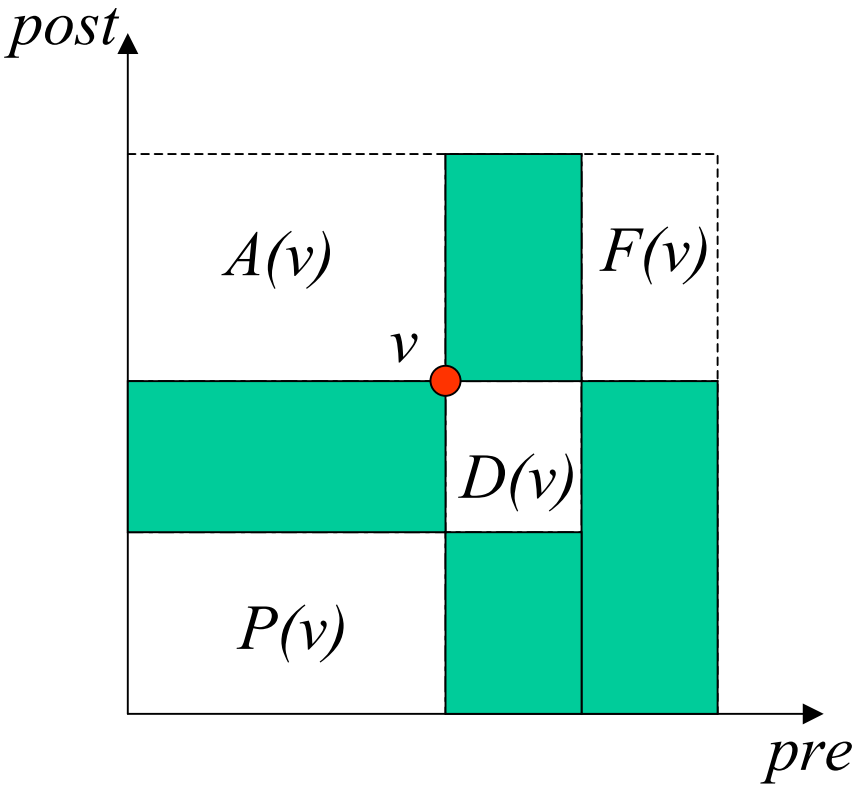
Trees and sequences



pre:	a	b	c	d	e	g	f	h	o	p
post:	d	e	c	g	b	o	p	h	f	a
rank:	1	2	3	4	5	6	7	8	9	10

Properties of *pre*/*post* ranks

$level(v) = pre(v) - post(v) + size(v) = |A(v)|$



$size(v) = |D(v)|$

$pre(v) = |A(v)| + |P(v)| + 1$

$post(v) = |D(v)| + |P(v)| + 1$

- empty space

String operations

string: $x = x_1, \ldots, x_n$

strictly increasing sequence: $1 = i_1 < i_2 < \ldots < i_k \check{} \; n$

subsequence of x: $x_{i1}, x_{i2}, \ldots, x_{ik}$

given strings x and y,
l.c.s. is the longest string that is a subsequence of both x and y,
e.g. **art** is l.c.s of **a**lgo**r**i**t**hm and p**ar**achu**t**e

string x is *sequence included* in string y if their l.c.s. is x,
e.g. **art** is sequence included in p**ar**achu**t**e

Ordered tree inclusion

Given:

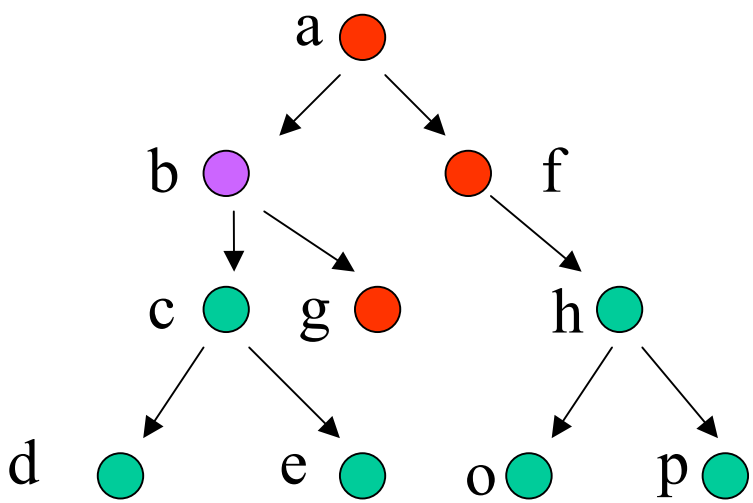
- data tree D
- query tree Q

Tree Q is included in D if

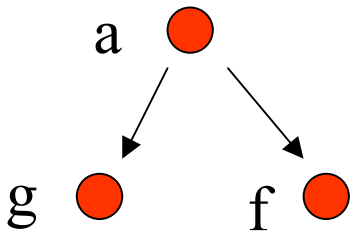
- $pre(Q)$ is sequence included in $pre(D)$, and
- $post(Q)$ is sequence included in $post(D)$.

Tree inclusion – example (true)

Data tree



Query tree



pre: **a** b c d e **g** **f** h o p

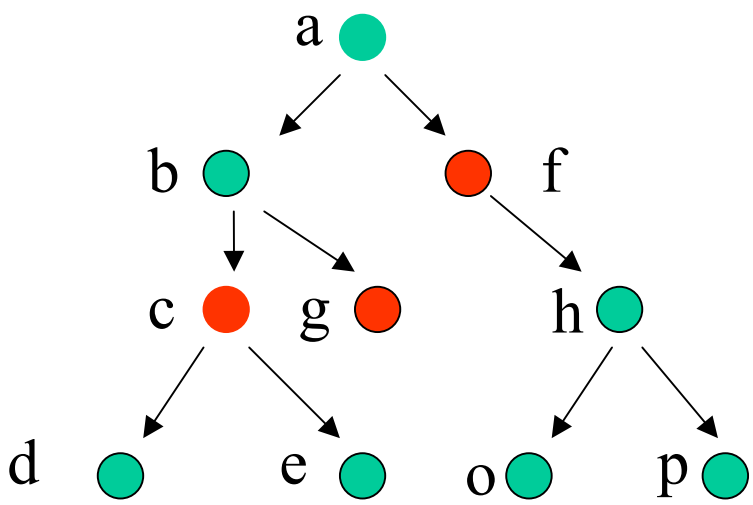
post: d e c **g** b o p h **f** **a**

pre: a g f

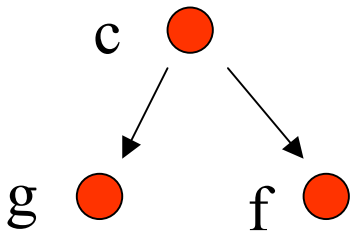
post: g f a

Tree inclusion – example (false)

Data tree



Query tree



pre: a b **c** d e **g** **f** h o p

post: d e c **g** b o p h **f** a

pre: c g f

post: g f c

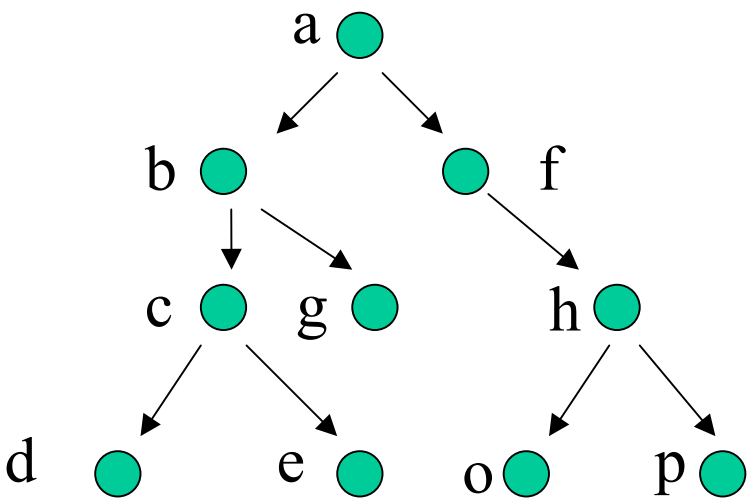
Tree signatures

Let T be an ordered labeled tree. The signature of T is a sequence,

$$sig(T) = \langle t_1, post(t_1); t_2, post(t_2); \ldots; t_m, post(t_m) \rangle$$

of $m = |T|$ entries, where t_i is a name of the node with $pre(t_i) = i$. The $post(t_i)$ is the postorder value of the node named t_i and the preorder value i.

Tree signature - example



A P A ? D

<a,10; b,5; c,3; d,1; e,2; g,4; f,9; h,8; o,6; p,7>

A ? D F

Extended signatures

The extended signature of T is a sequence

$$sig(T) = \langle t_1, post(t_1), ff(t_1), fa(t_1); \ldots ; t_m, post(t_m), ff(t_m), fa(t_m)\rangle$$

where: $ff(t_i)$ – pointer to (index or the preorder of) the first following, for not existing, $ff(t_i) = m+1$;

$fa(t_i)$ – pointer to (index or the preorder of) the first ancestor, for not existing, $fa(t_i) = 0$.

Analytic properties of signatures

- **descendants** $\mathrm{D}(t_i) = \{t_j \mid i < j < ff(t_i)\}$,
  $|\mathrm{D}(t_i)| = ff(t_i) - i - 1$; i.e. $size(t_i)$

- **following** $\mathrm{F}(t_i) = \{t_j \mid ff(t_i) = j = m\}$,
  $|\mathrm{F}(t_i)| = m + 1 - ff(t_i)$;

- **ancestors** $\mathrm{A}(t_i) = \{t_j \mid j < i \wedge post(t_j) > post(t_i)\}$,
  $|\mathrm{A}(t_i)| = ff(t_i) - post(t_i) - 1$; i.e. $level(t_i)$

- **preceding** $\mathrm{P}(t_i) = \{t_j \mid j < i \wedge post(t_j) < post(t_i)\}$,
  $|\mathrm{P}(t_i)| = i + post(t_i) - ff(t_i)$.

Other useful expressions

leaf detection $|\mathrm{D}(t_i)| = 0 \Rightarrow ff(t_i) = i + 1$ or $post(t_{i+1}) > post(t_i)$;

path slicing $t_i \cup \mathrm{A}(t_i) \mid t_i$ is a leaf;

inclusion test if $j < i < ff(t_j)$, t_i is in a sub-tree rooted at t_j;

siblings $[ff(t_i) - post(t_i) = ff(t_j) - post(t_j)] \wedge fa(t_i) = fa(t_j)$;

lowest common ancestor $t_k \in \{t_i \cup \mathrm{A}(t_i) \cap t_j \cup \mathrm{A}(t_j)\} \mid max(pre(t_k)$

XPath navigation

Using the analytic properties, the following XPath axes:

- **Child**
- **Descendant**
- **Parent**
- **Following**
- **Preceding**
- **Following-sibling**
- **Preceding-sibling**

can easily by determined in an obvious way.

Sub-signature

$$sub\_sig_S(T) = \langle\, t_{s1}, post(t_{s1});\ t_{s2}, post(t_{s2});\ \ldots\ ;t_{sk}, post(t_{sk}) \rangle$$

is a sub-sequence of $sig(T)$, defined by the ordered set $S = \{s_1, s_2, \ldots, s_k\}$ of indexes (preorder values) in $sig(T)$, such that $1 = s_1 < s_2 < \ldots < s_k = m$.

Ordered query tree

Given a query tree *Q* and an XML data tree *D*, the tree inclusion of *Q* in *D* is identified by a total mapping from nodes in *Q* to some nodes in *D*, such that the ancestor-descendant and the sibling structural relationships between nodes in *Q* are satisfied by the corresponding nodes in *D*.

We implicitly consider a *week inclusion* of *Q* in *D*:

parent-child relationships in *Q* are satisfied by **ancestor-descendant** relationships in *D*.

Evaluation

- Let Q and D be ordered labeled trees
- Suppose the data tree D and the query tree Q specified by signatures:

$$sig(D) = \langle d_1, post(d_1); d_2, post(d_2); \ldots ; d_m, post(d_m)\rangle$$
$$sig(Q) = \langle q_1, post(q_1); q_2, post(q_2); \ldots ; q_n, post(q_n)\rangle$$

- Tree Q is included in tree D if:

  on the level of node names, $sig(Q)$ is sequence-included in $sig(D)$ determining $sub\_sig_S(D)$ through the ordered set of indexes $S = \{s_1, s_2, \ldots, s_n\} \mid q_1 = d_{s1}, q_2 = d_{s2}, \ldots, q_n = d_{sn}$;

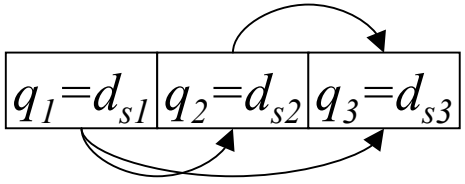
and (depending on the signature)

Using short signatures

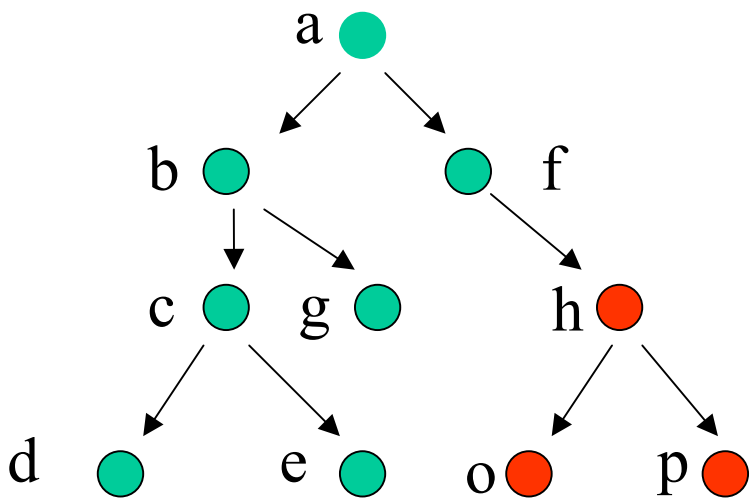
for all pairs of entries i and j in $sig(Q)$ and $sub\_sig_S(D)$

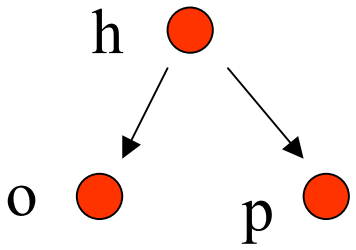
(i,j = 1, 2, ... $|Q|$ - 1 and i+j = |Q|),

$post(q_{i+j}) > post(q_i)$ implies $post(d_{s(i+j)}) > post(d_{si})$ and
$post(q_{i+j}) < post(q_i)$ implies $post(d_{s(i+j)}) < post(d_{si})$



Example - true

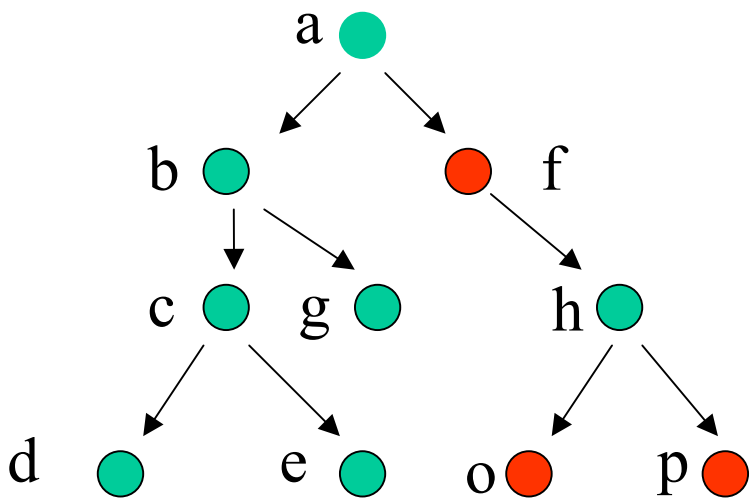


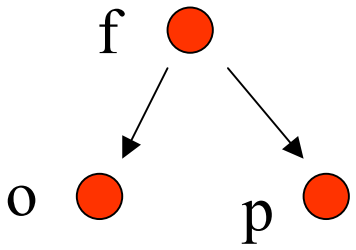


<h,3;o,1;p,2>

<a,10;b,5;c,3;d,1; e,2; g,4; f,9; ***h****,8;* ***o****,6;* ***p****,7>* $\Rightarrow$ *<h,8;o,6;p,7>*

Example - true

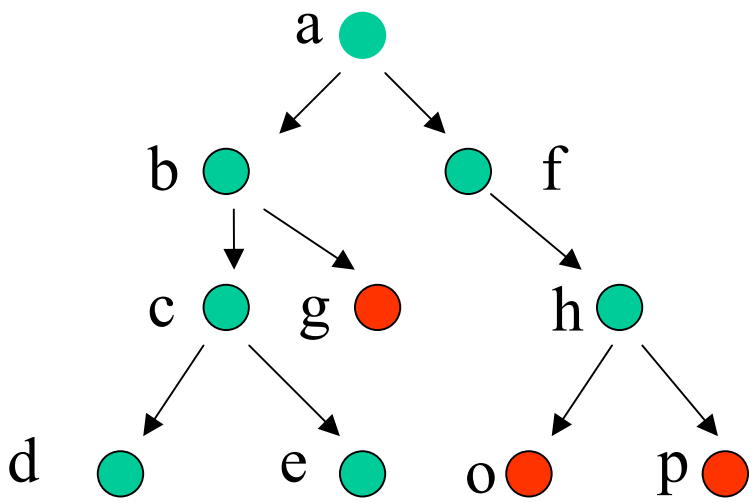


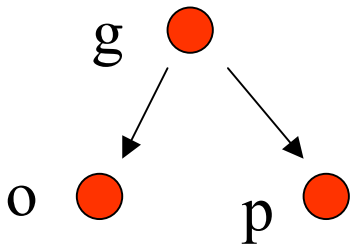


<f,3;o,1;p,2>

<a,10;b,5;c,3;d,1; e,2; g,4; ***f****,9; h,8;* ***o****,6;* ***p****,7> ⇒ <f,9;o,6;p,7>*

Example - false



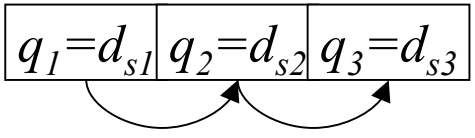


<g,3;o,1;p,2>

<a,10;b,5;c,3;d,1; e,2; ***g****,4; f,9; h,8;* ***o****,6;* ***p****,7>* ⇒ *<g,4;o,6;p,7>*

Using extended signatures

for $i = 1, 2, \ldots |Q| - 1$, if $post(q_i) < post(q_{i+1})$ (leaf node) then $ff(d_{si}) = s_{i+1}$, otherwise $ff(d_{si}) > s_{ff(qi)-1}$.



Query processing

- XQuery example:

```
for $a in // people
where
        $a/name/first="John" and
         $a/name/last="Smith"
return $a/address
```

An execution plan with content indexes

- let R_1 = *ContentIndexSearch*(last = "Smith");
- let R_2 = *ContentIndexSearch*({first = "John");
- let R_3 = *Parent*(R_1 , name);
- let R_4 = *Parent*(R_2, name);
- let R_5 = *Intersect*(R_3,R_4);
- let R_6 = *Parent*(R_5, people);
- let R_7 = *Child*(R_6, address);

Performance evaluation

query template:

```
for $a in //⟨e_name⟩
where ⟨pred($a)⟩
return
        ⟨result⟩ $a/⟨e_1⟩ ... $a/⟨e_n⟩ ⟨/result⟩
```

⟨e_name⟩	name selectivity,
⟨pred($a)⟩	content selectivity,
⟨e_n⟩	number of navigation steps (n).

Query type notation

name selectivity

content selectivity

number of iteration steps

Q N C n

{1,2}

{H(igh), M(edium), L(ow)}

Execution plan for signatures

- let R_1 = *ContentIndexSearch*(<pred>);
- let R_2 = *Parent*(R_1, <e_name>);
- let R_3 = *Child*(R_2, <e_1>);
- let R_4 = *Child*(R_2, <e_2>).

Execution plan for MPMGJN

Multi Predicate MerGe JoiN:

- let R_1 = *ContentIndexSearch*(<pred>);
- let R_2 = *ElementIndexSearch*(<e_name>);
- let R_3 = *ContainingParent*(R_2, R_1);
- let R_4 = *ElementIndexSearch*(<e_1>);
- let R_5 = *ContainedChild*(R_4, R_3);
- let R_6 = *ElementIndexSearch*({ <e_2>);
- let R_7 = *ContainedChild*(R_6, R_3).

Results for DBLP

query	signature	join	retrieved el.
QHH1	80	466	1
QHM1	320	738	1
QHL1	538	742	1
QMH1	88	724	1
QMM1	334	832	9
QML1	550	882	60
QLH1	95	740	38
QLM1	410	1421	1065
QLL1	1389	1282	13 805

Results for DBLP (cont.)

query	signature	join	retrieved el.
QHH2	90	763	1
QHM2	352	942	1
QHL2	582	966	1
QMH2	130	822	1
QMM2	376	1327	9
QML2	602	1220	60
QLH2	142	1159	38
QLM2	450	1664	1065
QLL2	2041	1589	13 805

Unordered query tree

Given a query tree *Q* and an XML data tree *D*, the tree inclusion of *Q* in *D* is identified by a total mapping from nodes in *Q* to some nodes in *D*, such that the ancestor-descendant structural relationships between nodes in *Q* are satisfied by the corresponding nodes in *D* – the sibling relationships are not considered.

We implicitly consider a *week inclusion* of *Q* in *D*:

parent-child relationships in *Q* are satisfied by **ancestor-descendant** relationships in *D*.

Evaluation

- Let D be an ordered and Q an unordered labeled trees
- Suppose the data tree D and the query tree Q specified by signatures:

$$sig(D) = \langle d_1, post(d_1); d_2, post(d_2); \ldots ; d_m, post(d_m) \rangle$$

$$sig(Q) = \langle q_1, post(q_1); q_2, post(q_2); \ldots ; q_n, post(q_n) \rangle$$

Evaluation (cont)

- on the level of node names, an ordered set of indexes $S=\{s_1, s_2, \ldots, s_n\}$ exists, such that $1= s_i = n$ and $d_{si} = q_i$, for $i=1, \ldots, n;$
- for all pairs of entries i and j,

  $i,j = 1, 2, \ldots, |Q|-1$ and $i+j = |Q|$:

  if $post(q_{i+j}) < post(q_i)$ then

  $post(d_{s(i+j)}) < post(d_{s(i)})$ and $s_{i+1} > s_i$

Approaches

- **permutation**: consider all and only such alternatives of Q, satisfying the ancestor-descendent relationships;
- **decomposition**:

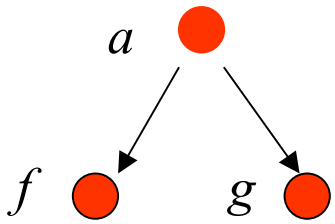
  - decompose the query into a set of paths

  - determine inclusions of the paths

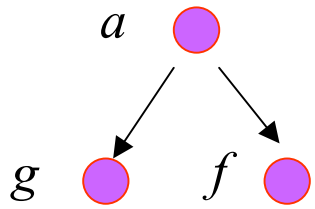
  - find structural consistent alternatives

Example

Query tree

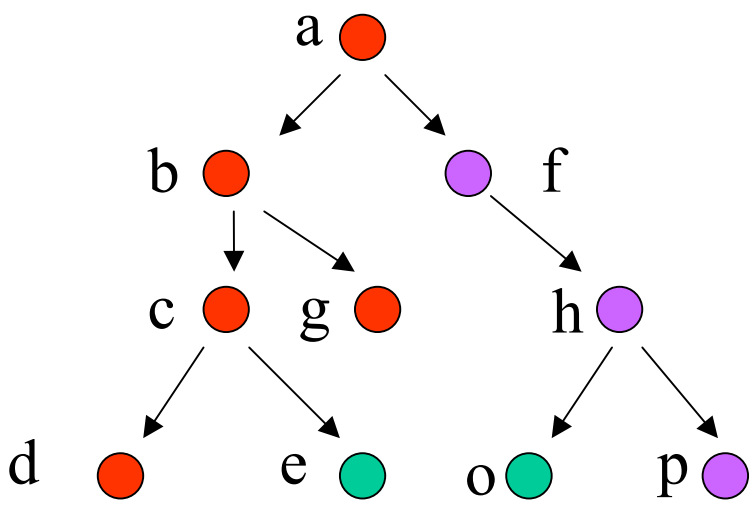


Query permutation



Path Decomposition:

Structurally consistent paths



P1: a,b,c,d
P2: a,b,g
P3: f,h,p

Only P1 and P2 are structurally consistent

Future directions

- **ranking** of results and **similarity** search
- **unordered** query trees
- **searching for structures** of a set (bag) of nodes
- **searching for nodes** (data) structured in a specific way
- any combination of the above