On Labeling Schemes for the Semantic Web

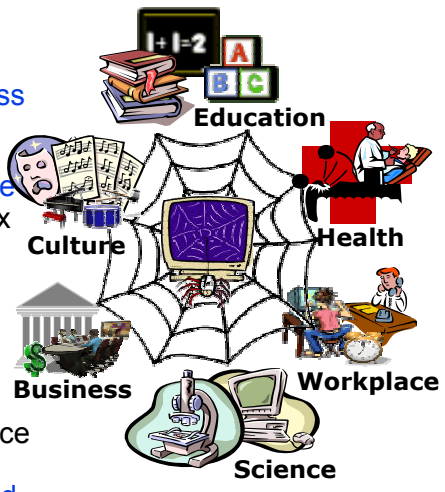
V. Christophides D. Plexousakis M. Scholl
S Tourtounis

Computer Science Department, University of Crete
and Institute for Computer Science - FORTH
Heraklion, Crete, Greece

CEDRIC/CNAM and INRIA
292 Rue St Martin
75141 Paris, Cedex 03, France

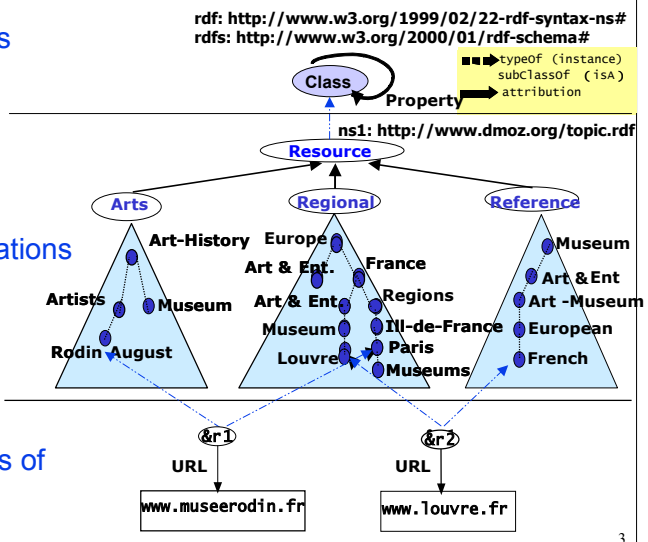
Semantic Web Portals & RDF/S

- More and more Portals adopt Semantic Web technology to provide:
  - a single point of useful, ubiquitous comprehensive, and integrated access to various information resources
- RDF enables to create complex resource descriptions using a standardized syntax
- RDFS allows to define expressive description schemes (taxonomies, ontologies) for resources using formal concepts/properties
- Having rich RDF/S schemes and resource descriptions (i.e., metadata) we can support more sophisticated browsing and querying functionality



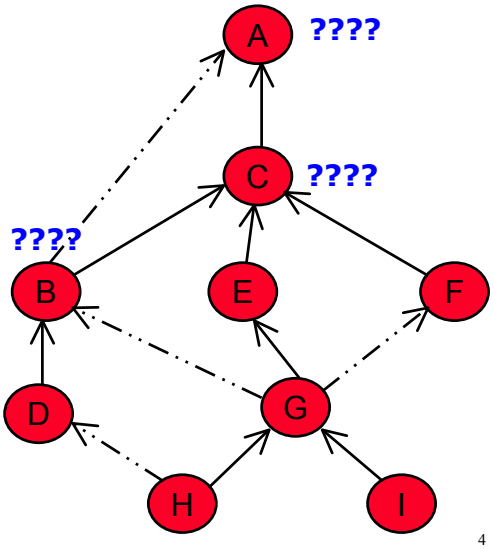
Advanced Querying/Browsing for SW Portals

- Core Functionality
  - ◆Ancestors/Descendents
  - ◆Leaves
  - ◆Precedings/Followings
  - ◆Siblings
  - ◆NCA
- Transitive closure computations over class (or property) subsumption semi-lattices
  - Optimize recursive traversal using range queries
    - gain up to 3-4 orders of magnitude



Our Objective

- Choose an adequate labeling scheme, taking into account that:
  - Portals' subsumption hierarchies range from Trees to DAGs
    - common labeling scheme with a reasonable extra storage & querying cost
  - Querying/Browsing Portals' catalogs relies on bulk class retrieval using filtering conditions on subsumption relationships
    - generated class labels should be efficiently processed by a database backend using standard index structures
- Evaluate core traversal functionality on top of an DBMS (PostgreSQL)
  - Both source and synthetic data

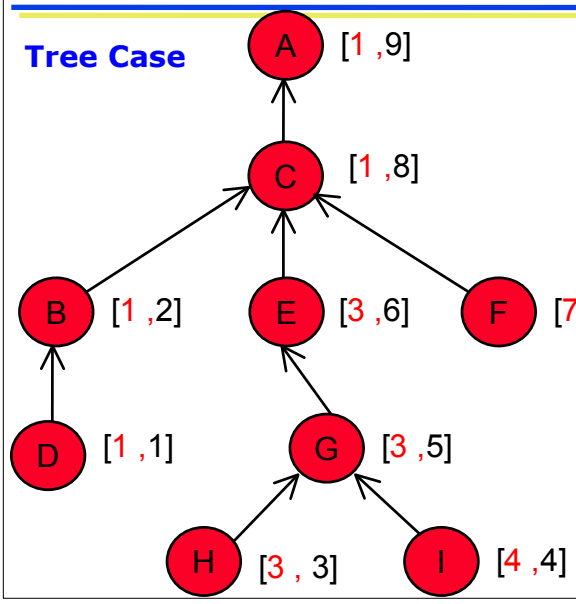


Outline

- Families of Labeling Schemes (object-oriented programming, knowledge representation systems, XML search engines)
  - bit-vector
  - prefix-based
  - interval-based
- Experimental Evaluation of two labeling schemes for both Trees and DAGs
  - Testbed: Open Directory Portal Statistics
  - Relational Representation
  - Translation into SQL
  - Query Execution Performance
- Conclusions and Future Work

Interval-based Labeling Scheme: Agrawal et al

Tree Case

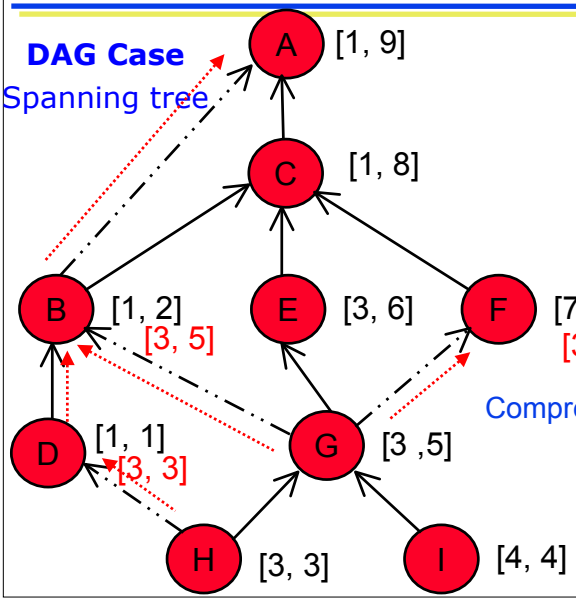


Interval of descendant is included in the interval of ancestor

Ordering of labels based on post numbers

1 < 2 < …< 9

Interval-based Labeling Scheme: Agrawal et al

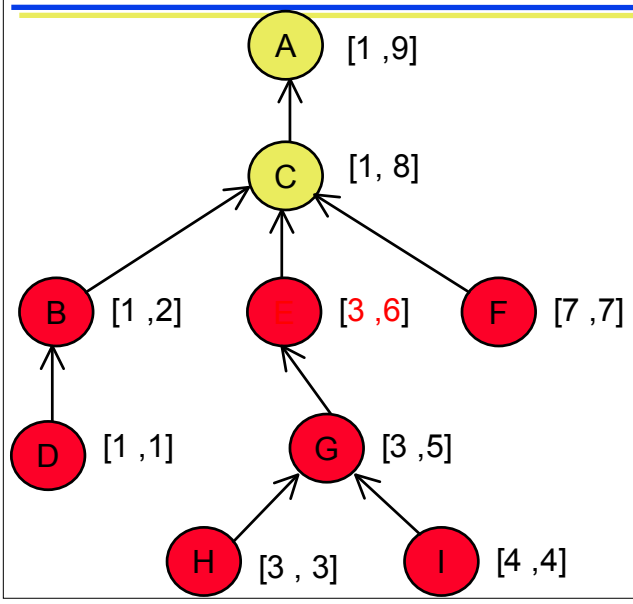


[1, 9] + [1, 2] + [3, 5]
Compression [1, 2] and [3, 5] are already included in [1, 9]

[1, 2] + [3, 3] + [3, 5]
Compression [3, 3] is included in [3, 5]

[1, 1] + [3, 3]

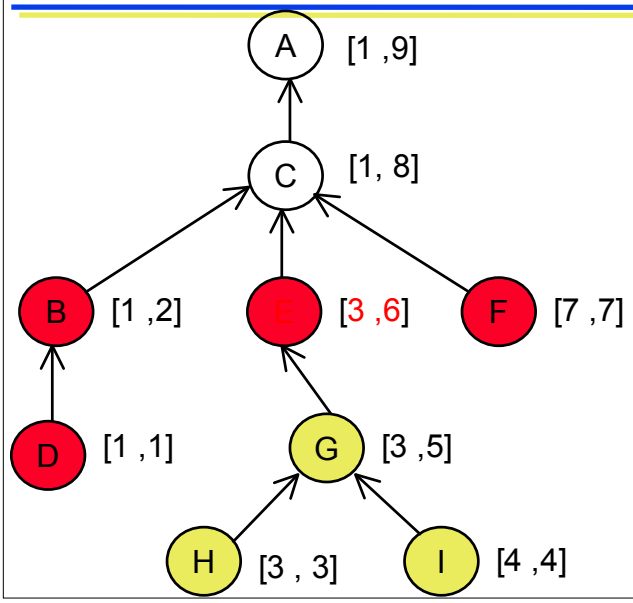
Agrawal et al Core Queries on Trees



- Ancestors(E)

```
post > post(E) ∧
index <= post(E)
```

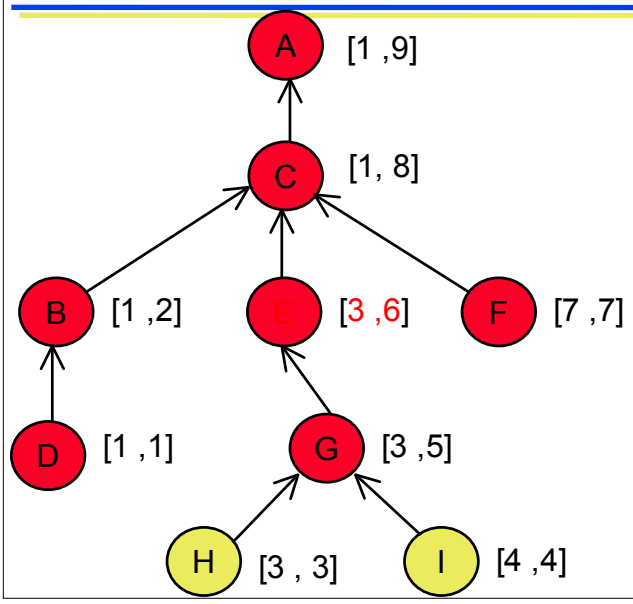
Agrawal et al Core Queries on Trees



- Descendants(E)

```
post < post(E) ∧
index >= post(E)
```


Agrawal et al Core Queries on Trees

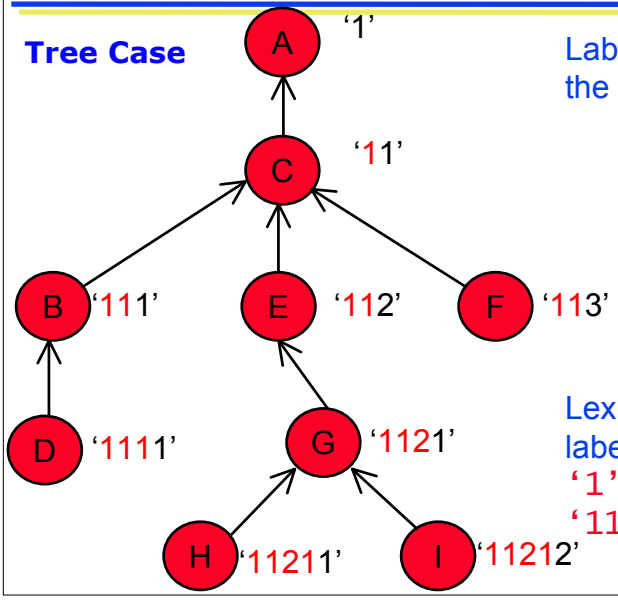


- Leaves(E)

```
post > post(E) ∧
index <= post(E)∧
index = post
```

Prefix-based Labeling Scheme: Dewey

Tree Case



Label of ancestor is a prefix of the label of the descendant

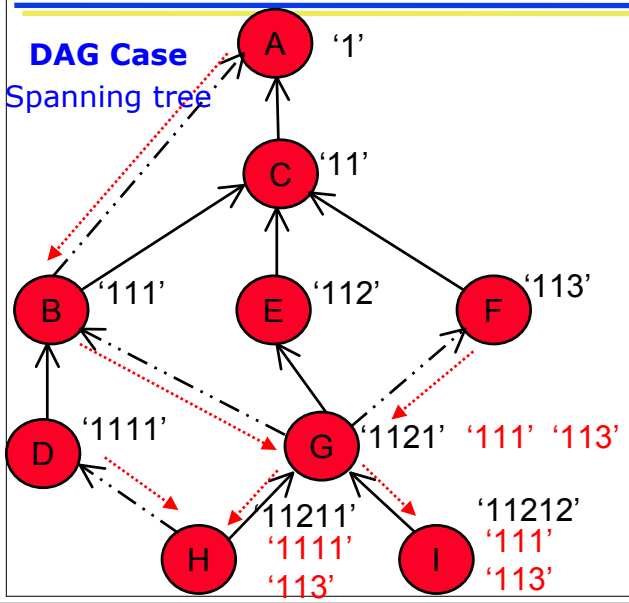
Lexicographic Ordering of labels

'1'< '11'< '111'< '1111'< '112' < ...

Prefix-based Labeling Scheme: Dewey

DAG Case

Spanning tree



Compression

'1' is a prefix of '111'

Tree edges

'1121'

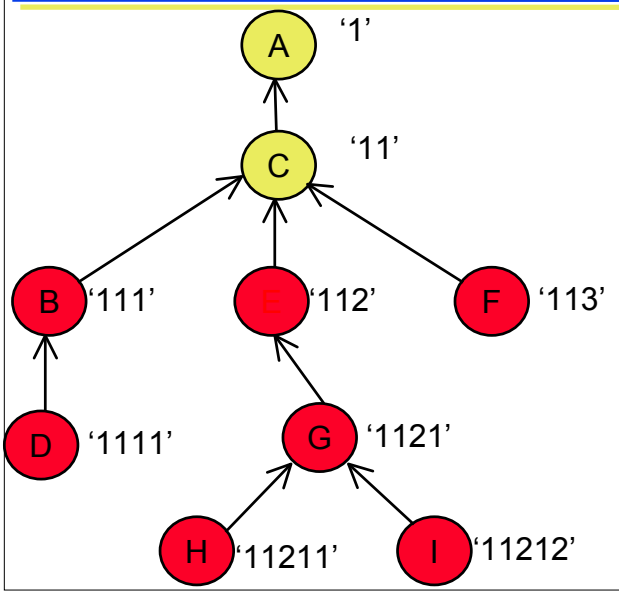
Non-tree edges

'111', '113', '1111'

Compression

'111' is a prefix of '1111'

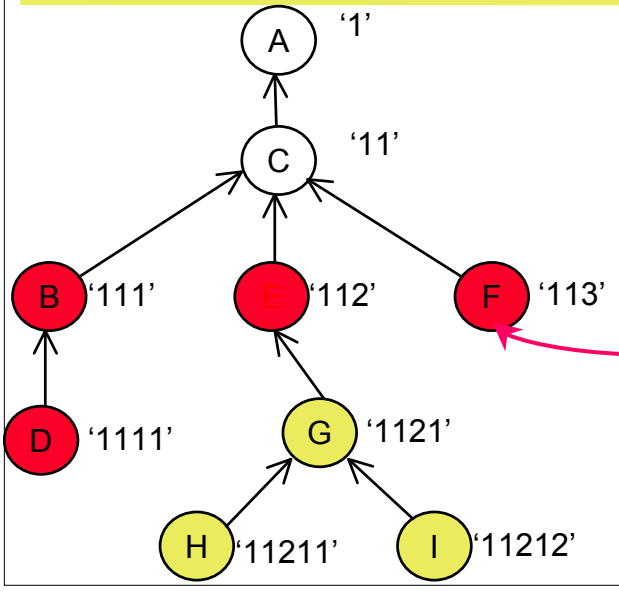
Agrawal et al Core Queries on Trees



- Ancestors(E)

  `prefixes(l(E))`

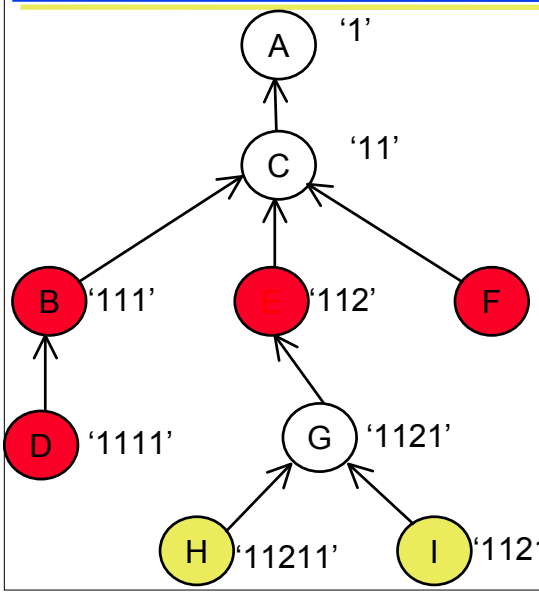
Agrawal et al Core Queries on Trees



- Descendants(E)

  `l(E) < l < next(l(E))`

Agrawal et al Core Queries on Trees



- Leaves(E)

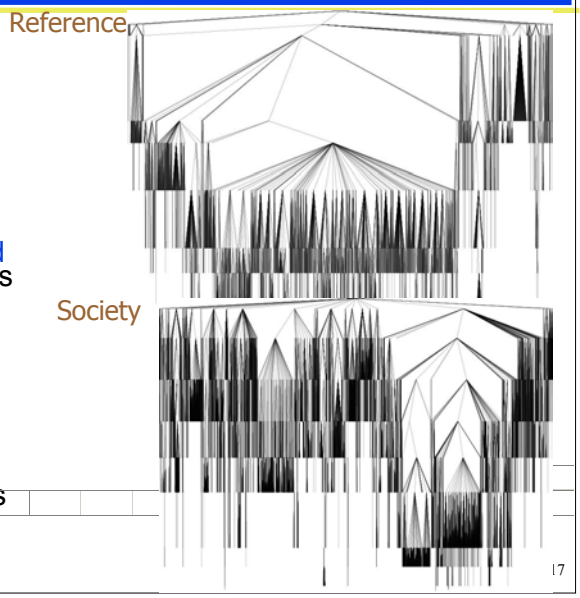
$l(E) < l < next(l(E)) \wedge \nexists\ u' : l(E) < l(u') < next(l(E)) \wedge l < l(u') < next(l)$

Experimental Evaluation

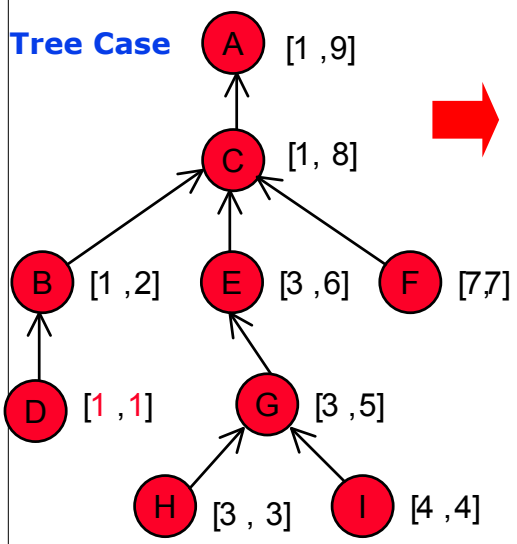
- Two labeling schemes
  - **Pinterval** (fixed label size): Extended postorder *interval-based* scheme by *Agrawal*, *Borgida* and *Jagadish*
  - **UPrefix** (variable label size): Prefix-based Unicode Dewey scheme
    - variable number of bytes to encode labels per level using Unicode Character Set (UTF-8)
- Testbed : The ODP Catalogue
  - Topic subsumption trees (single *isA*)
  - Synthetically generated DAGs (multiple *isA*)
- Evaluation Criteria for both Trees and DAGs
  - Storage volume of encoded classes
  - Execution time of basic queries
- DB Setting
  - *PostgreSQL* (7.2.1) with Unicode configuration – Ultra Sparc III, 750 MHz processor and 512 MB of main memory
  - Selection of 1000 buffers (8 KB)

Netscape Open Directory Portal (Dmoz)

- ODP Version 16-01-2001
- 253214 Topics
  - 17,77% of terms is replicated into more than one hierarchies
- 1715225 Resources
  - 6,93% is multiply classified under more than one topics
- Average depth 7,83 (Maximum 13)
- Average fan-in 1 (Maximum 314)
- 75,16% leaf topics
- 60% with 6-11 ancestor topics
- ODP subtopic trees are far from complete



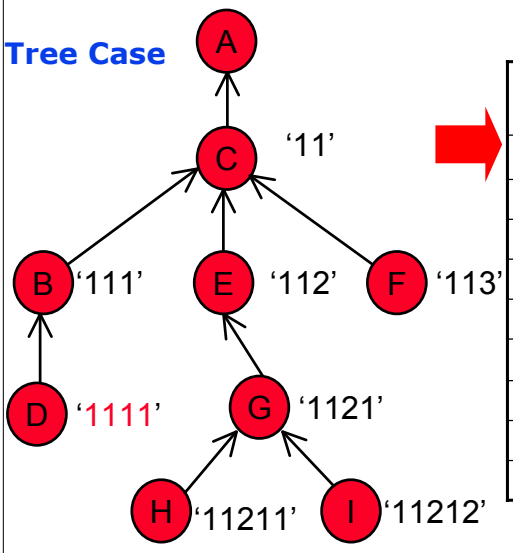
Relational Representation – PInterval



PInterval

index :int4	*Post : int4*	*father_post :int4*
1	1	2
1	2	8
1	8	9
3	3	5
4	4	5
3	5	6
3	6	8
7	7	8
1	9	–

Relational Representation – UPrefix

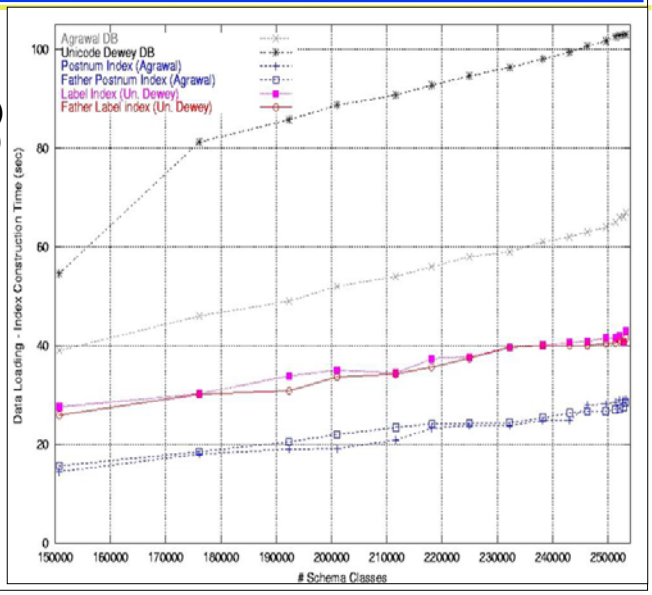


UPrefix

label: varchar(15)	*father_label: vachar(15)*
'1111'	'111'
'111'	'11'
'11'	'1'
'112'	'11'
'113'	'11'
'1121'	'112'
'11211'	'1121'
'11212'	'1121'
'1'	-

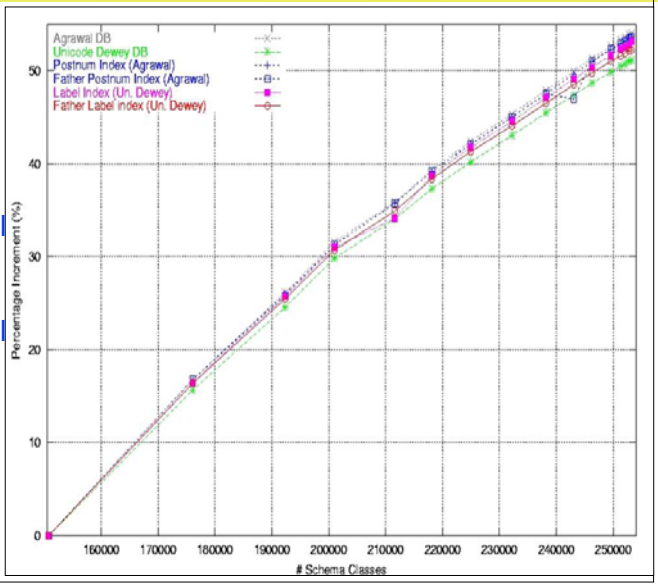
Storage Space

- UPrefix 21,2% >PInterval
- Index on UPrefix (label) 29,8% > PInterval (post)
- PInterval Size 12902 KB (1613 disk pages)
  Tuple = 52,17 b
- UPrefix Size 16376 KB (2073 disk pages)
  - Tuple 66,22 b
- 40 bytes generated by *PostgreSQL as tuple ids*

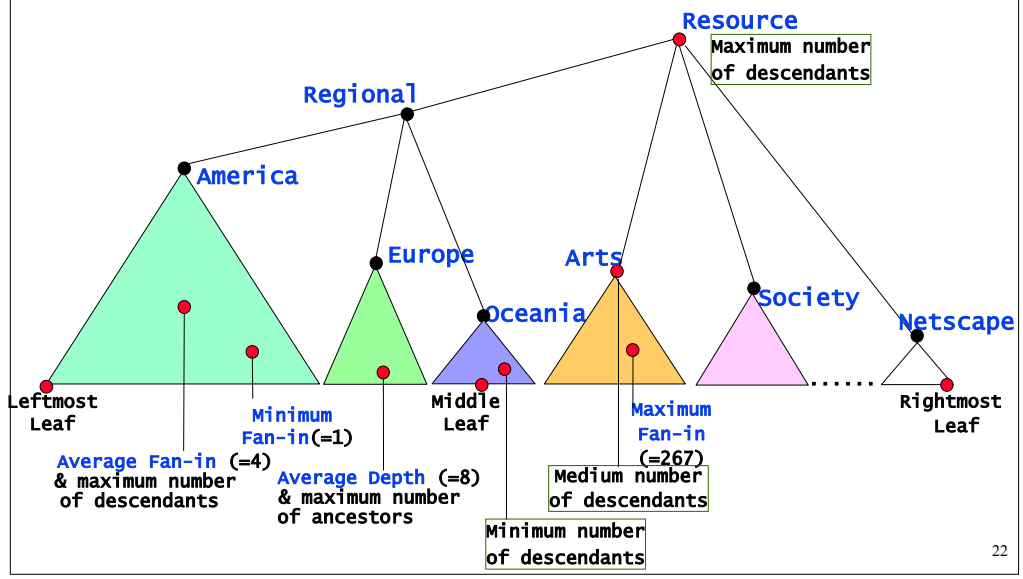


Loading Time

- Data Loading
  UPrefix 34,75% > Pinterval
- Index Construction
  UPrefix 32,21% > Pinterval



Testbed Queries on ODP



Evaluation of Testbed Queries in ODP Topics Tree

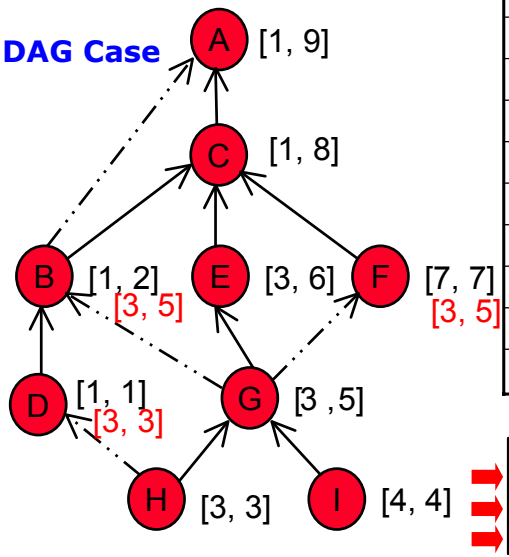
Query	PInterval			UPrefix		
	Case 1 %Select	Case 2 %Select	Case 3 %Select	Case 1 %Select	Case 2 %Select	Case 3 %Select
Subsumption check	0,00018	0,00018	0,00017	0,00017	0,00017	0,00017
Q1 Descendants	2,869392 100%	0,429730 10%	0,00027 0,0004%	3,296126 100%	0,57702 10%	0,00027 0,0004%
Q2 Ancestors	1,18904 0,002%	1,54799 0,00158%	0,00027 0,00355%	0,00163 0,002%	0,00178 0,00158%	0,00167 0,00355%
Q3 Leaves	2,957039 75,13%	0,493230 8,46%	0,00027 0,0004%	25,54267 75,13%	16,51963 8,46%	0,00055 0,0004%
Q4 Precedings	2,629188 100%	2,34529 50%	0,00024 0%	2,874457 100%	2,345290 50%	0,0005 0%
Q5 Followings	2,9721 100%	2,48579 50%	0,00024 0%	3,37825 100%	2,566292 50%	0,0005 0%
Q6 Siblings	0,00863 0,1236%	0,00054 0,002%	0,00047 0,0004%	0,00961 0,1236%	0,00056 0,002%	0,00049 0,0004%
Q7 nca	3,22742 0,0004%	0,00041 0,0004%	0,000438 0,0004%	0,0003945 0,0004%	0,0003945 0,0004%	0,0003945 0,0004%

- 3 Case studies of different selectivity and/or nodes in the hierarchy

Query Performance Comparison for Trees

- Comparable query performance for both labeling schemes
  - For queries with the same selectivity `PInterval` is slightly better than `UPrefix` (smaller # disk pages)
  - Leaves query in `UPrefix` is penalized by the nested condition
  - Ancestors and NCA queries in `UPrefix` run in practically constant time (stored procedures)
- Performance limitation of SQL queries employing user-defined functions (e.g., `next()` in `UPrefix`)
  - They are not evaluated using the indices (Sequential scans)
- PostgreSQL questionable choice of good evaluation plans
  - Index Scans even for queries >50% selectivity

Relational Representation – DPInterval

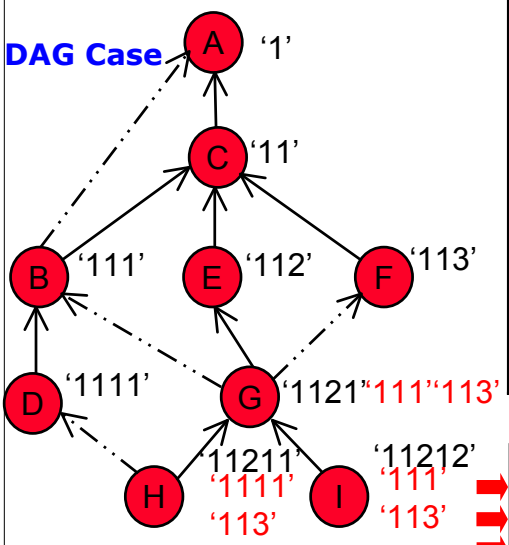


index :int4	Post : int4	fat_post :int4
1	1	2
1	2	8
1	8	9
3	3	5
4	4	5
3	5	6
3	6	8
7	7	8
1	9	-

DPInterval

3	3	1
3	5	2
3	5	7

Relational Representation – DUPrefix



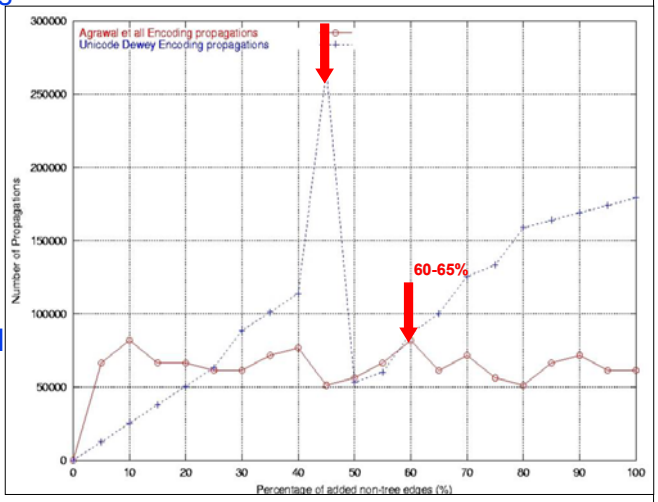
label: varchar(15)	fat_label: vachar(15)
‘1111’	‘111’
‘111’	’11’
‘11’	‘1’
‘112’	’11’
‘113’	’11’
‘1121’	‘112’
‘11211’	‘1121’
‘11212’	‘1121’
‘1’	-

DUPrefix

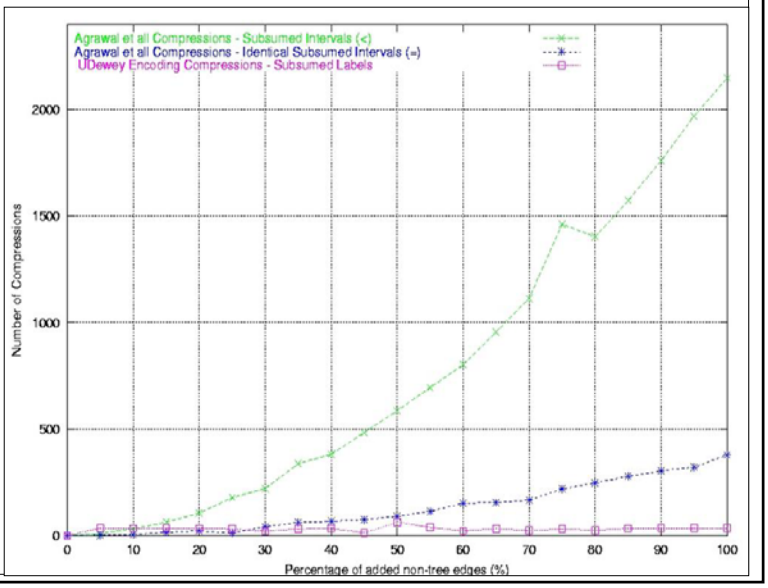
‘1121’	‘111’
‘11212’	‘111’
‘11211’	‘1111’

Comparing Label Propagations

- `DUPrefix` is 3 times bigger than `DPInterval` (with label compression)
  - DUPrefix = 179270 tuples
  - DPInterval = 62730 tuples
- `DPInterval` is (without compression) 3 times bigger than `DPInterval` (with compression)
  - DPInterval = 191812 tuples



Comparing Label Compressions



Core Query Functionality for DAGs

Query	DPinterval	DUPrefix
Ddescendants(v) {u| u ≺ v}	*descendants*(v) $\cup_{w \in Propdesc(v)}$ *descendants*(***w***)	*descendants*(v) ∪*Propdown*(v)
Dancestors(u) {u |u ≻ v}	*ancestors*(v) ∪ *Propup*(v)	*ancestors(v)* $\cup_{w \in Propansc(v)}$ *ancestors*(***w***)
Dleaves(u) {u | u ≺ v ∧ ∄ u': (u' ≺ v ∧ u' ≺ u)}	{u | u ∈ ***Ddescendants***(v) ∧ *Propdesc*(u) = 0 ∧ ∄ u' | u' ∈ ***Ddescendants***(u) ∧ *Propup*(u') =0}	{u | u ∈ ***Ddescendants***(v) ∧ *Propdown*(u) = 0 ∧ ∄ u' | u' ∈ ***Ddescendants***(u) ∧ *Propdown*(u') =0}
Dnca(v,w) {u | u ≻ v ∧ u ≻ w ∧ ∄ u': (u' ≻ v ∧ u' ≻ w ∧ u' ≺ u)}	{u | u ∈ ***Dancestors***(v) ∧ u ∈ ***Dancestors***(w) ∧ ∄ u' | u' ∈ ***Dancestors***(u) ∧ u' ∈ ***Dancestors***(w) ∧ u' ∈ ***Ddescendants***(u) }	{u | u ∈ ***Dancestors***(v) ∧ u ∈ ***Dancestors***(w) ∧ ∄ u' | u' ∈ ***Dancestors***(u) ∧ u' ∈ ***Dancestors***(w) ∧ u' ∈ ***Ddescendants***(u) }

SQL Translation of Descendants Query

```
select u.post
from PInterval u
where u.post < P and
      u.post >= I
Union All
select u.post
from PInterval u,
     DPInterval w
where w.ancestor = P
  and u.post <= w.post
  and u.post >= w.index
```

```
select u.label
from UPrefix u
where 'l' < u.label and
   u.label < 'l' || 'xFF'
Union All
select w.label
from DUPrefix w,
    (select u.label as label
     from UPrefix u
     where 'l'<= u.label and
     u.label < 'l' || 'xFF'
     ) u'
where w.ancestor = u'.label
```

Evaluation of Testbed Queries in ODP Topics DAGs

Query	DPInterval			DUPrefix		
	Case 1 %Select	Case 2 %Select	Case 3 %Select	Case 1 %Select	Case 2 %Select	Case 3 %Select
Q1 Descendants	11,92988 50%	0,617279 5,051%	0,000263 0,0004%	67,12124 50%	21,78665 5,051%	20,4409 0,0004%
Q2 Ancestors	3,790538 0,033%	2,6323 0,00454%	3,095479 0,00276%	0,00619 0,033%	0,00514 0,00454%	0,00472 0,00276%
Q3 Leaves	14,50642 21,55%	1,57837 2,211%	0,000087 0,0002%	95,58222 21,55%	48,192656 2,211%	47,297535 0,0002%
Q4 Siblings	0,002591 0,0626%	0,002442 0,0008%	0,00033 0,0004%	0,003011 0,0626%	0,002946 0,0008%	0,00032 0,0004%
Q5 nca	4,870160 0,0002%	4,691551 0,0002%	3,70287 0,0002%	0,00058 0,0002%	0,00057 0,0002%	0,00055 0,0002%

- Same input classes in queries with the case of trees
  - Addition of non-spanning tree edges → double size of `PInterval` and `UPrefix` & accordingly a decrease to queries selectivities

Query Performance Comparison for DAGs

- `DPIternal` outperforms `DUPrefix` up to 5 orders of magnitude for descendants and leaves queries
  - especially for high selectivity cases
- Labels compression complicates the translation into SQL of queries (*Propup* and *Propdown*) BUT
  - do not require costly duplicate elimination of labels (UNION ALL vs UNION)
- PostgreSQL questionable choice of good evaluation plans
  - Merge Joins even Hash Joins or Nested Loops are more efficient

Conclusions

- For voluminous subsumption hierarchies labeling schemes are 3-4 orders of magnitude better than transitive closure computations
- For tree-shaped hierarchies the storage requirements and query performances of `DPIternal` and `DUPrefix` schemes are comparable
- For DAG-shaped hierarchies we need for `DPIternal` (`DUPrefix`) 2.4 (2.7) times more space when label compression is considered
  - `DPIternal` is less sensitive to labels' propagation w.r.t. the actual position of the added DAG edges
  - Significant divergence when > 65% DAG edges are added
- Compared to the case of trees, Descendant and Leaf queries in
  - `DPIternal` are up to 5 times more costly
  - `DUPrefix` are up to 5 orders of magnitude more costly
- Ancestors and NCA queries in `DUPrefix` run practically in constant time for both cases
- When label propagation is omitted the two schemes exhibit almost the same storage requirement while query performances is slightly improved

Future Work

- Support for Incremental updates
  - Find good relabeling algorithms (worst case *log(n)* vs *n*)
- Extend interval-based scheme for DAGs by avoiding label propagation
  - Study formal characterization of DAGs which can be labeled using >= 2 dimensions
- Experimental evaluation using other DBMS than PostgreSQL