ISTI-CNR

# Implementing efficient XML query processing in ECD

Giuseppe Amato
ISTI-CNR
g.amato@iei.pi.cnr.it

# Overview

- Overall activity:
  - XXQuery
  - XXQuery processing/optimisation
  - Access methods (Indexes) for XXQuery processing
  - Efficient XML data Storing
  - Use of schemas (DTD or XML Schema)
- Completed/current activity
  - Access methods
    - Path index
    - XML signatures
    - Access methods for similarity search
  - XML query processing

# XXQuery

- XXQuery: approXimate Xquery
  - ✓ Xquery is a language for searching in XML collections
    - Exact search is only possible
  - ✓ XXQuery includes approximate and similarity operators

XQuery

```
for $a in /library//articles
where $a/topic = 'Computer'
return $a/abstract
```

XXQuery

```
for $a in /~library//articles
where $a/topic ~ 'Computer'
return $a/abstract
```

# Query processing/optimisation

- Definition of a physical query algebra on XML data
  - Supported by
    - efficient index based processing
    - efficient data (record) based processing
  - Translation of XXQuery queries into sequences of operations of the defined algebra
- Definition of cost execution models
- Definition of optimisation heuristics
  - Generating equivalent execution plans
  - Determining the most efficient one

# Query processing

- Path expression processing example.:

```
/~library//book[topic ~ 'Computer']
```

  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:

```
/~library//book[topic ~ 'Computer']
```

  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:


  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:


  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:

```
/~library//book[topic ~ 'Computer']
```

  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:

```
/~library//book[topic ~ 'Computer']
```

  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:


  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Query processing

- Path expression processing example.:

```
/~library//book[topic ~ 'Computer']
```

  - ✓ With relational databases this can be obtained as a sequence of join (containment join)
  - ✓ For path containing high frequency elements a single containment join might require hours

# Access methods

- Processing XXQuery involves
  - ✓ Exact content search
    - Specific attributes or elements content can be indexed
  - ✓ Approximate content search
    - Full text indexing
    - Ontology
    - Multimedia content indexing
  - ✓ Path traversal
    - Efficient decision on tree/path inclusion or other relationships

# XML data storing

# XML data storing

- Flat files
  - Natural for storing XML
  - No concurrency control
  - No recovery
  - Query processing efficiency ??
    - Blind sequential scan is not a solution
    - Ad-hoc index-structures should be designed
  - Good for generating output

# XML data storing

- Flat files
  - Natural for storing XML
  - No concurrency control
  - No recovery
  - Query processing efficiency ??
    - Blind sequential scan is not a solution
    - Ad-hoc index-structures should be designed
  - Good for generating output
- Relational databases
  - Data must be translated into relational form
  - Advantage
    - mature database systems
  - Disadvantage
    - overhead of translating data and queries
    - Data broken up into many pieces, increasing space overheads
    - Simple queries require a large number of joins
    - Generating query output might have an high cost

# Use of schemas

# Use of schemas

- Extension of term/path ontology by using schema mapping
  - ✓ Documents with different schemas may contain distinct paths having the same meaning
    - /library/book/author/name
    - /bib/book/info/creator/firstname

# Use of schemas

- Extension of term/path ontology by using schema mapping
  - Documents with different schemas may contain distinct paths having the same meaning
    - /library/book/author/name
    - /bib/book/info/creator/firstname
- Decision on index creation
  - Exact search indexes
  - Full text indexes
  - Multimedia indexes

# Use of schemas

- Extension of term/path ontology by using schema mapping
  - ✓ Documents with different schemas may contain distinct paths having the same meaning
    - /library/book/author/name
    - /bib/book/info/creator/firstname
- Decision on index creation
  - ✓ Exact search indexes
  - ✓ Full text indexes
  - ✓ Multimedia indexes
- XML data storing
  - ✓ Mapping schemas (DTDs or XML schemas) to relational schemas
  - ✓ Efficient storage
  - ✓ Efficient query processing

# Use of schemas

- Extension of term/path ontology by using schema mapping
  - ✓ Documents with different schemas may contain distinct paths having the same meaning
    - /library/book/author/name
    - /bib/book/info/creator/firstname
- Decision on index creation
  - ✓ Exact search indexes
  - ✓ Full text indexes
  - ✓ Multimedia indexes
- XML data storing
  - ✓ Mapping schemas (DTDs or XML schemas) to relational schemas
  - ✓ Efficient storage
  - ✓ Efficient query processing
- Decisions on content identification
  - ✓ E.g.: elimination of formatting tags for full text indexing

# Content identification

```
<abstract> This paper describes how to
  <em>index XML documents </em>. Several
  strategies can be used…</abstract>
```

```
<abstract>
   This paper describes how to
   <em>
      index XML documents
   </em>
   . Several strategies can be used…
</abstract>
```

# Current activity at ISTI

- Definition of access methods
  - ✓ XML Path index
  - ✓ XML Tree signatures
  - ✓ Access methods for similarity search
  - ✓ Query processing

# XML Path index

- Path can be resolved with relational databases using sequence of joins
  - ✓ Very high cost
- Our technique can process very efficiently typical patterns of XPath expressions:
  - ✓ For example:
    - `/dblp/inproceedings/author`
    - `//book/author`
    - `/dblp/book//`
    - `/dblp//author`
    - `//book//`
- Other complex patterns can be obtained as combination of them

# XML Path index

- Also attributes can be specified in a path
  - `/people/person/@name`
- Values associated with specific elements or attributes can also be indexed
  - `/people/person[@name=john]`
- Disk based and highly scalable

# XML Tree Signature

- Compact description of the structure of an XML document
  - ✓ They can be used to
    - Perform complex Query tree matching
    - Navigate the structure of XML documents
    - Perform containment tests efficiently
    - Perform path joins efficiently
    - Implementing projection (XQuery return clause)

# Access methods for similarity search

- Suppose an element refers to an image
- We want to support queries like
  - ✓ Give me all elements that refers to SIMILAR images
- Two basic similarity queries:

  - ✓ Range search
    - Threshold on the minimum similarity allowed
    - Ex.: Retrieve objects whose similarity with *Q* is at lest *x*

  - ✓ Nearest neighbours search
    - Threshold on the maximum number objects to be retrieved
    - Ex.: Retrieve the *k* objects closest to *Q*

# Query processing

- Path expression
  - Using Path index or Signature
- Hierarchical relationships
  - Parents
    - With signatures
  - Ancestors
    - Multi-predicate join or signatures
  - Children/descendants
    - Signatures
- Content filtering
  - XML text scan
  - Path index
  - B-Trees

# Preliminary experiments

- DBLP
  - 120 Mb, 3.181.399 elements
- Building indexes: about 10 minutes
  - Path index size: 208 Kb
  - Signature: 36Mb
  - Posting lists: 12Mb
  - Offsets: 48Mb

# Preliminary experiments

- Queries:
  - ✓ Pure XPath expressions with path indexes

    `/DBLP//AUTHOR, //INPROCEEDINGS/AUTHOR`

    - < 0.1 Sec
  - ✓ High frequency element filtering with no special indexing
    - `/DBLP//AUTHOR="Giuseppe Amato"`: 679.696 elements

    <15 Sec
  - ✓ High frequency element filtering with special indexing

    `/DBLP//AUTHOR="Giuseppe Amato"`

    - < 0.1 Sec
  - ✓ High frequency element parent

    `/DBLP//AUTHOR`: 679.696 elements

    <10 Sec

# Access methods for similarity search

| Method | Cost | Time |
|---|---|---|
| **Sequential scan** | 1,000,000 | 5 Minutes |
| **M-Trees** | 200,000 | 1 Minute |
| **D-Index** | 20,000 | 6 Seconds |
| **AM-Trees** | 2,000 | 0.6 Seconds |

# Conclusions

- Supporting efficient XML queries on large repositories is a challenge
- Relational database technology might help but it should be enhanced with new ad-hoc techniques
  - ✓ Access methods
  - ✓ Join algorithms
  - ✓ Storage strategies
- Object oriented and semi-structured databases deal with similar issues, however they still cannot compete with commercial relation databases