Using Data Mining for XML Data Storage and Compression

Dan Suciu

AT&T Labs

www.research.att.com/~suciu

Joint work with A. Deutsch, M. Fernandez, H. Liefke

Data Mining Applications

- "Data mining aims at the discovery of useful information from large collections of data" [Mannila'97]
- what is "useful" ?
- this talk: use data mining to help XML data processing

Outline

- XML storage
- XML compression   problem statement only

Part 1

XML Storage

XML – Example

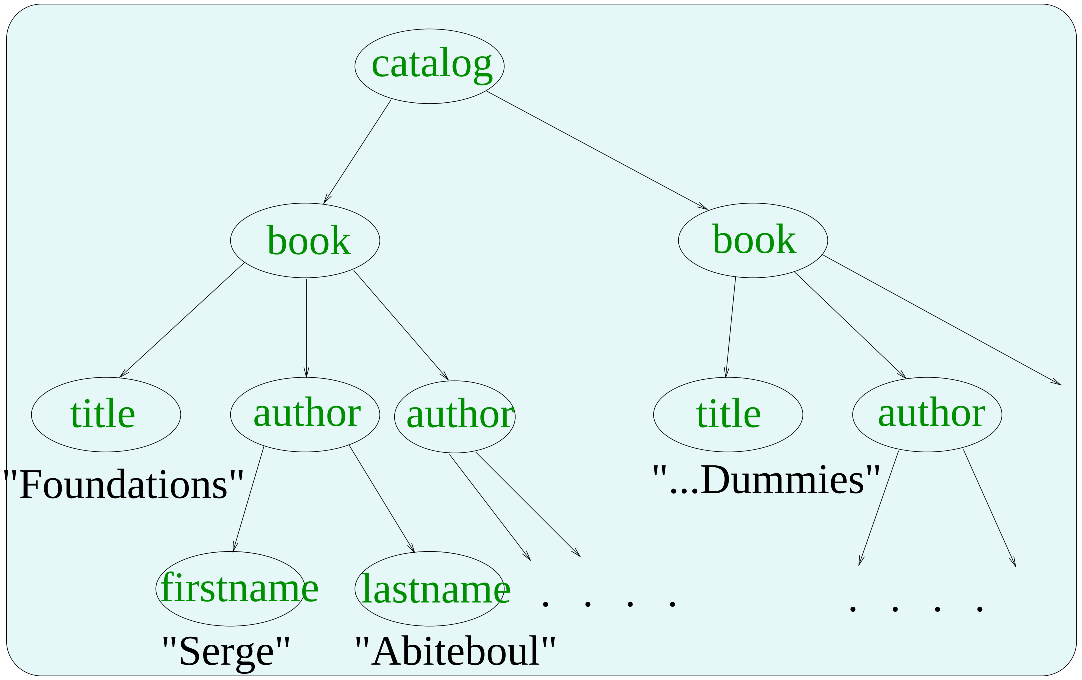
```
<bibliography> <book> <title> Foundations of Databases </title>
                      <author> Abiteboul </author>
                      <author> Hull </author>
                      <author> Vianu </author>
                      <publisher> Addison Wesley </publisher>
                      <year> 1995 </year>
               </book>
               . . .
</bibliography>
```

XML Differs from Relational Data

Relational data	XML
tabular	tree-like

Books

title	price
Foundations . . .	44.99
Databases for Dummies	29.99
Oracle DBA	39.99
MS Access	52.20



data, schema separate	self-describing
regular	irregular

XML is Semistructured Data

XML Queries

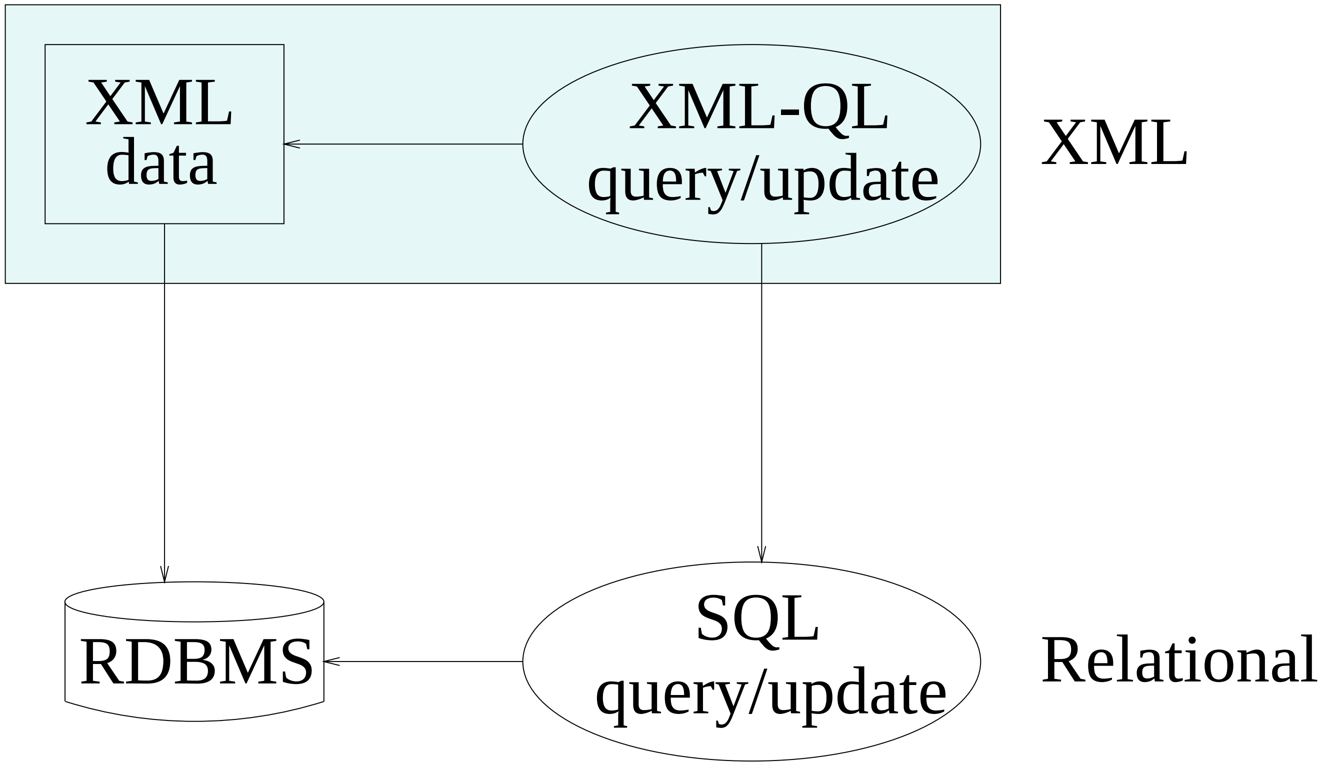
Patterns (XML-QL), path-expressions and qualifiers (XSL)

Retrieve all books written by Abiteboul after 1996 (XML-QL):

```
WHERE  <book>  <title> $T </title>
               <author> Abiteboul </author>
               <year> $Y </year>
       </book>,
       $Y > 1996
CONSTRUCT $T
```

The XML Storage Problem

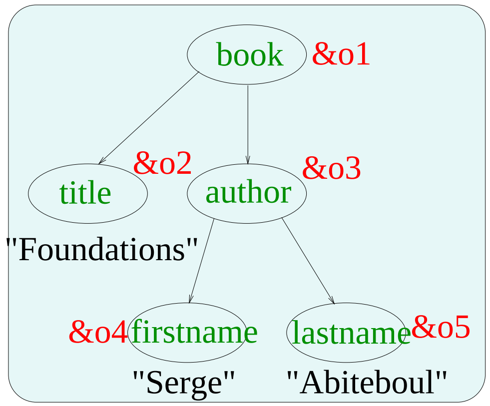
[Deutsch, Fernandez, Suciu'99]



Optimize: storage cost, execution cost for a workload

Ternary Relation

[Florescu and Kossman'99]



Edge

node1	element	node2
&root	book	&o1
&o1	title	&o2
&o1	author	&o3
&o3	firstname	&o4
&o3	lastname	&o5

Value

node	value
&o2	Foundations
&o4	Serge
&o5	Abiteboul

Simple and general; but need many joins to reconstruct objects

Using DTDs

[Shanmugasundaram et al., 99]

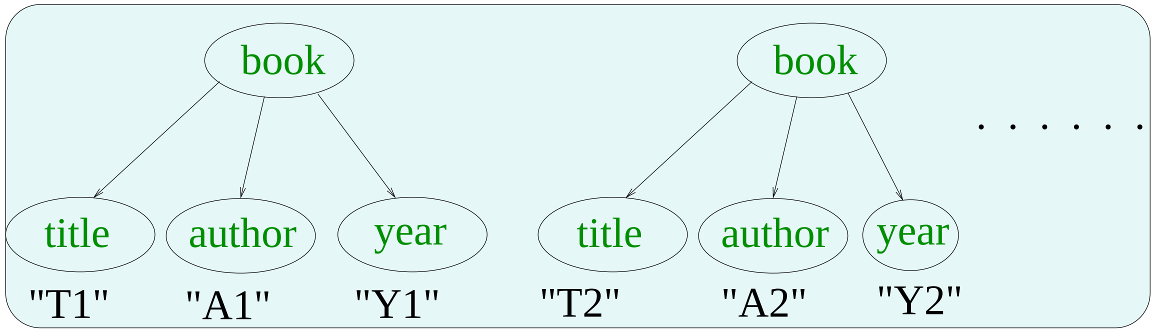
- derive schema from the DTD
- fewer joins to reconstruct data
- but: DTD's may be missing, may evolve
- also: DTD's often don't say what users want [Sahuguet'00]
  $(a \mid b \mid c)*$ instead of "a, b, c in any order"

The STORED Approach

[Deutsch, Fernandez, Suciu'99]

- exploit regularity in XML data
- use a bag of tricks to deal with irregular data
- use data mining to discover regular structures

STORED: Regular Structure

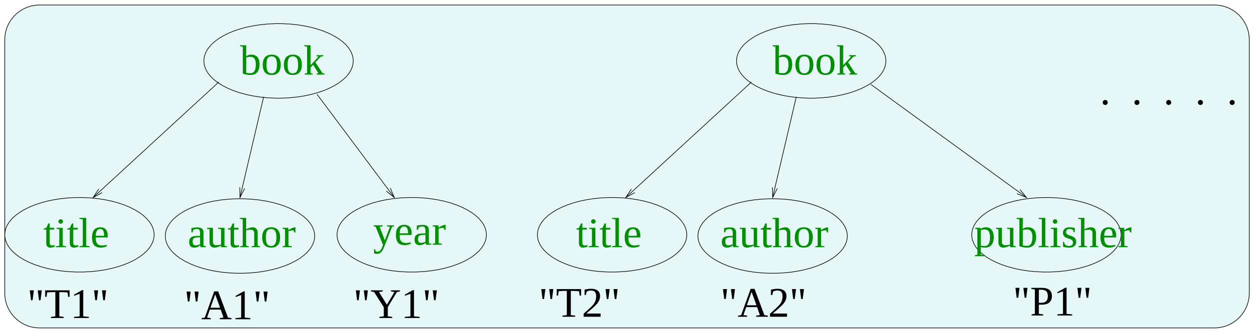


books

title	author	year
T1	A1	Y1
T2	A2	Y2
...		

For irregular data: apply bag of tricks (next).

STORED: Missing/Additional Elements

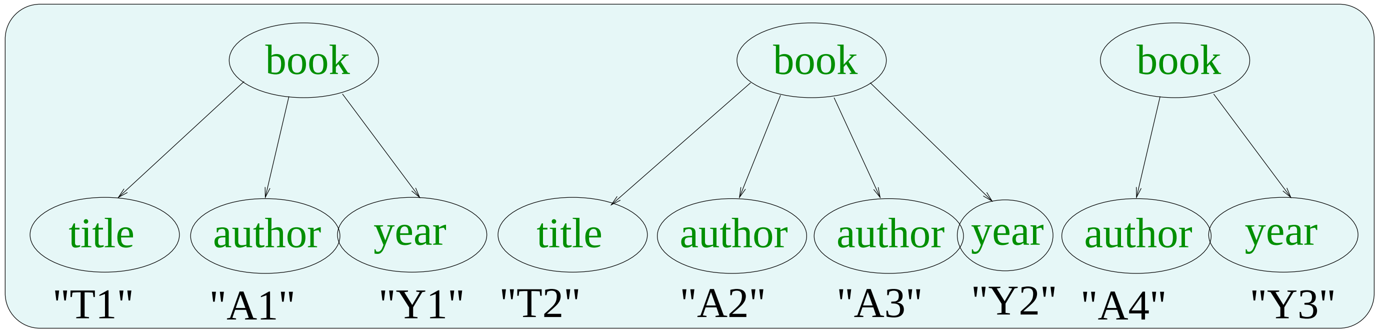


books

title	author	year	publisher
T1	A1	Y1	-
T2	A2	-	P1
...			

Use nulls

STORED: Multiple Occurrences of Same Elements

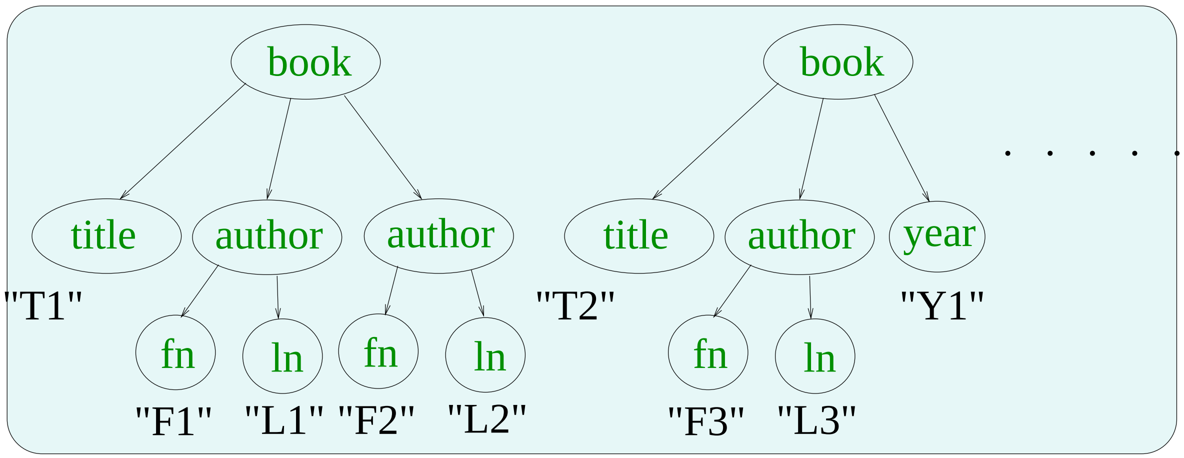


books

title	author[1]	author[2]	year
T1	A1	-	Y1
T2	A2	A3	Y2
-	A4	-	Y3
...			

Use multiple attributes

STORED: Nested Subelemenets

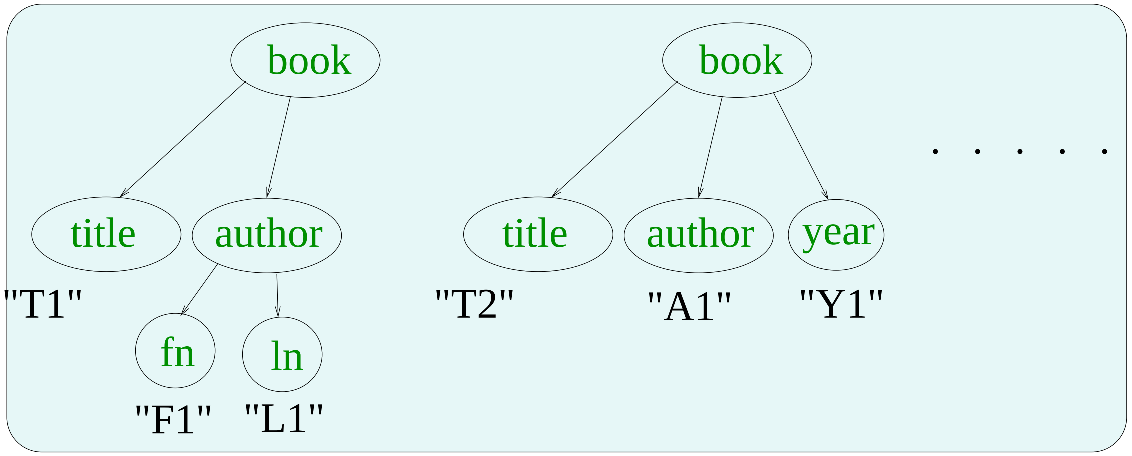


books

title	author[1]/fn	author[1]/ln	author[2]/fn	author[2]/ln	year
T1	F1	L1	F2	L2	-
T2	F3	L3	-	-	Y1

Flatten subelements

STORED: Horizontal/Vertical Partition



books1

title!	author/fn?	author/ln!
T1	F1	L1
...		

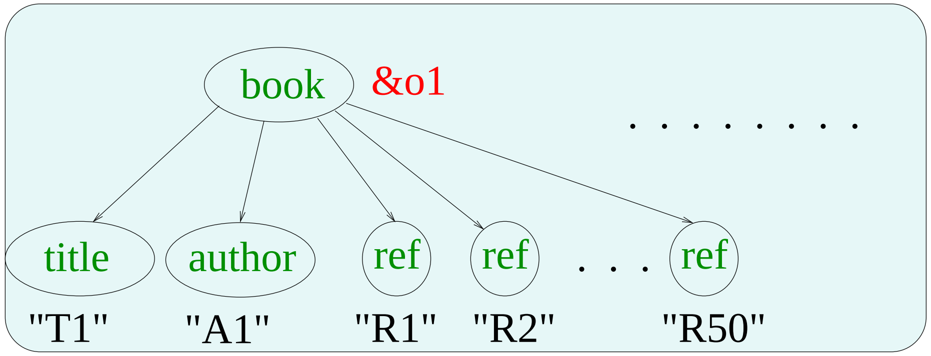
books2

title!	author?	year!
T2	A1	Y1
...		

Introduces redundacy: e.g. <book><title>T</title> </book>

Required (!) and optional (?) attributes.

STORED: Nested Collections



books

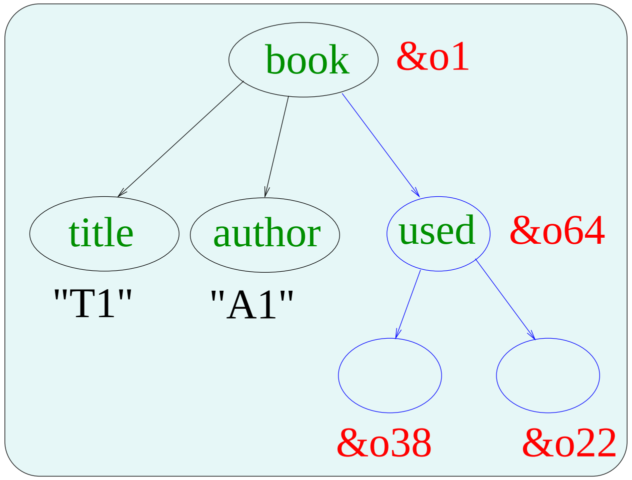
B-ID	title	author
&o1	T1	A1
...		

refs

ref	B-ID
R1	&o1
...	
R50	&o1
...	

Traditional relational encoding of nested collections

STORED: When Everything Fails



books

B-ID	title	author
&o1	T1	A1
...		

Edge

node1	tag	node2
&o1	used	&o64
...		

Overflow storage can store *any* data: updates are possible

Problem Statement

Input XML data instance D, query workload $Q_1, \ldots, Q_n$.

Output relational schema S, mapping m from D to S:

- minimize space cost of $m(D)$
- minimize processing cost of $m(Q_1) + \ldots + m(Q_n)$

Hard Constraints

- maximum number of relations K
- maximum attributes per relation A
- maxium disk space S

Problem NP Hard

verty simple setting: flat elements, no repeated attributes

<elm><a2/><a3/></elm>

<elm><a2/><a3/><a4/></elm>

<elm><a1></elm>

<elm><a1><a3></elm>

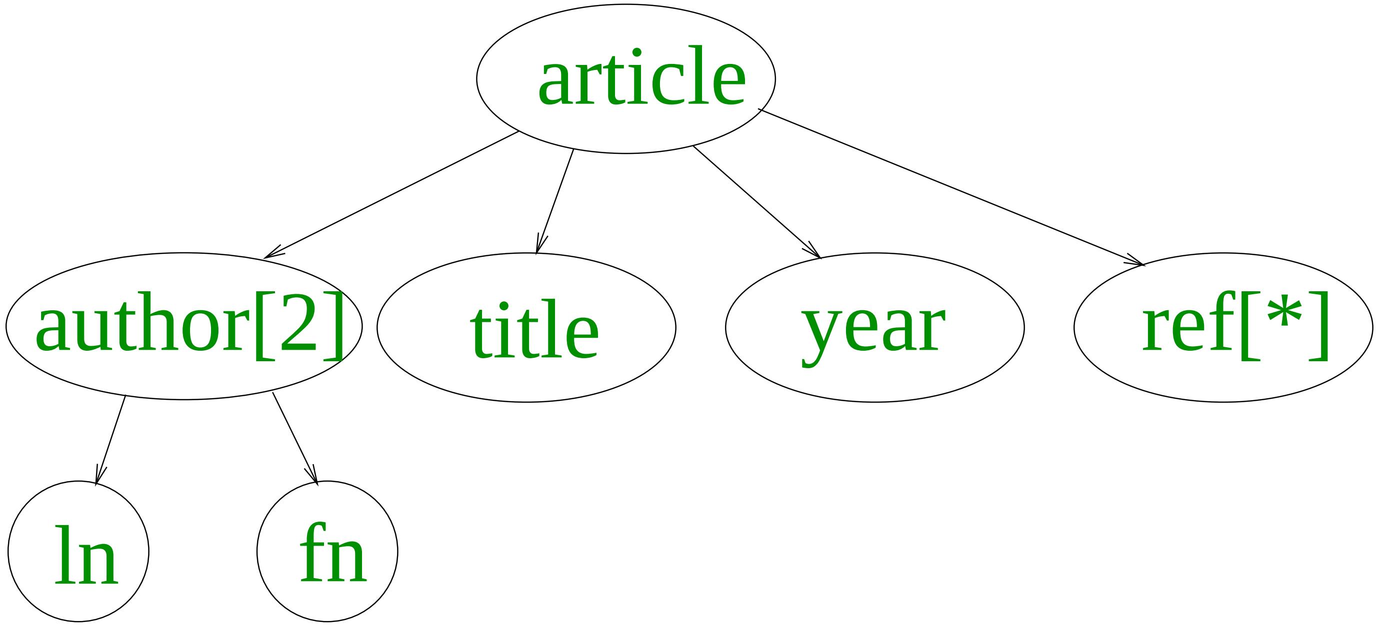
a1	a2	a3	a4	...	an
0	1	1	0	...	0
0	1	1	1	...	0
1	0	0	0	...	0
1	0	1	0	...	0
...					

cover with $\leq K$ rectagles of size $\leq S \Longrightarrow$ NP-hard in size of data

Use datamining and heuristics instead

Data Mining Approach

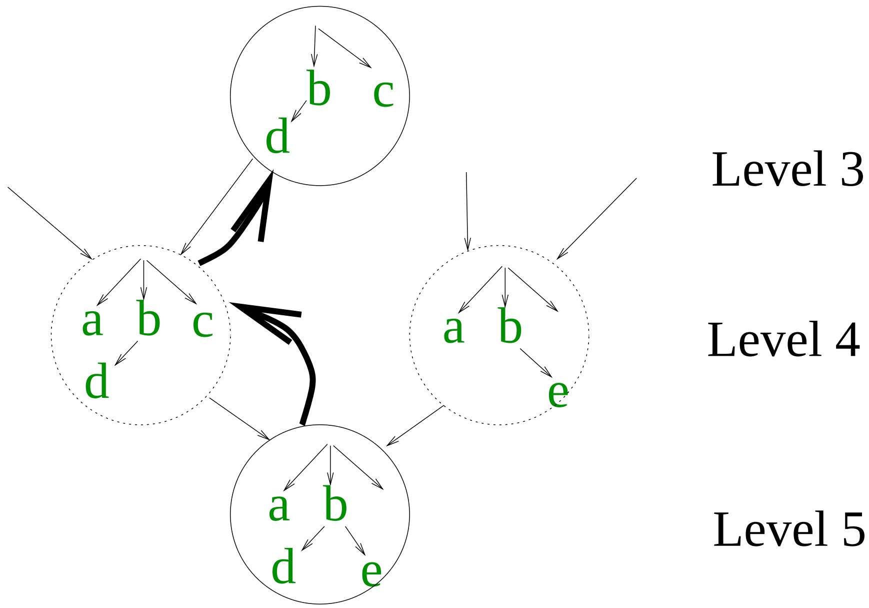
- extends data mining for semistructured data [Wang and Liu'99]
- tree patterns



- "item" = path; "item set" = pattern
- support = data support + query support

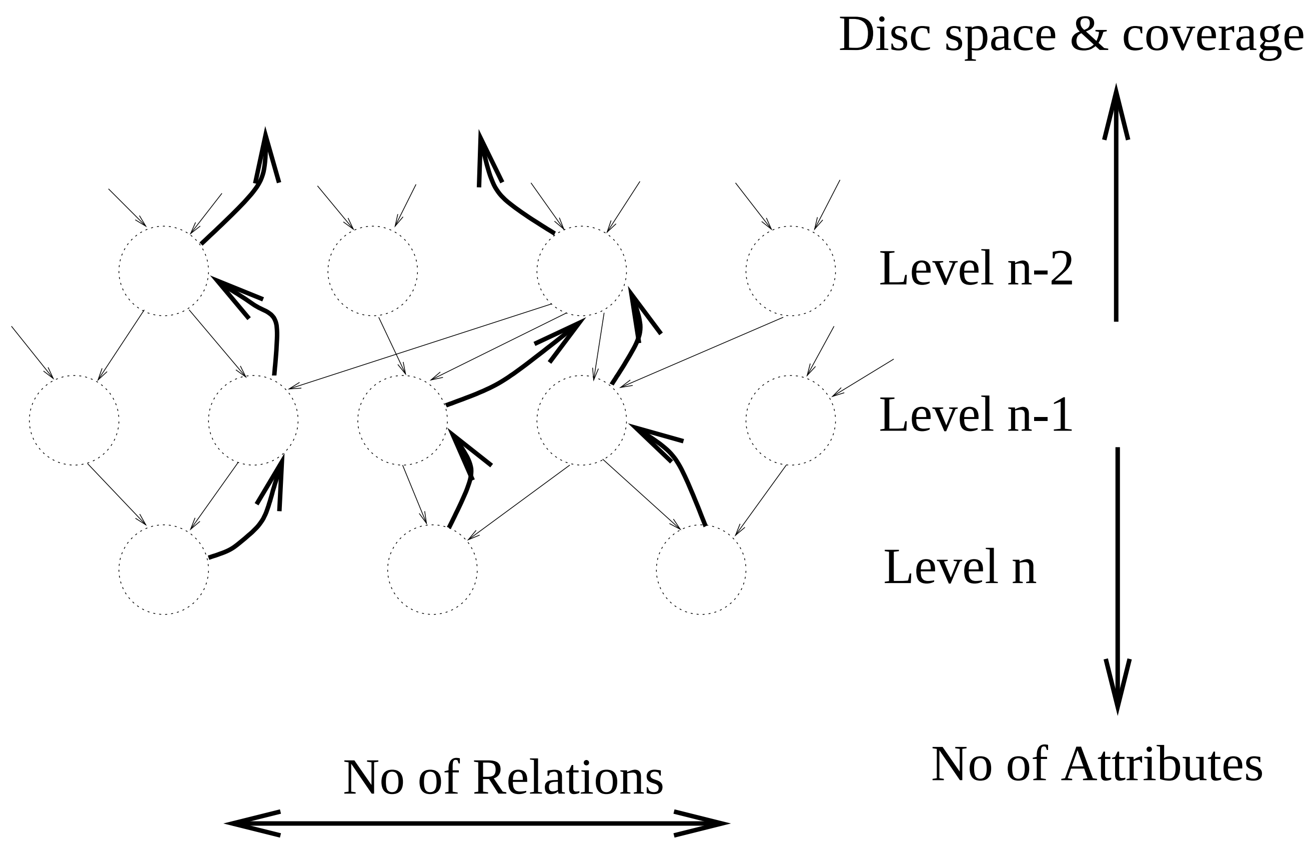
Deciding on Required v.s. Optional Attributes

- guide data mining steps by *data support* + *query support*
- keep backpointers to parent with high *data support*
- pattern ↦ all attrs, certain ancestor ↦ required attrs



a?	b/d!	b/e?	c!
...			

Choosing the Patterns



heuristics to choose K patterns ($\mapsto$ relations) that are spread apart

heuristics to choose subpatterns ($\mapsto$ required attrs)

Experiments: on DBLP

XML Data:

no. of publications	=	92,000
no. of edges	=	861,000
disc space	=	95M

Parameters:

Min Support	=	8.6%
A	=	8

Results: edge coverage ≈ 90%

Article1

author!	title?	pages?	year?	volume?	journal?	number?	url?

Article2

author[1]!	author[2]!	author[3]?	pages?	year?	volume?	number?

Future Work on Storage

- build query translation: *SilkRoute*, [Fernandez, Suciu, Tan'00]
- guide data mining steps by the storage cost function
- need evaluation of different storage methods

Part 2

XML Compression

- XML storage
- XML compression problem statement only

XML Compression: The Problem

- XML for exchange (in space or in time)
- but XML is verbose
- users prefer application-specific formats; examples:
  - Web Server Logs
  - EMBL
  - G2
- is XML doomed to fail ?

An Example: Web Server Logs

ASCII file 15.9MB (gzipped 1.6MB):

```
202.239.238.16|GET / HTTP/1.0|text/html|200|1997/10/01-00:00:02|-|4478|-|-|http://www.net.jp/|Mozill /3.1[ja](I)
```

XML-ized inflates to 24.2MB (gzipped 2.1MB):

```
<apache:entry>
  <apache:host> 202.239.238.16 </apache:host>
  <apache:requestLine> GET / HTTP/1.0 </apache:requestLine>
  <apache:contentType> text/html </apache:contentType>
  <apache:statusCode> 200</apache:statusCode>
  <apache:date> 1997/10/01-00:00:02</apache:date>
  <apache:byteCount> 4478</apache:byteCount>
  <apache:referer> http://www.net.jp/ </apache:referer>
  <apache:userAgent> Mozilla/3.1[ja](I)</apache:userAgent>
</apache:entry>
```

XMill

- specialized compressor for XML data [Liefke, Suciu'2000]
- XMill makes XML look "small" compared to other formats
- released end of 1999, 580 downloads

How XMill Works: Three Ideas

1. gzip Structure

```
<apache:entry>
  <apache:host> </apache:host>
  <apache:requestLine> </apache:requestLine>
  <apache:contentType> </apache:contentType>
  ...
</apache:entry>
```

+ gzip Data

```
202.239.238.16
GET / HTTP/1.0
text/html
200
...
```

=1.75MB

2. gzip Structure

```
<apache:entry>
...
</apache:entry>
```

+ gzip Data1

```
202.239.238.16
224.42.421.42
...
```

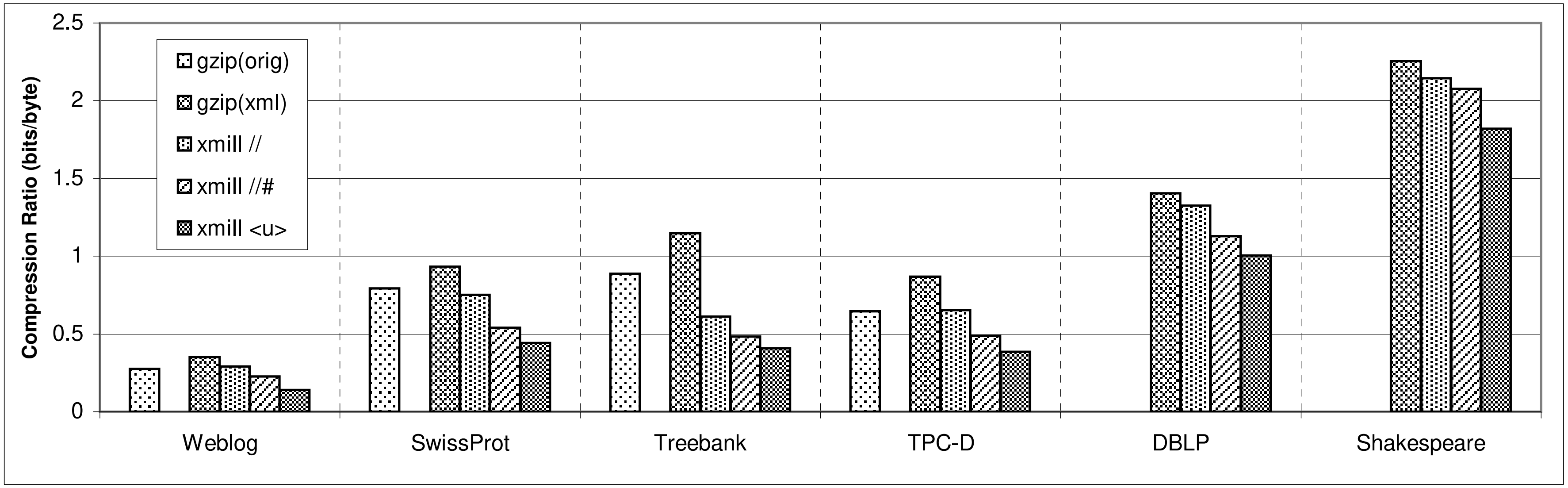
+ gzip Data2

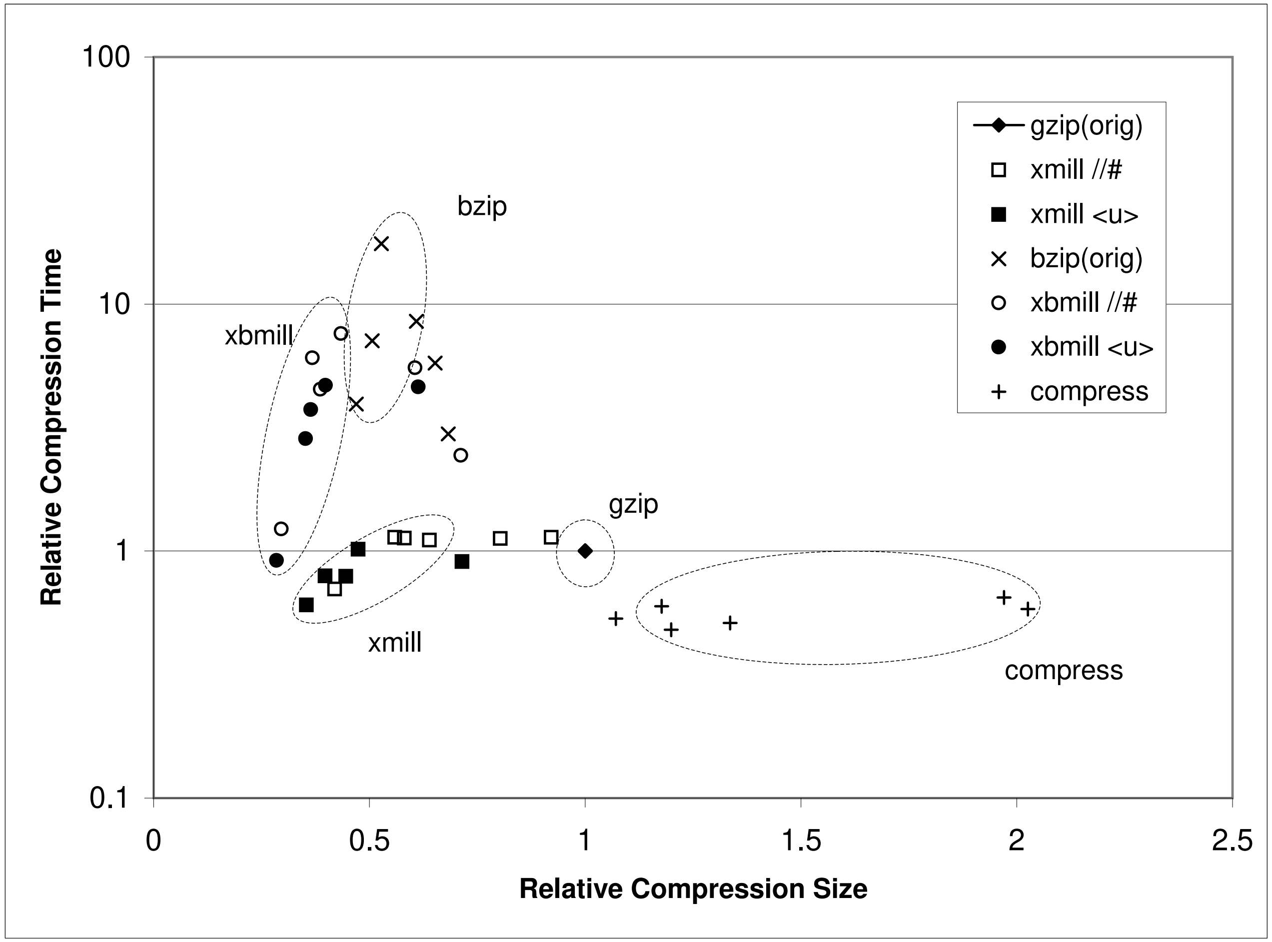
```
GET / HTTP/1.0
GET / HTTP/1.1
...
```

+ ... =1.33MB

3. gzip Structure + gzip c1(Data1) + gzip c2(Data2) + ... = 0.82MB

Experiments with XMill




100
10
1
0.1
0
0.5
1
1.5
2
2.5
Relative Compression Time
Relative Compression Size
gzip(orig)
xmill //#
xmill <u>
bzip(orig)
xbmill //#
xbmill <u>
compress
bzip
xbmill
gzip
xmill
compress

User Intervention

```
-p//apache:host=>seqcomb(u8 "." u8 "." u8 "." u8)
-p//apache:userAgent=>seq(e "/" e)
-p//apache:byteCount=>u
-p//apache:statusCode=>e
-p//apache:contentType=>e
-p//apache:requestLine=>seq("GET " rep("/" e) " HTTP/1." e)
-p//apache:date=>seq(u "/" u8 "/" u8 "-" u8 ":" di ":" di)
-p//apache:referer=>or(seq("file:" t)
                       seq("http://" or(seq(rep("." e) "/"
                                                 rep("/" e))
                                            rep("." e)))
                       t)
```

Problem

- derive regular expression automatically from data
- different from grammar induction [Charniak'93], because we only care about "high support" productions
- infer additional facts about the datatypes (e.g. integers, enumeration)

Conclusion

- have shown *applications* of data mining
- semistructured data/ XML is irregular: exploit regularities to improve processing/compression
- cost function may guide mining process

`www.research.att.com/~suciu`