XML Tree Structure Compression

Sebastian Maneth

NICTA & University of New South Wales

Joint work with N. Mihaylov and S. Sakr

Bolzano, Sept. 5th, 2008

NATIONAL ICT AUSTRALIA

Outline

1. Motivation
2. XMill's compression of XML tree structure
3. DAGs for XML tree structure compression
4. Sharing Graphs (= BPLEX output) for XML tree structure compression

Motivation

→ large part of an XML document consists of markup in the form of begin and end-element tags, describing the tree structure of the document

→ most XML file compressors **separate** the tree structure from the rest of the document (data values) and **compress them separate**

(for data values, classical compression methods can be used)

In this work:

→ want to find effective file compressed method for the tree structure of an XML document

XMill

Well-known XML file compressor: XMill by Liefke and Suciu [SIGMOD2000]

Idea → separate data values from tree structure
→ group similar data items together into containers
(similarity is based on tree structure path to the item)
→ compress all containers using conventional compression backends, such as Gzip, Bzip2, or PPM

How is the tree structure compressed?

Use (byte-aligned) symbols per each begin-element tag, and one fixed symbol for every end-element tag.
Compress resulting string using Gzip / Bzip2 / PPM / or ...

XMill

How is the tree structure compressed?

Example

<book>
<chapter></chapter>
<chapter><section/><section/><section/></chapter>
<chapter><section/><section/></chapter>
</book>

End element tag: /

Becomes

0 1 / 1 2 / 2 / 2 / / 1 2 / 2 / / /

Plus the symbol table ["book", "chapter", "section"]

0 1 2

Will be compressed using Gzip, Bzip2 or PPM

XMill

How is the tree structure compressed?

Example

<book>
<chapter></chapter>
<chapter><section/><section/><section/></chapter>
<chapter><section/><section/></chapter>
</book>

End element tag: /

Becomes

0 1 / 1 2 / 2 / 2 / / 1 2 / 2 / / /

Plus the symbol table ["book", "chapter", "section"]

0 1 2

Will be compressed using Gzip, Bzip2 or PPM

> We will use
> **XMill + Gzip**
> as *baseline compression value* in all our experiments!!

Our Approach

Use **in-memory (pointer-based) tree compression**,
& write *suitable binary encoding* to disk (possibly plus Gzip/Bzip2/PPM backends)

Pointer-based tree compressions considered:

1) **DAGs** (Directed-Acyclic Graphs)

→ obtained by sharing *common subtrees* of the XML tree structure
use standard algorithm based on hashing distinct subtrees

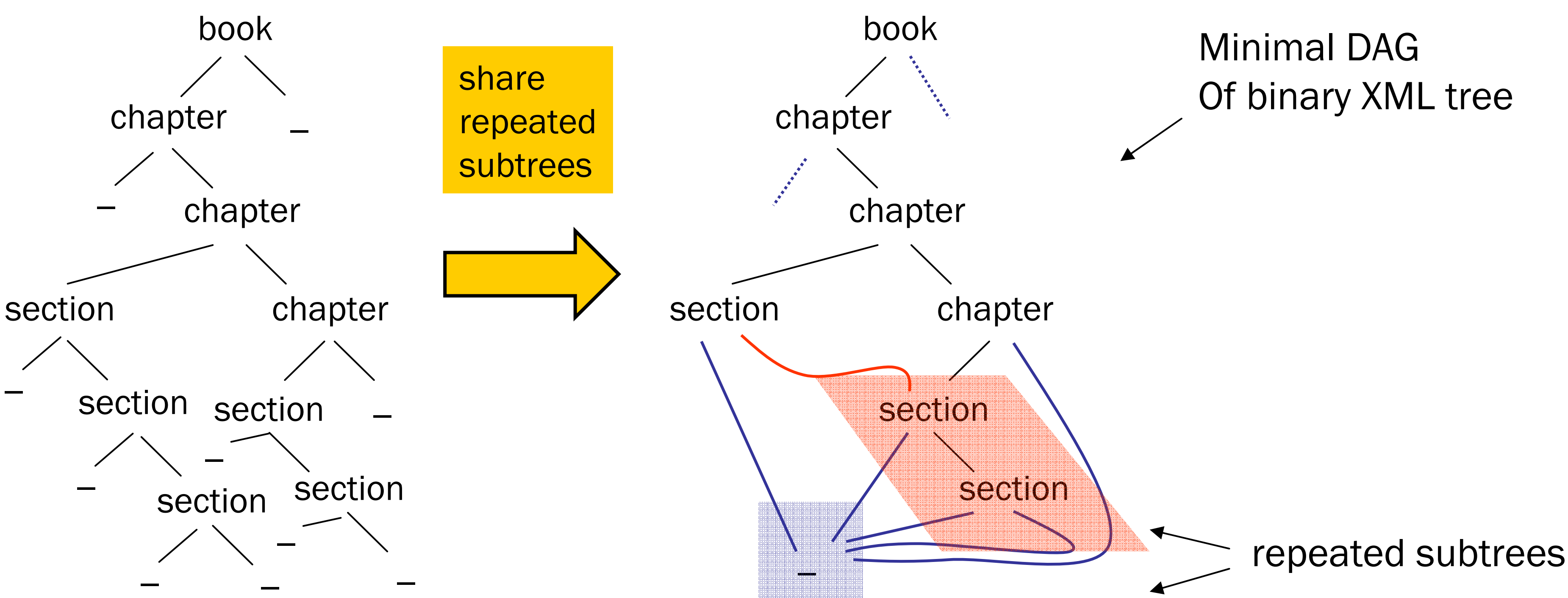
2) **Sharing graphs** [Lamping, POPL 1990]

→ obtained by sharing *common connected subgraphs* in XML tree
use BPLEX algorithm [Busatto, Lohrey, Maneth, Inf. Syst. 2008]

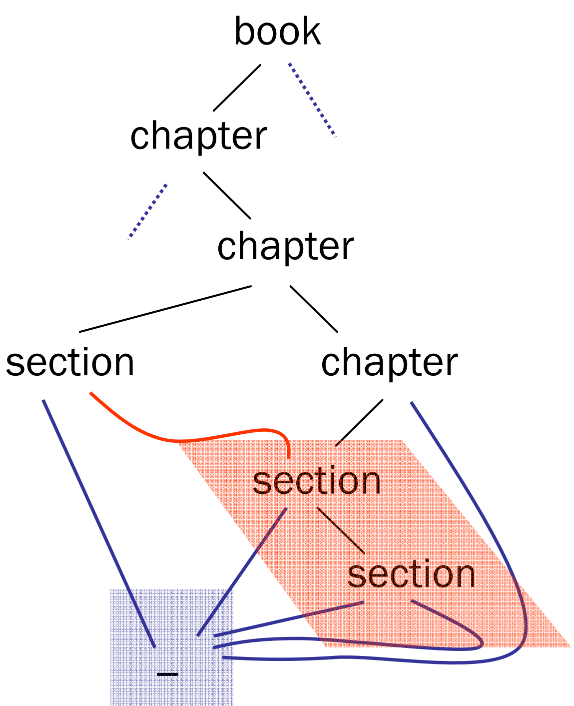
1. Minimal DAG

1) **DAGs** (Directed-Acyclic Graphs)
→ obtained by sharing *common subtrees* of the XML tree structure

Example --- working on *binary XML tree* (first-child/next-sibling encoding)



1. Minimal DAG



(0 : ,
1 : section[0, 0],
2 : section[0, 1],
3 :book[chapter[0, chapter[section[0, 2], chapter[2, 0]]], 0]).

Sequential representation of minimal DAG

- Use own ***suitable binary endocing*** → DAG
- Gzip → DAGGzip
- Bzip2 → DAGBzip2
- ppm → DAGPPM

Final compressed codewords: DAG, DAGGzip, DAGBzip2, DAGPPM

1. Minimal DAG

→ Test `DAG,DAGGzip,DAGBzip2,DAGPPM` on diverse XML dataset:

including
* files used by Liefke/Suciu for XMill
* several Wikipedia XML files
* files from EXI W3C working group

Etc.

Documents, used in Experiments

Document	Size (KB)	Tags	# Nodes	Depth
1998statistics.xml	717	47	54,581	7
Catalog-01.xml	6,624	51	372,459	9
Catalog-02.xml	65,875	51	3,705,071	9
Dictionary-01.xml	3,481	25	513,574	9
Dictionary-02.xml	34,311	25	5,077,549	9
EnWikiNew.xml	7,834	21	665,825	6
EnWikiQuote.xml	5,034	21	437,682	6
EnWikiSource.xml	21,849	21	1,902,189	6
EnWikiVersity.xml	9,530	21	828,229	6
EnWikTionary.xml	160,373	21	14,520,656	6
EXI-Array.xml	7,156	48	226,524	10
EXI-Factbook.xml	2,087	200	86,581	6
EXI-Invoice.xml	457	53	26,130	8
EXI-Telecomp.xml	5,402	39	177,634	7
EXI-Weblog.xml	2,216	13	178,375	4
JST_gene.xml	7,932	27	388,029	8
JST_snp.xml	24,667	43	1,169,686	9
Lineitem.xml	30,270	19	1,985,776	4
Medline.xml	80,248	79	5,394,921	8
Mondial.xml	409	23	22,423	5
Nasa.xml	9,958	62	792,467	9
NCBI_gene.xml	13,042	51	645,917	8
NCBI_snp.xml	135,853	16	6,879,757	5
Sprot.xml	206,993	49	21,634,330	7
Treebank.xml	31,450	252	3,843,775	38
USHouse.xml	144	44	11,889	17

→ Size (KB) means
XML tree structure only.

Files with data values are,
e.g., 457MB (Sprot.xml) or
190MB (NCBI_snp.xml)

Note

→For every text and attribute node we have special dummy nodes in our tree structure.

1. Minimal DAG

→ Test DAG,DAGGzip,DAGBzip2,DAGPPM on diverse XML dataset:

including
* files used by Liefke/Suciu for XMill
* several Wikipedia XML files
* files from EXI W3C working group

Etc.

Most important observation

→ Minimal DAG does **not** give rise to best compression

→ Only share subtrees of a certain size (more than TRESH-many nodes)

NATIONAL ICT AUSTRALIA

1. Minimal DAG

→ Test DAG,DAGGzip,DAGBzip2,DAGPPM on diverse XML dataset:

including
* files used by Liefke/Suciu for XMill
* several Wikipedia XML files
* files from EXI W3C working group

Etc.

Optimal TRESH-values for our datasets:

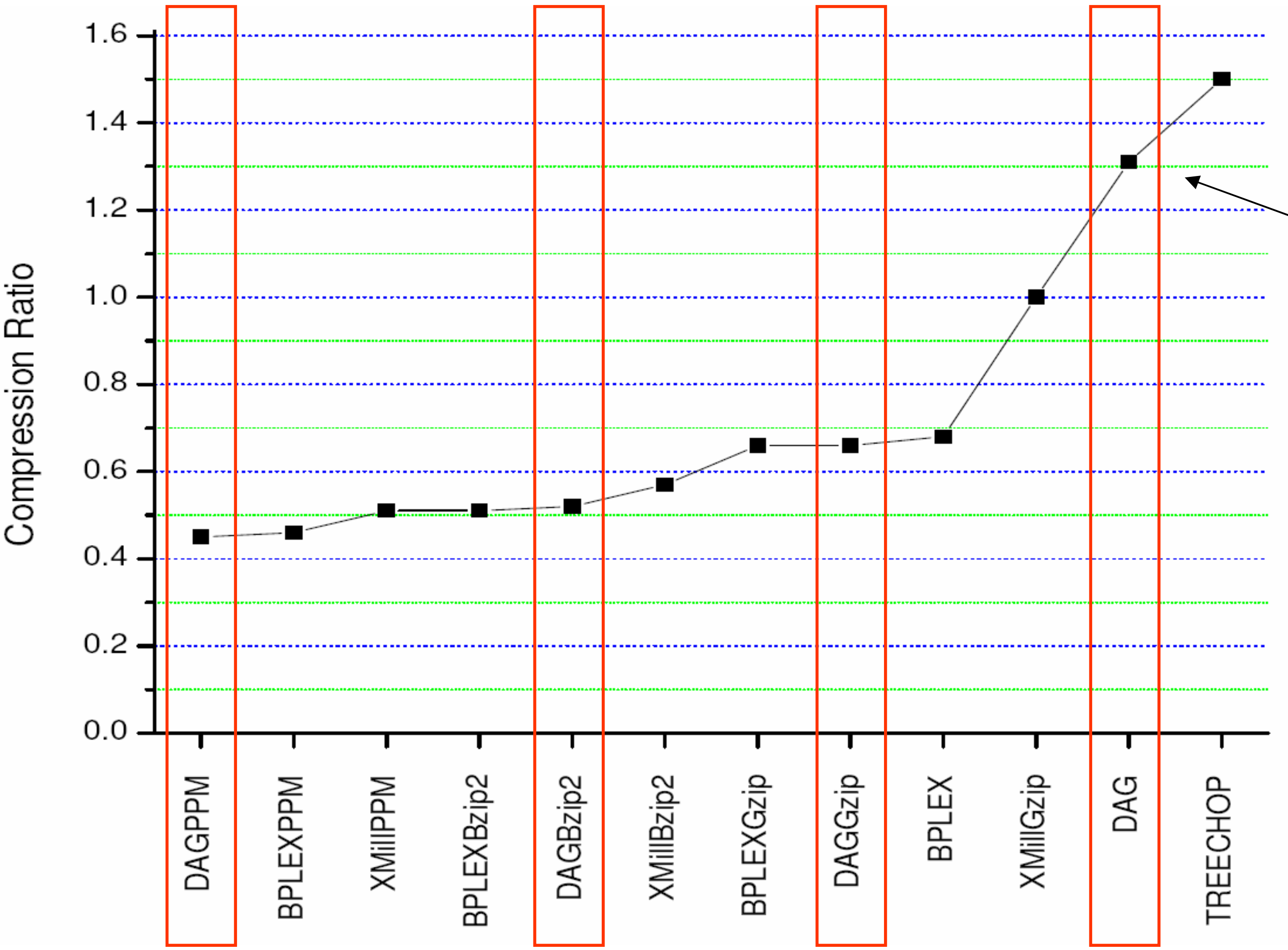
TRESH=14 for DAG
TRESH=1000 for DAGGzip
TRESH=3000 for DAGBzip2 and DAGPPM

Most important observation

→ Minimal DAG does **not** give rise to best compression

→ Only share subtrees of a certain size (more than TRESH-many nodes)

1. DAGs, Results



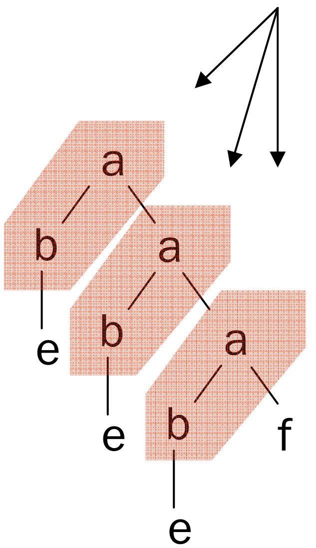
With no backend compression, DAG gives worse compression than XMill with Gzip (=baseline = 1.0)

2. Sharing Graphs (SLT grammars)

Idea, share repeated (connected) subgraphs in binary XML tree.
Represent them as trees with parameters.

Example

repeated subgraph



share repeated subtgraphs

using BPLEX algorithm

```
(1: a[b[y1],y2],
 2: 1[c,1[d,1[e,f]]])
```

Sharing graph
(in tree-grammar notation)

2. Sharing Graphs (SLT grammars)

Known, for usual XML documents:

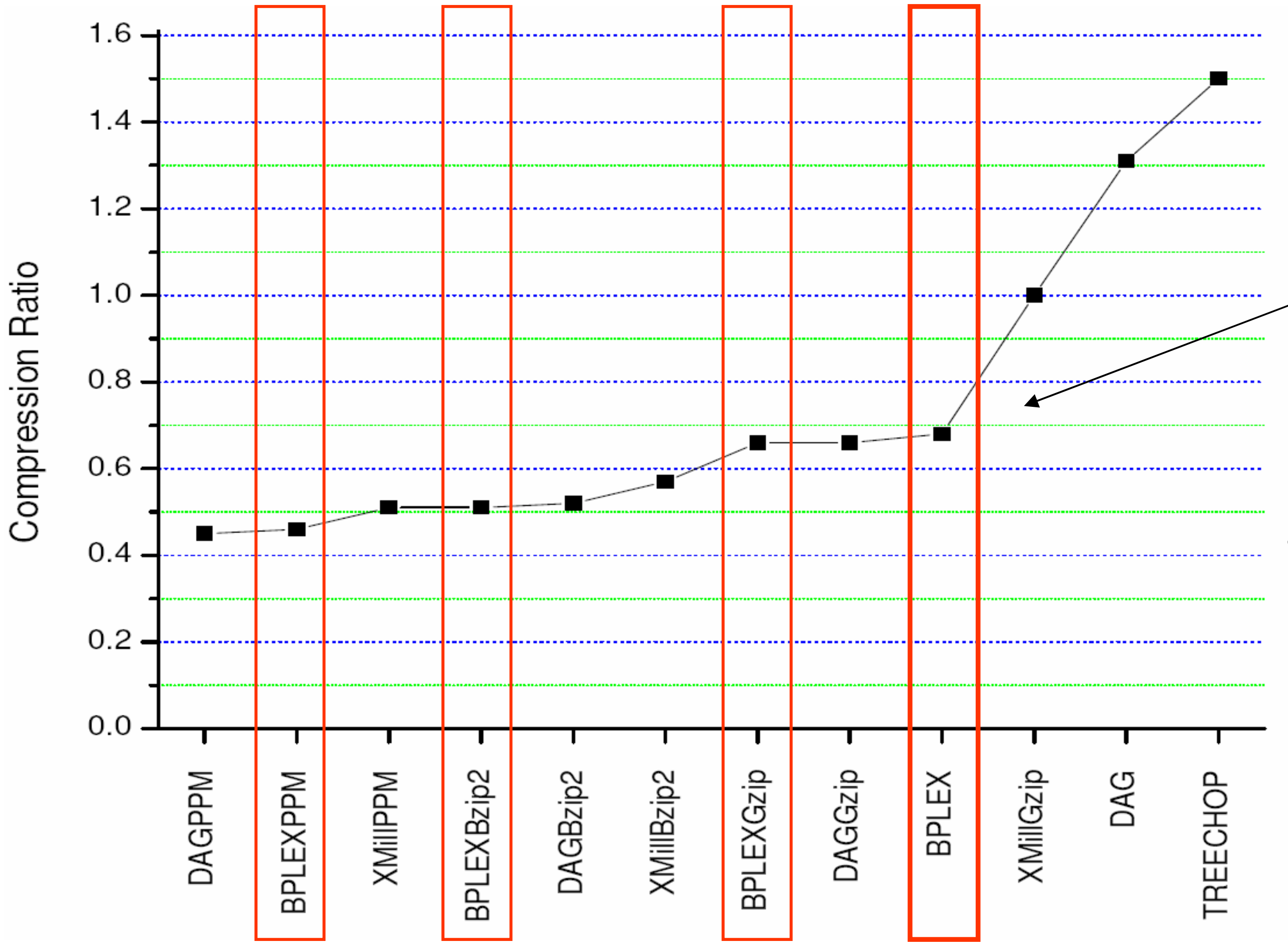
BPLEX algorithm produces *pointer-structures* (sharing graphs) with
Approx. *2-3 times less pointers* than the DAG.

Consider `BPLEX,BPLEXGzip,BPLEXBzip2,BPLEXPPM`

→ again, do **not** use "minimal sharing graphs", but introduce a TRESH value, similar as for DAGs

Then, optimal performance on our datasets by using
TRESH=14 for `BPLEX`
TRESH=14 for `BPLEXGzip`
TRESH=10,000 for `BPLEXBzip2`
TRESH=30,000 for `BPLEXPPM`

2. SLT grammars, Results



With no backend compression, BPLEX gives **better** compression than XMill with Gzip!! (=baseline = 1.0)

On average, size is 68% of XMillGzip.

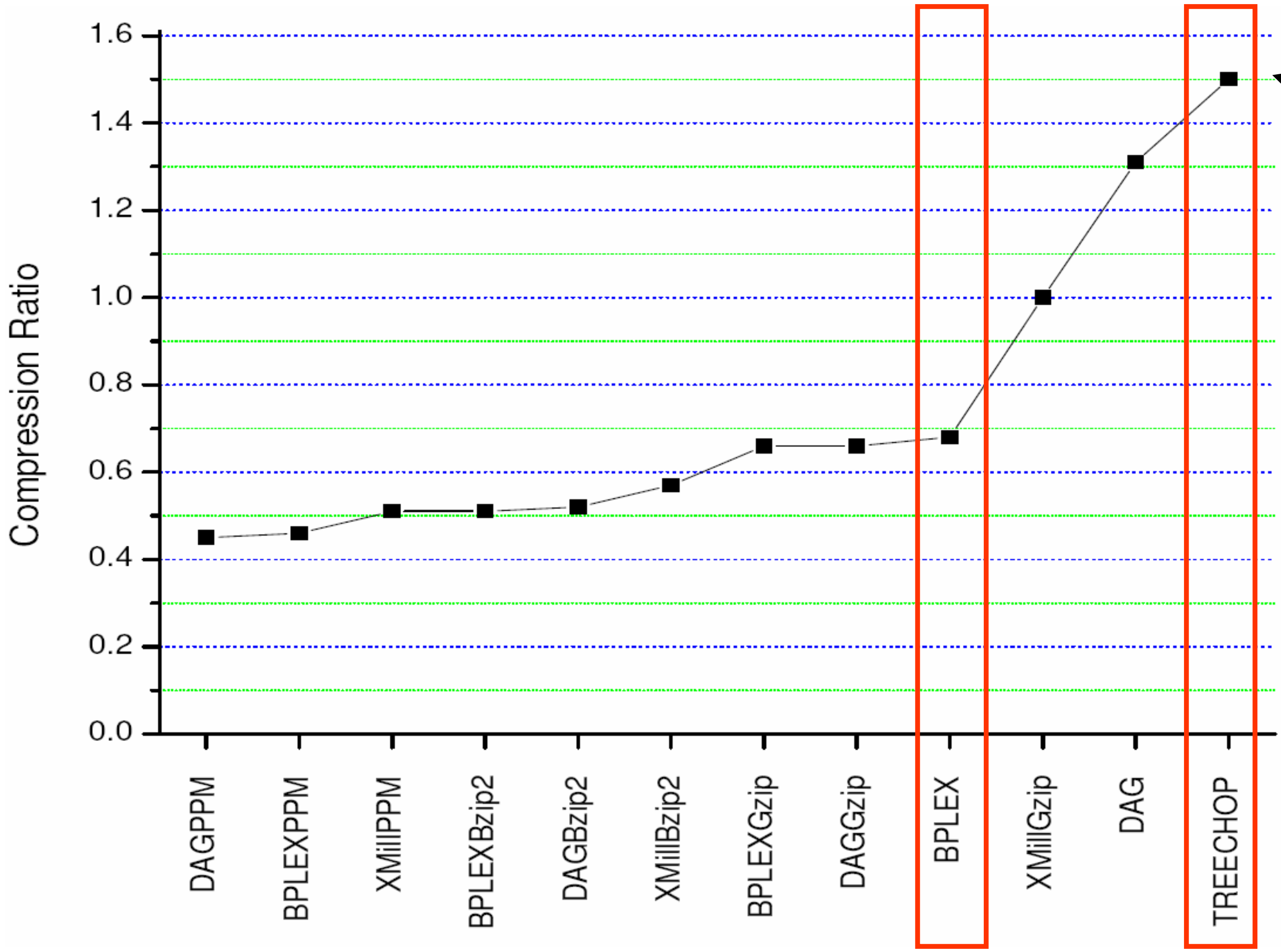
2. SLT grammars, Results

Note, the "*suitable binary encoding*" in BPLEX to obtain 68% of XMillGzip, is a Huffman-coding of a natural representation of the pattern trees.

This encoding can be use with little overhead, to execute queries (such as XPath or XQuery, or any DOM program) directly on the compressed structure!!

It gives rise to a **VERY SMALL queryable representation**, smaller than any other queryable representation known from the literature.

2. SLT grammars, Results



Next best, queryable format.

Approx. 2x bigger than BPLEX

Other queryable formats require much much more space still!!

Conclusions

For file compression of XML tree structures, DAGs are suitable.

→ they can be obtained quickly, using hashing
→ using Gzip-backend, they are only 70% of the size of XMillGzip

---

For in-memory compression, e.g., as a queryable data structure, BPLEX-outputs are extremely well suitable

→ they can be queried with little overhead, for Core XPath queries
even with speedup wrt running over uncompressed tree [Lohrey,Maneth2007]

→ using **no** backend, they are only 68% of the size of XMillGzip

NATIONAL ICT AUSTRALIA

Thank you!

.. for your attention,
and special thanks to xxx.yyy for giving this presentation at XANTEC'2008.

For questions, please send email to

sebastian.maneth@nicta.com.au

--------- THE END ----------