XML “Bloat”

-Verbose data representation.
  - interoperability
  - human-readability
- XML processing uses much more memory.
  - benefits?
- “Premature optimisation is the root of all evil”
  - SOAP packets ~ 1MB
DOM

Standard DOM implementations: pointer-based.

DOM (Document Object Model) – basic in-memory representation.
In-memory representation = 300% to 800% of file size
Navigate tree; retrieve information associated with node.

For static documents, is there a “pointerless” representation, with full DOM functionality?  Succinct DOM
Succinct DOM

- Supports most static methods of DOM Core L3
  - document changes not supported.
- Supports $x R y$, $R \in \{\text{ancestor, descendant, preceding, following, preceding/following-sibling}\}$
- Good worst-case behaviour:
  - navigation, $x R y$ etc. in $O(1)$ time.
  - good upper bounds on space usage.
- “Very fast” with “reasonable compression”.
- Leicester Research Archive (Google title)
  - SDOM library (linux/i686), test harness, Valgrind
  - 3.5x faster to 7x slower than Xerces-C
  - (i) significantly smaller than original XML file or (ii) comparable to “query-friendly” XML compressors
Previous Work

- Many XML compressors: (XMill, Xgrind, XPress, XCQ, Xbzip...)
  - Query-friendly (Xgrind, Xpress, XCQ, Xbzip ...) focus on XPath queries, not navigation
  - Navigation typically slow – order of milliseconds.
  - SDOM/(-CT)
    - (at worst) few times slower than memory access.
    - no direct support for Xpath queries.

- DOM implementations
  - SEDOM – basic operations slow (ms), CR between SDOM and SDOM-CT (allows updates)
  - DDOM* – space better than Xerces (allows updates)
  - TinyTree – space much better than Xerces (static)
  - ISX (WWW’07):
    - similar approach to ours, seems 5x-10x slower from paper, significantly worse CR, but fast updates.

- Summary: Succinct DOM occupies new point in Space/Time tradeoff, first to surpass performance of Xerces with good compression.
Succinct DOM

• Based on Succinct Data Structures.
  ▫ “Information-theoretic” optimality.
  ▫ Succinctness ≠ compression.
• Highly optimised components [GRRR’06, DRR’07]
Tree Structure

- Space usage?
  - Naïve: $\geq 2$ pointers/node
  - Succinct: $\log\left(\frac{1}{n}\left(\frac{2n-2}{n-1}\right)\right) = 2n - O(\log n)$
Tree Structure

Xerces: 5 ptrs/node  
(160 bits/node)  
Succinct: 2 bits/node

- Parenthesis sequence:
  - findclose/open, enclose
    - enclose(10) = 1
  - $O(1)$ time
    - $2.86n$ bits [GRRR ‘06]
Tree Nodes

- Numbered in document order
  - Text nodes
    - numbered consecutively 1..t in “document order”
  - Non-text tree nodes
    - numbered consecutively 1..e in “document order”
    - index into array of “short codes”, where each entry is log(p + 12) bits.

Bit-vector data structure with Rank operation

Number of distinct element names

Specify other kinds of nodes e.g. processing-instruction.
Text Nodes

- “String value” of text node

```xml
<book catalogue="java">
  <author>Selman</author>
  <title>Java3d programming</title>
  <year>2000</year>
</book>
```
Text Nodes

- Text nodes numbered 1..t

Text data “array”

Selman r p Java 3D ... m i n g r p 2 0 0

Naïve: 4-8 bytes per text node.
  - Average text node: 10-12 characters – uncompressed.
- Offset array compression [DRR ‘07]: ~ 1 byte/node.
Text data “array”

(All text nodes concatenated in document order; offsets given by offset “array”)

- Array of uncompressed characters
  - text node value is null-terminated: can return pointer into array.
  - Pure succinct representation.

- Bzip-compress 16KB blocks, decompress on demand.
  - cache 4 uncompressed blocks.
- Store array in FM-index or Compressed Suffix Array.
  - seems slower than (1)
- Use (1) if locality in text access; (2) otherwise.
- Other alternatives e.g. hashing.
Node class

- Most important class in DOM for navigation.
  - represents a node in the DOM tree.
  - SDOM Node: 2 ints + ptr to Document object.
- SDOM initially has no Node objects
  - Created upon traversal; no check to see if a tree node has an existing Node object:
    - `y = x.getFirstChild();`
    - `z = x.getFirstChild(); /* z != y */`
- No garbage collection...
- Better use TreeWalker class
  - NextNode/PreviousNode methods are very fast.
Experimental evaluation
## Files used (UW repository)

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Nodes</th>
<th>Max. Depth</th>
<th>Max node degree</th>
<th>#text chars</th>
<th>#attr Chars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orders</td>
<td>5MB</td>
<td>300,004</td>
<td>3</td>
<td>30001</td>
<td>1MB</td>
<td>NEG</td>
</tr>
<tr>
<td>Lineitem</td>
<td>32MB</td>
<td>2,045,954</td>
<td>3</td>
<td>120351</td>
<td>6MB</td>
<td>0</td>
</tr>
<tr>
<td>XPATH</td>
<td>50MB</td>
<td>2,522,571</td>
<td>5</td>
<td>42075</td>
<td>13MB</td>
<td>NEG</td>
</tr>
<tr>
<td>Treebank_e</td>
<td>82MB</td>
<td>7,312,613</td>
<td>36</td>
<td>112769</td>
<td>57MB</td>
<td>NEG</td>
</tr>
<tr>
<td>SwissProt</td>
<td>110MB</td>
<td>10,599,084</td>
<td>5</td>
<td>100000</td>
<td>35MB</td>
<td>13MB</td>
</tr>
<tr>
<td>DBLP</td>
<td>128MB</td>
<td>10,595,379</td>
<td>6</td>
<td>657717</td>
<td>64MB</td>
<td>7MB</td>
</tr>
<tr>
<td>XCDNA</td>
<td>594MB</td>
<td>25,221,153</td>
<td>7</td>
<td>82237</td>
<td>256MB</td>
<td>0</td>
</tr>
</tbody>
</table>

Also Xmark files: not shown.  
Machine: Pentium 2GB RAM. Ubuntu. g++ 3.3.
## SDOM space usage

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>SDOM</th>
<th>Xerces</th>
<th>Saxon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orders</td>
<td>5MB</td>
<td>37%</td>
<td>451%</td>
<td>157%</td>
</tr>
<tr>
<td>Lineitem</td>
<td>32MB</td>
<td>28%</td>
<td>399%</td>
<td>161%</td>
</tr>
<tr>
<td>XPath</td>
<td>50MB</td>
<td>33%</td>
<td>383%</td>
<td>137%</td>
</tr>
<tr>
<td>Treebank</td>
<td>82MB</td>
<td>84%</td>
<td>866%</td>
<td>266%</td>
</tr>
<tr>
<td>SwissPrt</td>
<td>110MB</td>
<td>60%</td>
<td>704%</td>
<td>272%</td>
</tr>
<tr>
<td>DBLP</td>
<td>128MB</td>
<td>68%</td>
<td>737%</td>
<td>240%</td>
</tr>
<tr>
<td>XCDNA</td>
<td>594MB</td>
<td>50%</td>
<td>491%</td>
<td>136%</td>
</tr>
</tbody>
</table>

- DOM implementations (with uncompressed text)
- Percentages relative to file sizes.
- Textual data left uncompressed!
SDOM-CT space usage

- Comparison with:
  - Query-Friendly XML compressors
    - XBZipIndex, XPRESS, XQZip, XGRIND.
  - XMill

- SDOM-CT is a “middle-of-the-road” compressor.
- about 25% of SDOM-CT’s space is purely to support fast navigation.
Speed test

- **Basic test:** several kinds of traversals
  - Document-order
  - Reverse document-order
  - Upward path enumeration
- **Full test:**
  - Document-order traversal plus:
    - traverse all attributes and values
    - search for a (uncommon) substring in all text nodes.
- **Compare traversals using** `Node` and `TreeWalker NextNode` methods.
- **All XML documents (except XCDNA) fit comfortably in main memory even for Xerces.**
Basic Test (Document/RDO)
Upward Path Enumeration

![Graph showing time (secs) for different datasets with Xerces and SDOM]
Full Test
## Text compression

<table>
<thead>
<tr>
<th>Files</th>
<th>Text</th>
<th>libBzip2-blocks</th>
<th>FMIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>path-order</td>
<td>doc-order</td>
</tr>
<tr>
<td>Orders</td>
<td>1MB</td>
<td>22%</td>
<td>30%</td>
</tr>
<tr>
<td>Lineitem</td>
<td>6MB</td>
<td>21%</td>
<td>31%</td>
</tr>
<tr>
<td>XPATH</td>
<td>13MB</td>
<td>9%</td>
<td>13%</td>
</tr>
<tr>
<td>Treebank</td>
<td>57MB</td>
<td>42%</td>
<td>45%</td>
</tr>
<tr>
<td>SwissProt</td>
<td>49MB</td>
<td>17%</td>
<td>29%</td>
</tr>
<tr>
<td>DBLP</td>
<td>71MB</td>
<td>28%</td>
<td>35%</td>
</tr>
</tbody>
</table>

Difference between path-order and doc-order is fairly small for bzip2 (cf. Xmill).
“Spike” caused by use of vector for accumulating Textual data as it is processed by SAX.

Easy fix for much “preprocessing” space usage.
Conclusion/Future Work

• Succinct DOM: very fast, reasonable compression, and robust performance.
  ▫ Incorporate into higher-level application (Xalan/Saxon/OpenOffice)?
  ▫ Optimise parsing further?
  ▫ Succinctness + secondary storage?

• Commercialisation?:
  ▫ Saxon £250 license
  ▫ Intel high-performance XSLT library $500
  ▫ IBM XML parsing hardware ~ $5k.