Tradeoffs in XML Database Compression

James Cheney

Database Seminar

March 20, 2006

XML Compression

- XML (presented as text) is generally verbose
- XML representation of most data typically larger than equivalent “custom” formats
- Often, gzip or bzip2 used to compress XML text
  - However, may miss XML-specific compression opportunities
  - Also, have to uncompress and then parse XML text to SAX events/DOM tree
- Goal: Faster, better, or cheaper XML compression

Prior work

- XMill (Liefke, Suciu 2000): first (serious) XML compression work
  - containerize/vectorize XML document, then compress with gzip/bzip2
- XMLPPM (C. 2001): uses statistical modeling, better compression but slower
- SCMPPM (Adiego, de la Fuente, Navarro DCC 2004), XAust (Hariharan, Shankar 2005): use different statistical models, reports improvement over XMLPPM
- Vectorized XML (BGK2003), Bplex (Busatto, Lohrey, Maneth 2005): in-memory compression of XML document trees; not used for compressing whole file to disk

Prior work

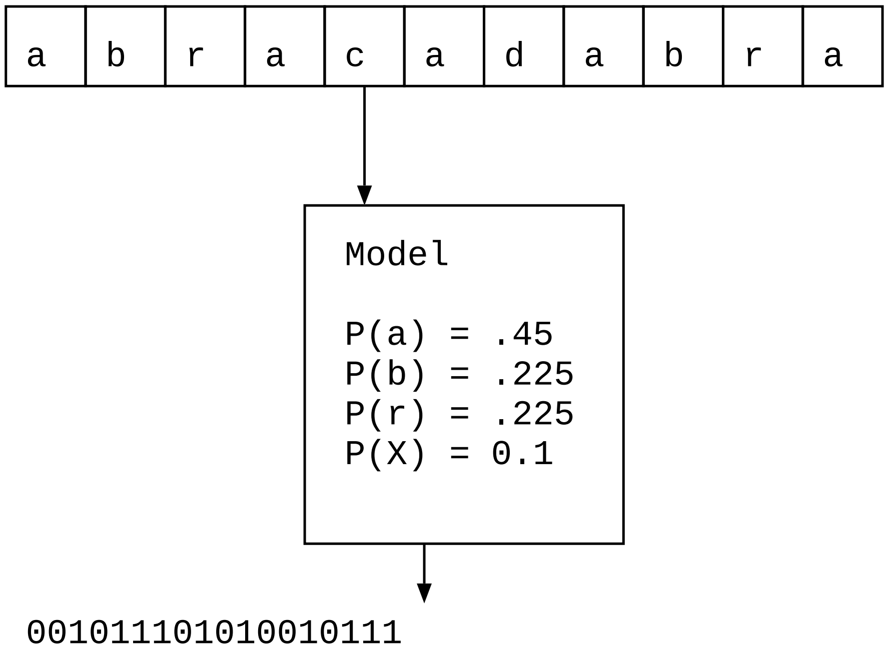
- XMill (Liefke, Suciu 2000): first (serious) XML compression work
  - containerize/vectorize XML document, then compress with gzip/bzip2
- XMLPPM (C. 2001): uses statistical modeling, better compression but slower
- SCMPPM (Adiego, de la Fuente, Navarro DCC 2004), XAust (Hariharan, Shankar 2005): use different statistical models, reports improvement over XMLPPM
- Vectorized XML (BGK2003), Bplex (Busatto, Lohrey, Maneth 2005): in-memory compression of XML document trees; not used for compressing whole file to disk

Origin of this work

- After reading about the above, I wanted to see how the new models obtained better compression.
- I implemented equivalent models and played with the other authors' source code but found that they did not compress as well for my data
- This talk:
  - Describe the different approaches,
  - explain why existing experimental comparisons are incomplete,
  - and present experiments that explain the discrepancies (and should help direct future work)

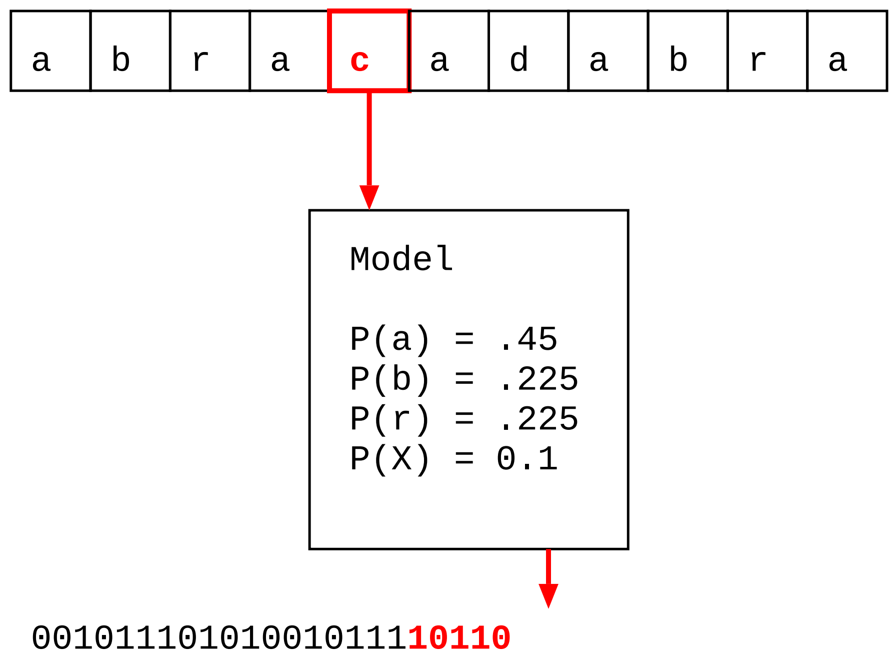
Statistical models

- Statistical text compression: compresses text by building a *model* that predicts next symbol



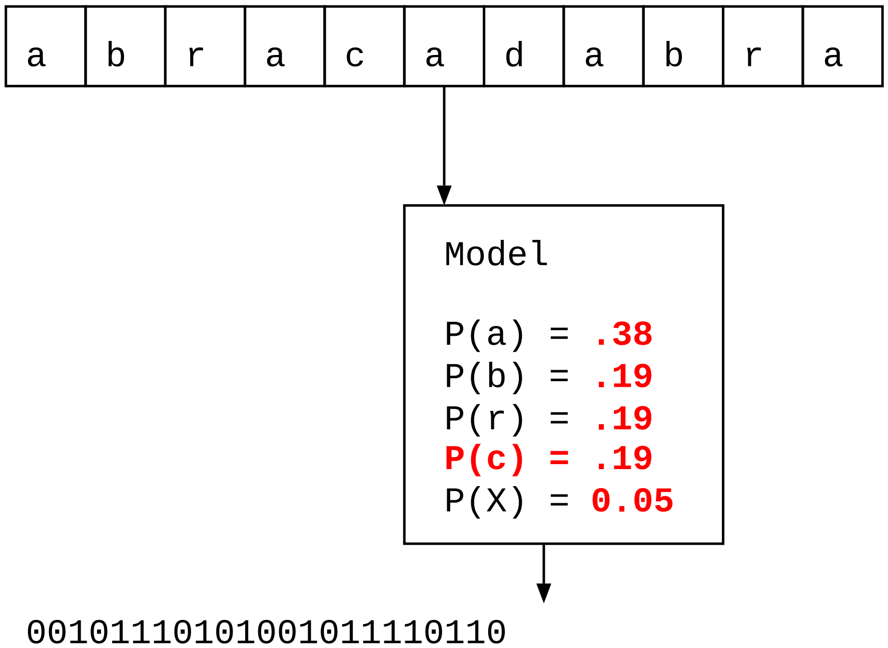
Statistical models

- Statistical text compression: compresses text by building a *model* that predicts next symbol



Statistical models

- Statistical text compression: compresses text by building a *model* that predicts next symbol
- *Adaptive* approach: interleave model building and prediction/compression. Requires only one pass over data, but has to "learn" model as it goes



Statistical XML compression

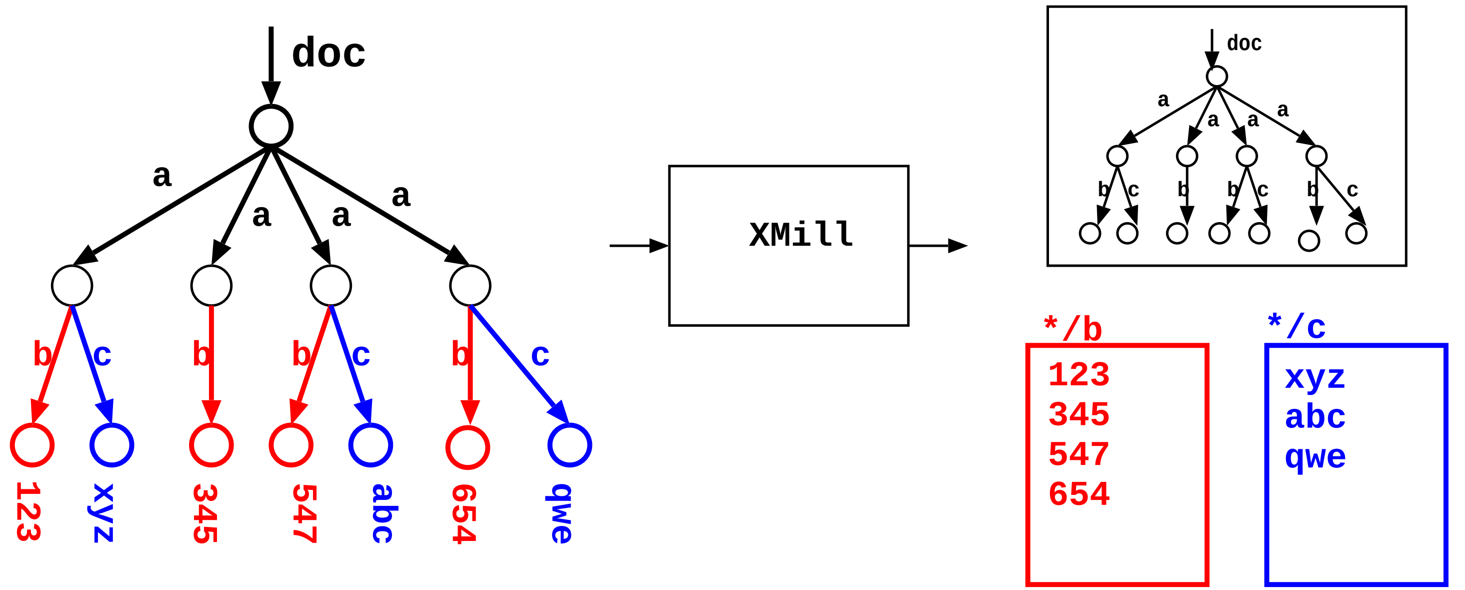
- Note: Most of the “interesting” content of most XML documents is unstructured text

	gzip			xmlppm		
file	struct	total	%struct	struct	total	%struct
DBLP	9.9MB	52.4MB	19%	667KB	33.4MB	2.0%
Medline	2.7MB	20.2MB	14%	539KB	13.7MB	3.9%
XMark	4.1MB	38.1MB	11%	287KB	27.6MB	1.0%
PSD	13.6MB	108MB	12%	2.5MB	79.6MB	3.1%

- Existing techniques already compress structure well (less than 1–20% of document)
- So, in this work, I focused only on modeling/compression of unstructured text in XML
- Compressing the structure is treated as a small fixed cost

XML compression strategy

- Statistical approach to XML compression: Mostly use statistical text compression, but “leverage” hierarchical structure somehow
- XMill used a similar idea, but *reorganized* XML text to make it easier for gzip to compress.



Approach #1: Multi-model

- Idea: Switch between n models, one model $M(e)$ per element name e
- Use $M(e)$ to encode the text immediately under e

```
M(book)       "\n  "            " "             "\n"
M(title)            "Gone..."
M(author)                          "Marg..."
M(chapter)                                          "..."
```

- Used in SCMPPM, XAust
- I'll call this the *Structured Contexts Model* (SCM) approach

Approach #2: Single-model

- Idea: Use a *single* model for text, but “prime” models with element symbols
- Priming symbols are “free” since can be inferred from tree context (this is part of the fixed cost we’re ignoring)

```
(00) "\n  " (01) "Gone..." (00) " " (02) "Marg..." (00) "\n"
```

- where (00), (01) etc are priming symbols for various element tags
- Used in XMLPPM, so I’ll call it the XMLPPM approach

Prior experiments

- XMLPPM: wide variety of XML documents, max size <1MB, used 1MB memory for statistical models
  - When limit reached, statistical model restarts
- SCMPPM: used large TREC documents with 8 elements, very little structure; statistical models used 1MB each (maximum of 8MB for TREC)
- XAust: used large documents such as DBLP; no memory upper limit

Flaws in prior experiments

- XMLPPM: didn't consider large documents, memory variation
- SCMPPM, XAust: didn't consider small documents, memory variation
  - Can't tell whether reported compression gain is due to using more memory or more accurate modeling
  - SCM approach may allocate much more memory than it ever uses
  - SCM approach may eventually attain much better compression, but may converge very slowly (benefiting only large files)
- Not enough data to draw any conclusions about relative merits of these approaches

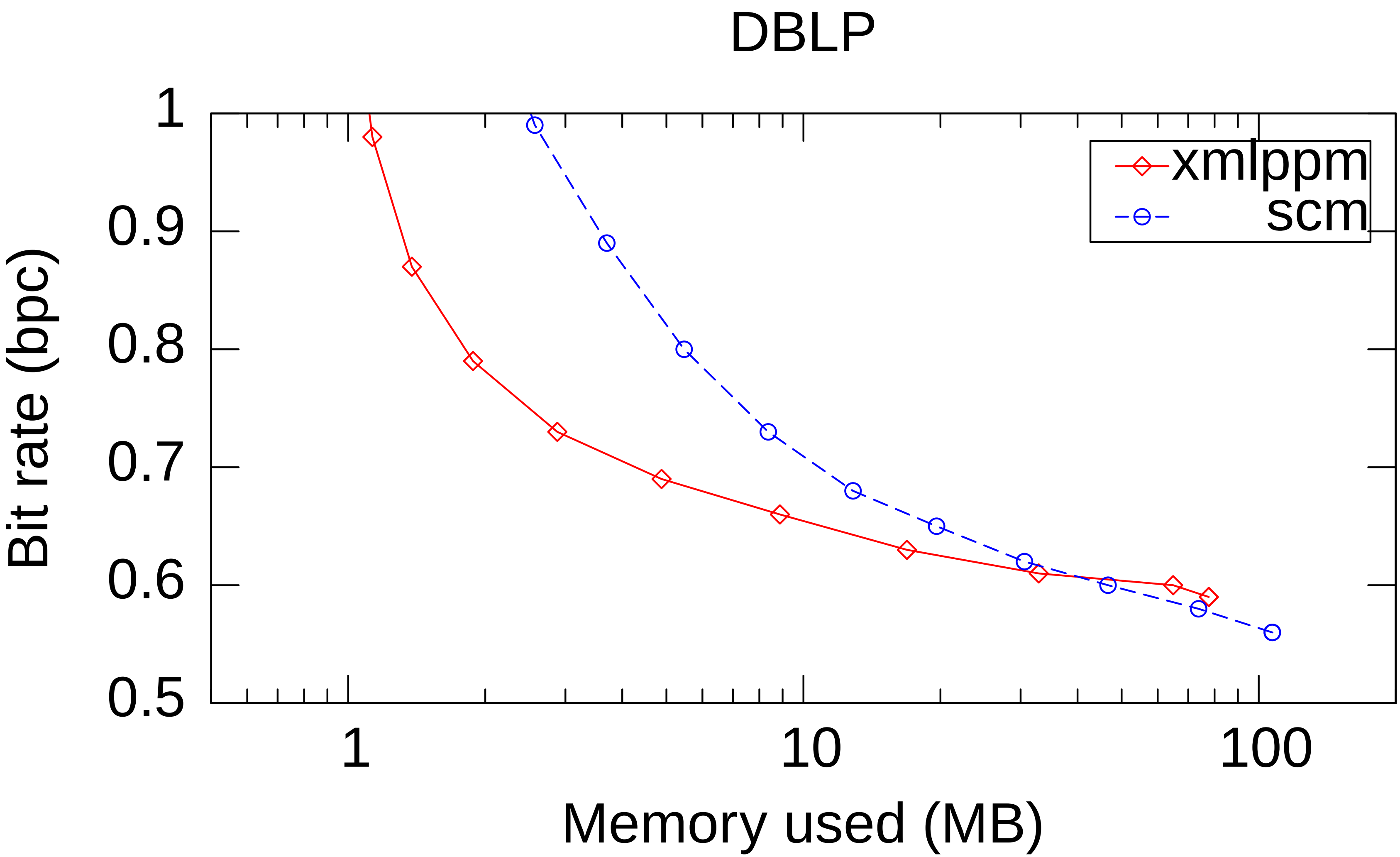
Experimental methodology

- Three "experiments":
  1. Memory vs. compression rate: for a wide range of model sizes, measured *compression rate* vs. *memory used*
  2. Memory footprint: for a wide range of model sizes, measured *memory allocated* vs. *memory used*
  3. Convergence rate: compressed *prefixes* of large files, and measured *prefix length* vs. *compression rate*
- Note: Measured "real" memory use by OS-reported RSS size.
  - Imperfect, but measuring exact memory use is difficult; approximate OS-view measurements probably good enough

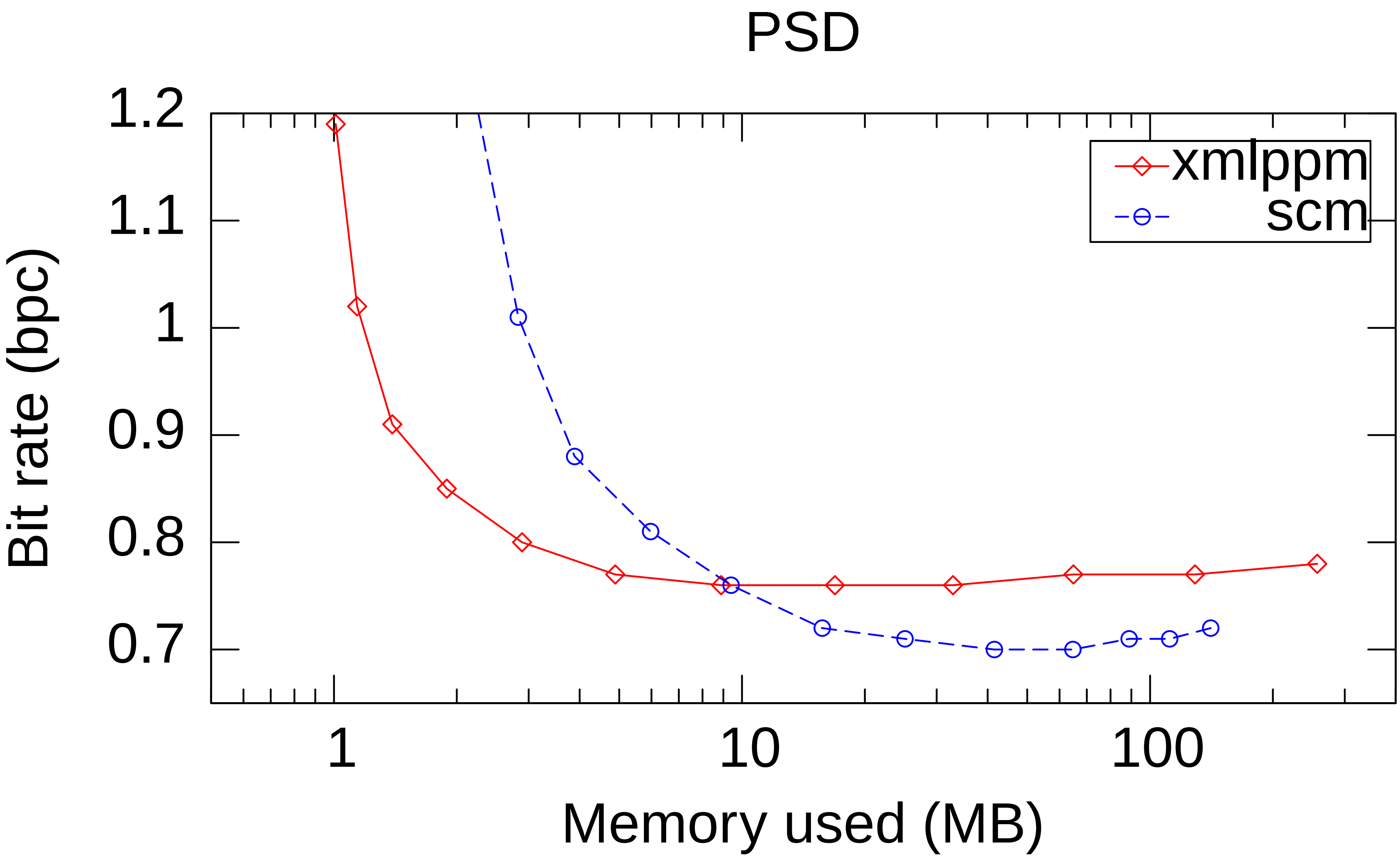
Experiments

- Used two large "typical" data sets:
  - DBLP (bibliography, 300MB uncompressed)
  - PSD (protein sequence database, 717MB uncompressed).
- Model size ranges: 4KB–32MB for SCM, 4KB–256MB for XMLPPM.
  - Note: for model sizes $>$ 32MB, SCM runs out of memory
- Prefix ranges: 10, 20, 50, 100, 200, 500, ... size of document
- Experiment machine: Athlon 3000+ (1.8Ghz), 512MB, FC3

Memory use vs. compression rate



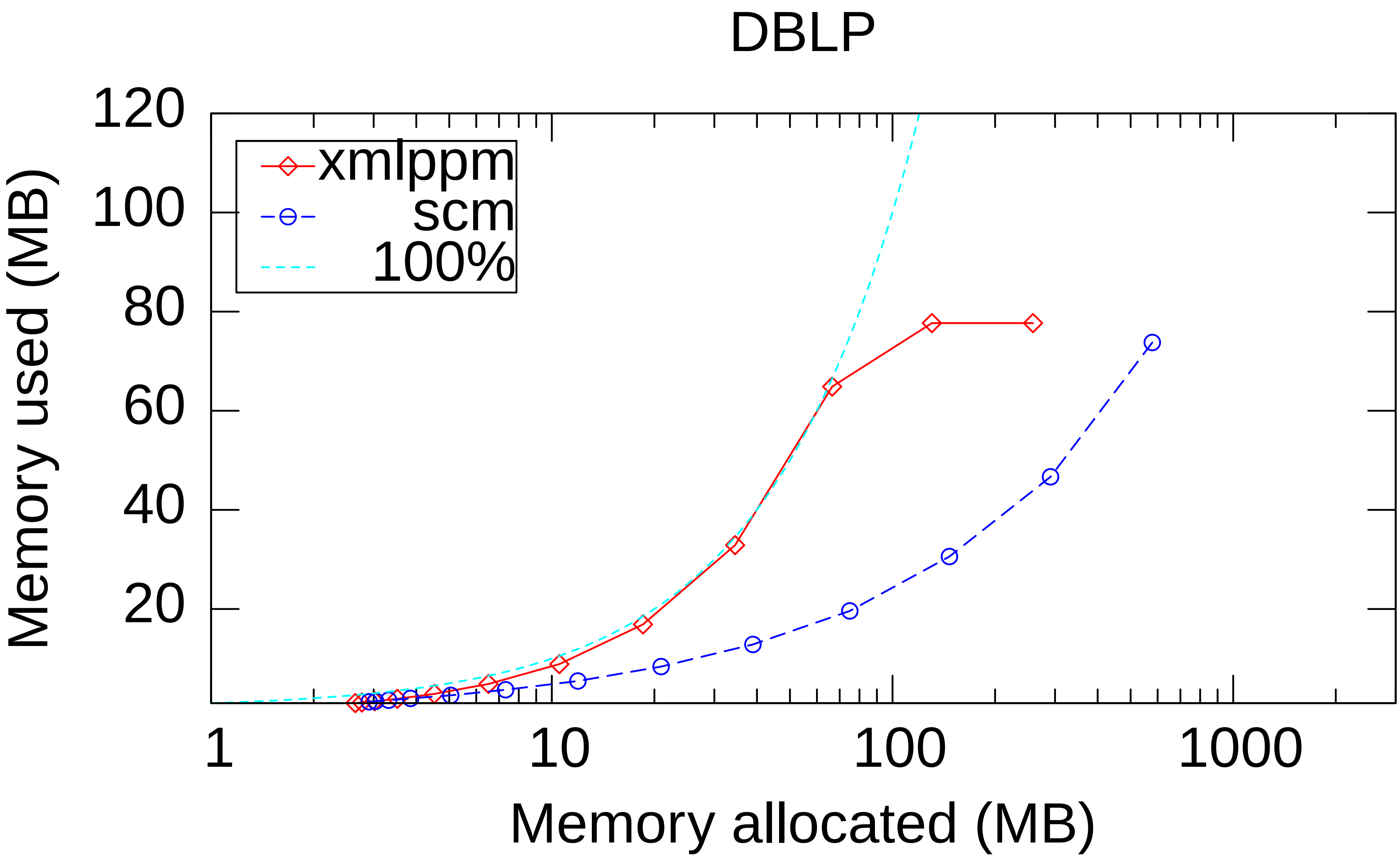
Memory use vs. compression rate



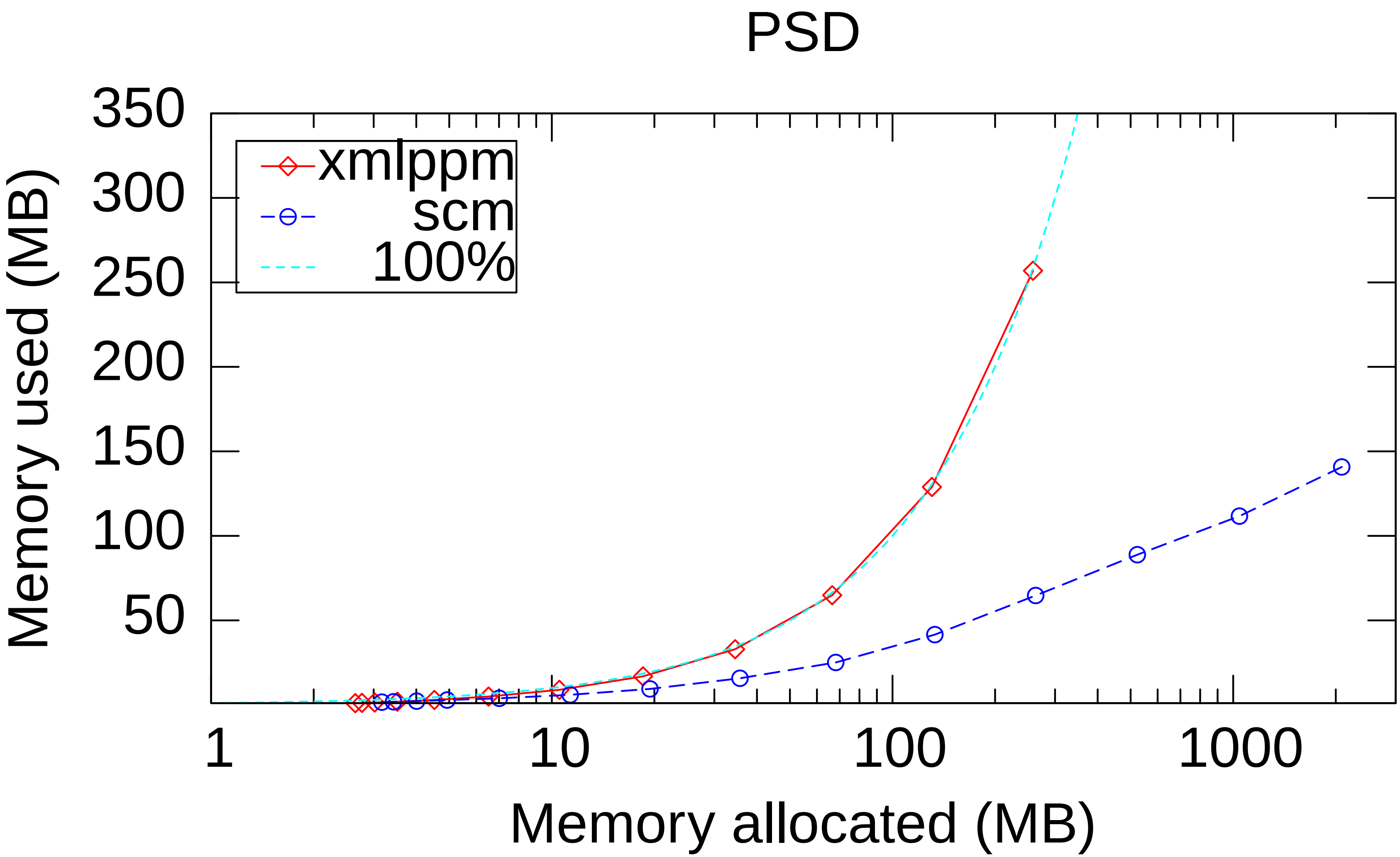
Memory use vs. compression rate

- For DBLP, improvement for SCM is minor (5%), needs over 40MB to achieve this.
- For PSD, SCM can perform around 10% better, improves after 10MB.
- Why?
  - separate SCM models initially have a lot of redundant information so need more memory to get same compression as XMLPPM
  - But eventually, models specialize and compression benefits outweigh memory cost of overlap
- Interestingly, both approaches tend to “overfit” for PSD when large amounts of memory available
- More memory doesn’t always help!

Memory utilization



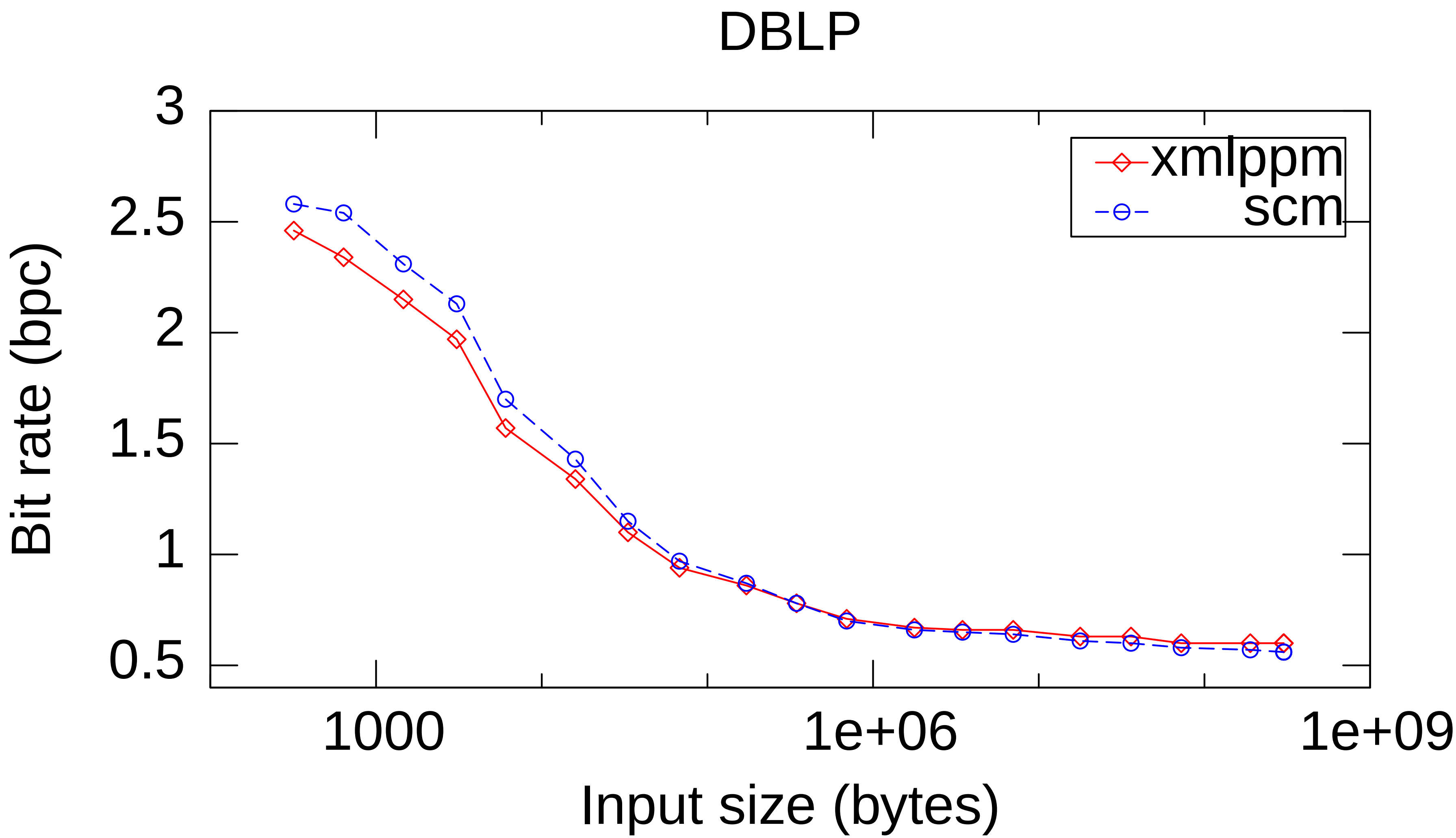
Memory utilization



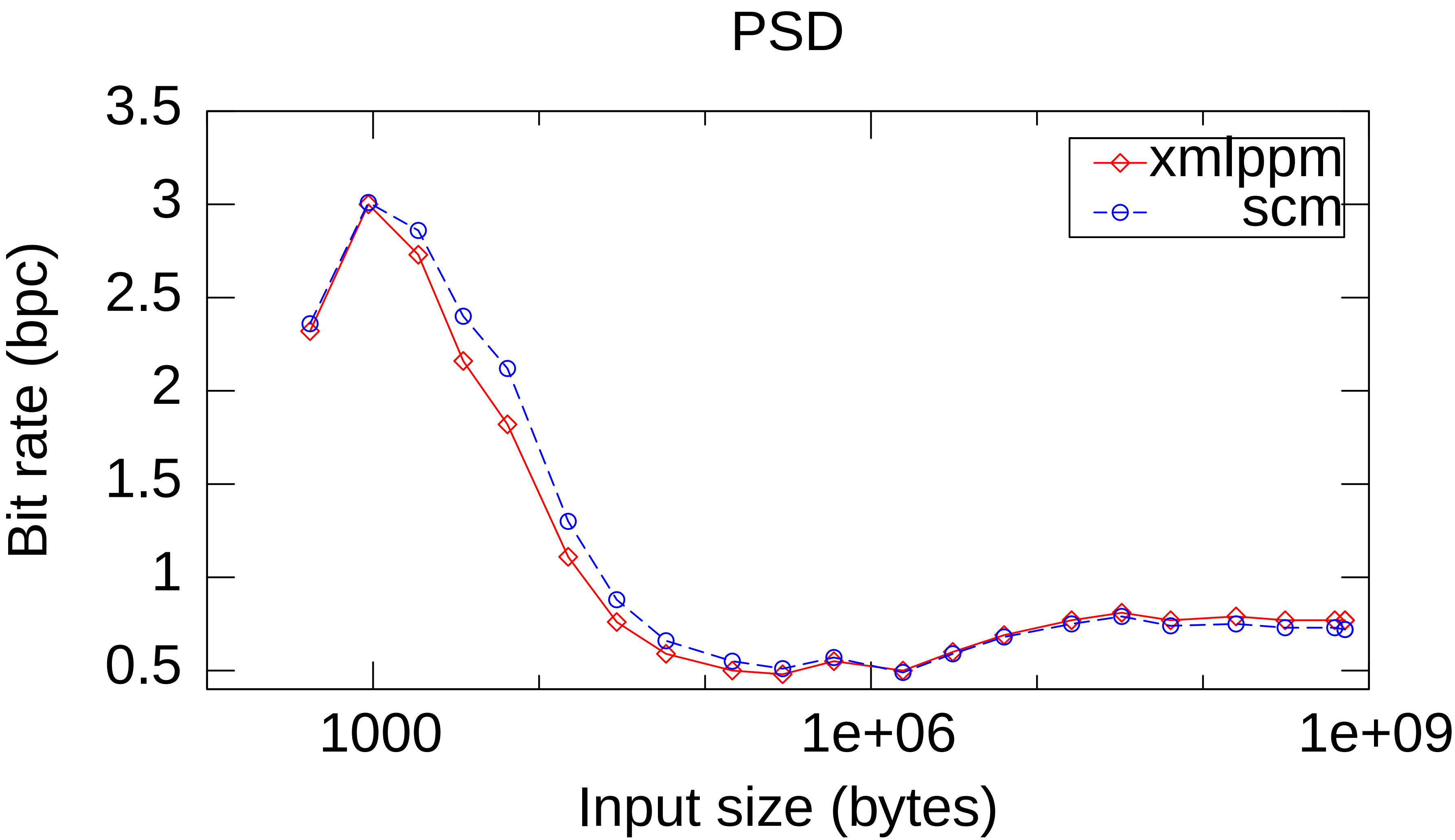
Memory utilization

- For both, SCM has 20-30% utilization, while XMLPPM has near 100% (until allocation exceeds requirements)
  - This means that most of the models in SCM never get “full”
  - All of the text is concentrated in around 25% of the models
- Interesting, but may not matter since modern operating systems allocate pages lazily
- as long as the program doesn’t try to allocate more memory than the machine actually has
  - this is why SCM fails for model size > 32MB

Convergence rate



Convergence rate



Convergence rate

- Overall trend: SCM performs worse for small files, but eventually wins out
- Why?
  - because SCM separates text under different elements, each model learns any common text separately
  - but because XMLPPM lumps all text into a single model, eventually it does worse because of averaging
- Crossover point is at around 500KB–2MB
  - Most Web/network applications of XML are way smaller
  - Most XML representations of DBs are way larger

Conclusions

- The SCM approach does provide better compression...
- provided you give it lots of memory and lots of data
  - Of course, for “archiving” XML DBs (DBLP, PSD, etc), this is fine!
- However, the XMLPPM approach is better for small documents or using small amounts of memory
  - This may make it preferable for on-the-fly compression of XML “messages” or fragments of XML within a DB
  - webpages, RDF, RSS feeds, SOAP RPCs
  - Or low-memory devices such as PDAs, mobile phones

Meta-conclusions

- XML compression research is still pretty open area
- However, so far experiments have focused on compression rate and ignored other costs
- In this work, we investigated tradeoffs between memory use and compression rate
- Data suggests several interesting directions
  - Can a single model provide good compression for both small and large documents?
  - Can a single model provide good compression for both low and high memory?