Compressed Materialised Views of Semi-Structured Data

Richard Gourlay, Brian Tripney, John Wilson
Department of Computer and Information Sciences
University of Strathclyde, Glasgow, UK
{rsg, brian, jnw}@cis.strath.ac.uk

Abstract

Query performance issues over semi-structured data have led to the emergence of materialised XML views as a means of restricting the data structure processed by a query. However preserving the conventional representation of such views remains a significant limiting factor especially in the context of mobile devices where processing power, memory usage and bandwidth are significant factors. To explore the concept of a compressed materialised view, we extend our earlier work on structural XML compression to produce a combination of structural summarisation and data compression techniques. These techniques provide a basis for efficiently dealing with both structural queries and value-based predicates. We evaluate the effectiveness of such a scheme, presenting results and performance measures that show advantages of using such structures.

1 Introduction

Query processing over relational database structures can be improved by the creation of materialised views, which provide reorganised data appropriate for particular query plans. A typical example is the use of precomputed data structures for on-line analytical processing (OLAP) queries. Whilst relational systems will continue to provide support for a variety of applications, the growth in importance of semistructured data is reflected in the expanding use of XML especially in the context of web-based applications. In this scenario, materialised views are used particularly to enhance the performance of queries by caching the results of previous searches on client platforms, thereby obviating the need for repeated contact with remote servers. The use of cached materialised views is of particular importance in mobile applications where a server may be unavailable at the moment that a query is generated. Given that applications operating in this environment are typically running on small devices such as mobile phones and using the limited bandwidth capabilities provided by wireless connections, there is operational advantage in optimising the use of communication infrastructure, processor time and data storage. The verbose representation of XML that has contributed to its general adoption as a standard for Internet-based communication between applications provides a challenge for the efficient use of these resources. Compressing materialised XML views can provide a benefit in this scenario as long as the view can be queried using the full functionality of XPath expressions. This can be achieved by using XML tree summarisation techniques to deal with the structural elements of queries and dictionary compression techniques to address query validation. Dictionary techniques work well with relational structures since domains provide a natural way to partition attribute values. Semistructured data loses this benefit so it is necessary to address the issue of how such structures can be usefully partitioned to generate the most efficient views.

2 Compressed View Model

Materialised views that typically result from XPath queries may be fragments of XML graphs or relation-based representations of the original structure. For XML fragments (as with the underlying structures themselves), the structural elements can be compressed by retaining only a skeleton representation of the graph and dictionaries can be used to support non-redundant storage of leaf values.

Semi-structured data can be represented as a graph with vertices used to indicate data items and structural interrelationships shown by arcs. Arbitrary graphs can be encoded in XML representation through the use of special ID:IDREF pairs. Although graphs are an accurate representation of XML views, tree representations provide a useful simplification. A sequence of distinct arcs in a graph is called a *path*. Paths may be either

```
<bibliography>
  <publications>
    <proceedings editor="p3">
      <title>Databases</title>
    </proceedings>
    <book author="p2">
      <title>Programming</title>
    </book>
    <book author="p1">
      <title>Databases</title>
    </book>
  </publications>
  <people>
    <person id="p1">
      <name>Millar</name>
    </person>
    <person id="p2">
      <name>Smith</name>
    </person>
    <person id="p3">
      <name>Stewart</name>
    </person>
  </people>
</bibliography>
```

Figure 1. Example XML Document

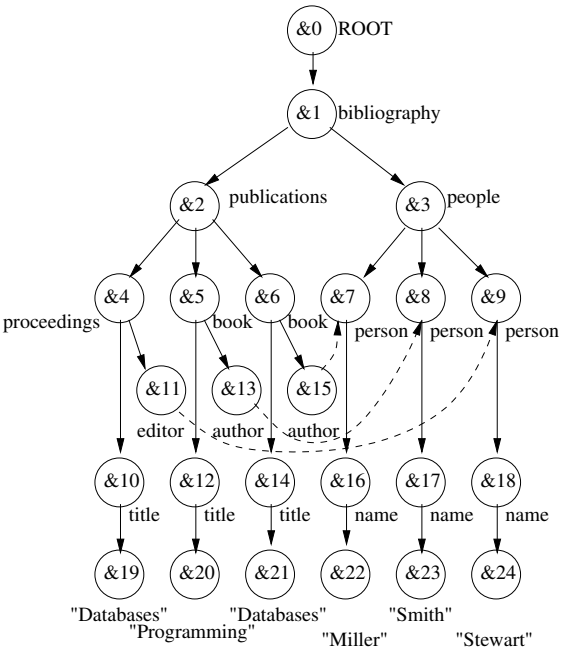

Figure 2. DataGraph for Example Document

chains (if the vertices in the path are distinct) or *circuits* (if the initial vertex and the final vertex are the same). Removing a single arc from each of the circuits in a graph produces a *spanning tree* that still connects all the data items in the graph. By removing different arcs, a number of different spanning trees can be produced from a single graph.

Query 1
//book[/author & /title/DATA='Databases']

Query performance over XML fragments in materialised views can be supported in much the same way as performance over the underlying data structure. The structural index approach illustrated by the work of Kaushik et al [17] allows a resolution of path location steps in linear time. A family of indexes ((j,k)-F+B-index) can be constructed using a range of values for forward or backward bisimilarity. Given the sample document shown in Figures 1 and 2 and excluding the path from the root vertex, the longest forward facing path in the query graph of Query 1 has length two: (book/title/DATA). There are no backward directed paths so the (2,0)-F+B-index shown in Figure 3 is the smallest covering index for the structural part of the query.

Value predicates in the query graph are replaced by a structural leaf predicate with the special tag label DATA. The family of (j,k)-F+B-indexes does not incorporate atomic data, so no member of this index family is covering for the validation part of any query. A materialised view that is covering for both structural and validation elements of a range of queries can however be generated by incorporating atomic data into the (j,k)-F+B-index structure. Entropy-based compression techniques for atomic data provide the basis for a compact representation of tree fragment views [13] whilst at the same time allowing validation without prior decompression of the tree fragment or externally referenced structure.

Although (2,0)-F+B-index graph allows for the fastest querying of the structure, the data is gathered into one single DATA partition, mixing data held in the title and name tags. When it comes to evaluating the predicate part of Query 1: (DATA='Databases') the optimally partitioned graph for the query would be the (2,1)-F+B-index shown in Figure 4. This splits the data held in the title and name tags into separate partitions allowing for more efficient compression and predicate querying. However the split adds a cost of three vertices into the structure, reducing the efficiency of structural querying of the graph.

A materialised view of the document can be generated holding the summarised structure of the document with links to dictionaries holding the compressed text of the document. An example of the model is shown in Figure 5, where the structure and partitions are gathered from the F+B-index with pointers from the compressed partitions into the relevant partition dictionaries.

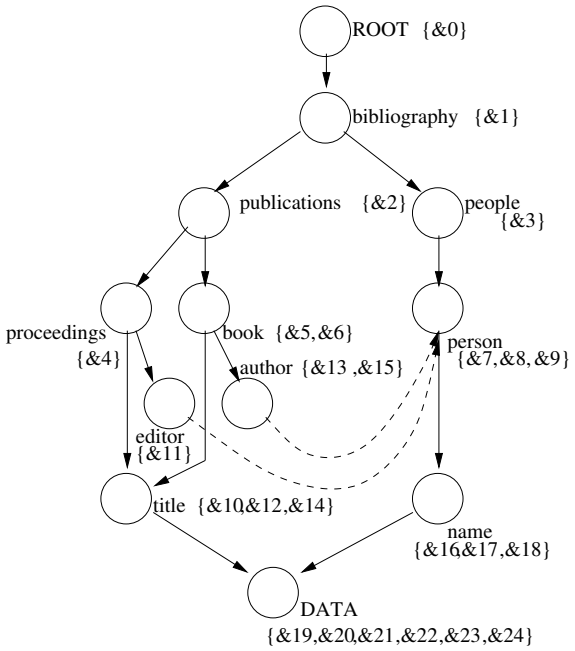

Figure 3. (2,0)-F+B-index

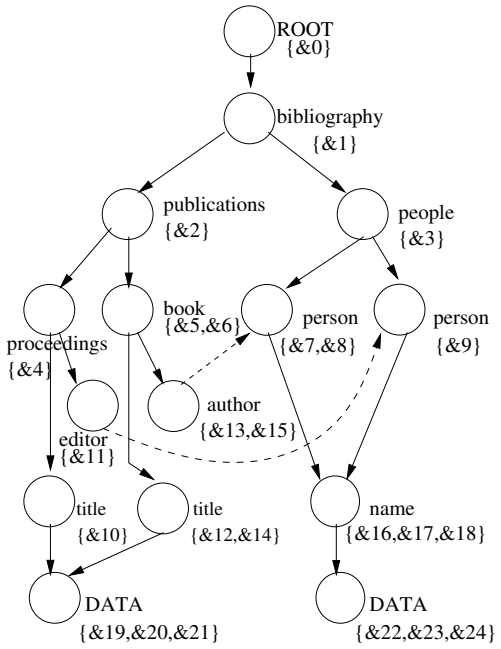

Figure 4. (2,1)-F+B-index

3 Related Work

Approaches for improving the performance of queries over XML data structures focus on the use of indexes and view materialisation. XPath views are typically used to preserve data structures that may be relevant to subsequent queries. These structures may be represented as sub-trees [1] or as relations [4]. Strategies for the maintenance of materialised XPath views are necessary if they are to be useful once the underlying data structure has been updated [27]. Materialised XPath views may provide sufficient coverage to enable the resolution of subsequent queries and there is interest in how to identify what portions of such queries can be answered by views and how queries can be reorganised to optimise view usage [32].

Several methods of efficiently compressing XML data have been developed. The earliest XML-aware compressors took advantage of context-based compression to reduce the data size from text based compressors. A comparative study by Ng et al [26] showed the compression ratios achievable by non-queryable XML compressors are far greater than those of queryable compressors. As an example of the latter class of compressors, XGrind [28], utilises a separate Huffman coding built for each separate element and attribute type to compress the values whilst tokenising the structural elements to maintain the document structure. This allows the document to be parsed as the original XML document with only elements of interest being decompressed. Cheney [9, 10] shows a scheme that can achieve 10–25% reduction from even most efficient text based compressors (zip, gzip etc.). While these schemes greatly reduce the size of the data there is no way to directly query this structure, the data must first be fully decompressed before being accessible to queries. Ferragina et al [14] developed a data structure combining several benefits of simple zip compression with fast access by linking two compressed array structures, one for the encoding of the structure and one for the node data.

The next broad range of XML compressors are data structures which build on XMill [21] to allow the analysis and storage of the data into a queryable form [11, 3, 23]. These allow efficient path queries over the data while still needing large parts of the data to be decompressed in the worst cases. We have already investigated the use of dictionary compression techniques for representing XML data structures [25] and other investigators have also examined the decomposition of XML into array-based structures [12].

Early work on different kinds of index structures for semi-structured data focused on query optimisation for the Lore system [22]. The main thrust of this work was the development of heuristics that determine when to use each of the four specific forms of indices (value, text, link and path index) provided by their experimental base. Halverson et al [15] identify the

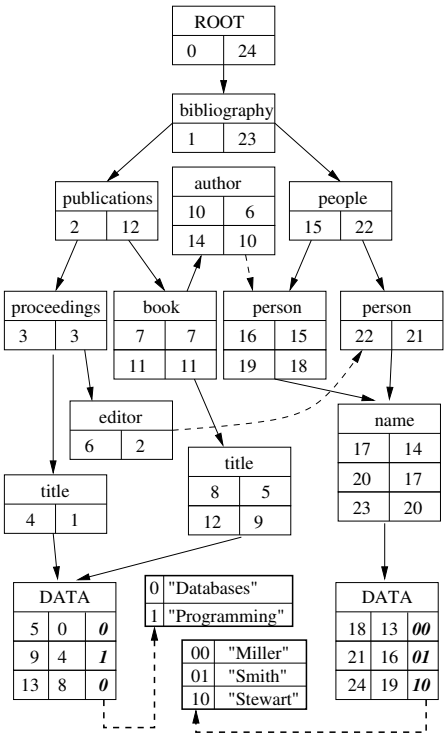

Figure 5. NSGraph with Dictionaries

need to combine pattern matching techniques based on inverted lists with the navigational approach typical for XML tree traversal algorithms, pointing out the lack of integration between these two lines of research. They provide a cost model for query answering in each of these domains, identifying query classes that are better suited to either approach or to a combination of both. Full path indexing can be supplemented by dynamic indexing that responds to query load [8]. Sub-graphs and feature extraction can also be used as the basis of index construction [33, 34]. Indexing techniques for supporting cross-border queries in mobile applications are also emerging [20]. Vectorization of XML, developed by Choi and Buneman [12, 5] combines the XMill [21] approach for compact representation of atomic data with the approach for skeleton compression by sharing subtrees [7] to address XML join queries. Their fundamental assumption is that the skeleton of typical XML documents is small and thus can be kept in memory. The actual data is only used in the last stage of their join algorithm, avoiding unnecessary I/O operations. Kaushik et al [18] extend their original work [16, 19] on structural indices for path expressions to include keyword constraints on the contained atomic data. They propose a general strategy to combine structural indices with inverted lists in order to address this class of queries efficiently and test their approach using the Niagara system [24]. As their value indices are based on techniques developed in the context of information retrieval systems, their resulting query system includes support for finding the k most relevant results. Structural and index values are combined by Amato et al [2] by extending the structure to incorporate the values for some elements or by incorporating B^+-Tree value indexes within the structure. Even within unstructured data there are often regular substructures that allows for a mapping to semi or fully structured data, Buneman et al [6] describes such a mapping to an edge-labelled graph structure. Our initial work [30] investigated effective partitioning schemes for semi-structured data, this was aimed at providing efficient structures for various classes of queries. We extended this work introducing NSGraph[31] which utilised numbering schemes in order to connect the structural elements within a partition to entries within an associated data dictionary.

4 Experimental Work

Although NSGraph comprises both structure and data values, we currently simplify the consideration of these components by dealing with them individually - the data values being compressed separately from the structure. This compression is achieved using a minimal-bit representation. To this end we create a dictionary for each data grouping, tokenising each data value using the smallest number of bits possible.

For example, in Figure 5 we use two dictionaries, one for titles and one for names. Titles consists of two unique data values, therefore each title value can be represented using a single bit token. For the three unique values associated with parent node *Name*, two bits must be used for each token. For each data grouping the token is only as large as $log_2(no.\ of\ unique\ values\ in\ group)$ rounded to the next whole number of bits.

As NSGraph does not yet deal with data values, we simulate this data compression using a Delphi implementation that deals solely with the data values and not the XML structure. Our simulation accepts the data values stripped from the XML files in comma-separated format. This simulation is not optimised and consequently we also state a theoretical compression achievable using this kind of minimal-bit representation calculated as follows:

Using a minimal token approach, the size of each dictionary should be $(token\ size * no.\ of\ entries) + \Sigma(uncompressed\ entry\ size)$. The compressed data size is then $no.\ of\ records * \Sigma(size\ of\ tokens\ in\ record)$,

	DataGraph	(2,2)F+B	(2,0)F+B	(0,2)F+B	(0,0)F+B	Data Set
v	1400565	3199	101	85	44	Legal
e	1400564	3198	1704	84	84	Legal
v	285004	23	14	23	14	Orders
e	285003	22	22	22	22	Orders
v	255004	21	13	21	13	Modified Orders
e	255003	20	20	20	20	Modified Orders

Table 1. Vertex and Edge Counts in Various Graph Simulations

with the total theoretical size being that of the compressed data plus all dictionaries.

The level of structural compression is evaluated by comparing the number of elements in the original document (shown in Table 1 as v in the DataGraph) and the number of partitions generated by NSGraph at various levels of bisimilarity. Similar results are shown for the number of edges found in the original DataGraph and NSGraph variations (shown as e in Table 1).

The variation shown as (0,0)F+B equates to the data being partitioned by label name alone ignoring the structure surrounding element. The (2,0)F+B variation gives an example of skeleton partitioning where only the label name and outgoing paths of the element are examined. The (0,2)F+B variation is similar to the previous example but only looking at incoming paths to the element. The final evaluated variation, (2,2)F+B, shows the partitions produced by separating elements on the basis of both incoming and outgoing paths being identical.

To evaluate the minimal-bit representation we make use of both benchmark and real-world data sets. Firstly, we have taken a subset of the TPC-H benchmark, based on the Orders element, to produce a series of test data containing varying numbers of orders. These files range from the smallest file containing details of 1000 orders up to the largest with 15000 (Orders). Part of the orders data is an artificially-created string drawn from a limited pool of words. To see the effect this has on compression, we have produced a second series of test data with these strings omitted (Modified Orders). Finally, we make use of a real-world set of legal data. Files in this series contain details of between 1000 and 13000 convictions taken from a sentencing information system (Legal).

For purposes of comparison, we use XGrind to compress the same data in XML format. This produces a compressed XML structure and associated metadata, the sizes of which are aggregated to produce the total XGrind compressed size. We evaluate the size contributed by the structural elements of the NSGraph by counting the edges and nodes produced. In general, these elements will provide a negligible contribution to the storage used when compared with that required for the compressed data values.

5 Results

Table 1 gives the results of structural summarisation of the sample data sets. The first column shows the number of vertices and edges found in the generated DataGraph of the various data sets. This gives a baseline with which to compare the various levels of bisimilarity used in producing the summarisations.

For all data sets there is a reduction in the number of both vertices and edges in the generated NSGraph variations. In the Legal data set this reduction is over 99.7% at the highest level of bisimilarity tested ((2+2)F+B). Other variations in the bisimilarity also restrict the number of vertices and edges but comparison with the (2,2)F+B case indicates that there is a certain amount of partition mixing in these results. The two TPC-H data sets give significant reductions with (2,2)F+B index, a 99.99% reduction in both the number of vertices and edges, the smaller differences with the other variations show less partition mixing and a more stable structure. Both TPC-H data sets give very similar results as the structure of the data changes only slightly between the two versions. The values for these indexes give an indication to the potential savings in memory that are available by using this approach to represent the structural elements of materialised views.

The results of data compression over the three data sets are shown in Figures 6, 7 & 8. As our simulated NSGraph data compression and XGrind take different input formats it would be inappropriate to directly compare the compression achieved by each. Therefore we compare the output size of both programs against the size of the raw data values used as input. In all cases the results include the size of the dictionaries or other information required to decompress the output.

Figure 6 shows the effect of compression over the TPC-H orders data. The simulated compression gives

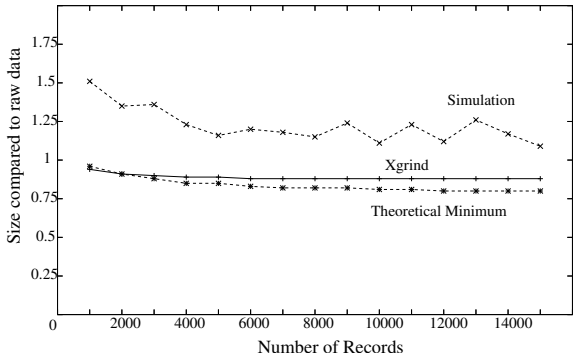

Figure 6. Orders data from TPC-H Benchmark

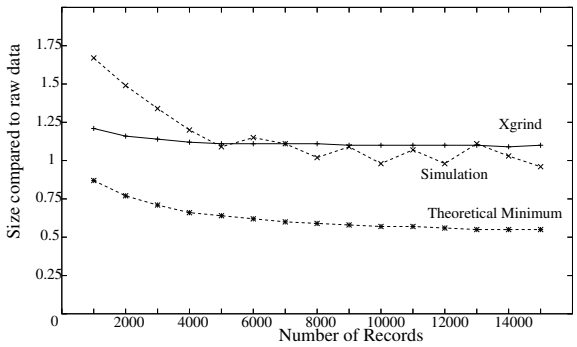

Figure 7. Modified Orders (Comment removed)

an output slightly larger than the raw data. The theoretical minimum for the minimal-bit representation corresponds closely to the size of structure produced by XGrind. The results in Figure 7 show the effect of removing the large comment element from the structure. Here the simulated NSGraph compression is comparable with the size of the XGrind structure. Figure 8 shows that the level of compression that is achieved by the NSGraph simulation is more advantageous than that produced by XGrind. In addition, the results produced by the NSGraph simulation now approach the theoretical minimum.

6 Discussion

Our results show that XGrind is more effective than the simulated NSGraph data compression over the TPC-H Orders data, but XGrind and the simulation are broadly comparable for the modified Orders data set. When comparing the theoretical minimal bit representation against XGrind, results for the full Orders data set are roughly equal, but the theoretical representation is better over the modified data set.

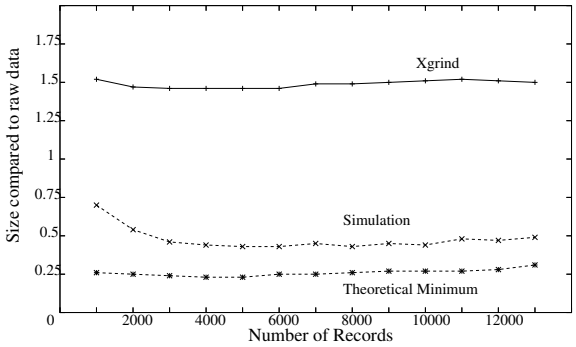

Figure 8. Legal data

The difference between the Orders and Modified Orders data sets is that the latter has had the generated text element removed. A class of target applications for the the methods that we are developing is in the area of distributing news items to mobile clients. In this scenario, only the metadata would be pushed to the client although longer text items may be pulled if they are requested by the client. For this kind of application, the Modified Orders data provides a good model for the metadata element.

Conventional benchmark data provides a major challenge for the entropy coding techniques we are developing since the random nature of the data is not an accurate reflection of the structure of many real-world applications. This becomes apparent when considering the results produced by the Legal dataset. Both the simulated NSGraph compression and the theoretical results provide more efficiently compressed structures than XGrind. It may also be seen that the simulation results for this data set are closer to theoretical minimum than for other datasets.

Differences between the simulated compression and the theoretical minimum are apparent for all data sets. These are due to overheads in the Delphi implementation and we expect that more efficient implementation techniques will enable us to build a complete NSGraph system that more closely approaches the optimum data representation. Results given do not include the size of the NSGraph structural element, however the levels of structural summarisation discussed below indicate the contribution to overall size will be negligible compared to the compressed data element.

Our previous work has shown that increasing the levels of bisimulation both in a forward and backward direction concentrates elements with increasingly similar structures [29]. Partitioning dictionaries using the same approach also results in an increasing number of separate, but more closely related, dictionary struc-

tures. The NSGraph system is able to deal with queries containing both structural and atomic value predicates without the need to store the original XML data structure. Combining both structural summarisation and compressed data exploits the advantages of both techniques to provide an efficient solution for representing XML trees.

The potential demonstrated by the simulated and theoretical dictionary compressed view technique when applied to semi-structured data presents an opportunity to exploit the redundancy that is found in many real-world data structures. This is as true of XML as it is of relational databases. Although ID:IDREF relationships in XML structures allow non-redundant modelling, data designers will typically use redundant methods to achieve the same effect. Examples of this practice include the PubMed corpus, which has been modelled with redundant author names rather than with the use of IDREFs to represent this information non-redundantly. The conventional approach of representing materialised views as tree fragments or as relations preserves this redundancy. Where memory and processor resources are limited, this approach simply preserves the redundancy of the underlying structure. Using compressed tokens in the manner described here removes this redundancy and makes optimal use of resources. This is especially important for applications running on small mobile devices, which typically have severe limitations in memory and processor capability. The implication of our results in the context of mobile pull-based data intensive applications is that the number of cache faults will usually be minimised by limiting the level of bisimulation. This is consistent with the outcome that could be expected from increasing the fragmentation of the data structure representation. Power utilisation can also be minimised by optimising the representation of the data and hence reducing processor load. It has the added advantage of limiting the need to download additional blocks of data from the server in order to resolve specific queries.

The preliminary results we report are being followed up by an implementation of the compressed materialised view model that we have simulated. We are also exploring the automatic detection of optimal (or near optimal) partitioning of the data in order to maximise the efficiency of the partition compression. We expect to be able to extract benefit by altering the compression scheme used in the generated partitions contingent on the characteristics of the partitions that have been captured. Huffman encoding of string data or zip compression of larger data items may also provide improved compression ratios in structures that incorporate large text elements.

7 Conclusions

Compressed materialised XML views can provide a benefit in the efficient use of resources including communication bandwidth, processor time and data storage but such views must still be accessible to the full functionality of XPath expressions. Such views can be generated by using XML tree summarisation techniques to deal with the structural elements of queries and dictionary compression techniques to address query validation. Dictionary techniques work well with relational structures since domains provide a natural way to partition attribute values. Semi-structured data loses this benefit so it is necessary to address the issues of how such structures can be efficiently partitioned to generate the most efficient structure to produce the compressed views.

We are currently evaluating a system that can combine the benefits of an F+B-index structural summarisation with the benefits of dictionary compression of the data. The intention is to produce a coherent data structure that can utilise the fast response characteristics of structural indexing whilst also being able efficiently to access a compressed representation of the data within the structure. The purpose of this system is to evaluate the effect that combining data compression and structural indexing has on the size of the in-memory representation of semi-structured data. The objective of the current research is to build on the simulation results we report and to determine whether several mobile device specific problems - memory limitations, battery and processor usage - could be efficiently addressed in such a combined system.

References

[1] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental maintenance for materialized views over semistructured data. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 38–49, 24–27 1998.

[2] G. Amato, F. Debole, P. Zezula, and F. Rabitti. YAPI: Yet another path index for xml searching. In *Proc of ECDL*, pages 176 – 187, 2003.

[3] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A. Pugliese. Xquec: Pushing queries to compressed xml data. In *VLDB*, pages 1065–1068, 2003.

[4] A. Balmin, F. Özcan, K. S. Beyer, R. Cochrane, and H. Pirahesh. A framework for using materialized xpath views in xml query processing. In *VLDB'2004: Proceedings of the 30th international conference on Very large data bases*, pages 60–71, 2004.

[5] P. Buneman, B. Choi, W. Fan, R. Hutchison, R. Mann, and S. D. Viglas. Vectorizing and querying large xml repositories. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 261–272, Washington, DC, USA, 2005. IEEE Computer Society.

[6] P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 336–350, London, UK, 1997. Springer-Verlag.

[7] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proc. of VLDB*, pages 141–152, 2003.

[8] Q. Chen, A. Lim, and K. W. Ong. D(k)-index: an adaptive structural summary for graph-structured data. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 134–144, New York, NY, USA, 2003. ACM Press.

[9] J. Cheney. Compressing xml with multiplexed hierarchical ppm models. In *Data Compression Conference*, pages 163–, 2001.

[10] J. Cheney. Tradeoffs in xml database compression. In *DCC*, pages 392–401, 2006.

[11] J. Cheng and W. Ng. Xqzip: Querying compressed xml using structural indexing. In *EDBT*, pages 219–236, 2004.

[12] B. Choi and P. Buneman. XML vectorization: A column-based XML storage model. Technical Report MS-CIS-03-17, University of Pennsylvania, 2003.

[13] W. P. Cockshott, D. McGregor, and J. Wilson. High-performance operations using a compressed database architecture. *Computer J.*, 41(5):283–296, 1998.

[14] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and searching xml data via two zips. In *Proc. World Wide Web Conference 2006 (WWW'06)*, 2006.

[15] A. Halverson, J. Burger, et al. Mixed mode XML query processing. In *Proc of VLDB*, pages 225–236, 2003.

[16] R. Kaushik, P. Bohannon, et al. Covering indexes for branching path queries. In *Proc. of SIGMOD*, pages 133–144, 2002.

[17] R. Kaushik, R. Krishnamurthy, et al. On the integration of structure indexes and inverted lists. In *Proc. of ICDE*, page 829, 2004.

[18] R. Kaushik, R. Krishnamurthy, et al. On the integration of structure indexes and inverted lists. In *Proc. of SIGMOD*, pages 779–790, 2004.

[19] R. Kaushik, P. Shenoy, et al. Exploiting local similarity for indexing paths in graph-structured data. In *Proc. of ICDE*, pages 129–140, 2002.

[20] J. Lee, Y. Lee, S. Kang, H. Jin, S. Lee, B. Kim, and J. Song. Bmq-index: Shared and incremental processing of border monitoring queries over data streams. In *The 7th International Conference on Mobile Data Management (MDM'06)*, 2006.

[21] H. Liefke and D.Suciu. XMill: An efficient compressor for XML data. In *Proc. of SIGMOD*, pages 153–164, 2000.

[22] J. McHugh and J. Widom. Query optimization for XML. In *Proc. of VLDB*, pages 315–326, 1999.

[23] J.-K. Min, M.-J. Park, and C.-W. Chung. Xpress: A queriable compression for xml data. In *SIGMOD Conference*, pages 122–133, 2003.

[24] J. F. Naughton, D. J. DeWitt, et al. The Niagara internet query system. *IEEE Data Eng. Bull.*, 24(2):27–33, 2001.

[25] M. Neumüller and J. N. Wilson. Improving XML processing using adapted data structures. In *Proc. of WEBDB*, pages 63–77, 2002.

[26] W. Ng, W. Y. Lam, and J. Cheng. Comparative analysis of xml compression technologies. *World Wide Web*, 9(1):5–33, 2006.

[27] A. Sawires, J. Tatemura, O. Po, D. Agrawal, A. E. Abbadi, and K. S. Candan. Maintaining xpath views in loosely coupled systems. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 583–594. VLDB Endowment, 2006.

[28] P. M. Tolani and J. R. Haritsa. Xgrind: A query-friendly xml compressor. In *ICDE*, pages 225–234, 2002.

[29] J. Wilson, R. Gourlay, R. Japp, and M. Neumüller. Extracting partition statistics from semistructured data. In *1st International Workshop on XML Data Management Tools and Techniques (XANTEC 2006)*, 2006.

[30] J. N. Wilson, R. Gourlay, R. Japp, and M. Neumueller. Extracting partition statistics from semistructured data. In *17th International Workshop on Database and Expert Systems Applications (DEXA 2006)*, pages 497–506, Piscataway NJ, September 2006. IEEE.

[31] J. N. Wilson, R. Gourlay, R. Japp, and M. Neumueller. A resource efficient hybrid data structure for twig queries. In S. Amer-Yahia, Z. Bellahsene, E. Hunt, R. Unland, and J. XuYu, editors, *Database and XML Technologies : 4th International XML Database Symposium (XSym 2006)*, pages 77–91. Springer Berlin / Heidelberg, September 2006.

[32] W. Xu and Z. M. Ozsoyoglu. Rewriting xpath queries using materialized views. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 121–132. VLDB Endowment, 2005.

[33] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346, New York, NY, USA, 2004. ACM Press.

[34] N. Zhang, M. T. Ozsu, I. F. Ilyas, A. Aboulnaga, and D. R. Cheriton. Fix: Feature-based indexing technique for xml documents. Technical Report CS-2006-07, School of Computer Science, University of Waterloo, 2006.