Extracting Partition Statistics from Semistructured Data

John N. Wilson Richard Gourlay Robert Japp
Mathias Neumüller
Department of Computer and Information Sciences
University of Strathclyde, Glasgow, UK
{jnw,rsg,rpj,mathias}@cis.strath.ac.uk

Abstract

The effective grouping, or partitioning, *of semistructured data is of fundamental importance when providing support for queries. Partitions allow items within the data set that share common structural properties to be identified efficiently. This allows queries that make use of these properties, such as branching path expressions, to be accelerated. Here, we evaluate the effectiveness of several partitioning techniques by establishing the number of partitions that each scheme can identify over a given data set. In particular, we explore the use of parameterised indexes, based upon the notion of forward and backward bisimilarity, as a means of partitioning semistructured data; demonstrating that even restricted instances of such indexes can be used to identify the majority of relevant partitions in the data.*

1 Introduction

Efficient resolution of *branching path expressions* over XML data requires that data items having a structure that matches thatspecified by the query can be identified without traversing the complete data set. This can be achieved with a suitable form of index graph. Such graphs can provide a structural summary of the complete data set and allow data items with similar structure (and semantics) to be collected together and accessed efficiently. Consequently, index graphs implicitly define a partitioning of the data, with the number, and nature, of partitions identified varying with choice of indexing technique. In this paper, we explore the effectiveness of a number of forms of index graph through an exploration of the number of partitions that each index can identify over a given data set.

The different forms of index that can be used with semistructured data each define a set of *partitions* over the data. Each vertex on the index graph corresponds to exactly one partition, with all data therein sharing some notion of common structural properties. The size of the index graph, and therefore the number of partitions identified, varies dramatically with choice of indexing technique. There is a tradeoff between the size of the index graph and accuracy consequently the choice of index and the number of partitions identified, will vary considerably depending on which form of index has been selected.

Constructing a suitable index graph, and identifying a set of partitions, is a fundamental step in the efficient processing of semistructured data. In addition to the benefits of efficient access provided by the index graph, partitions provide a valuable opportunity for optimisation. Firstly, it is likely that data within a given partition will be accessed together. Secondly, partitions capture regular aspects of the data set, presenting an opportunity to exploit techniques aimed at such data (which have been extensively studied for the relational model). Index graphs, and the partitions that they identify, are central to the efficient processing of XML and can be used as the basis of query processing models in the absence of globally valid schemata or when cost-based optimisers are unavailable.

The remainder of this paper is structured as follows. Section 2 summarises the necessary concepts and terminology, while Section 3 describes a number of techniques for partitioning semistructured data. Section 4 presents the main contribution of this paper: an evaluation of how effective different indexing techniques are at identifying partitions of the data. Section 5 concludes the paper.

2 Background

The data graph representation of semistructured data is a *directed node-labelled graph*. The vertices of this graph represent data items, with the edges representing structural relationships. Directly creating a data graph from the flat-file representation of XML gives a tree rather than a graph:

it is only when additional relationships, such as those encoded using ID:IDREF references, are included that a graph containing cycles can be obtained. Here, we refer to a tree view of the data that is both a spanning tree and is rooted at the root vertex as a *distinguished spanning tree*. The forms of index graph of interest here are all targeted at path expressions (i.e. they do not index the values of the atomic data) and provide a graph derived from the data graph that can associate multiple data items with a single vertex.

Branching path expressions are a form of query that can be performed over a data graph. These queries are comprised of a sequence of labels (a *label path*), and can contain both forward and backward separators. Evaluating such a query consists of finding all vertices on the data graph that have a path leading to them matching the path given by the labels with forward separators (this is referred to as the *primary path*) that also satisfy the conditions given by the backward separators. If a given index graph is smaller than the data graph it summarises, then it is possible to evaluate a query more efficiently using the index than it would be to traverse the complete data set.

The use of *bisimilarity* as a means of partitioning semi-structured data was first demonstrated by Buneman et al. [1]. Such indexes group vertices on the data graph according to the equality of the complete set of outgoing and incoming edges to a given vertex. This produces a refined set of partitions over a given data set that can be used as an aid to the efficient processing of branching path expressions. However, such index graphs can become too large to be useful. Kaushik et al. [5] introduce a parameterised index based upon bisimilarity, with a view to using such parameters to limit the size of the index graph. More recently, He and Yang [4] proposed a multi-resolution index upon bisimilarity: a form of index with the degree of bisimilarity varied according to the needs of each node.

3 Partitioning Techniques

Partitions can be categorised in one of three ways (depending on the type of vertices contained therein). Firstly, a partition comprised solely of vertices representing atomic data is said to be an *atomic partition*. Secondly, a partition that contains only complex vertices, those vertices that combine the information stored in other vertices by means of a set of outgoing edges, is said to be a *complex partition*. Finally, a partition that contains both atomic and complex vertices is said to be a *mixed partition*. Such distinctions will be considered when contrasting different partitioning schemes.

Technique 1 (Label Partitions) This is the simplest partitioning technique discussed here. For a given data set (Figure 1), all vertices carrying an identical label are grouped together. Using this approach, there will be exactly one

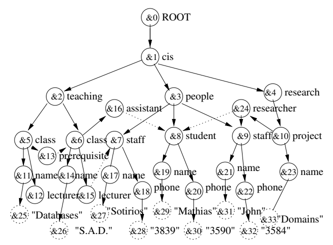

Figure 1. An example data graph

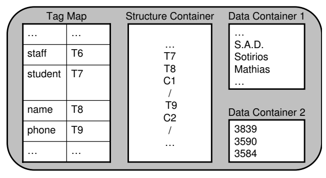

Figure 2. XMill containers.

atomic partition and one complex partition for each source-specific tag present in the data graph (including the root vertex).

Technique 2 (Parent Partitions) This technique is based on the equality of the label of a node's parent node in the distinct spanning tree system [7]. A vertex on a data graph may have more than one incoming edge (parent node), so it becomes possible for a node to be present in more than one partition. The total number of partitions that can be identified is bounded by the size of the label alphabet. Since the root vertex has no incoming edge it does not belong to any partition. Inversely, there is no partition corresponding to the tag label `DATA`, as atomic vertices have no outgoing edges.

Figure 2 illustrates the transformation of our example data graph as used with XMill. This approach causes all data items representing a 'name' to be merged into one atomic partition, regardless of whether they are the names of people, projects or courses. This effect of combining unrelated information is referred to as *partition mixing*.

Technique 3 (Path Partitions) This partitioning technique attempts to avoid partition mixing by partitioning the vertex set based upon the entire path from the root node to a given vertex. This may result in a given vertex being part of more than one partition. In fact, cyclic graphs will result in an unbounded number of partitions being identified. However, the equivalent technique based upon the distinguished spanning tree is bounded by the size of the node set giving a usable definition of partitioning. This is equivalent to the strong DataGuide [3] and can result in *partition splitting* (Figure 3).

Technique 4 (Depth Partitions) The notion of partitioning that we are using here allows for the characterisation of partitions based upon node level. Here, all nodes that occur at the same depth in the tree view of the data are grouped

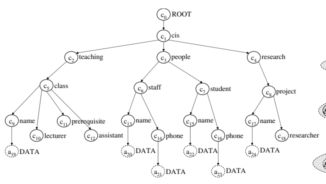
Figure 3. Strong DataGuide.

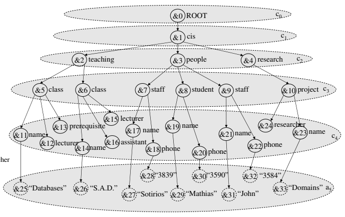
Figure 4. Depth partitions.

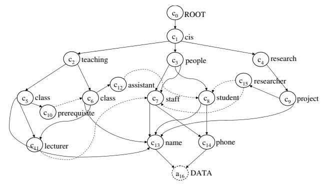
Figure 7. The skeleton representation.

Source	Size (MB)	Note
Xmark	10	Synthetic data, scale factor 0.1
PubMed	31	www.ncbi.nlm.nih.gov/entrez
Ensembl Rat	36	www.ensembl.org
MAGE	75	www.mged.org

Figure 8. Data structures investigated.

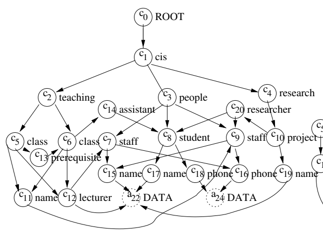
Figure 5. The A(1)-index graph.

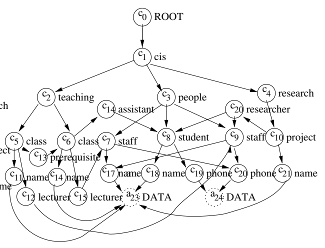
Figure 6. The (1,1)-F+B-Index graph.

together. Although such a partitioning may initially seem to be of limited use, certain queries can be accelerated with an index based upon such a partitioning. For example, the query `/*/*/*`, can be resolved efficiently using this technique. If this technique is used on the graph view of the data then, as was the case with Path Partitions, an unbounded number of partitions can be identified. Given that the partitions identified do not take account of node labelling, mixed partitions can be identified. Figure 4 shows such a partitioning imposed on the data graph.

Technique 5 (Skeleton Partitions) This technique is based on the concept of forward bisimilarity of a node [2], i.e. the equality of the collection of outgoing paths. This technique states that two nodes in the tree are said to be bisimilar if they have the same label and the same ordered sequence of bisimilar child nodes. This allows common substructures in a document to be identified, for example, the two `staff` nodes shown in Figure 1; however, this technique splits the two `class` nodes because of the differences in their respective sub-trees. Given that only outgoing edges are considered when computing bisimilarity, all atomic data is grouped in a single atomic partition.

Technique 6 (Local Backward Bisimilarity Partitions) Here the entire set of incoming edges is used to determine the partition of each node [6]. This gives an index that is similar to that of path partitioning, whilst avoiding nodes being present in multiple partitions based on each incoming edge. Since, in practice, path expressions are often of limited length, the lengths of the paths used when calculating the bisimilarity can also be limited. The definition of bisimilarity then becomes recursive, stating that a vertex is k-bisimilar to another if they have equal labels and have the same set of incoming edges from vertices that are $(k-1)$-bisimilar. Thus, the parameter k can be used to control the balance between index size and index coverage. Figure 5 shows the index graph of the example source based on 1-bisimilarity. Again, every vertex defines a partition of the data set. The two `staff` vertices that were contained in a single partition when using skeleton partitions are now split As seen previously, all names are merged into a single partition regardless of whether they refer to people, projects or courses. Increasing the value of k would overcome this limitation.

Technique 7 (Forward and Backward Bisimilarity Partitions) This method combines the structural properties identified by techniques such as skeleton partitions, with the contextual properties that can be identified using backward bisimilarity [5]. The resulting index graph is covering for general branching path expressions without value predicates. As with local backward bisimilarity, the lengths of paths considered can be limited. Here, two parameters, k_b and k_f, that restrict the lengths of incoming and outgoing paths respectively, can be used to reduce the complexity of the final index graph. Figure 6 shows the index graph based on (1,1)-F+B-bisimilarity, The `name` and `lecturer` vertices occurring below `class` vertices have been split, although their incoming paths of length one are identical. This is because they can be distinguished based on their siblings by using a branching path expression whose length does not exceed one. For example, the branching path expression `class[/prerequisite]/name` selects vertex **&11** but not vertex **&14**, which were combined in partition c_{11} in Figure 5 but are split into the partitions c_{11} and c_{14}.

4 Partition Statistics

Our analysis using various data sources (Figure 8) explores both the expressive power of a given partitioning

Source	Size |V|	Label	Parent	Path	Depth	Skeleton
Total number of partitions						
Xmark	322,327	77	77	933	14	13,662
PubMed	1,288,555	87	87	197	9	15,374
Ensembl Rat	2,556,204	103	103	295	18	340
MAGE	3,139,711	31	31	76	11	10,699
Total number of atomic partitions						
Xmark	322,327	1	34	405	1	1
PubMed	1,288,555	1	44	80	1	1
Ensembl Rat	2,556,204	1	77	102	1	1
MAGE	3,139,711	1	11	25	1	1
Total number of complex partitions						
Xmark	322,327	77	39	528	5	13,661
PubMed	1,288,555	86	37	117	4	15,373
Ensembl Rat	2,556,204	103	1	103	2	339
MAGE	3,139,711	31	20	51	6	10,689
Total number of mixed partitions						
Xmark	322,327	0	4	0	8	0
PubMed	1,288,555	0	5	0	4	0
Ensembl Rat	2,556,204	0	25	0	15	0
MAGE	3,139,711	0	0	0	4	0

Figure 9. Static partitions counts.

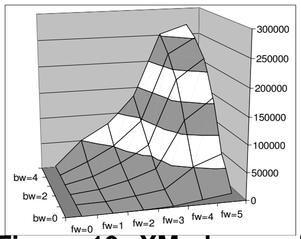

Figure 10. XMark: Depth $= 1$

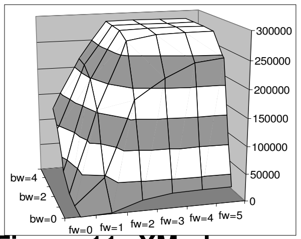

Figure 11. XMark: Depth $= 3$

technique and the complexity of the underlying data source. Partition Techniques 1–5 are classified as *fixed-partitioning* techniques, in that they do not support the use of parameters to influence the operation of the algorithm. With *parameterised-partitioning* techniques however, a range of parameter combinations are possible.

4.1 Fixed-Partitioning Statistics

Figure 9 shows the number of distinct partitions that can be identified over our sample data when using partitioning Techniques 1–5 (the partitioning techniques that do not make use of parameters). The results shown in Figure 9 clearly demonstrate that competing partitioning techniques produce significantly different numbers of partitions for a given data.

In contrast to the regular structure of the Ensembl file (Figure 9), the XMark data exhibits few indications of a regular structure since its data, by design, is convoluted. This is indicated by the results presented in Figure 9. In the XMark data set there is no correspondence between the number of complex partitions identified by label partitioning and path partitioning (or indeed, any of the chosen schemes). A large number of partitions are identified using the skeleton partitioning technique, even though the size of the data graph is significantly smaller than the data graph of the other sample documents. This is a consequence of similar elements occurring in many contexts in XMark generated files.

Of the remaining two example files, PubMed and MAGE, the results shown indicate that they do have some degree of varying content (i.e. tags appearing in different contexts), but that these real data sources do not have the same complexity as the synthetically generated XMark data. When using the skeleton partitioning technique, the number of partitions identified for the PubMed and MAGE files is comparable to that of the XMark data, although, again, it should be noted that there is a significant difference in the size of the data sets.

With the exception of the regularly structured Ensembl file, the number of partitions identifiable using the Skeleton technique greatly exceeds that of the more simple partition techniques. This suggests that techniques based upon bisimulation are likely to be the most applicable when accelerating branching path expressions or when clustering the data according to its context.

4.2 Parameterised-Partitioning Statistics

In contrast to partitioning Techniques 1–5, Techniques 6 and 7 define a family of partition structures, with one or more parameters being used to specify the exact nature of any given instance of the partitioning. Points in the parameter space that correspond to either stable or rapidly changing numbers of partitions are of interest. Knowledge of such points may be beneficial when automating parameter choice for such indexes. The sets of partitions that can be identified using Technique 6 (local backward bisimilarity) are a subset of the sets of partitions that can be identified using Technique 7 (forward and backward bisimilarity). Thus, only the results obtained using Technique 7 are given here.

Figures 10 and 11 show the numbers of partitions obtained using a parameterised index for tree depths of 1 and 3. The results are shown as 3D-surfaces to allow the interaction of the two parameters that control the size of the forward and backward bisimilarity to be seen. For any given parameter combination (i, j), the set of partitions identifiable will include all partitions, or a refinement thereof, that are identifiable using a lower value of i and/or j. Thus, as the bisimilarity is extended in either direction, the number of identifiable partitions increases until such a point is encountered when all the partitions identifiable using this technique have been encountered. The maximal number of partitions that can be identified using such indexes is equal to the number of partitions identified by the F&B index [5].

Two trends emerge from the results presented in Figures 10 and 11. Firstly, it is possible identify a maximal set of partitions using relatively low values for the three parameters that define the index. Secondly, the rate at which we approach this maximal number of partitions increases with tree depth. For the XMark document, it was found that the

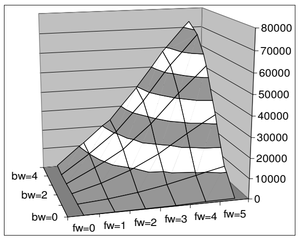

Figure 12. Pubmed: Depth $= 1$

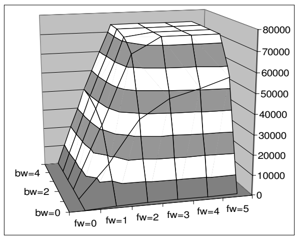

Figure 13. Pubmed: Depth $= 3$

maximal number of partitions that could be identified using this technique was very close to the number of vertices on the data graph: 285,372 partitions compared to 322,327 vertices on the data graph.

Figures 12 and 13 show the number of parameters that can be identified in the PubMed file using parameterised indexes of tree depths 1 and 3. In each case, the overall shape of the surfaces is similar to that obtained when using XMark data, although the surfaces are smoother and more closely resemble that which would be obtained over normally distributed data. These smoother surfaces indicate that the structure of the PubMed data is more regular than that of XMark. The most significant difference in the data gathered over PubMed is that the maximal number of partitions identified is significantly smaller than the size of the original data graph: 73,874 partitions compared to a data graph of 1,288,555 vertices. Here, the full F&B index would be an order of magnitude smaller than the original data graph. Again, it can be seen that increasing the tree depth increases the rate at which the number of partitions grows.

Overall, it can be seen that even restricted instances of parameterised covering indexes have a complexity similar to that of the full F&B index [5]. Thus, if it is intended to provide a partitioning (index graph) that is smaller than that obtained with F&B index then the parameters for the parameterised index can only be drawn from a limited range. When compared to the size of the data graph, the maximal number of partitions identified over our two sample documents differed considerably. For XMark the maximal number of partitions approaches the size of the data graph, whereas for PubMed the maximal number of partitions was an order of magnitude smaller than the size of the data graph. The input data determines whether it is viable to use higher values for the index parameters.

5 Conclusions

This paper summarised seven different techniques for partitioning semistructured data, contrasting their effectiveness over various XML documents. As expected, the number of identifiable partitions varied greatly with choice of partitioning technique; however, it was also found the number of partitions varied greatly with the nature of the source data. Partitioning convoluted data sources, such as XMark data, can result in an index graph that approaches the complexity of the data graph. However, this was not found to be the case for real sources of semistructured data. Hence, index graphs that potentially have a complexity equal to that of the data graph may well be viable over real data sources.

Of the seven techniques described, those based upon bisimulation (Techniques 5, 6 and 7) identified significantly more partitions for data with varying structure than the more simplistic partitioning techniques. Furthermore, employing both forward and backward bisimulation (Technique 7) allowed significantly more partitions to be identified than is possible using bisimulation in only one direction (e.g. Skeleton Partitions). This suggests that partitioning on forward and backward bisimulation is the most appropriate technique for both accelerating branching path expressions and for clustering the atomic data according to context. Additionally, it was demonstrated that the number of partitions identified using parameterised indexes quickly approaches the maximum as the parameters controlling the forward and backward bisimilarity are increased. Hence, in situations were it is desired to limit the size of the index graph (and the number of partitions) then it will be necessary to draw values for these parameters from a restricted set of values.

References

[1] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimisation techniques for unstructured data. In *Proc. of the 1996 ACM SIGMOD*, pages 505–516, 1996.

[2] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed XML. In *Proc. of 29th VLDB*, pages 141–152, 2003.

[3] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proc. of 23rd VLDB*, pages 436–445. Morgan Kaufmann, 1997.

[4] H. He and J. Yang. Multiresolution indexing of XML for frequent queries. In *Proc. of 20th ICDE*, pages 683–694. IEEE Computer Society, 2004.

[5] R. Kaushik, P. Bohannon, J. Naughton, and H. Korth. Covering indexes for branching path queries. In *Proc. of the 2002 ACM SIGMOD*, pages 133–144, 2002.

[6] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Proc. of the 18th ICDE*, pages 129–140, 2002.

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