Proposal to the EPSRC

Vectorised XML?

Peter Buneman
Division of Informatics
University of Edinburgh

June 12, 2002

Contents

PART 1: Previous Research and Track Record

Peter Buneman has recently joined the University of Edinburgh as Professor of Database Systems, a topic in which he has worked for the past 20 years or more, mostly at the University of Pennsylvania. In a previous incarnation he worked at Edinburgh on a number of topics related to Cognitive Science and also on the mathematics of phylogeny. His work in databases covers a number of areas including query languages, data models, data integration, partial information, semistructured data and data provenance. He has also worked on a variety of issues in scientific databases, He has served on the programme committees for most major database conferences, he has been programme chair of several of them including ACM Sigmod, ACM Pods and ICDT. He has served on the editorial board of a number of database journals and is currently an associate editor for ACM Transactions on Database Systems. He is a Fellow of the ACM and a recipient of a Royal Society Wolfson Merit Award.

Those aspects of Buneman's work connected with this proposal are briefly described below. They are query languages, semistructured data and data archiving.

In the early 1990's Buneman and his colleagues worked on the principles of programming with collection types. A relational table is a *set* of tuples whose components are all taken from some *atomic* domain. The challenge was to find the "right" extension to relational algebra that would provide for new collection types other than sets (lists and multisets) and would relax the atomic value constraint so that entries in tables could, for example, be other tables. They were able to show that a language based on the categorical notion of a monad provided the generalisation that was needed. That it was the "right" generalisation was established through a series of complexity and conservativity results. A practical language was implemented and used in various bioinformatic data integration problems. It has also been used as a core language for a typed XML algebra.

Following this Buneman worked on semistructured data, laying down a data model, a query language and optimisation techniques for this form of untyped data. He is co-author of a book on this topic. He has also worked on various forms of constraint for semistructured data including path constraints, partial type systems for semistructured data, and key constraints for XML.

Buneman's most recent work has focused on the problem of *data provenance* – understanding where data has come from. With the proliferation of databases on the Web, the provenance of data is becoming increasingly difficult to record or even to characterise. In bioinformatics, for example, there are literally hundreds of public databases. Most of these are not source data: their contents have been extracted from other databases by a process of filtering, transforming, and manual correction and annotation. Thus, describing the provenance of some piece of data is a complex issue. These "curated" databases have enormous added value, yet they usually fail to keep an adequate description of the provenance of the data they contain.

A key requirement in recording provenance is the keeping of archives of all states of a database. Archiving is particularly important for scientific databases, where the data evolves, but – in the interests of accuracy and verifiability of findings derived from the data – it is important to keep all previous versions of the data. Buneman and colleagues have developed a technique based on the notion of timestamps whereby an element appearing in multiple versions of the database is stored only once along with a compact description of versions in which it appears. A prototype has been developed with encouraging results. For two widely used databases in bioinformatics, one can archive as often as one wants for a whole year, and the size of the archive is less than 15% larger than the size of any one of the versions. Moreover an individual version can be efficiently extracted. Querying an archive is one of the goals of this proposal.

Further information on these projects and publications see `http://db.cis.upenn.edu`

The Division of Informatics of the University of Edinburgh is one of the top-ranked computer science departments in the UK. The Division is committed to building up a world-class group in databases, and it is ideally situated to do this.

- Two additional posts (up to reader level) in databases have been allocated. One of these is expected to be filled by the end of the year.
- Martin Grohe, a world-class researcher in finite model theory has recently joined the division. Grohe has made several contributions to the principles of databases. He has been an invited speaker at ACM Principles of Database Systems (PODS).
- Kousha Etessami will also join the Division in the coming months. He has also contributed to database theory.
- Christoph Koch (see below) will be joining as a post-doctoral student in July.

- Edinburgh is host to the National e-Science Center (NeSC), directed by Malcolm Atkinson, who has collaborated with Buneman. One of the main activities of the Center is to promote interest in Scientific Databases.
- Through NeSC, Buneman has already formed contacts with people – some of them involved in this proposal – working in biological and astronomical databases. There is a rapidly growing group in bioinformatics at Edinburgh.
- Within Edinburgh there are other possibilities for collaboration, notably Werner Nutt at Heriot Watt who is a reader in Database Systems and Richard Baldock at the Western General MRC who has led the Mouse Atlas project.

The purpose of this grant is to support a postdoc and general research requirements (travel and equipment). Christoph Koch, who is arriving in Edinburgh in July 2002, is the primary candidate for the postdoc. Short-term funds are available for this position, but longer-term funding will be needed. Koch has applied to other funding agencies, and if his applications are successful, the funds requested in this proposal would be used for a second postdoctoral researcher or a research assistant.

Christoph Koch received his MS and PhD degrees in Computer Science from TU Vienna. He has several years of experience in the IT industry and most recently, while doing his PhD research on the integration of scientific databases, had a three years stay at the European Organisation for Nuclear Research (CERN), Geneva, Switzerland. He has written about 20 publications on database and Artificial Intelligence topics and coauthored work that received the best paper award at PODS, the most prestigious database theory conference, in 2002. His current research interests are on database theory, particularly on data extraction and integration, XML and Web data management, and scientific databases.

Named as CO-PI and collaborator on this proposal are J.D. Armstrong and Robert Mann. Armstrong has been involved in the development of database systems and algorithms for handling DNA sequence information, mutation data and customised laboratory information management systems. Armstrong was recently been appointed to the Division of Informatics at the University of Edinburgh and teaches Algorithms and Biological Data. Mann is a postdoctoral researcher at the Edinburgh University Institute for Astronomy. His work involves mining astronomical data, which contains large datasets represented in XML.

Currently held grants

All of Buneman's current grants are all held in the USA at the University of Pennsylvania (Penn). He is active and former PI on two grants "Data Provenance" (Digital Libraries II, funded by NSF, DARPA, NLM, LoC, NEH, NASA) $500k, and "A Deterministic Model for Semistructured Data" (NSF) $300k. He is a co-PI on a number of grants involving scientific databases including a grant from Glaxo Smith Kline on data integration (formerly PI, approx $1m) and an NSF IT grant for linguistic databases, the TalkBank project held at Penn and CMU (approx $4m.)

Budget Justification

The proposal is to support one post-doc and tarvel and equipment for both the post-doc and the investigators and collaborators on this proposal. The funding for the post-doc is requested at a higher rate commensurate with the academic standing and industrial experience of Christoph Koch (see above) who has spent some time in industry and scientific establishments and has already been working for some time as a post-doc.

Equipment and Travel For experimental work, especially with astronomical data, a machine with substantial main memory will be required. However it is anticipated that such a machine will be available through independent funds already avaailable to provide infrastructure in support if e-science. The equipment requested here is therefore the standard for any proposal in computer science. The travel is for participation by the PI, Co-PI, collaborators and supported postdoc in the major database conferences, most of which are in Europe or the USA, but only infrequently in the UK. These include ACM SIGMOD/PODS (USA) VLDB (Europe/USA) ICDT (Europe) ICDE (Europe/USA) SSDBM (Europe/USA) EDBT (Europe.)

PART 2: Description of the Proposed Research and its Context

1 Introduction

The received wisdom on storing tables in a relational database is to store each tuple contiguously in secondary storage. A simple alternative is to store the columns contiguously, so that a table is represented as a set of *vectors* all of the same length. The advantage of vectorisation is that queries involving a small number of attributes can be evaluated efficiently and that it is often possible to do more main-memory query processing. What is asked here is can we "vectorise" XML? This would provide similar advantages for storage and query evaluation, especially for archives and large bodies scientific data, much of which is now transmitted in XML.

2 Background

The widely accepted storage technique for relational databases is to store each tuple contiguously. An alternative, which is almost as old as relational databases [3] is *vertical partitioning* in which one stores the columns contiguously. The benefit is that queries that only involve a small number of columns require less i/o. Moreover, there are dramatic performance improvements to be made if the columns can be stored and manipulated as vectors in main memory. The idea has re-emerged in various places: in [9, 4] for object-oriented databases and in [1] in which it is used to speed up transfer between main memory caches. It has also been used commercially in Sybase IQ and recently by Aleri Inc. where it is combined with vector processing language technology and has been successfully used in a number of financial database applications to support data integration and OLAP. There are, of course, drawbacks to the idea. Updates, especially deletions, are prohibitively expensive if naively implemented. Also, there is a sense in which vertical partitioning is already widely used: an index typically stores information about a small subset of columns contiguously. We shall use the term "vectorising" for that form of vertical partitioning which stores the columns of a table as vectors and, in particular, preserves the *order* in each column.

At first sight, vectorising XML, which is not tabular in nature and can be highly irregular in structure, does not make sense. But there is some recent work by Liefke and Suciu [17] that demonstrates the feasibility of the idea for the purpose of compression. What is proposed here it to use this decomposition and variations on it for the purpose of *querying and programming* with XML data. There are a number of potential benefits from such a storage technique:

- It is a robust form of native storage for XML that supports programming interfaces and efficient evaluation of a wide variety of basic queries.
- It can co-exist with other indexing and storage techniques.
- The relatively high cost of updates in vectorised databases is less important for XML which one does not usually think of as a storage model for databases with high transaction rates. It is certainly the case that scientific databases are "write-mostly".
- It preserves *order*. Many existing storage models and query languages "lose" the order in an XML document, however XML is intrinsically a representation for ordered data and order is important in many scientific applications [18].

In case the advantages sound too good to be true, the author should strike a note of caution. There are a number of competing proprietary and academic storage schemes for XML, and it is not clear whether one could win out with a new one funded on a small budget. Nevertheless, the idea of vectorisation brings up enough interesting challenges that it is very likely that some useful research and techniques will emerge from its study. Our starting point is briefly to describe what we mean by vectorising XML.

3 The Liefke-Suciu decomposition

XML is a form of semistructured data that represents a middle-ground between structured text and databases. That it will serve as a universal medium for data exchange is not in doubt. Whether we shall store large XML databases (as opposed to using conventional databases and using XML just for data exchange) depends on our ability to find XML storage models and query languages that are matched to those models. For example, if XML algebra [12] is to perform as effectively as relational algebra, we shall need a corresponding storage model and implementations of the fundamental operations.

XML is a textual representation of a labeled tree in which the internal nodes are labeled with

```
⟨recipe-book⟩
    ⟨recipe⟩
        ⟨contributor⟩ Annie ⟨/contributor⟩
        ⟨name⟩ Salsa ⟨/name⟩
        ⟨comment⟩
            Peter ⟨i⟩ loves ⟨/i⟩ this stuff
        ⟨/comment⟩
        ⟨ingredient⟩
            ⟨name⟩ tomatoes ⟨/name⟩ ⟨qty⟩ 1kg ⟨/qty⟩
        ⟨/ingredient⟩
        ⟨ingredient⟩
            ⟨name⟩ onions ⟨/name⟩ ⟨qty⟩ 200g ⟨/qty⟩
        ⟨/ingredient⟩
    ⟨/recipe⟩
```

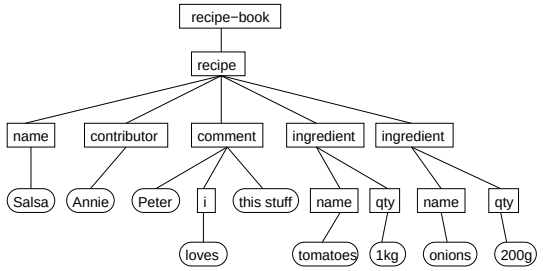
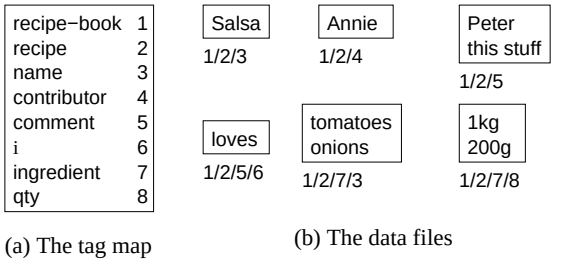


Figure 1: Some XML and its representation as a tree

tags (corresponding roughly to table and attribute names) and terminal nodes contain text (corresponding to data).

The Liefke-Suciu (LS) decomposition takes an XML file (Figure 1) and produces a collection of output files (Figure 2). These consist of (a) a *tag map* file that encodes the XML tags, (b) a collection of *data files* that contain the text at the terminal nodes of the tree, and (c) a *skeleton* file that describes the tree structure of the XML source. The XML tags are encoded as integers simply to save space. The data files are named by the paths at which the data is to be found and are shown, for illustration, with each segment of text on a separate line. The skeleton file is a compact description of the XML tree without the character data. It can be readily understood as a left-first or document-order walk round the tree. Each time a tagged node is encountered we place its code in the skeleton file. If we encounter a character data node, we put a "#" in the skeleton file and we write the character data into a file whose name is the current path to that node. When we leave a tag node for the last time we put a "+" in the skeleton file.

This does not complete the description of the LS technique, which was designed for data compression. Paths may be grouped into containers (e.g. all paths ending in name) and the data files correspond to containers. The containers are then individually compressed using a variety of compression algorithms. We have also swept under



(a) The tag map

(b) The data files

(c) The skeleton

Figure 2: Files produced by the Liefke-Suciu decomposition

the carpet various details such as attributes, entities and white space.

The analogy with vectorisation should be clear. The "vectors" or columns correspond to the data files. The difference is that we need to carry the skeleton in order to reconstruct the original XML or to do any processing with it. What we consider in the rest of this proposal paper is how we could make the LS decomposition – or variations on it – to make querying and programming XML more efficient.

4 Uses of vectorisation

4.1 SAX-style parsing

SAX (A simple API for XML) [21] is the basic parser for XML. It performs a single traversal of the XML source and as it does so calls procedures such as StartElement when an opening tag is encountered, EndElement, for a closing tag, and Characters when it is reading character data. Even before we consider main-memory processing, we can use the LS decomposition to implement efficient SAX traversals.

A SAX interface It is easy to construct a SAX interface that works directly of the LS decomposition. In fact this is roughly what one does if one wants to reconstruct the original XML. The author wrote a naive (non-validating) SAX interface in Python and was able to obtain a five-fold increase in speed over a Python standard [20] parser on a program that simply traversed the data and reconstructed the XML file. This is really not surprising, since most of the work was done by the standard parser when the XML was decomposed, and the data is being stored in "parsed" form. Given this, and the benefits of LS compression, a full SAX interface to the compressed data seems like a good idea.

	Source	Lazy	Skeleton
Baseball	672	106	85
Shakespeare	7646	561	539

Uncompressed

	Source	Lazy	Skeleton
Baseball	66	11	1.3
Shakespeare	2139	40	31.0

Compressed

Figure 3: File for some simple queries (sizes in kb)

Lazy SAX parsing An almost immediate consequence of LS compression is that one can build a "lazy" SAX interface[1]. Rather than opening all the data files when the interface is initialised, as the traversal proceeds one keeps a count, for each data file, of how many items should have been read from each data file. It is only when the user program wants some data that the appropriate file is opened and the reader "catches up" to produce that data. Thus files that contain data that is never needed are never read. For example (Figure 1) a query that asks for the names of the recipes that contain onions will only read the data files 1/2/4 and 1/2/7/4. We noted in the introduction that vectorisation is particularly useful for "fat" tables. XML has a hierarchical structure, which means that many-one relationships can be stored in a single hierarchy; also, in forcing data into a single hierarchy, designers often resort to un-normalised representations. What this means is that XML documents are typically very "fat", and the benefits of vectorisation are exceptional.

As examples of the benefits of this technique the author tried two simple experiments with XML data available on the web. The first was to find the names and teams of all players with an earned run average greater than 8.0 from baseball statistics [14]; the second was to find the titles of all Shakespeare plays in which Falstaff appears as a player. The Shakespeare XML was taken from [5], and all the plays were run together as a single XML document.

The results are shown in Figure 3, which shows the sizes of the source data, the total size of the files used by "lazy" SAX parsing and the size of the skeleton file. While there is a substantial reduction in i/o, the skeleton accounts for most of the i/o used by lazy parsing. On the "compressed" side we see the corresponding file sizes when all files are compressed using the standard gzip program [23]. Here the skeleton file reduces disproportionally. This is partly due to the relatively crude representation used in the author's implementation (2 bytes for an opening tag, one byte for a closing tag, with the "presence of character data flag" being absorbed into the preceding tag byte(s).) However, a more important reason is the regularity of these files. This is something we shall discuss later. The Shakespeare example might be regarded as unfair because the query does not mention the text of play (the spoken lines) which accounts for 60% of the total source data. However, by analogy with scientific databases – treating the text as "data" and the annotations as "metadata" – this query is typical of exploratory queries that do not mention the "data".

As further evidence that the principle works, the figures in table 4 were very recently obtained by Byron Choi, a research student at the university of Pennsylvania. They are taken against an XML version of the Eupoean Molecular Biology Database and compare XPath queries performed by Flat file (FF) Vectorising (Vect) Edge Table(ET) and Inlining (Inl). The last two [13, 1] are relational storage techniques for XML implemented in DB2. The queries were 1. `/` – retrieve the whole file in XML,
2. `/Entries/Entry/Identification` and
3. `/Entries/Entry/Identification/EntryName`,
all "hand coded". The skeleton (s) was 16K and the tag map (i) was negligible. Only queries 2 and 3 had "tabular" answers and were encoded as relational queries. Reconstruction of XML – needed for 1 – is not expressible in relational algebra and was not implemented. It should be stressed that these results are preliminary. All of them will benefit from indexing and tuning.

4.2 Indexing

The LS decomposition invites numerous possibilities for indexing. It is clear that keeping pointers (byte offsets) from the data files into the skeleton file and *vice versa* may reduce i/o in finding, say, a data element associated with a node in the skeleton. At the same time, insertion of pointers will enlarge the size of the files (especially the skeleton) so the benefits of this are unclear – especially when we may want to keep a file in main memory. A middle ground is to introduce *synchronisation marks* into the file. Such a mark is a number that is placed somewhere in the skeleton, and the same number is placed at corresponding positions in the data files. A main-memory array is kept that gives the offsets of each synchronisation mark in the skeleton and each data file.

[1] Because of the way the SAX interface is specified, without playing some extremely dirty ticks, one cannot build a lazy interface that conforms exactly to the specification, but the modification is extremely small

Now, in almost any kind of searching or indexing we establish a location in one of the data files. We scan backwards for a synchronisation mark, use the synchronisation table to index into the corresponding marks in the skeleton and other needed data files, and scan forwards. To do this we need to augment either the skeleton or the synchronisation table with enough information to determine the path at each mark. It is up to us to choose the density of synchronisation marks: we need to keep enough so that scanning does not take long (and backward scanning can happen within the input buffers) but not so much that the synchronisation table cannot fit into main memory. The same technique can also be used to connect the skeleton and data files with a redundant copy of the source XML file.

Another use of the LS decomposition and synchronisation marks is for streaming XML. One can imagine a scenario in which XML is transmitted not as a single file but by sending the skeleton and data files through concurrent channels and uses synchronisation marks for re-combining these files. This in turn offers a number of possibilities for distributed query optimisation on streaming XML.

	FF *(sec, io)*	*Vect* *(sec, io)*	*ET* *(sec, io)*	*Inl* *(sec, io)*
1	5.09s, 45.6M	5.00s, 20.5M + s + i	–	–
1	3.54s, 45.6M	1.40s, 320K + s + i	11.35s, 90.8M	1.74s, 440K
2	3.43s, 45.6M	1.19s, 95K + s + i	8.65s, 90.8M	1.45s, 184K

Figure 4: Some initial comparisons against EMBL

4.3 Exploiting Structure

All the techniques and proposals discussed so far are independent of any kind of imposed structure (DTD, XML Schema) on the data. What benefits are there to having some form of type? We should remark in passing that validating LS-decomposed XML is extremely efficient. One might think that having a DTD would provide efficient storage of the skeleton, but this is not always the case. For example, one could encode the Baseball skeleton by encoding the sequence of choices in the DFAs for each regular expression in the DTD. This is *not* as efficient as LZ77 encoding. The latter finds regularities (e.g. that players are typically either pitchers or batters) in the data that are not expressed in the DTD. Having a rich catalogue of base types, as in XML Schema, is clearly advantageous.

However there are places in which we certainly can exploit structure. One such is in the representation of (multi-dimensional) arrays. First note that if the XML representation of a array is a sequence of sequence of ..., then the data file will contain the array in the form that is usual in main-memory processing. But we can also compress the skeleton by simply recording the tags with their dimensions. XML is not typically thought of as a medium for expression or even transmission of array data, but vectorisation allows us to extend its functionality.

4.4 Ordered Data

Some, but not all, XML storage schemes lose the (horizontal) order in an XML tree [11, 13, 22]. At the same time some, but not all, query languages [8, 10] lose order – at least in complex queries and in the name of efficiency. These languages and storage schemes are well matched, but they are inappropriate for textual data, and they are inappropriate for scientific data where it is frequently important to store and query the sequential structure of data (time series, arrays etc.)

One of the main goals of this project is to investigate XML programming and query languages for ordered data: their implementation against the vectorised scheme we have described here and their optimisation. The author is aware of two such languages. The first, XDuce [15], is a full (Turing-complete) programming language; but it should be possible either to augment it with or to recognize (in the code) the application of list-like operators such as *map* and *fold* about which some optimisation techniques are understood [7]. XDuce is typed, which means we may statically determine which data files will be needed to support a program.

A second language is XQuery. Most implementations of XQuery are untyped, and XQuery itself relies on XPath [8], which may not respect order. However recent work on an algebra for XQuery suggests that there may emerge some "superset of a subset" of XQuery which is statically typed and is order-sensitive. Again, we would want to experiment with, and perhaps influence the design of, this language.

Another long-term challenge is to combine array processing with database query languages. Ideas on how to do this surface from time to time [19, 16], but they have not come to market perhaps because there is no physical representation that will

efficiently support both paradigms. Will vectorisation offer a solution? Another approach would be to take a vector processing language (APL and its derivatives are widely used on Wall street) and extend it with primitives for XML-style tree traversal.

5 Applications

It has been claimed that this decomposition of XML may be particularly suitable for scientific databases. The author is in contact with two areas in which this idea may work. The first is Astronomy. Quoting from an astronomer[2] the obstacles include: "forcing a complex structure into a set of tables so we can use an RDBMS" and "Lack of suitable query languages. SQL can't do what we want." Indeed the astronomers (and physicists) sometimes make use of column storage, but it is done in an *ad hoc* fashion. Why not an XML query language or an XML query language with some suitable vector-processing operations embedded in it? Robert Mann (colaborator on this proposal) points out that there is a new XML standard for exchange of tabular data in astronomy `http://www.us-vo.org/VOTable/`. This raises the exciting possibility that an astronomer could perform *ad hoc* queries – using the techniques described above – directly on this data without having first to go through the pain of constructing, loading and indexing a database.

In bioinformatics the story is similar. The difference is that size is less of an issue, but the structure can be more complex, and it can undergo frequent changes. Again we believe a semistructured approach that provides the ability to do the moral equivalent of adding columns may help. Douglas Armstrong (co-PI) has a number of projects, (Flytrap, Fly-brain [2]) that employ structured files or relational databases that suffer from the limitations described above. Typical datasets often include time series and multidimensional arrays. In addition to having a biologist on board, Armstrong's systems will also provide suitable test data for the project. The data analyses involve supervised machine learning techniques taking into consideration biological context and algorithms for designing optimal RNAi constructs.

A final application is in scientific archiving. The author has recently been involved in work to achieve efficient compression of all the previous states of a scientific database [6]. The prime function of an archive is to recover efficiently any of the previous database states. However one might want to query the temporal history of a part of the database. The technique interacts extremely well with LS decomposition. It may therefore be a better storage medium for querying archival data. In Figure 5 the central line shows the size of successive versions of the Swissprot database over a five-year period. The top line is the size of the archive file, which contains all previous versions. At the end of that period it is less than twice the size of the last version. The archive file is still an XML file and can be compressed using LS. The bottom line shows how effective that compression is. The other lines show, for comparison, how the new technique compares with a conventional "diff"-based technique. We have good reason to believe that the LS-decomposed file will support efficient temporal queries over the data, e.g. "when did this entry last change?"

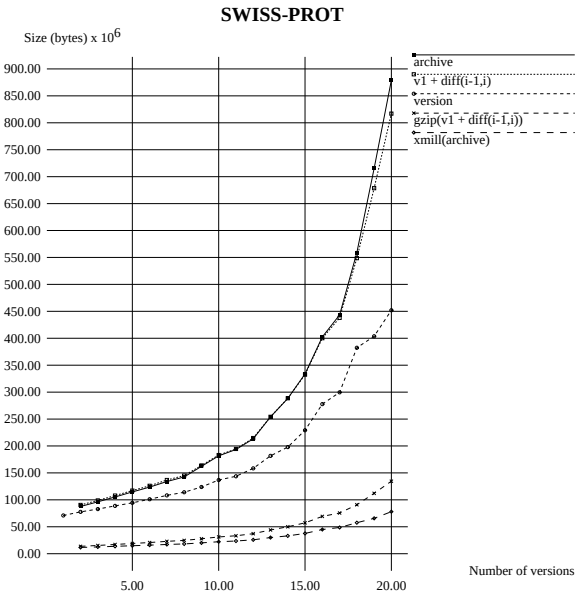


Figure 5: Archiving Swissprot

Work plan

As in any scientific investigation, the likelihood of sticking to a precise plan is low. Here, an idea of how the project should progress over a three year period.

Year 1. Implementation of the basic technique with indexing. Implementation of a rudimentary query language and Evaluation on some scientific data sets with some "hand coded" queries. Release of a prototype.

[2] Clive Page, personal communication

Year 2. Design and implementation of a query language with vector processing operations. Build interfaces for typed languages (e.g. XDuce) and investigate optimisation.

Year 3. Evaluation of query language work. If successful, generate a "product" consisting of database, API and query language. Further evaluation on scientific data sets.

Dissemination. This will be via the normal channels of publications and demonstrations at the major conferences and through the distribution of prototypes.

References

[1] A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis. Weaving relations for cache performance. In *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[2] J. D. Armstrong, K. Kaiser, A. Müller, K.-F. Fischbach, N. Merchant, and N. J. Strausfeld. Flybrain - an atlas and database of the drosophila nervous system. *Neuron*, 15:17–20, 1995.

[3] Don S. Batory. On searching transposed files. *TODS*, 4(4):531–544, 1979.

[4] Peter A. Boncz, Annita N. Wilschut, and Martin L. Kersten. Flattening an object algebra to provide performance. In *ICDE*, pages 568–577, 1998.

[5] Jon Bosak. Shakespeare in XML. `ibiblio.org/xml/examples/shakespeare/`.

[6] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. Archiving scientific data. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2002. To appear.

[7] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of Programming with Collection Types. *Theoretical Computer Science*, 149:3–48, 1995.

[8] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[9] George P. Copeland and Setrag Khoshafian. A decomposition storage model. In Shamkant B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, pages 268–279. ACM Press, 1985.

[10] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, August 1998.

[11] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. "storing semistructured data with stored". In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 431–442. ACM Press, 1999.

[12] Denise Draper, Peter Fankhauser, Mary Fernndez, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. Xquery 1.0 formal semantics. `http://www.w3.org/TR/2002/WD-query-semantics-20020326/` .

[13] Daniela Florescu and Donald Kossmann. Storing and querying xml data using an rdmbs. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[14] Elliotte Rusty Harold. Complete 1998 Major League Statistics in XML. `http://www.ibiblio.org/xml/examples/baseball/`.

[15] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language (preliminary report). In *WebDB (Informal Proceedings)*, pages 111–116, 2000.

[16] Leonid Libkin, Rona Machlin, and Limsoon Wong. A query language for multidimensional arrays: design, implementation, and optimization techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pages 228–239, 1996.

[17] Hartmut Liefke and Dan Suciu. XMill: an efficient compressor for XML data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 153–164, 2000.

[18] D. Maier and B. Vance. A Call to Order. In *Proceedings of the Twelfth ACM Symposium on Principles of Databases Systems*, pages 1–16, 1993.

[19] Peter Buneman. The Fast Fourier Transform as a Database Query. Technical Report MS-CIS-93-37, University of Pennsylvania, 1993.

[20] SIG for XML Processing in Python. `www.python.org/sigs/xml-sig/`.

[21] SAX: The Simple API for XML, version 2. `www.megginson.com/SAX`.

[22] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *The VLDB Journal*, pages 302–314, 1999.

[23] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inform. Theory*, 23(3):337–343, May 1977.