Data Provenance: Some Basic Issues

Peter Buneman, Sanjeev Khanna and Wang-Chiew Tan

University of Pennsylvania

Abstract. The ease with which one can copy and transform data on the Web, has made it increasingly difficult to determine the origins of a piece of data. We use the term *data provenance* to refer to the process of tracing and recording the origins of data and its movement between databases. Provenance is now an acute issue in scientific databases where it central to the validation of data. In this paper we discuss some of the technical issues that have emerged in an initial exploration of the topic.

1 Introduction

When you find some data on the Web, do you have any information about how it got there? It is quite possible that it was copied from somewhere else on the Web, which, in turn may have also been copied; and in this process it may well have been transformed and edited. Of course, when we are looking for a best buy, a news story, or a movie rating, we know that what we are getting may be inaccurate, and we have learned not to put too much faith in what we extract from the Web. However, if you are a scientist, or any kind of scholar, you would like to have confidence in the accuracy and timeliness of the data that you are working with. In particular, you would like to know how it got there.

In its brief existence, the Web has completely changed the way in which data is circulated. We have moved very rapidly from a world of paper documents to a world of on-line documents and databases. In particular, this is having a profound effect on how scientific research is conducted. Let us list some aspects of this transformation:

- A paper document is essentially unmodifiable. To "change" it one issues a new edition, and this is a costly and slow process. On-line documents, by contrast, can be (and often are) frequently updated.
- On-line documents are often databases, which means that they have explicit structure. The development of XML has blurred the distinction between documents and databases.
- On-line documents/databases typically contain data extracted from other documents/databases through the use of query languages or "screen-scrapers".

Among the sciences, the field of Molecular Biology is possibly one of the most sophisticated consumers of modern database technology and has generated a wealth of new database issues [15]. A substantial fraction of research in genetics is conducted in "dry" laboratories using *in silico* experiments – analysis

of data in the available databases. Figure 1 shows how data flows through a very small fraction of the available molecular biology databases[1]. In all but one case, there is a *Lit* – for literature – input to a a database indicating that this is database is *curated*. The database is not simply obtained by a database query or by on-line submission, but involves human intervention in the form of additional classification, annotation and error correction. An interesting property of this flow diagram is that there is a cycle in it. This does not mean that there is perpetual loop of possibly inaccurate data flowing through the system (though this might happen); it means that the two databases overlap in some area and borrow on the expertise of their respective curators. The point is that it may now be very difficult to determine where a specific piece of data comes from. We use the term *data provenance* broadly to refer to a description of the origins of a piece of data and the process by which it arrived in a database. Most implementors and curators of scientific databases would like to record provenance, but current database technology does not provide much help in this process for databases are typically rather rigid structures and do not allow the kinds of *ad hoc* annotations that are often needed for recording provenance.

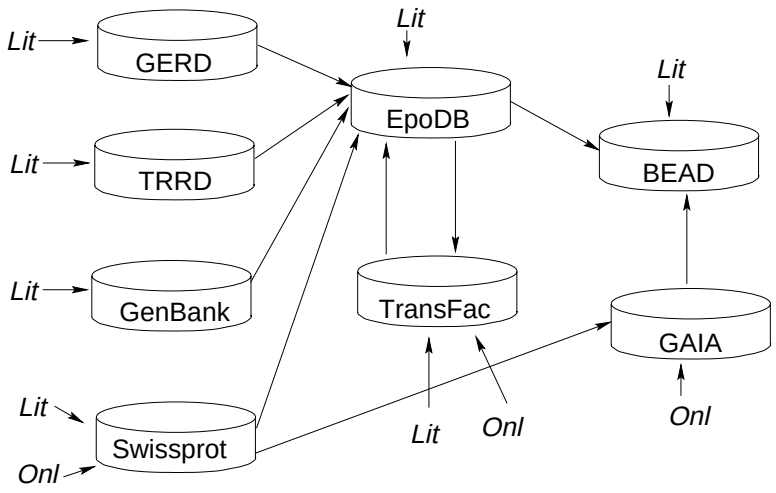

Fig. 1. The Flow of Data in Bioinformatics

The databases used in molecular biology form just one example of why data provenance is an important issue. There are other areas in which it is equally acute [5]. It is an issue that is certainly broader than computer science, with legal and ethical aspects. The question that computer scientists, especially theoretical computer scientists, may want to ask is what are the technical issues involved

[1] Thanks to Susan Davidson, Chris Stoeckert and Fidel Salas of the Bioinformatics Center at Penn for providing this information.

in the study of data provenance. As in most areas of computer science, the hard part is to formulate the problem in a concise and applicable fashion. Once that is done, it often happens that interesting technical problems emerge. This abstract reviews some of the technical issues that have emerged in an initial exploration.

2 Computing Provenance: Query Inversion

Perhaps the only area of data provenance to receive any substantial attention is that of provenance of data obtained via query operations on some input databases. Even in this restricted setting, a formalization of the notion of data provenance turns out to be a challenging problem. Specifically, given a tuple t in the output of a database query Q applied on some source data D, we want to understand which tuples in D contributed to the output tuple t, and if there is a compact mechanism for identifying these input tuples. A natural approach is to generate a new query Q', determined by Q, D and t, such that when the query Q' is applied to D, it generates a collection of input tuples that "contributed to" the output tuple t. In other words, we would like to identify the provenance by *inverting* the original query. Of course, we have to ask what we mean by contributed to? This problem has been studied under various names including "data pedigree" and "data lineage" in [1, 9, 7]. One way we might answer this question is to say that a tuple in the input database "contributes to" an output tuple if changing the input tuple causes the output tuple to change or to disappear from the output. This definition breaks down on the simplest queries (a projection or union). A better approach is to use a simple proof-theoretic definition. If we are dealing with queries that are expressible in positive relational algebra (SPJU) or more generally in positive datalog, we can say that an input tuple (a fact) "contributes to" an output tuple if it is used in *some* minimal derivation of that tuple. This simple definition works well, and has the expected properties: it is invariant under query rewriting, and it is compositional in the expected way. Unfortunately, these desirable properties break down in the presence of negation or any form of aggregation. To see this consider a simple SQL query:

```
SELECT name, telephone
FROM employee
WHERE salary > SELECT AVERAGE salary FROM employee
```

Here, modifying any tuple in the `employee` relation could affect the presence of any given output tuple. Indeed, for this query, the definition of "contributes to" given in [9] makes the whole of the `employee` relation contribute to each tuple in the output. While this is a perfectly reasonable definition, the properties of invariance under query rewriting and compositionality break down, indicating that a more sophisticated definition may be needed.

Before going further it is worth remarking that this characterization of provenance is related to the topics of truth maintenance [10] and view maintenance [12]. The problem in view maintenance is as follows. Suppose a database (a view) is generated by an expensive query on some other database. When the source

database changes, we would like to recompute the view without recomputing the whole query. Truth maintenance is the same problem in the terminology of deductive systems. What may make query inversion simpler is that we are only interested in what is in the database; we are not interested in updates that would add tuples to the database.

In [7] another notion of provenance is introduced. Consider the SQL query above, and suppose we see the tuple `("John Doe", 12345)` in the output. What the previous discussion tells us is *why* that tuple is in the output. However, we might ask an apparently simpler question: given that the tuple appears in the output, *where* does the telephone number `12345` come from? The answer to this seems easy – from the `"John Doe"` tuple in the input. This seems to imply that as long as there is some means of identifying tuples in the `employee` relation, one can compute where-provenance by tracing the variable (that emits `12345`) of the query. However, this intuition is fragile and a general characterization is not obvious as discussed in [7].

We remark that (where) provenance is also related to the view update problem [3]: if `John Doe` decides to change his telephone number at the view, which data should be modified in the `employee` relation? Again, where provenance seems simpler because we are only interested in what is in the view and not in what is not.

Another issue in query inversion is to capture other query languages and other data models. For example, we would like to describe the problem in object-oriented [11] or semistructured data models [2] (XML). What makes these models interesting is that we are no longer operating at the fixed level of tuples in the relational model. We may want to ask for the why- or where-provenance of some deeply nested component of some structure. To this end, [7] studies the issue of data provenance in a "deterministic" model of semistructured data in which every element has a canonical path or identifier. Work on view maintainence based on this model has also been studied in [14]. This leads us to our next topics, those of citing and archiving data.

3 Data Citation

A digital library is typically a large and heterogeneous collection of on-line documents and databases with sophisticated software for exploring the collection [13]. However many digital libraries are also being organized so that they serve as scholarly resources. This being the case, how do we *cite* a component of a digital library. Surprisingly, this topic has received very little attention. There appear to be no generally useful standards for citations. Well organized databases are constructed with *keys* that allow us uniquely to identify a tuple in a relation. By giving the attribute name we can identify a component of a tuple, so there is usually a canonical path to any component of the database.

How we cite portions of documents, especially XML documents is not so clear. A URL provides us with a universal locator for a document, but how are we to proceed once we are inside the document? Page numbers and line

numbers – if they exist – are friable, and we have to remember that an XML document may now represent a database for which the linear document structure is irrelevant. There are some initial notions of keys in the XML standard [4] and in the XML Schema proposals [16]. In the XML Document Type Descriptor (DTD) one can declare an ID attribute. Values for this attribute are to be unique in the document and can be used to locate elements of the document. However the ID attribute has nothing to do with the structure of the document – it is simply a user-defined identifier.

In XML-Schema the definition of a key relies on XPath [8], a path description language for XML. Roughly speaking a key consists of two paths through the data. The first is a path, for example `Department/Employee`, that describes the set of nodes upon which a key constraint is to be imposed. This is called the *target set*. The second is another path, for example `IdCard/Number` that uniquely identifies nodes in the target set. This second part is called the key path, and the rule is that two distinct nodes in the target set must have different values at the end of their key paths. Apart from some details and the fact that XPath is probably too complex a language for key specification, this definition is quite serviceable, but it does not take into account the hierarchical structure of keys that are common in well-organized databases and documents.

To give an example of what is needed, consider the problem of citing a part of a bible, organized by chapter, book and verse. We might start with the idea that books in the bible are keyed by name, so we use the pair of paths `(Bible/Book, Name)`. We are assuming here that `Bible` is the unique root. Now we may want to indicate that chapters are specified by number, but it would be incorrect to write `(Bible/Book/Chapter, Number)` because this says that that chapter numbers are unique within the bible. Instead we need to specify a *relative key* which consists of a triple, `(Bible/Book, Chapter, Number)`. What this means is that the `(Chapter, Number)` key is to hold at every node specified by by the path `Bible/Book`.

A more detailed description of relative keys is given in [6]. While some basic inference results are known, there is a litany of open questions surrounding them: What are appropriate path languages for the various components of a key? What inference results can be established for these languages? How do we specify foreign keys, and what results hold for them? What interactions are there between keys and DTDs. These are practical questions that will need to be answered if, as we do in databases, use keys as the basis for indexing and query optimization.

4 Archiving and Other Problems Associated with Provenance

Let us suppose that we have a good formulation, or even a standard, for data citation, and that document A cites a (component of a) document B. Whose responsibility is it to maintain the integrity of B? The owner of B may wish to update it, thereby invalidating the citation in A. This is a serious problem in

scientific databases, and what is commonly done is to release successive versions of a database as separate documents. Since one version is – more or less – an extension the previous version, this is wasteful of space and the space overhead limits the rate at which one can release versions. Also, it is difficult when the history of a database is kept in this form to trace the history of components of the database as defined by the key structure. There are a number of open questions :

- Can we compress versions so that it the history of A can be efficiently recorded?
- Should keeping the cited data be the responsibility of A rather than B?
- Should B figure out what is being cited and keep only those portions?

In this context it is worth noting that, when we cite a URL, we hardly ever give a date for the citation. If we did this, at least the person who follows the citation will know whether to question the validity of the citation by comparing it with the timestamp on the URL.

Again, let us suppose that we have an agreed standard for citations and that, rather than computing provenance by query inversion (which is only possible when the data of interest is created by a query,) we decide to annotate each element in the database with one or more citations that describes its provenance. What is the space overhead for doing this? Given that the citations have structure and that the structure of the data will, in part, be related to the structure of the data, one assumes that some form of compression is possible.

Finally, one is tempted to speculate that we may need a completely different model of data exchange and databases to characterize and to capture provenance. One could imagine that data is exchanged in packages that are "self aware"[2] and somehow contain a complete history of how they moved through the system of databases, of how they were constructed, and of how they were changed. The idea is obviously appealing, but whether it can be formulated clearly, let alone be implemented, is an open question.

References

1. A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE*, pages 91–102, 1997.
2. Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web. From Relations to Semistructured Data and XML*. Morgan Kaufman, 2000.
3. T. Barsalou, N. Siambela, A. Keller, and G Wiederhold. Updating relational databases through object-based views. In *Proceedings ACM SIGMOD*, May 1991.
4. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium (W3C), Feb 1998. `http://www.w3.org/TR/REC-xml`.
5. P. Buneman, S. Davidson, M. Liberman, C. Overton, and V. Tannen. Data provenance. `http://db.cis.upenn.edu/~wctan/DataProvenance/precis/index.html`.

[2] A term suggested by David Maier

6. Peter Buneman, Susan Davidson, Carmem Hara, Wenfei Fan, and Wang-Chiew Tan. Keys for XML. Technical report, University of Pennsylvania, 2000. `http://db.cis.upenn.edu`.
7. Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory*, 2001. To appear, available at `http://db.cis.upenn.edu`.
8. James Clark and Steve DeRose. *XML Path Language (XPath)*. W3C Working Draft, November 1999. `http://www.w3.org/TR/xpath`.
9. Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.
10. Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.
11. R. G. G. Cattell et al, editor. *The Object Database Standard: Odmg 2.0*. Morgan Kaufmann, 1997.
12. A. Gupta and I. Mumick. Maintenance of materialized views: Problems, techniques, and applications. IEEE Data Engineering Bulletin, Vol. 18, No. 2, June 1995., 1995.
13. Michael Lesk. *Practical Digital Libraries: Books, Bytes and Bucks,*. Morgan Kaufmann, July 1997.
14. Hartmut Liefke and Susan Davidson. View maintenance for hierarchical semistructured data. In *International Conference on Data Warehousing and Knowledge Discovery*, 2000.
15. Susan Davidson and Chris Overton and Peter Buneman. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology*, 2(4):557–572, Winter 1995.
16. World Wide Web Consortium (W3C). *XML Schema Part 0: Primer*, 2000. `http://www.w3.org/TR/xmlschema-0/` .