A Fingerprint Method for Scientific Data Verification

Micah Altman
Institute for Quantitative Social Science, Harvard University
1737 Cambridge St. # 325
Cambridge, MA 02138

Micah_Altman@harvard.edu

Abstract - **This article discusses an algorithm (called "UNF") for verifying digital data matrices. This algorithm is now used in a number of software packages and digital library projects. We discuss the details of the algorithm, and offer an extension for normalization of time and duration data.**

INTRODUCTION

In digital library systems, migrating digital objects among formats is necessary both for effective use and preservation. However, a consequence of such reformatting is that we are often required to re-verify the intellectual content of the object, in order to avoid data loss and misinterpretation.

The Universal Numeric Fingerprints (UNF's) is an algorithmic tool used to verify that a data matrix or related digital object that is produced in one software environment and/or format has been correctly interpreted when moved to a different environment and/or format.

In outline, the algorithm uses three steps to compute a fingerprint: First, an approximation algorithm is used to compute a reduced-fidelity version of the target digital object. Second, this reduced version is put into a normalized serial form. Third, a hash function is used to compute a unique fingerprint from the resulting serialized object.

While these steps could in theory be applied to any digital object, in practice it is necessary to create a specific normal form and approximation method must be defined for each type of object to be processed. The UNF is implementation is specially tailored for use with scientific data. We describe the approach, implementation, and algorithmic details below.

RELATED APPROACHES

The UNF approach shares some similarities to some published approaches to audio and video fingerprinting [1,2] The UNF algorithm however is specifically tailored to scientific and statistical data vectors. Audio and video fingerprinting methods typically rely on type-specific feature extractions, and are not directly applicable to scientific databases. Furthermore, unlike audio and video fingerprints, which typically compute a long sequence, UNF's use a more compact representation, suitable for use in scholarly citations.

Other approaches have been developed that attempt to verify that one object is a derivative of another object, regardless of file format. These methods operate through insertion or alteration of data in unused or unnoticed portions of the object, forming a digital watermark. Research into digital watermarks have produced algorithms that are designed to be robust to lossy transformations of the object. And hence some types of image objects can be identified as a derivative of another even when the derivative is manifested in a different file format. (For a survey see [3].)

Watermarks have significant shortcomings when used to establish the semantic equivalence of two arbitrary digital objects. Watermarking algorithms cannot be used to establish that two independently created objects are semantically equivalent, since these will not share the same watermark. Conversely, two objects could have identical watermark information added, but contain completely different semantic content. Nor can watermarks be used to verify that a derivative is identical to a watermarked digital object, if the derivative was created from the original digital object before the watermark was applied to that original digital object. Furthermore, watermarks are not practical for some objects, such as numeric data and source code files, where the alterations created by the watermarking process tend to alter the semantic content of the digital object.

IMPLEMENTATION AND USE

The UNF was developed in the course of research into improving the numerical accuracy of statistical software [4]. It was incorporates type-specific algorithms for scientific data such as numeric vectors, matrices, and related objects.

Where possible, UNF algorithms are implemented as plug-ins for the software applications being used to manipulate or analyze the digital objects of interest. This strategy is sed so that the UNF can be computed directly from the internal data structure used by the software product performing the data display and manipulation. This allows the UNF to be used not just to verify format migration, but also to verify that the data has been correctly interpreted by a software application. (Some of the errors we frequently encountered in [4], and which the UNF detects, included: omitted records, omitted

T. Sobh (ed.), *Advances in Computer and Information Sciences and Engineering*, 311–316.

variables, recoding of missing values, and loss of numeric precision.)

Furthermore, this implement strategy obviates the need to write specific parsers for each data format – since the host software application parses the object before calculating UNF. Instead, we need only supply standard interfaces for computing UNF'sof vectors, matrices, and other related structures.

The general algorithm was first described in [4] along with a sample implementation. Version 3 was the first version to be implemented in publicly available software: The UNF package for R was made available through the Comprehensive R Archive Network (CRAN), a code archive developed part of the R Project [5]. (This version of the software also introduced the name "Universal Numeric Fingerprint".)

UNF plug-ins are currently available for the statistical software packages "R", "SAS", and "Stata"; along with a stand-alone application, and C++ libraries. The UNF technology has been incorporated in a number of other digital library and preservation practices: UNF's are used to support format migration in the "Virtual Data Center" (VDC) digital library system for federated data sharing. [6] The successor of the VDC, the Dataverse Network Project (DVN) [7] which provides digital library software and virtual-arching services, uses UNF's for data verification.

Both systems now generate scholarly citations for all data collections, based on the standard for the citation of quantitative data developed in [8]. In this citation standard, the UNF is an integral part of the citations, and is used to verify that a given data collection matches the version cited. The citation's combination of a global unique identifier, UNF, and bridge service URL combine to support permanence, verifiability, and accessibility.

For use in citations to data, when displayed, the UNF is formatted so that it is self-identifying and can easily be printed. An example of a printed UNF is UNF:3:ZNQRI14053UZq389x0Bffg?==, where UNF: identifies the rest of the string as a UNF, :3 means that the fingerprint uses version 3 of the UNF and hash algorithm, and everything after the next colon is the actual fingerprint. For a particular algorithm and number of significant digits, the fingerprint is always the same length. Thus, the UNF includes enough self-identifying information so that the algorithm used may be updated to newer versions over time without disturbing old citations.

UNF's are also used by the The Data Preservation Alliance for the Social Sciences (Data-PASS), and . This is a partnership of six major U.S. archival institutions collaborating to acquire and preserve data at-risk of being lost to the research community. To support this goal, Data-PASS developed shared technical infrastructure and best practices for metadata, confidentiality, security, processing and other elements of the archival management process. UNF's are used for citation and verification in the Data-PASS shared catalog infrastructure, and are part of the best practices for metadata identified by the alliance. (See [9,10]).

In these systems, standards, and practices, UNF's are used as a replacement for or supplement to file-level cryptographic hashes and summary statistics. (In a sense, the UNF is a uniquely tailored summary statistic for the entire object.) Like a cryptographic hash, the UNF detects changes to the data object. Unlike a standard hash, it is sensitive only to semantically important changes – changes of format or insignificant changes of precision will not yield differing UNF's.

ALGORITHMIC CONSIDERATIONS

To summarize, the abstract algorithm consists of the following steps: An approximation algorithm is used to compute the approximated semantic content of the digital object. This approximated content is then put into a normalized form. A hash function is used to compute a unique fingerprint for the resulting normalized, approximated object, and the hash is stored. When the object is reloaded into the same or another application, this process is repeated, and the value generated at load time is compared to the stored value. These steps are discussed in more detail in the remainder of this section.

The first part of what is needed is a normalization function f () that maps each sequence of numbers (or more commonly, blocks of bits) to a single value:

$$f\{i_0, i_1, ..., i_n\} \rightarrow c \tag{1}$$

To verify the data, we would need to compute f once when initially creating the matrix. Then recompute it after the data has been read into our statistics package. For robust verification, we should choose **f**() such that small changes to the sequence are likely to yield different values in **f**().

Normalization of objects alone, while it can be used as a basis of establishing identity across formats in limited cases, is inapplicable when reformatting of the object changes the precision, accuracy, or level of detail of an object in trivial ways. This is a well known issue is video and audio formats, in reformatting complex text documents, and surprisingly occurs commonly even in reformatting purely numerical databases.

Any type-specific approximation function **A**(), may be employed.[1] This approximation process, **A**(), accepts as input a digital object, *O*, of specified type, and an approximation-level parameter, *k*.

[1] For audio and video type-specific approximation is used, such as decimation, spatial or frequency downsampling, and/or numerical cutoff filtering. For examples of decimation, downsampling and cutoff algorithms see: [11], [12].