26

Compacting Anglo-Saxon cemetery data

Jeremy Huggett
Research Centre for Computer Archaeology, North Staffordshire Polytechnic

26.1 Introduction

This paper is not about databases as such, nor is it simply an account of a data management system produced as part of a research project designed to record Anglo-Saxon cemeteries, although this will be referred to in passing. Instead, this paper is concerned with the way in which data in general is handled within a machine-based environment. The aim is to discuss the ways in which archaeological data may be recorded and stored more efficiently on computer, and to describe the benefits, as well as some of the disadvantages, that can result.

I start from the premise that archaeological data are not always recorded on computer as efficiently as they might be. When the inevitable happens and disc space runs out, the tendency is to upgrade the storage capacity rather than to make better use of existing facilities. Rather than list specific occasions when this has happened, this paper will argue that anyone who includes character or text string entries in their database is unlikely to be making best use of their disc space.

The rapidly falling price of storage media, and hardware in general, is often cited as a reason against the need for compact data storage, but efficient storage is not the only recommendation for data compaction. Reducing the amount of disc space required by a data set can have a knock-on effect on such factors as processing time and ultimately means that much larger quantities of data can be recorded and manipulated without upgrading the hardware.

Two main levels of data compaction may be defined: substitution, and compression. To some extent these are organised in ascending order of programming complexity, but these techniques may be employed individually or in combination with an increasing saving in storage requirements. If both techniques are employed together the savings may be quite dramatic. To use my own system as an example, it is estimated that the entire Anglo-Saxon burial data set consisting of around 30,000 burials could be held in approximately 800 kilobytes of disc space. By way of comparison, if the same data set were to be held on disc in its expanded form, it would require something in the order of 30,000 kilobytes or 30 megabytes.

26.2 Substitution

The substitution of data by a code is a feature of some archaeological data recording systems, with Dominic Powlesland's excavation recording system being perhaps the most obvious example (Powlesland 1985). A recent book on archaeological data processing describes three types of code—the full keyword, the abbreviated keyword, and the numeric code (Richards & Ryan 1985, p. 122). Of these, only the numeric code offers a real saving in terms of disc space.

There are also the additional benefits which accompany the use of a numeric code: rapid sorting and processing of data, speed of entry, and the facility to use the number system for statistical analyses for instance.

Replacing a character string, which might correspond to a site name or a descriptive category, with a single number can result in a substantial reduction of the size of the record. A 30 character string which would occupy 30 bytes of storage space can easily be replaced with a single 2-byte integer.

However, there are problems with codes, particularly because they may be difficult to remember and require a code reference book. In addition, once coded, the data may be difficult to verify. In fact, such criticisms are very minor, and with some care at the design stage need not apply. To illustrate this, the process developed to record an Anglo-Saxon burial will be briefly described.

The fields are defined at the initialisation stage, as is usual, and during a recording session they appear as 'questions' which have to be answered by the user. Data substitution occurs at this stage, since rather than entering a text description for each field, the user enters the corresponding numeric code. An expert user will know the codes, but if required, each question will appear on screen accompanied by a menu listing the various options along with their numeric code. Thus, there is no need for a code book, since the program itself is able to prompt the user if necessary and the codes remain transparent to the user. Each set of numeric codes is only unique within the field—a number '1' may be entered in any of the fields, but it will have a different meaning depending on the context. Some fields have too many options to be conveniently displayed on screen—the artefact name, for instance. In this case, a keyword entry system is used, whereby the user enters the keyword, or an abbreviation of it, and the system places the unique number associated with that entry into the record. One side-benefit of this approach is that it acts as an automatic spelling checker.

These menu options could be entered in advance, along with the field names, at the initialisation stage. This assumes that all the possible options are known before data capture commences, and that the descriptions of all graves and their contents could easily be reduced to a fixed set of templates. However, this would be an extremely inflexible approach since no allowance would be made for burials which did not fit conveniently into any of the pre-set categories. It would also mean that a large number of possible options would have to entered, many of which might never be used.

Instead, the approach that was adopted was to develop a system which allowed the menus to expand dynamically as required. Menus are initialised with the option 'Unknown' where applicable, and further options are only added when they are required to deal with a particular burial. If needed, the user simply enters a number one larger than the greatest shown on the menu, and is then prompted for the new option. The next time that menu appears on screen, the new entry has been added to it. Similarly, if no match is found for a keyword provided, the system checks with the user that the entry is correct before adding it to the lookup file and assigning a new number to it. A maximum of 30 characters is allowed for each menu entry, and these are only stored once in a lookup file rather than in every single record.

In this way, a burial record is built up, consisting of a series of numbers corresponding to the descriptive terms and other categories (Fig. 26.1). The level of compaction in this example may be calculated using the grave record as an example. A record with seventeen 30-byte fields totalling 510 bytes is reduced to seventeen 2-byte integers totalling 34 bytes: a reduction of 93%.

Field name	Descriptive Record	Substituted Record
CEMETERY NAME	Nassington	34
GRAVE NUMBER	9	9
BURIAL RITE	Inhumation	1
GENERAL STRUCTURE	Flat Burial	1
EXTERNAL STRUCTURES	None	1
INTERNAL STRUCTURES	None	1
GENERAL BURIAL TYPE	Single Burial	1
RELATIONSHIP WITH OTHER GRAVES	None	1
GRAVE NUMBER	—	0
SEX	Male	2
AGE	Adult (18–40 yrs)	5
ORIENTATION	315	315
GENERAL POSITION	Extended, Supine	2
HEAD POSITION	Facing Up	1
LEFT ARM POSITION	Along Side	1
RIGHT ARM POSITION	Along Side	1
LEG POSITION	Parallel (together)	1
GRAVE GOODS POINTER	No Grave Goods	0

Fig. 26.1: A substituted grave record

26.3 Compression

However, storing data as 2-byte integers is still wasting space. Two bytes can hold a single number ranging from 0 to 65535, yet most applications are unlikely to need numbers which are larger than a few hundred. Since zero can be held in 1 bit, and 100 needs only 7 bits for instance, it can be seen that not all of the 16 bits may actually be filled. In other words, a 2-byte integer is same as a fixed-length record: more than large enough to store the biggest number likely to be required, but consequently wasteful of space if only small numbers are used.

The system developed uses a technique of bit-packing to compress the contents of several fields into a single half word or 2-byte number. The number of bits required to hold each field has to be defined at the initialisation stage. This imposes an apparent limitation in that it assumes that the upper limit for the field is known. In other words, a decision has to be made at the initialisation stage as to whether there are likely to be five categories in a particular field or ten. However, generous limits can be set with very little additional cost in storage overheads—seven bits can store numbers up to 127 for instance, but simply adding one extra bit allows for numbers up to 255 (Fig. 26.2).

Having defined the size of each field in terms of the number of bits required, a code is generated which is then used for the subsequent encryption of the entries. Using the number of bits allocated to each field, the number of fields which can be compressed into each 2-byte integer is calculated, together with the power shift required to compress each field entry. This process is carried out once only, at the initialisation stage, and the codes are then loaded into memory at the start of each run. For example, eleven bits are assigned to the cemetery name field, allowing for 2047 cemeteries. This leaves five bits free in the first compacted entry. The next field is the grave number, which is assigned nine bits, allowing for 511 graves per cemetery. The grave number is compressed into the remaining five bits of the first compacted entry, with the remaining four bits carried over to the second entry. The second compacted entry therefore has sixteen bits free less the four bits carried over from the grave number: a total of twelve bits left. Into this space is compressed the burial rite field (one bit), general

Bits	Largest number
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023
11	2047
12	4095
13	8191
14	16383
15	32768
16	65535
n	2**n-1

Fig. 26.2: Upper limits for bit-packed entries

grave structure (three bits), external structures (three bits), internal structures (three bits) and burial type (three bits, with one carried over into the third compacted entry). The end result is that all seventeen fields, together with a pointer to the grave goods record, are compressed into six 2-byte numbers. Ignoring the grave goods pointer, the seventeen fields, which were originally reduced to 34 bytes, are finally reduced to ten bytes (Fig. 26.3).

The important figure is what this means in terms of an overall reduction in storage requirements. Compared with a fully expanded grave record, a 97% reduction is achieved using the substitution and compression techniques described above.

26.4 Conclusions

The figures, I think, speak for themselves. Using compaction techniques the savings in storage space are dramatic: in the Anglo-Saxon example, the record is reduced to 3% of its former size. In spite of this, the process of data compaction is often viewed with suspicion, and there seems to be a general reluctance to apply compaction techniques. The reasons for this are not clear.

As mentioned above, the increasing availability of cheap mass storage devices is often used as an argument against the need for efficient data storage. While there is no denying that the price of hard disc drives is tumbling, it should hardly need to be pointed out that data always expands to fill the space available. The attitude that the hardware can always be expanded is extremely short-sighted, if not reckless. Archaeology in this country is a publicly-financed business, and has a responsibility to spend wisely what little money it has. The purchase of unnecessary computer hardware is hardly the most serious waste of resources there is, but in many cases the purchase of a computer represents a considerable proportion of the capital budget of a small unit. In addition, running out of disc space may be a result of sloppy programming rather than an excess of data.

Substituted record	Bits per field with split entries		Compressed record
34	11	first half word	18466
9	5		
	4		
1	1		
1	3		
1	3	second half word	18736
1	3		
1	2		
	1		
1	3		
0	9	third half word	16386
2	3		
5	3		
315	9	fourth half word	1071
2	4		
1	3		
1	3		
1	3	fifth half word	585
1	3		
0	3		
	14		
	2	sixth half word	0

Fig. 26.3: Compression of a substituted record

A major criticism of data compaction is that it may be difficult to interpret the raw data held on file. This is certainly the case with the techniques employed on the Anglo-Saxon burial data set described above—it would probably take ten minutes and several sheets of calculations to decode a single burial record by hand. However, even a straightforward text file undergoes a form of encryption when it is written to disc, and its subsequent interpretation is controlled by the operating system. Coding data should hold no fear for archaeologists—most archaeologists do it all the time—and if anything, using a computer to code data simplifies the process. In addition, any program which is capable of coding a record can be used to decode it again.

There is, of course, one drawback with data compaction—the actual process of encryption and decoding requires additional processing, and therefore a small increase in time is to be expected. In most cases, however, it is not necessary to decode a record completely. Searches, frequency counts and other forms of statistical processing can all be carried out on the numeric record, so that the only decoding involved is that which converts the compressed record back into the substituted numeric record. Indeed, all such data handling techniques can be performed more efficiently and rapidly on a numeric record rather than an expanded character record. The only time that a record needs to be decoded completely is when it is output to the screen or printer for validation, listing, or archiving on paper.

Any increase in processing time is limited to the input/output phases and is very small—barely noticeable on a small micro, and on a mini or mainframe computer the response still appears instantaneous. On the basis of the dramatic savings in storage alone, I would suggest that it is a small price to pay. However, the benefits extend further than the physical reduction in size of a data set. Processing the record in numeric form is much more efficient in terms of computer time than handling text records. In addition, the compaction of data enables records to be blocked together so that a number of data records may be read or written in a single operation, thus reducing processing time by increasing the number of records held in memory and cutting the number of disc accesses required. Consequently, any increase in processing time resulting from the compaction of data will be vastly outweighed by the reduction of time spent accessing the disc, since fewer disc accesses will be required to transfer the same amount of data, and by the increased efficiency in processing the data once they have been read into memory.

In conclusion, two final points may be made. First, the techniques outlined above are only two possible ways of compacting data. For example, Dominic Powlesland employs rather different methods of substitution and compression with similar success, although the overall level of compaction is lower (Powlesland 1985). Finally, the advantages of data compaction are not restricted to those who start out by compressing their data—these techniques may be applied retroactively to data sets which are already held on computer. Anyone facing the onset of a data storage crisis in the future could do worse than consider a more efficient method of storage rather than automatically move up to the next rung on the ladder of hardware escalation.

References

POWLESLAND, D. 1985. 'Random access and data compression with reference to remote data collection: 1 and 1 = 1', *in* M. A. Cooper & J. D. Richards, (eds.), *Current Issues in Archaeological Computing*, pp. 23–33, International Series 271, British Archaeological Reports, Oxford.

RICHARDS, J. D. & N. S. RYAN 1985. *Data Processing in Archaeology*, Cambridge University Press, Cambridge.