Compressing semi-structured text using hierarchical phrase identification

Craig G. Nevill-Manning, Ian H. Witten
Computer Science, University of Waikato, Hamilton, New Zealand
email {cgn, ihw}@waikato.ac.nz

Dan R. Olsen Jr.
Computer Science, Brigham Young University, Provo, Utah, USA
email olsen@cs.byu.edu

1 Introduction

Many computer files contain highly-structured, predictable information interspersed with information which has less regularity and is therefore less predictable—such as free text. Examples range from word-processing source files, which contain precisely-expressed formatting specifications enclosing tracts of natural-language text, to files containing a sequence of filled-out forms which have a predefined skeleton clothed with relatively unpredictable entries. These represent extreme ends of a spectrum. Word-processing files are dominated by free text, and respond well to general-purpose compression techniques. Forms generally contain database-style information, and are most appropriately compressed by taking into account their special structure. But one frequently encounters intermediate cases. For example, in many email messages the formal header and the informal free-text content are equally voluminous. Short SGML files often contain comparable amounts of formal structure and informal text. Although such files may be compressed quite well by general-purpose adaptive text compression algorithms, which will soon pick up the regular structure during the course of normal adaptation, better compression can often be obtained by methods that are equipped to deal with both formal and informal structure.

Semi-structured text represents a compromise between the demands of intelligibility and efficiency: it is readable by humans, yet is arranged in a way that facilitates automatic processing. Historically, formally-structured databases with rigid structure have been favored over informally-structured text because of their greater efficiency. Fixed-length records make indexing and access to information much simpler than free text. When responding to specific queries, data files with a rigid, predefined format can be accessed more efficiently than can looser textual representations of the same information—for an entire arsenal of database technology can be summoned to provide efficient retrieval.

However, in many situations the balance seems slowly to be shifting in favor of semi-structured information storage. Many factors are encouraging this trend: more emphasis on flexibility; changing balance between human and machine costs; increasing globalization; cheaper and more plentiful storage, processing, and bandwidth resources; improved software technology. With the increasing availability and convenience of full-text indexing, access to information in more flexible formats can also be performed efficiently. It is becoming feasible to build high-use services such as web indexes based on textual databases.

But information in semi-structured form is often repetitive and verbose. Even when compressed using standard text compression methods, it occupies far more storage than a specially-tailored database file that represents the same information. From the point of view of storage efficiency, you cannot do better than encode data in a way that takes account of its particular structure. Some full-text retrieval systems also address the storage problem, at least to some extent, since they compress the textual database.

However, they treat all information as free text and do not take account of any structure that is present to increase the amount of compression obtained.

This paper takes a compression scheme that infers a hierarchical grammar from its input, and investigates its application to semi-structured text. Although there is a huge range and variety of data that comes within the ambit of “semi-structured,” we focus attention on a particular, and very large, example of such text. Consequently the work is a case study of the application of grammar-based compression to a large-scale problem.

The structure of this paper is as follows. We begin by identifying some characteristics of semi-structured text that have special relevance to data compression. We then give a brief account of a particular large textual database, and describe a compression scheme that exploits its structure. In addition to providing compression, the system gives some insight into the structure of the database. Finally we show how the hierarchical grammar can be generalized, first manually and then automatically, to yield further improvements in compression performance.

2 Semi-structured text

We characterize semi-structured text as data that is intended for automatic processing, but which is also human-readable. Figure 1 shows four representative examples: fragments of HTML, an email message, a genealogical database, and a structured data file. Of course, the ratio of free text to structured fields varies greatly in such files. Compression schemes that perform well on text also perform well on semi-structured text: they often perform

a
```
<TABLE>
<TR><TD VALIGN=TOP>Query type:</TD>
<TD>
<INPUT TYPE="radio" NAME="querytype" VALUE="b
<INPUT TYPE="radio" NAME="querytype" VALUE="r
</TD>
</TR>
<TR><TD VALIGN=TOP>Search type:</TD>
<TD>
<INPUT TYPE="radio" NAME="collection" VALUE="n
<INPUT TYPE="radio" NAME="collection"
VALUE="nz
<INPUT TYPE="radio" NAME="collection" VALUE="n
</TD></TR>
</TABLE>
```

b
```
Replied: Tue, 17 Oct 1995 09:53:52 +1300
Replied: "M.Apperley@lucy.cs.waikato.ac.nz (Mar
Return-Path: M.Apperley@lucy.cs.waikato.ac.nz
Return-Path: <M.Apperley@lucy.cs.waikato.ac.nz
 by waikato.ac.nz (PMDF V5.0-4 #11755) id <01HW
Received: from [130.217.248.11] (judie.cs.waikato.
 by cs.waikato.ac.nz (5.x/SMI-SVR4) id AA17856;
Date: Mon, 16 Oct 1995 10:25:16 +1300
From: M.Apperley@lucy.cs.waikato.ac.nz (Mark A
Subject: OZCHI Proceedings
X-Sender: mapperle@lucifer.cs.waikato.ac.nz
Resent-To: ihw@cs.waikato.ac.nz
Message-Id: <9510152125.AA17856@cs.waikato.

Thanks for offering to explore the possibility of ACM
Mark
```

c
```
@attribute 'education-allowance' {'yes','no'}
@attribute 'statutory-holidays' real
@attribute 'vacation' {'below_average','average','g
@attribute 'longterm-disability-assistance' {'yes','n
@attribute 'contribution-to-dental-plan' {'none','half'
@attribute 'bereavement-assistance' {'yes','no'}
@attribute 'contribution-to-health-plan' {'none','half'
@attribute 'class' {'bad','good'}
@data
1,5,?,?,?,40,?,?,2,?,11,'average',?,?,'yes',?,'good
3,3.7,4,5,'tc',?,?,?,?,'yes',?,?,?,?,'yes',?,'good'
3,4.5,4.5,5,?,40,?,?,?,?,12,'average',?,'half','yes',
2,2,2.5,?,?,35,?,?,6,'yes',12,'average',?,?,?,?,'go
3,4,5,5,'tc',?,'empl_contr',?,?,?,12,'generous','yes
```

d
```
0 @26DS-KX@ INDI
1 AFN 26DS-KX
1 NAME Dan Reed /OLSEN/
1 SEX M
1 BIRT
2 DATE 22 JUN 1953
2 PLAC Idaho Falls,Bonneville,Idaho
1 FAMC @00206642@
0 @00206642@ FAM
1 HUSB @NO48-3F@
1 WIFE @93GB-DD@
1 CHIL @26DS-KX@
1 CHIL @21B7-WR@
0 @26DN-7N@ INDI
1 NAME Mary Ann /BERNARD (OR BARNETT)
```

Figure 1 Examples of semi-structured text: (a) HTML source, (b) email, (c) textual data file, and (d) genealogical data even better because the structured information is more predictable. However, it should be possible to leverage the special qualities of semi-structured text to improve on the performance of general-purpose schemes.

One feature of semi-structured text is repeating template structures, such as the INPUT TYPE lines in Figure 1a. Each radio button in a World Wide Web page is defined using the <INPUT TYPE...> tag, which has a specific format evident in the five similar lines. The buttons are grouped into table cells, which are introduced by the <TD> tag and end with the corresponding </TD> tag. A scheme that takes advantage of the predictability of this structure may yield better compression than a general-purpose scheme.

A salient aspect of all semi-structured text is the use of keywords: fixed tokens that fulfil a structural role, and therefore occur more frequently and predictably than words in free text. Examples are evident in all the fragments in Figure 1. Standard text compression methods will benefit from the frequency of such terms, but will not reap full benefit from their relatively predictable context.

Semi-structured files sometimes display global as well as local structure. Figure 1c shows an excerpt from a structured file of data about employment contracts. The first part lists attribute names and values, one per line, in a simple format, effectively defining data types. Each line of the second part gives a list of attribute values, each value being drawn from those given in the corresponding attribute definition. Using the attribute information to encode the values would compress the file very significantly.

Yet another source of redundancy in semi-structured text is predictability within the free text. The DATE tag in Figure 1d is always followed by a day of month, a month name, and a year, while the NAME tag is followed by some number of first names and then the family name in uppercase, delimited by virgules.

3 The example text

Our example text takes the form of a genealogical database, expressed in a semi-structured textual form that is specifically designed to represent such information. There is great variability in the kinds of genealogical information that must be stored. Genealogical evidence comes from a wide variety of source records from a variety of cultural norms. This creates a strong requirement for flexibility. Despite this there is a high degree of structure. Places, dates and significant events all occur with great regularity and where similar information occurs, a similar representation is used.

THE GENEALOGICAL DATABASE

The LDS Church, for various reasons, maintains the most comprehensive collection of on-line genealogical information in the world. The two largest of these databases are the International Genealogical Index, which contains birth, death and marriage records, and the Ancestral File, which contains linked pedigrees for families all over the world. The former contains the records of over 265 million people while the latter contains records for over 21 million; they are growing at a rate of 10% to 20% per year. In uncompressed GEDCOM format (see below), these databases total 62.5 Gbytes in size. They are currently compressed in an *ad hoc* manner and occupy 20 Gbyte and 5 Gbyte, respectively.

GEDCOM FORMAT

The GEDCOM standard for exchanging genealogical information stores information as textually encoded trees. Each node has a tag which identifies its type and some textual content. The content of a node is highly stylized depending on the node's type. Any node can have child nodes that provide additional information. This form is very flexible, easily transmitted and yet is amenable to automatic processing because of its tree structure. Its variability, however, makes it more suited to full text retrieval than to traditional database queries.

The records are variable-length, and may have any combination of fields. Each record is a line of text with a level number, tag and textual contents. The level numbers provide a hierarchy within a record in a scheme reminiscent of COBOL data declarations. Lines at the zero level can have labels that are used elsewhere in the file to refer to the entire record. These records are generally individuals (INDI) or families (FAM).

Figure 1d shows an example of an individual record, a family record, and the beginning of another individual record. The first gives name, gender, birth date, birthplace, and a pointer to the individual's family. The family record, which follows directly, gives pointers to four individuals: husband, wife, and two children—one of which is the individual himself. This example, however, gives an impression of regularity which is slightly misleading. For most of the information-bearing fields such as NAME, DATE, and PLACE, there are records that contain free text rather than structured information. For example, the last line of Figure 1d shows a name given with an alternative. The DATE field might be "Abt 1767" or "Will dated 14 Sep 1803." There is a NOTE field (with a continuation line code) that frequently contains a brief essay on family history.

4 SEQUITUR

Nevill-Manning *et al.* (1994) describe a scheme, dubbed SEQUITUR, that infers a hierarchical description of a sequence based on repeating subsequences. This outperforms most other dictionary-based compression methods on the Calgary corpus, although it does not do quite so well as the best statistical methods.

SEQUITUR creates a grammar in which each rule represents a repeating subsequence within the sequence. Short rules can be used within longer ones, resulting in a compact hierarchy that also provides a plausible, human-readable structure for the sequence. Figure 2 shows a short sequence and the grammar that is produced for it. *S* is the start symbol, and rule *S* expands to the original sequence. The short rule *A* is reused in rule *B* to describe a longer repetition in the original sequence. SEQUITUR's grammars obey two constraints: no digram appears twice, and every rule is used more than once. The heart of the algorithm is the enforcement of these two constraints by rule creation and deletion respectively.

In general, the resulting grammar contains a rather long first rule, and a large number of much smaller rules—although this is not evident in the small example of Figure 2. The first rule contains all the unstructured, non-repeating, parts of the input sequence. For example, when compressing a large segment of text on a character-by-character basis, the first rule is typically nearly half the size of the original text, in terms of the number of non-terminals and terminals together. Typically, the number of different non-terminals—that is, the number of rules—is around 4% of the number of characters in the original input, and the average length of the right-hand side of a rule is just over two symbols.

This scheme improves on other dictionary schemes in three ways. First, the entire grammar is kept in memory at once, so phrases that are separated by many symbols can still be matched. This contrasts with most dictionary compression schemes, which search

abcdbcabcdbc	S → BB A → bc B → aAdA

Figure 2 A short sequence and the grammar that SEQUITUR produces

for matches in a window of a few thousand characters to bound processing time. SEQUITUR's grammar not only provides an efficient in-memory record of the sequence, but it also enables fixed-time lookup for matching phrases.

Second, because of the way in which Ziv-Lempel schemes add phrases to the dictionary, many phrases become redundant. In LZ78 (Ziv and Lempel, 1978), new phrases are produced from older ones by adding a single symbol. This means that phrases are generated on the way to a long repeated phrase, and are not used again. In LZMW, a variation of LZ78, new phrases are constructed by concatenating the two previous phrases encoded. While this allows phrases to grow more quickly, it still gives rise to unused phrases which consume code space. SEQUITUR's phrases must occur at least twice to avoid deletion, and a phrase is removed if it is superseded by a longer one.

Third, SEQUITUR's hierarchy of phrases represents the dictionary more concisely because long phrases often contain shorter ones. Representing dictionary entries in terms of other dictionary entries reduces the size of pointers required to specify both the starting point and the length of a new dictionary entry.

The key to SEQUITUR's suitability for semi-structured text is its emphasis on hierarchical structure, and its ability to match repetitions that are widely separated and arbitrarily long.

5 Compression of the genealogical database

In order to assess the performance of various compression methods on semi-structured text, we extracted 38 000 individual records and 17 000 family records from the genealogical database, totalling 9 Mbyte in GEDCOM format. Table 1 shows the results of a number of standard compression programs on this sample, along with those from SEQUITUR in several different configurations. In this section, we describe the standard programs, then the use of SEQUITUR. In the following section, we discuss how the grammar can be generalized and show how SEQUITUR's grammar can be interpreted.

STANDARD COMPRESSION PROGRAMS

The LDS genealogical databases are currently stored in a system called AIM which performs special-purpose compression designed specifically for the GEDCOM format. This begins by assembling a dictionary of the most frequently occurring one-, two- and three-word phrases. Any words or phrases that do not occur at least three times are dropped. This dictionary then is encoded with the 63 most frequent entries in one byte and the remaining entries in two bytes. In addition to this there are special encodings for dates and other types. This first-level approach provides about 40% compression, which is considerably worse than any of the other schemes we tested. However, the files so produced are then compressed using the standard STACKER compression product. This provides a further 40% compression for a total compression in AIM of 16%.

Table 1 shows the results of a number of compression programs on the 9 Mbyte sample of the genealogical database that we used for evaluation. The standard compression programs (except MG) do not support random access to records of the database, and so they are not suitable for use in practice because random access is always a *sine qua non* for information collections of this size.

The first block of Table 1 summarizes the performance of some popular byte-oriented schemes. Unix *compress* illustrates what is obtainable using a robust implementation of Ziv and Lempel's LZ78 technique (Ziv and Lempel, 1978). The *gzip* utility is an exemplary implementation of the earlier LZ77 method (Ziv and Lempel, 1977), and achieves substantially better compression. PPMC (Bell *et al.*, 1990), a statistical data

	scheme	dictionary	code indexes	word indexes	total size (Mbyte)	compression
original	no compression				9.18	100.0%
byte-oriented	COMPRESS				2.55	27.8%
	GZIP				1.77	19.3%
	PPM				1.42	15.5%
word-oriented	WORD-0	–	–	–	3.20	34.8%
	WORD-1	–	–	–	2.21	24.1%
	MG	0.14	–	2.87	3.01	32.8%
SEQUITUR		0.11	–	1.07	1.18	12.9%
SEQUITUR	codes generalised	0.11	0.40	0.60	1.11	12.1%
	dates generalised	0.11	0.64	0.31	1.06	11.5%
	gender generalised	0.11	0.64	0.30	1.05	11.4%
	names generalised	0.11	0.76	0.17	1.04	11.3%

Table 1 Compression rates of various schemes on the genealogical data

compression scheme that normally outperforms dictionary schemes, performs extremely well on the data, giving a compression rate of over six to one. For all these schemes, compression rates are about twice as great as they are on *book1* from the Calgary corpus (Bell *et al.*, 1990), which indicates the high regularity of this database relative to normal English text.

The next block of Table 1 summarizes the performance of some word-oriented compression schemes. These schemes split the input into an alternating sequence of words and non-words—the latter comprising white space and punctuation. WORD uses a Markov model that predicts words based on the previous word and non-words based on the previous non-word, resorting to character-level coding whenever a new word or non-word is encountered (Moffat, 1987). We used both a zero-order context (WORD-0) and a first-order one (WORD-1). MG is a designed for full-text retrieval and uses a semi-static zero-order word-based model, along with a separate dictionary (Witten *et al.*, 1994). In this scheme, as in WORD-0, the code for a word is determined solely by its frequency, not on any preceding words. This proves rather ineffectual on the genealogical database, indicating the importance of inter-word relationships. WORD-1 achieves a compression rate that falls between that of *compress* and *gzip*. The relatively poor performance of this scheme is rather surprising, indicating the importance of sequences of two or more words as well perhaps as the need to condition of inter-word gaps on the preceding word, and vice versa.

None of these standard compression programs perform as well as the *ad hoc* scheme used in AIM, except, marginally, PPM.

APPLYING SEQUITUR

In order to apply the SEQUITUR method to this data, it was resolved to take a word-based approach to compression. Virtually all words were separated by single spaces, and so it was not necessary to have a separate sequence of non-words. Instead, a single space was defined as the word delimiter. In the rare occasions where extra spaces occurred they were prepended to the next word—this generally happened only with single-digit dates. Other punctuation was appended to the preceding word; again, this decision did not materially increase the dictionary size. The dictionary was encoded separately from the word sequence, which was represented as a sequence of numeric dictionary indexes.

The word indexes were compressed using SEQUITUR as if they were symbols drawn from a large alphabet. The compression scheme was identical to that described by Nevill-Manning *et al.* (1994). The dictionary was compressed in two stages: front coding followed by compression by PPMC. Front coding involves sorting the dictionary, and whenever an entry shares a prefix with the preceding entry, replacing the prefix by its length. For example, the word *baptized* would be encoded as *7d* if it were preceded by *baptize*, since the two have seven letters in common. A more principled dictionary encoding was also performed, but failed to outperform this simple approach.

The input comprised 1.8 million words, and the dictionary contained 148 000 unique entries. The grammar that SEQUITUR formed had 71 000 rules and 648 000 symbols, 443 000 of which were in the top-level rule. The average length of a rule (excluding the top-level one) was nearly 3 words. This grammar, when encoded using the method described by Nevill-Manning *et al.* (1994), was 1.07 Mb in size. The dictionary compressed to 0.11 Mb, giving a total size for the whole text of 1.18 Mb. This represents almost eight to one compression, some 20% improvement over the nearest rival, PPMC.

6 Generalizing tokens for increased compression

Examination of SEQUITUR's output reveals that significant improvements could be made quite easily by making small changes to the organization of the input file. We first describe how this was done manually, by using human insight to detect regularities in SEQUITUR's output, next we show how the grammar can be interpreted, and finally show how the process of identifying such situations can be automated.

MANUAL GENERALIZATION

Of the dictionary entries, 94% were codes used to relate records together for various familial relationships. Two types of codes are used in the database: individual identifiers such as *@26DS-KX@*, and family identifiers such as *@00206642@*. These codes obscure template structures in the database—the uniqueness of each code means that no phrases can be formed that involve them. For example, the line *0 @26DS-KX@ INDI* in Figure 1d occurs only once, as do all other *INDI* lines in the file, and so the fact that *0* and *INDI* always occur together is obscured: SEQUITUR cannot take advantage of it. In fact, the token *INDI* occurs 33 000 times in the rules of the grammar, and in every case it could have been predicted with 100% accuracy by noting that *0* occurs two symbols previously, and that the code is in the individual identifier format.

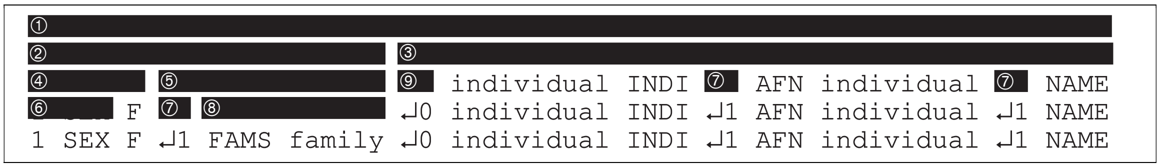
This prediction can be implemented by replacing each code with the generic token *family* or *individual*, and specifying the actual codes that occur in a separate stream. Replacing the code in the example above with the token *individual* yields the sequence *0 individual INDI* , which recurs many thousands of times in the file and therefore causes a grammar rule to be created. In this grammar, *INDI* occurs in a rule that covers the phrase *↵0 individual INDI↵1 AFN individual↵1 NAME* (↵ denotes the end of line). This is now the only place that *INDI* occurs in the grammar.

Overall, the number of rules in the grammar halves, as does the length of the top-level rule, and the total number of symbols. The compressed size of the grammar falls from 1.07 Mb to 0.60 Mb. The total size of the two files specifying the individual and family codes is 0.40 Mb, bringing the total for the word indexes to 1.0 Mb, a 7% reduction. Including the dictionaries gives a total of 1.11 Mb to recreate the original file.

Separating the dictionaries represents the use of some domain knowledge to aid compression, so comparisons with general-purpose compression schemes is unfair. For a

① → ②③
② → ④⑤
③ → ⑨individual INDI⑦AFN individual⑦NAME
④ → ⑥ F
⑤ → ⑦⑧
⑥ → 1 SEX
⑦ → ↵1
⑧ → FAMS family
⑨ → ↵0

b



c

... ① Sybil Louise /MCGHIE/ (① Eliza) Jane /LOSEE/ (① Margaret) /SIMMONS/ (① Marie) Elizabeth /BERREY/ ① Athena Augusta /ROGERS/ (① William) Henry /WINN/ ...

Figure 3 A phrase: (a) hierarchical decomposition; (b) graphical representation; (c) examples of use

this reason, PPMC was applied to the same parts as SEQUITUR, to determine what real advantage SEQUITUR provides. PPMC was first applied in a byte-oriented manner to the sequence of word indexes. It compressed these to 1.07 Mb, far worse than SEQUITUR's 0.60 Mb. In an attempt to improve the result, PPMC was run on the original file with generic tokens for codes, yielding a file size of 0.85 Mb—still much worse than SEQUITUR's result. Note that this approach does outperform running PPMC on the unmodified file. Finally, the WORD-1 scheme was applied to the sequence of generalized codes, but the result was worse still.

INTERPRETING THE GRAMMAR

We now turn to the structure of the grammar. Occam's razor asserts that simpler theories should be preferred to more complex ones. Although this implies that compressed representations provide good theories about the structure of a sequence, most compression schemes produce models that are incomprehensible. SEQUITUR's model, on the other hand, has a natural, readable representation as a grammar. We can gain insight into its performance by examining the model that SEQUITUR builds.

Figure 3a shows nine of the 71 000 rules in SEQUITUR's original grammar, with ungeneralized codes, renumbered for clarity. Rule ① is the second most widely used rule in the grammar: it appears in 261 other rules.[1] The other eight rules are all those that are referred to, directly or indirectly, by rule ①: Figure 3b shows the hierarchy graphically. The topmost black bar in Figure 3b represents rule ①. The next two bars are rules ② and ③, the contents of rule ①. The hierarchy continues in this way until all of the rules have been expanded.

Rule ① represents the end of one record and the beginning of the next. Rule ⑨ is effectively a record separator (recall that each new record starts with a line at level 0), and this occurs in the middle of rule ①. Although grouping parts of two records together achieves compression, it violates the structure of the database, in which records are

[1] The most widely used rule is *2 PLAC*, which occurs 358 times, indicating that the text surrounding the place tag is highly variable. However, the structure of the rule itself is uninteresting.

integral. However, the two parts are split apart at the second level of the rule hierarchy, with one rule, ②, for the end of one record, and another, ③, for the start of the next. The short rules ⑦ and ⑨ capture the fact that every line begins with a nesting level number. There is also a rule for the entire SEX field indicating the person is female, which decomposes into the fixed part: *1 SEX*, and the value *F* on the end, so that the first part can also combine with *M* to form the other version of the SEX field. There is a similar hierarchy for the end of a male record, which occurs 259 times.

As for the usage of this rule, Figure 3c shows part of the top-level rule. Here, rules have been expanded for clarity: parentheses are used to indicate a string which is generated by a rule. This part of the sequence consists mainly of rule ① in combination with different names. Separate rules have been formed for rule ① in combination with common first names.

AUTOMATIC GENERALIZATION

In order to automate the process of identifying situations where generalization is beneficial, it is first necessary to define the precise conditions that give rise to possible savings. In the case described above, the rule *INDI ↵1 AFN* occurred many times in the grammar, and accounted for a significant portion of the compressed file. Conditioning this phrase on a prior occurrence of *↵0* greatly increases its predictability. The problem is that other symbols may be interposed between the two. One heuristic for identifying potential savings is to scan the grammar for pairs of phrases where the cost of specifying the distances of the second relative to the first (predictive coding) is less than the cost of coding the second phrase by itself (normal coding).

Figure 4 gives two illustrations of the tradeoff. In the top half of Figure 4, using normal coding, the cost of coding the *B*s is three times the cost of coding an individual *B*: $\log_2$(frequency of *B* / total symbols in grammar) bits. For predictive coding, the statement “A predicts B” must be encoded once at the beginning of the sequence. Reducing this statement to a pair of symbols, *AB*, the cost is just the sum of encoding *A* and *B* independently. Each time that *A* occurs, it is necessary to specify the number of intervening symbols before *B* occurs. In the example, *A*<3> signifies that the next *B* occurs after three intervening symbols. These distances are encoded using an adaptive order-0 model with escapes to introduce new distances.

The bottom half of Figure 4 shows a more complex example, where two *A*s appear with no intervening *B*, and a *B* occurs with no preceding *A*. The first situation is flagged by a distance of ∞, and the second is handled by encoding *B* using normal coding.

Table 2 lists pairs of phrases ranked according to the number of bits saved by using one phrase to predict the other. At the top of the list is the prediction *↵0⇒ INDI ↵1 AFN*, which is the relationship that we exploited by hand—the intervening symbol between *↵0* and *INDI ↵1 AFN* is an individual code. Predictions 2, 3 and 6 indicate that the codes should be generalised after the *AFN*, *FAMS* and *FAMC* tags respectively. Taken together,

	manual	predictive
to encode the Bs in	encode ...B...B...B...	encode “A predicts B”, and A<3>...A<4>...A<4>
AabcB...AdefgB...AhijkB	cost: 3 × entropy(B)	cost: entropy(A) + entropy(B) + entropy(3, 4, 4)

	manual	predictive
to encode the Bs in	encode ...B...B...B	encode “A predicts B”, and A<3>...A<∞>...A<4>...B
AabcB...A...AhijkB...B	cost: 3 × entropy(B)	cost: entropy(A) + entropy(B) + entropy(3, ∞, 4) + entropy(B)

Figure 4 Examples of two ways of coding symbol *B*

	Prediction	Normal (bits/symbol)	Predicted (bits/symbol)	Saving (total bits)
1	↲ 0 ⇒ INDI ↲ 1 AFN	3.12	0.01	2297.90
2	INDI ↲ 1 AFN ⇒ ↲ 1 NAME	3.12	0.01	2297.23
3	FAMS ⇒ ↲ 0	2.25	0.81	638.30
4	↲ 1 SEX ⇒ ↲ 2	0.96	0.06	655.90
5	BAPL ⇒ ↲ 1 ENDL	1.57	0.76	508.53
6	FAMC ⇒ ↲ 1 FAMS	2.57	1.47	427.14
7	↲ 1 BIRT↲ 2 DATE ⇒ ↲ 2 PLAC	1.88	1.11	381.96
8	↲ 1 NAME ⇒ ↲ 1 SEX	1.25	0.82	315.42
9	ENDL ⇒ ↲ 1 SLGC	2.15	1.58	265.88

Table 2: Predictions based on part of the GEDCOM database

these four predictions achieve the 7% improvement described in the last section. Predictions 5, 7 and 9 indicate that dates should be generalised. Doing this by replacing dates with the token *date*, and encoding the sequence of dates in a separate file, reduces the total size of the compressed files a further 5%. Prediction 4 indicates that the *SEX* field can be generalised by replacing the two possible tags, *F* and *M*. Acting on this reduces the size by a further 1%. Finally, prediction 8 indicates that names should be generalised, resulting in a final compressed size of 1.04 Mb, or a ratio of almost nine to one. The final block of Table 1 summarizes these improvements.

SEQUITUR produces deterministic grammars, and this process is analogous to adding non-deterministic rules to the grammar. The new non-terminal is the generic token (e.g. *date)*, and it heads many rules, each right-hand side being one date. It is planned to integrate this generalisation process into SEQUITUR.

7 Conclusion

Semi-structured text proves to be an interesting domain for compression, and one that provides a strong incentive for hierarchical grammar-based techniques such as SEQUITUR. Although there is a huge range and variety of semi-structured text, we have gained significant insight by studying one particular example in detail—insight that we believe will apply widely to different genres of semi-structured text. We have shown that the grammar-based approach outperforms the best text compression methods—by 20%, in this example; that significant improvement can be obtained by performing some small generalizations on the grammar—giving a further 10% improvement; and that opportunities for such generalizations can be detected automatically. The full method achieves compression of nearly 9:1 on this data, considerably better than PPM's 6.5:1.

References

Bell, T.C., Cleary, J.G. and Witten, I.H. (1990) *Text Compression.* Prentice Hall, Englewood Cliffs, NJ.

Nevill-Manning, C., Witten, I.H. and Maulsby, D. (1994) "Compression by induction of hierarchical grammars." *Proc Data Compression Conference*, edited by J.A Storer and M. Cohn. IEEE Press, pp. 244–253.

Witten, I.H., Moffat, A., and Bell, T.C. (1994) *Managing Gigabytes: Compressing and indexing documents and images*, Van Nostrand Reinhold, New York.

"GEDCOM Standard: Draft release 5.4", Family History Department, The Church of Jesus Christ of Latter-day Saints, Salt Lake City, Utah.

Ziv, J. and Lempel, A. (1977) "A universal algorithm for sequential data compression," *IEEE Trans. Information Theory, IT-23* (3), 337-343, May.

Ziv, J. and Lempel, A. (1978) "Compression of individual sequences via variable-rate coding," *IEEE Trans. Information Theory, IT-234*(5), 530-536, September.