Fast Convergence with a Greedy Tag-Phrase Dictionary

(EXTENDED ABSTRACT)

Tony C. Smith and Ross Peeters
Computer Science, University of Waikato, Hamilton, New Zealand.
tcs@cs.waikato.ac.nz

1 Introduction

The best general-purpose compression schemes make their gains by estimating a probability distribution over all possible next symbols given the context established by some number of previous symbols. Such context models typically obtain good compression results for plain text by taking advantage of regularities in character sequences. Frequent words and syllables can be incorporated into the model quickly and thereafter used for reasonably accurate prediction. However, the precise context in which frequent patterns emerge is often extremely varied, and each new word or phrase immediately introduces new contexts which can adversely affect the compression rate.

A great deal of the structural regularity in a natural language is given rather more by properties of its grammar than by the orthographic transcription of its phonology. This implies that access to a grammatical abstraction might lead to good compression. While grammatical models have been used successfully for compressing computer programs[4], grammar-based compression of plain text has received little attention, primarily because of the difficulties associated with constructing a suitable natural language grammar. But even without a precise formulation of the syntax of a language, there is a linguistic abstraction which is easily accessed and which demonstrates a high degree of regularity which can be exploited for compression purposes—namely, lexical categories.

For example, an explicit-context model is able to take advantage of the re-occurrence of, say, *the three little pigs* in a particular text. But if the compressor subsequently comes across *the three large pigs*, the word *large* will generate a new context and thus inhibit the model's ability to predict *pigs* with any certainty—all other things being equal. If, however, it is known that *the* is a determiner, that *three*, *little* and *large* are adjectives and that *pigs* is a noun, and if the context recorded for *the three little pigs* is the category sequence *determiner adjective adjective noun*, then the context predicts each phrase equally well. Certainly some penalty must be incurred to encode the new word, but by transferring some of the entropy to the lexical category itself the compression algorithm can recover more quickly from minor perturbations in the explicit sequence. That is, for the purpose of prediction it is often more useful to know how often the pattern *determiner adjective adjective noun* occurs in a text than how often *the three little pigs* occurs, and this can be used to achieve good compression.

Lexical categories have been shown to assist in giving good compression results when incorporated into context models[5, 6]. This paper describes a greedy *dictionary-based* model that maintains a dictionary of tag-phrases, along with separate lexicons for each unique tag. The text is tagged with part-of-speech (POS) labels and then given to the encoder,

the/D grey/A cat/N saw/V a/D black/A dog/N

the/D dog/N saw/V the/D cat/N

the/D dog/N chased/V the/D grey/A cat/N

Figure 1: Three sample sentences after tagging.

which uses the tags to construct the phrase dictionary in a manner similar to LZ78. The output is a sequence of arithmetically encoded phrase numbers coupled with the information needed to match the correct word with each tag in the phrase. Each unique word (defined as each novel word/tag pair) is transmitted once when it is first encountered, then retained in the appropriate dictionary and thereafter arithmetically encoded according to the empirical distribution for that dictionary whenever the word is encountered.

We present results from some empirical tests showing that this "tag-phrase dictionary" technique (hereafter referred to as TPD) achieves nearly identical compression as that obtainable using PPM, an explicit-context model. This goes against the widely held view that greedy dictionary schemes require much larger samples of text before they can compete with statistical context methods[1]. Some interesting theoretical issues pertaining to text compression in general are implied, and these are also discussed.

2 The algorithm

Before the text is encoded, it is passed through a tagger which labels each word with an appropriate POS category—for example, *cat* might be tagged as a noun (N), *chased* as a verb (V), *black* as an adjective (A), *the* as a determiner (D), and so forth, as shown in Figure 1. Words belonging to more than one lexical category will ideally be tagged according to their syntactic role in each instance where they occur, but all ambiguity is handled by the tagger itself. The AMALGAM tagger was used for the results given in this paper.

The phrase dictionary

The tagged-text is given to the encoder as a stream of *word/tag* pairs. The encoder parses the input into a dictionary of phrases in a manner similar to LZW (a variant of LZ78 proposed by Welch[7]), where each phrase is constructed as a trace of lexical tags from the input. The phrase dictionary is maintained as a multi-way trie, initialised to include each unique tag as an individual phrase. As each word/tag pair is read, the encoder finds the longest matching trace for the tag sequence and then arithmetically encodes the appropriate phrase number according to its frequency at that point. A new phrase is created by appending the first unmatched tag as a leaf node to the phrase just encoded. The process continues from the root of the trie looking for the longest trace that begins with the last unmatched symbol.

The decoder works by building the trie in a complementary manner. Starting with the same initial trie structure, new phrases are added by appending the first tag of each new phrase as a leaf to the node corresponding to the last symbol of the previous phrase.

Lexical dictionaries

Given that a phrase merely depicts a sequence of lexical categories, additional information is encoded with the phrase number to identify the correct word for each tag. When a new word/tag pair is encountered, the word is explicitly encoded for output and then added to the appropriate dictionary as given by the tag. If the word is already in the dictionary, it is arithmetically encoded according to the empirical distribution for that category derived from counts maintained for each word. The output from the encoder is thus a variable-length tuple of the form $(\sigma_j, w_1, w_2, ..., w_n)$, where σ_j is the encoded phrase number and w_i is the encoded word for the i-th category symbol in σ_j.

The decoder maintains lexical dictionaries in a complementary manner, adding new words as they are encountered and maintaining the necessary counts for deciphering the adaptive arithmetic codes.

Encoding example

The process of encoding the three sentences in Figure 1 is demonstrated in Figure 2. The system is initialised with a trie consisting of a root node with leaf nodes for each possible tag that can occur in the input (Figure 2a), and the lexical dictionaries are empty. Analogous to LZW, both the encoder and decoder must know in advance the minimum size of the tag set. This can be avoided by presuming a much larger tag set than is likely to be used as the statistical burden of the unused branches rapidly becomes insignificant.

The encoding starts by reading in the first tuple ("the", D) (Figure 2b). A match on the tag causes the algorithm to move down a level to the phrase labelled 1. No match is found for the next tuple ("grey", A) as there is no A branch from phrase 1. A new node is appended and given the next available phrase number. The phrase number for the last matching node (phrase 1) is encoded, along with codes identifying the words associated with its tags. Since "the" is not in the determiner dictionary, it is explicitly encoded and then added to the dictionary.

Moving back to the root of the trie, the unmatched tuple is still waiting to be processed. A match with tag A causes the algorithm to move down to the node labelled 2. The next tuple ("cat", N) fails to find a match, so a new node is appended and given the next available phrase number. Phrase 2 is encoded, along with an explicit code for the word "grey" which is subsequently added to the dictionary for adjectives.

To get a better feel for the algorithm, we pick up the process when it is about to encode the last four words (Figure 2c). At the root of the trie, the tuple ("chased", V) is waiting to be encoded and a match causes the encoder to move down to node 4. A match with the next pair ("the", D) causes a move down to the phrase represented by node 8. No match is found for the next pair ("grey", A) so a new node is created, node 15, and appended to the phrase. The output consists of the last phrase number reached, phrase 8, and the words associated

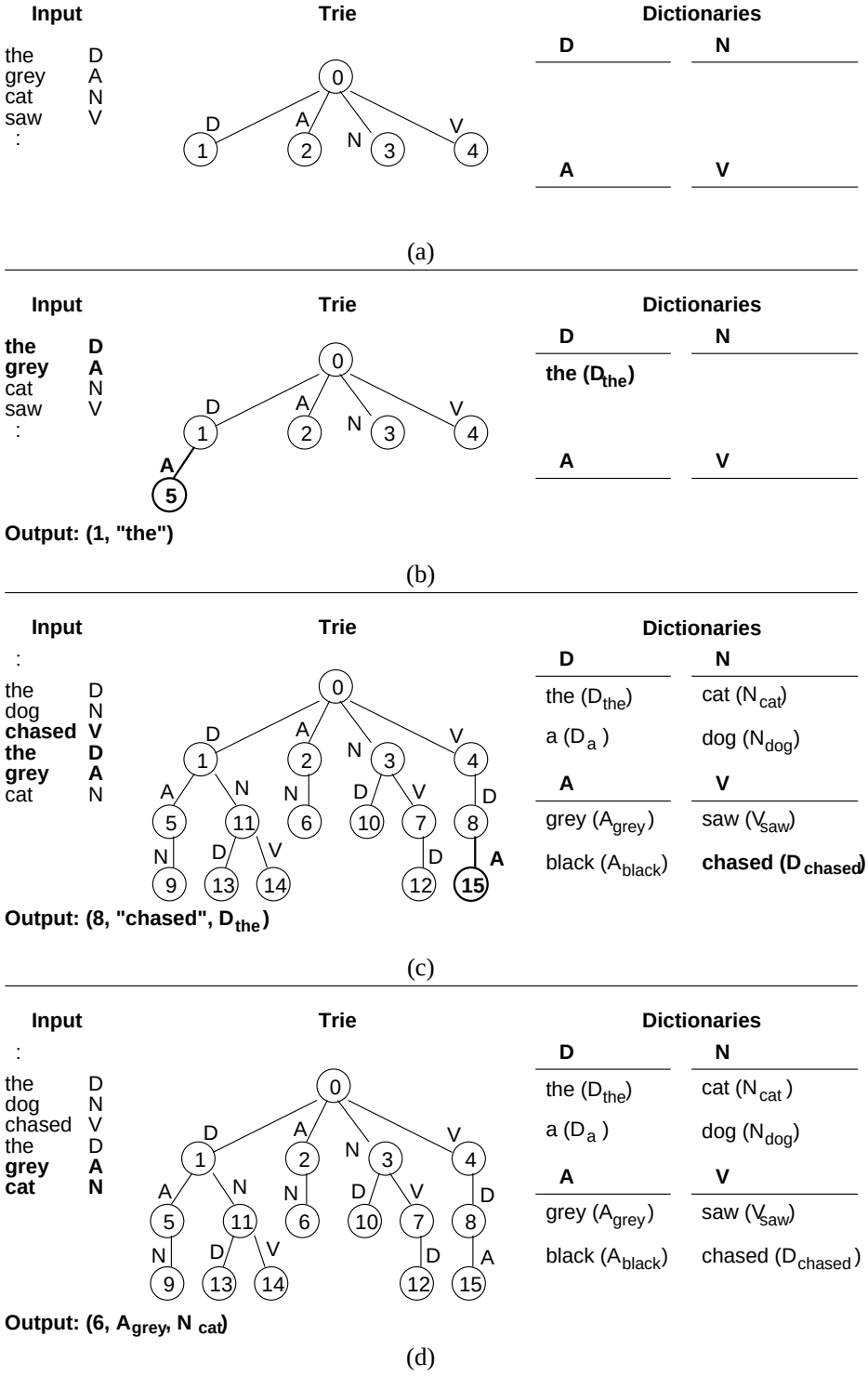

Figure 2: Example of the encoding process

with its tags. As "chased" is a new verb, it is explicitly encoded and added to the verb dictionary. Because "the" is already in the dictionary for determiners, it is arithmetically encoded (represented in the figure as D_{the}).

The encoder returns to the root of the trie, and a match with the remaining two tuples ("grey", A) and ("cat", N) brings it to node 6 (Figure 2d). Both words are already in their respective dictionaries, thus the output is a tuple consisting of an arithmetic code for phrase 6 and for the two words (depicted in the figure as A_{grey} and N_{cat}).

Decoding

Decoding the compressed text is done in a similar manner to encoding. As each tuple is read, the path in the trie corresponding to the phrase number is traversed. Any new word is taken from the input and added to the dictionary given by its corresponding tag in the phrase. The arithmetic code for a known word can be derived from its frequency, allowing it to be retrieved from the appropriate dictionary. The trie is built up by appending the first tag of each subsequent phrase to the node corresponding to the last tag of the previous phrase.

3 Compression performance

Experiments were conducted on seven different corpora to compare the performance of *TPD* with three other well-known compression schemes: *UNIX compress*, a variant of LZ78; *gzip*, a variant of LZ77; and *PPMD5*, a fifth-order context model using escape method D. The texts were tagged by the AMALGAM project—an e-mail service which uses an automatic tagging mechanism based upon Brill's [2] rule-based tagger. While TPD is capable of lossless compression over the full range of ASCII values, all of the texts were converted to 27-letter English ('a' to 'z' and a blank space) to alleviate problems associated with tagging interword character sequences.

Results from the experiments are depicted in Figure 3. They show that the context-based scheme of PPMD5 consistently outperforms the dictionary-based schemes of *compress* and *gzip*, which might initially suggest that the two general paradigms could be classed according to their compression performance. However, TPD, another dictionary-based scheme, surprisingly performs comparably to the context-based approach. It is more than 30% better than *compress* and *gzip* and only 4% worse than PPMD5 on average.

Effects of tagging accuracy

Automatic taggers typically claim to give about 95% accuracy[3]. To determine the effect of tagging accuracy on compression, an experiment was devised to compare the results for automatically-tagged texts against those obtained for their manually-checked versions. The texts used were the LOB Corpus tagged with the LOB tag set and the Wall Street Journal tagged with the PENN tag set. The results are shown in Table 1.

A perhaps surprising result is that the computer-tagged texts compress just as well as their manually-tagged versions—ever so slightly better, in fact. This may be because the

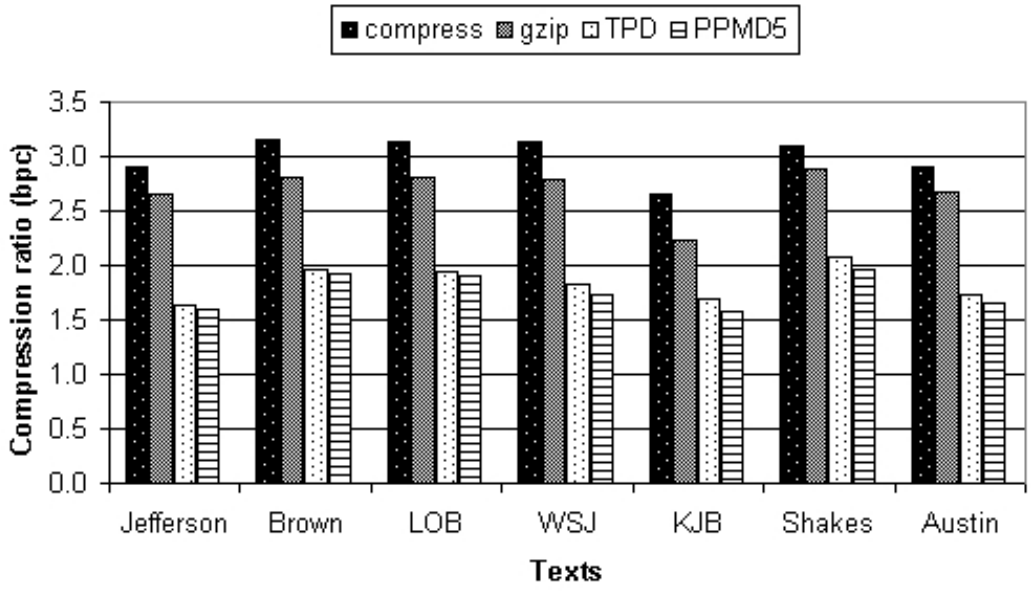

Figure 3: Comparison of four compression methods.

automatic tagger is rule-based, providing greater regularity in the way a tag is used and a more uniform distribution of tags, both of which are advantageous for TPD.

Tagged-Text	**Manual (bpc)**	**Computer (bpc)**
LOB Corpus	1.950	1.949
Wall Street Journal	1.838	1.838

Table 1: Results of computer tagged and manually checked texts

Improved lexical coding

The semiadaptive lexical dictionaries used by TPD in the previous experiments do not store or predict words in a particularly sophisticated manner. Words are stored explicitly, and their arithmetic codes are derived simply from their observed frequencies within each category. In an effort to improve this aspect of the model, the dictionaries were modified to use the previous word from the input to help predict the next word from the specified category—a kind of bigram technique based upon Teahan and Cleary's WTW method[6]. The results, depicted in Figure 4, show an across the board improvement, even outperforming PPMD5 in some cases.

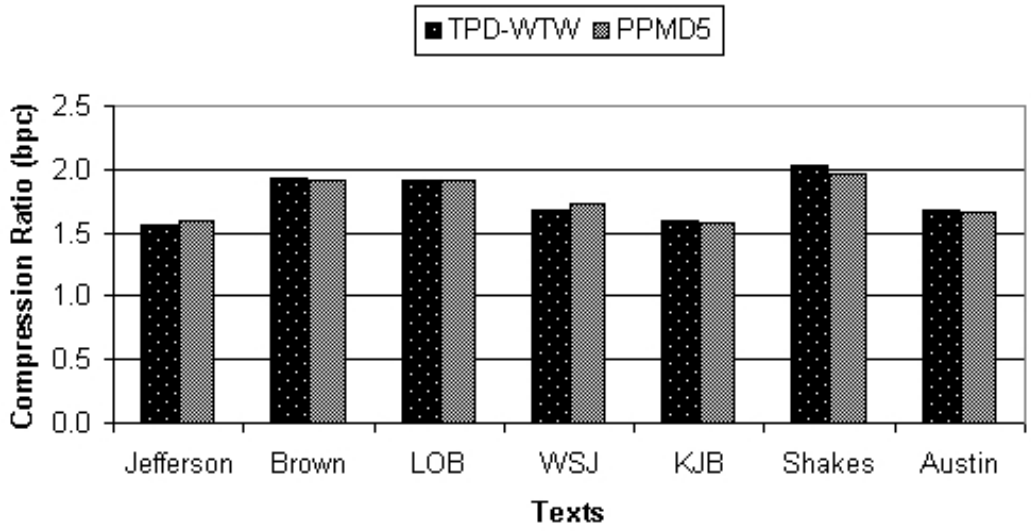

Figure 4: TPD results using bigram lexical prediction.

Modeling each category with a letter-based trie (in a manner similar to that described by Teahan and Cleary) was also considered to try and take advantage of morphological regularities and possibly gain additional compression. In the end, however, it was decided that outperforming context models by marginal amounts was not a satisfactory research goal. The more interesting result is that a greedy dictionary method can be made to achieve comparable compression to statistical context methods for the same sample text.

4 Analysis

It is not clear why TPD's compression performance should be so similar to that of PPMD5. This section describes two possible explanations: one relates to an abstract similarity in the compression models; the other relates to a possible equivalence in entropy for two different levels of linguistic abstraction.

The partition trade-off

A context in PPMD5 defines a set of possible next symbols—a category, if you like—over which it adaptively maintains a distribution. Because each context is given as an explicit sequence of symbols, the number of categories tends to be quite large, but the number of symbols in each category tends to be quite small. In contrast, TPD maintains a small number of categories, but at the price of having some of them contain a large number of symbols.

There is an expected trade-off between the number of categories and a model's ability to predict any one symbol. More tags mean more phrases, which leads to longer phrase codes, but fewer words for each tag, meaning shorter word codes. Fewer tags, however, mean better prediction of phrases and thus shorter phrase codes, but the number of words for each tag increases thereby widening the probability distribution and increasing the average word code length.

An experiment was devised to examine how the size of the tag set affects compression results. Three versions of a manually checked LOB Corpus were used, each tagged with a different variation of the LOB tag set. The first used the original set, where root categories (such as adjectives) are broken down into specialised subgroups (such as superlatives, comparatives, and attributives). In the second version, some of the larger related root categories were collapsed together, and in the third all root categories were combined. The results, given in Table 2, indicate a marginal decrease in compression as the number of tags decreases. This could imply that an even more specialised tag set (such as the context-defined categories of PPM perhaps) might lead to even better compression.

Tag set	**Number of tags**	**Compression ratio (bpc)**
Original	153	1.950
Reduced	42	1.960
Base	25	1.967

Table 2: Results using different number of tags

Ergodicity in linguistic abstraction

The trade-off in the number of categories seems an insufficent explanation for the comparable compression ratios of PPM and TPD because the manner in which the categories are applied for prediction is vastly disparate. PPM uses a context to define a category from which it predicts the next symbol. Once that symbol has been predicted, the context is shuffled along by lopping off the first symbol of the previous context and appending the symbol just predicted. In this respect, the same amount of context is used to predict each symbol (ignoring escapes to lower-order models).

In contrast, TPD predicts the next phrase without using prior context, it disambiguates the categories in isolation, and then it throws everything away. Apart from counts, each phrase and word is predicted without reference to preceding symbols. The similar compression ratios from such different approaches may imply a fundamental equivalence in the entropy of two different linguistic abstractions—namely phonology and syntax.

Phonological regularity

Each language has its own phonetic vocabulary from which it constructs syllables and words. Allowable phonemic combinations are given by the phonological constraints of the lan-

Phrases produced by TPD	**Generalised phrases**
1. AT JJ NN	1. AT JJ NN
2. AT JJ JJ NN	2. AT JJ JJ NN
3. AT NN IN AT NN	3. AT NN (5)
4. IN AT JJ NN	4. IN (1)
5. IN AT NN	5. IN AT NN

Table 3: Common phrases identified by TPD

guage, and exhibit a great deal of structure and regularity. Orthographic systems are essentially phonological transcriptions, and thus preserve many of these regularities as character sequences. The PPM context-model records traces of character sequences and thus, in a sense, models the phonetic vocabulary and the phonological constraints given by the language. Character-based models are well-suited for capturing morphological structure, but do less well when it comes to making predictions across word boundaries.

Syntactic regularity

Each language also has syntactic constraints governing how words can be combined to form sentences. Like phonology, syntax also demonstrates a high degree of regularity, but it is not expressed in explicit sequences of symbols or sounds; rather it is manifest in a higher level of abstraction over word types and more subtle grammatical features. While the regular grammar constructed by TPD is, in general, too weak a formalism for sufficiently characterising natural language grammar, some syntactic regularities are captured in the model[1].

The left-hand column of Table 3 shows some of the most frequent phrases in the trie produced by compressing the LOB Corpus (*NN*–noun, *JJ*–adjective, *AT*–article, *IN*–preposition). These phrases are similar to the simple nounphrase and prepositional-phrase rules commonly given in standard CFG descriptions of English syntax, including some of the embedded structures—for example, phrase 1 is embedded in phrase 4 and phrase 5 is embedded in phrase 3, as shown in the right-hand column of the table. In this respect, TPD appears to be successful in capturing phrasal regularity. Given the abstract nature of dependencies between phrases, prediction from contexts that straddle phrase boundaries is not significant, thus TPD loses little when it throws the previous phrase away.

Entropic equivalence

That the same degree of compression is achievable from two different levels of linguistic abstraction implies a kind of ergodicity between the models. That is, the same level of

[1] Technically, the phrase dictionary of TPD is a Markov model. Each tag in a phrase can be replaced by a weighted finite state model for which each transition corresponds to a word in the lexical dictionary associated with that tag.

generalisation is achieved such that both models might be regarded as equally perspicuous accounts of linguistic structure—just different kinds of structure. While PPM generalises primarily over morphological syntax, TPD generalises over lexical syntax. In light of theories which attempt to draw analogies between compression and learning[8], the fact that the different models approach the same level of source entropy may pertain to a relationship between the problem of learning phonology and the problem of learning syntax—but this is difficult to substantiate.

The results from TPD satisfy a more pragmatic issue. The mathematical equivalence of dictionary-based models and context models is well established, but the results obtained from TPD now provide empirical evidence that their practical differences (for text compression, anyway) may have been superficial. The advantage previously enjoyed by context modeling was perhaps simply a consequence of applying greedy dictionary methods to the wrong level of abstraction.

5 Acknowledgements

We would like to thank Stuart Inglis, David Bainbridge and Geoff Holmes for their insightful comments. In addition, we would like to thank Bill Teahan for providing most of the texts, including the Jefferson corpus which he scanned himself, and for his assistance in carrying out this research.

References

[1] T.C. Bell, J.G. Cleary, and I.H. Witten. *Text Compression.* Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[2] E. Brill. *A corpus-based approach to language learning.* PhD thesis, University of Pennsylvania, 1993.

[3] Eugene Charniak. *Statistical language learning.* MIT Press, Massachusetts, 1993.

[4] R. Davies. Computer program compression. Master's thesis, University of Waikato, Hamilton, New Zealand, 1996.

[5] R. N. Horspool and G. V. Cormack. Constructing word-based text compression algorithms. In *Proceedings DCC'92*, pages 62–71. IEEE Computer Society Press, 1992.

[6] W. J. Teahan and J. G. Cleary. Models of english text. In J.A. Storer and M. Cohn, editors, *Proceedings of DCC '97*. IEEE Computer Society Press, 1997.

[7] T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[8] J. G Wolff. Language acquisition, data compression and generalization. *Language and Communication*, 2(1):57–89, 1982.