Offline Dictionary-Based Compression

N. Jesper Larsson* and Alistair Moffat†

1. Introduction

Dictionary-based modelling is the mechanism used in many practical compression schemes. For example, the members of the two Ziv-Lempel families parse the input message into a sequence of phrases selected from a dictionary, and obtain compression since a reference to the phrase can be more compact than the phrase itself.

In most implementations of dictionary-based compression the encoder operates *online*, incrementally inferring its dictionary of available phrases from previous parts of the message, and adjusting its dictionary after the transmission of each phrase. Doing so allows the dictionary to be transmitted implicitly, since the decoder simultaneously makes similar adjustments to its dictionary.

An alternative approach – the topic explored in this paper – is to use the full message (or a large block of it) to infer a complete dictionary in advance, and include an explicit representation of the dictionary as part of the compressed message. Intuitively, the advantage of this *offline* approach is that with the benefit of having access to all of the message, it should be possible to optimize the choice of phrases so as to maximize compression performance. Indeed, we demonstrate that very good compression can be attained by an offline method without compromising the fast decoding that is a distinguishing characteristic of dictionary-based techniques.

Several nontrivial sources of overhead – in terms of both computation resources required to perform the compression, and bits generated into the compressed message – have to be carefully managed as part of the offline process. To meet this challenge, we have developed a novel phrase derivation method and a compact dictionary encoding. In combination these two techniques produce the compression scheme RE-PAIR, which is highly efficient, particularly in decompression.

It should also be noted that while offline compression involves the disadvantage of having to store a large part of the message in memory for processing, the difference between doing this and storing the growing dictionary of an online compressor is illusory. Indeed, incremental dictionary-based algorithms maintain an equally large part of the message in memory as part of the dictionary; similarly, online predictive symbol-based context models occupy space that may be linear in the size of that part of the message on which prediction is based.

Our scheme is offline only while inferring the dictionary, and during decompression bits are read and phrases written in a fully interleaved manner. Moreover, during decoding only a compact representation of the dictionary must be stored. Thus, during decompression, our approach has a space advantage over both incremental dictionary-based schemes and over context-based source models.

*Department of Computer Science, Lund University, Sweden.
†Department of Computer Science, The University of Melbourne, Australia.

2. Dictionary-Based Compression

The goal of dictionary-based modelling to derive a set of *phrases* that can be used to economically represent the message. Furthermore, since in an offline method the phrase table must be transmitted as part of the compressed message, the derivation scheme used should allow a compact encoding of the phrase set. This latter requirement does not apply to incremental dictionary-based methods.

2.1. Previous Approaches. To facilitate a compact encoding of the phrase table we employ a *hierarchical* scheme where longer phrases are encoded through references to shorter ones. This is in some ways similar to the LZ'78 mechanism [**12**], and the extension developed by Miller and Wegman [**5**]. The drawback of the aggressive phrase construction policies of LZ'78 mechanisms is that the dictionary is diluted by phrases that do not in fact get productively used. In our proposal every phrase is used at least twice during the coding of the message.

Our derivation scheme is also related to the grammar-based compression method SEQUITUR of Nevill-Manning and Witten [**9**]. In SEQUITUR the input message is processed incrementally, and rules in a context-free grammar are created and then revised in a symbol-by-symbol manner. But because SEQUITUR processes the message in a left-to-right manner, and maintains its two invariants (uniqueness and utility) at all times, it does not necessarily choose as grammar rules the phrases that might eventually lead to the most compact representation. Hence, SEQUITUR is best categorized as an online algorithm with strong links to the LZ'78 family, and the obvious question is whether a holistic approach to constructing a grammar to represent the message can yield better compression.

Our scheme also has points in common with the compression regime described by Manber [**4**]. To obtain fast searching of compressed text, Manber considers a compression mechanism based upon character digrams, for both the text and the pattern. We also replace frequent pairs, but continue the process recursively until no more pairs of symbols can be reduced. Hence the name of our program, RE-PAIR, for *recursive pairing*.

2.2. Recursive Pairing. The phrase derivation algorithm used in RE-PAIR consists of replacing the most frequent pair of symbols in the source message by a new symbol, reevaluating the frequencies of all of the symbol pairs with respect to the extended alphabet, and then repeating the process until there is no pair of adjacent symbols that occurs twice:[1]

1. Identify symbols a and b such that ab is the most frequent pair of adjacent symbols in the message. If no pair appears more than once, stop.
2. Introduce a new symbol A and replace all occurrences of ab with A.
3. Repeat from step 1.

The message is reduced to a new sequence of symbols, each of which represents either a unit symbol or a pair of recursively defined symbols. A zero-order entropy

[1] In independent work, Nakamura and Murashima [**8**] propose a scheme that shares this pair substitution with our own, but differs in representation of the phrase table and message encoding.

code for the reduced message is the final step in the compression process; and the penultimate step is, of course, transmission of the dictionary of phrases.

We have not specified in which order pairs should be replaced when there are several pairs of equal maximum frequency. While this does influence the outcome of the algorithm, it appears to be of minor importance. The current implementation resolves ties by choosing the least recently accessed pair, which avoids skewness in the hierarchy by discriminating against recently created pairs.

2.3. Implementation. This section describes an implementation that accomplishes phrase derivation in overall linear time and space. Many options, particularly influencing space requirements, are available; for brevity a number of alternatives are omitted. It is supposed throughout that the original message is n symbols long.

The sequence of symbol numbers is stored as records in an array. Each record contains three words, one of which holds the symbol number. The other two words are used to thread the records into a series of doubly linked lists, one for each *active pair*, which denotes a combination of two adjacent symbols that is still under consideration for replacement by a single symbol.

To gain constant-time access to pairs, a hash table is maintained, with an entry for each active pair, and a pointer to the first appearance of each active pair in the symbol array. The third access structure is a specialized priority queue, implemented as an array of $\lceil \sqrt{n}\, \rceil - 1$ linked lists recording the active pairs that occur less than $\lceil \sqrt{n}\, \rceil - 1$ times, and one linked list recording the more frequent ones. The hash table and priority queue make use of the same set of underlying records, each of which holds a counter for the number of occurrences of that active pair, as well as the pointer to the first location at which that pair occurs.

When the count of a pair decreases as a result of its left or right part being absorbed in a pair replacement, that pair either remains on the final priority list or is moved to a list residing at a lower index in the array. That is, the count of any existing active pair never increases. Moreover, the count of any new active pairs introduced during the replacement process cannot exceed the count of the pair being replaced. Hence, the maximum count is a monotonically decreasing entity, and locating the next most-frequent active pair can be done in constant time per pair reduced. The total time consumed by all executions of step 1 is $O(n)$.

To account for the replacement operations in step 2, observe that since the length of the sequence decreases for each replacement, the total number of replacements is $O(n)$. Replacement of a single appearance of pair ab by a new symbol A involves the following sequence of operations, each of which must be accomplished in constant time:

- A. Locate the first or next sequence entry associated with ab. Identify the adjacent symbols x and y to establish the context $\ldots xaby \ldots$.
- B. Decrement the counts of the adjacent pairs xa and by.
- C. Replace ab in the sequence, leaving $\ldots xAy \ldots$.
- D. Increase the counts of the pairs xA and Ay, creating records for them and adding them to the hash table and priority queue if this is the second time they have been generated during the processing of the ab pairs.

Care must be taken for sequences of identical symbols. For example, replacing aa with A should yield two occurrences of A for the subsequence $aaaa$, not three. This is a simple matter of remembering the last few positions encountered in scanning or replacing pairs, and excluding any pair that overlaps one that was just counted.

Operations A and C can be accomplished in $O(1)$ time. Initially the array is dense, and the symbols that are immediately adjacent to pair ab are readily determined. However, adjacency in a physical sense cannot always be guaranteed, since step C of each pair replacement leaves one record vacant. Indeed, after a series of replacements there may be many now-unused array records between the records for two consecutive symbols. To deal with this problem, the empty space is also threaded, allowing gaps to be traversed in $O(1)$ time. Finally, in steps B and D, entries are moved from one linked list in the priority queue to another. These movements can also be performed in $O(1)$ time, because each pair record includes the index of the list that contains it.

2.4. Memory Space. In the full paper we show that application of a periodic compaction process allows the total memory space during encoding to be restricted to $5n + 4k^2 + 4k' + \lceil \sqrt{n}\,\rceil$ words, where n is the number of symbols in the source message, k is the cardinality of the source alphabet; and k' is the cardinality of the final dictionary. In practice both k and k' are small relative to n. In the decoder two words of memory are required for each phrase in the hierarchy.

3. Compression Effectiveness

3.1. Symbolwise Equivalent. To understand the structure of dictionary-based models it is helpful to consider the structure of their *symbolwise equivalent* models – models that process one character at a time with an entropy coder [**1, 3**].

Consider the final sequence of phrases. Suppose that there are k' distinct phrases in the sequence, and n' phrases in total. Then each occurrence of a phrase that appears ℓ times in the final sequence generates approximately $-\log_2(\ell/n')$ bits in the compressed message, since the final phrases are entropy coded. Let one such phrase, of length r, be described by $x_1 x_2 \ldots x_r\$$, where \$ represents an *end of phrase* symbol, and let ℓ be its frequency. Let $N(c \mid s)$ be the number of phrases in the final sequence (of the n') which have sc as a prefix. For example, $N(\mathrm{a} \mid \lambda)$ is the count of the number of phrases that commence with character 'a' (λ is the empty string) and $N(\mathrm{b} \mid \mathrm{a})$ is the number of phrases that commence with 'ab'.

Now consider the expression

$$-\log_2 \frac{N(x_1 \mid \lambda)}{n'} - \log_2 \frac{N(x_2 \mid x_1)}{N(x_1 \mid \lambda)} - \cdots - \log_2 \frac{N(\$ \mid x_1 x_2 \cdots x_r)}{N(x_r \mid x_1 x_2 \cdots x_{r-1})} ,$$

which telescopes to

$$-\log_2 \frac{N(\$ \mid x_1 x_2 \cdots x_r)}{n'} = -\log_2 \frac{\ell}{n'} ,$$

since $N(\$ \mid x_1 x_2 \cdots x_r) = \ell$, the number of times the phrase $x_1 x_2 \cdots x_r$ appears. That is, the overall code for each phrase can be interpreted as a zero-order code for the first symbol, with probabilities evaluated relative to the commencing letters of the set of phrases; followed by a first-order probability for the second symbol, with probabilities evaluated in the context of first letters of the set of phrases, and so on.

3.2. Sources of Redundancy. Given the symbolwise equivalent, it is unlikely that the RE-PAIR mechanism can outperform a well-tuned context-based model, since the latter uses a high-order prediction for every character in the message, whereas, like other dictionary-based methods, RE-PAIR essentially resets its prior context to the empty string at the start of each phrase. However, the same improvements as have been suggested for other dictionary-based models can be used if compression effectiveness is to be maximized. For example, Gutmann and Bell [**2**] suggest that the probability for each phrase be conditioned upon the last character of the previous phrase. (A full first-order model on phrases is, of course, pointless.)

Another way in which compression effectiveness can be improved is to note that the final sequence contains no repeated symbol pairs, nor any pairs that constitute phrases in the dictionary. That is, if phrase-pair AB has previously appeared in the final message, or if $C = AB$ is in the phrase table (for some C), then, if phrase A occurs, the next phrase cannot be B. In this case phrase B, and any others that match the criteria, can be *excluded*, and the remaining probabilities adjusted upwards.

Both conditioning and exclusions are complex to implement, and if compression effectiveness (rather than compression efficiency) is the goal, then a full context-based mechanism is a better basic choice of algorithm. Our current implementation includes neither of these two improvements.

4. Encoding the Phrase Table

The hierarchical organization of the phrase table offers a natural way to encode it compactly as backward-referring pairs. This is achieved by encoding the phrases in *generations*, the first generation being the phrases that consist of two primitive symbols, the second generation being the phrases that are constructed by combining first-generation objects, and so on.

Let the primitive symbols be generation 0, and the number of items up to and including generation i be k_i. Define k_i for $i < 0$ to be zero, $k = k_0$ to be the size of the input alphabet, and s_i to be the size of generation i. That is, $k_i = \sum_{j \le i} s_j$. Finally, note that each phrase in generation i can be assumed to have at least one of its components in generation $i - 1$, as otherwise it could have been placed in a earlier generation. Therefore, the universe from which the phrases in generation i are drawn has size $k_{i-1}^2 - k_{i-2}^2$. Items are numbered from 0, so that the primitives have numbers 0 through $k - 1$, and generation i has numbers k_{i-1} through $k_i - 1$. Each item is a pair (l, r) of integers, where l and r are the ordinal symbol numbers of the left and right components.

Given this enumeration, the task of transmitting the dictionary becomes the problem of identifying and transmitting the generations; and to transmit the ith generation, a subset of size $s_i = k_i - k_{i-1}$, drawn from the range $k_{i-1}^2 - k_{i-2}^2$, must be represented. This section considers three strategies for the low-level encoding of the generations: arithmetic coding with a Bernoulli model, spelling out the pairs literally, and binary interpolative encoding.

l							
6	33	34	35	36	37	38	39
5	26	27	28	29	30	31	32
4	19	20	21	22	23	24	25
3	12	13	14	15	16	17	18
2				8	9	10	11
1				4	5	6	7
0				0	1	2	3
	0	1	2	3	4	5	6 r

l							
6	7	15	23	30	35	38	39
5	6	14	22	29	34	37	36
4	5	13	21	28	33	32	31
3	4	12	20	27	26	25	24
2				19	18	17	16
1				11	10	9	8
0				3	2	1	0
	0	1	2	3	4	5	6 r

FIGURE 1. Pair enumeration of the horizontal and chiastic slides respectively, when $k_{i-1} = 7$ and $k_{i-2} = 3$.

4.1. Bernoulli Model. If the s_i combinations that comprise the ith generation are randomly scattered over the $k_{i-1}^2 - k_{i-2}^2$ possible locations, then an arithmetic coder and a Bernoulli model will code the ith generation in the minimum possible number of bits. For efficiency reasons we do not advocate the use of arithmetic coding for this application; nevertheless, calculating the cost of doing so over all generations gives a good estimate of the underlying entropy of the dictionary, and is reported as a reference point in the experimental results below.

4.2. Literal Pair Enumeration. On the other hand, the most straightforward way of encoding the generations is as pairs of numbers denoting the ordinal numbers of the corresponding left and right elements, encoding $(l_1, r_1), (l_2, r_2), \ldots$ as the number sequence $l_1, r_1, l_2, r_2, \ldots$. A few optimizations to limit the range of these integers, and thereby reduce the required number of bits to encode them, are immediately obvious: numbers are contained in previous generations; pairs must have one of their elements in the immediately prior generation; and the pairs in each generation may be coded in lexicographic order. Given these observations, the left elements in the sequence grow slowly, with long sequences of equal elements, while the right elements are more varied. Experimentally, the most efficient encoding (of the ones tested) is to use a zero-origin gamma code (see Witten, Moffat, and Bell [**11**] for details) for transmitting the input alphabet as well the differences between successive left elements in the pair sequence; and a binary code for the corresponding right elements, tracking the remaining range (for the current left element).

4.3. Interpolative Encoding. There are many other ways of representing a subset of values over a constrained range. When the subset is expected to be non-random over the range – as is the case here – the interpolative coding method of Moffat and Stuiver [**6**] can be used. In this method a sorted list of integer values in a known range is represented by first coding the middle item as a binary number, and then recursively transmitting the left and right sublists, both within the narrowed range established by the value of the middle item. When the middle item lies towards one of the ends of the range, all subsequent codes in that section of the list will thus be shorter than if a gap-based mechanism had been used.

To actually encode the phrases with interpolative coding the two-dimensional pairs data must be converted to single numbers. A direct approach is to enumerate the possible pairs using the same lexicographically sorted ordering as the literal pairs. The resulting enumeration, which we call *horizontal slide*, is shown in the left half of Figure 1. The horizontal slide is not symmetric in its arguments, and any two-dimensional clusters in the matrix are broken up into several parts. Furthermore, the lower left part of the matrix can be expected to have the higher density, and the interpolative coding should be able to exploit this clustering. This leads to the enumeration shown in the right part of Figure 1, which we refer to as the *chiastic slide*. With this scheme, (l, r) in generation i gets number

$$\chi(l,r) = \begin{cases} 2l\,(k_{i-1} - k_{i-2}) + k_{i-1} - r - 1 & \text{when } l < k_{i-2}, \\ (2r+1)(k_{i-1} - k_{i-2}) + l - k_{i-2} & \text{when } r < k_{i-2}, \\ l\,(2k_{i-1} - l) + k_{i-1} - r - k_{i-1}^2 - 1 & \text{when } k_{i-2} \le l \le r, \\ r\,(2k_{i-1} - r - 2) + k_{i-1} - l - k_{i-1}^2 - 1 & \text{when } k_{i-2} \le r < l. \end{cases}$$

Calculating $\chi(l, r)$ is a costly operation if performed for each pair, and the closed form for $\chi^{-1}(x)$, for decoding, includes division as well as a square root. Fortunately, since the encoding is performed generation-wise and numbers are strictly increasing, values can be precomputed or accumulated, and incremental processing is fast.

5. Tradeoffs

5.1. Encoder. One way of saving encoding time, at the potential expense of compression effectiveness, is to bring pair replacement to a premature halt, and transmit the sequence as it stands at that time. The description above stipulated that pair replacement should continue until no pair occurs twice, but this *all the way* threshold can be modified if faster encoding, or a tighter bound on the dictionary space requirement, is required. The decoding algorithm is unaffected by this change, and acts in exactly the same manner as previously.

5.2. Decoder. The decoder offers a particularly convenient tradeoff between throughput and memory usage. The simplest decoding strategy is to, for each symbol number decoded, undertake an in-order traversal of the phrase hierarchy, and output a character as each leaf is encountered. The alternative is to expand all of the phrases to form strings, and then output strings directly as symbol numbers are decoded, which is extremely fast. Moreover, the transition between these two extremes is adjustable. For a given amount of memory the most frequent (or recent) strings can be held in full, and others at least partially expanded recursively. One possible implementation of this tradeoff is to retain a sliding window of recently decoded text (which can be combined with output buffering), recursively expanding only phrases that are not already in the buffer.

6. Experimental Results

Our prototype implementation of the RE-PAIR mechanism, with input partitioned into 1 MB blocks, gives the compression results shown in Table 1. The six

file	*size*	*p.-ent.*	*lit. p.*	*hori.*	*chi.*	*stat.*	*total*
E.coli	4,529	0.19	0.21	0.14	0.12	2.00	2.12
bible.txt	3,952	0.38	0.38	0.36	0.36	1.53	1.89
world192.txt	2,415	0.42	0.42	0.39	0.38	1.40	1.78
average							1.93
WSJ-20MB	20,480	0.43	0.43	0.41	0.39	1.83	2.22
Random-1	128	0.40	0.82	0.44	0.44	8.13	8.57
Random-2	128	5.33	5.93	5.23	5.02	0.00	5.02

TABLE 1. Results for the Large Canterbury Corpus (LCC) and three other files, using a 1 MB block size. The *size* column shows the original file sizes in kilobytes; *p.-ent.* is the phrase table entropy estimate calculated as described in Section 4.1; the *lit. p.*, *hori.*, and *chi.* columns show the space required for phrase tables encoded as *literal pairs* (Section 4.2), and with interpolative coding (Section 4.3) using the *horizontal slide* and *chiastic slide* respectively, measured in bits per symbol of the original file. The *stat.* column gives the space for the sequence part of each compressed file compressed with a static minimum-redundancy code, and *total* is the sum of the *chi.* and *stat.* columns. As a reference point, `gzip` and a fifth-order PPM obtain average compression of 2.30 and 1.70 bits per character respectively over the LCC; and 2.91 and 1.76 bits per character on the file WSJ-20MB. On the two random files PPM achieves 9.35 and 5.13 bits per character.

test files are the three files of the *Large Canterbury Corpus* or LCC;[2] the file WSJ-20MB which is 20 MB extracted from the *Wall Street Journal* (English text, including SGML markup); file Random-1 consisting of a random sequence of 8-bit bytes; and file Random-2 containing a sequence of 65,536 random 8-bit bytes, followed by an exact repetition of that sequence. The final column in Table 1 shows the overall compression attained when the chiastic slide dictionary representation is combined with a semi-static minimum-redundancy coder for the reduced message [**7, 10**]. As expected, compression is good, but not quite as good as the PPM context-based mechanism.

Table 2 gives some detailed statistics for the RE-PAIR mechanism, again using 1 MB blocks. The number of pairs formed is considerably smaller than the length of the block, and the phrases isolated on the non-random files can be very long indeed.

Table 3 shows the execution time of the three compression mechanisms on the file WSJ-20MB. The prototype implementation of RE-PAIR was undertaken in Java to take advantage of the favourable extensibility properties of an object-oriented context during the development of our program. This implementation, which makes extensive use of dynamic memory, runs slowly compared to other compressors written in C. However the RE-PAIR decoder has been implemented in C as well as Java, and executes approximately 50 times faster in that language. Hence, we believe that a C implementation of the encoder will operate at speed comparable to `gzip` and PPM.

[2]Files and compression results available at `http://corpus.canterbury.ac.nz/`.

file	*max. pairs*	*av. phr.*	*longest*	*av. len.*
E.coli	39,778	16,587	1,800	6.2
bible.txt	28,681	26,994	548	9.3
world192.txt	29,112	24,072	393	10.2
WSJ-20MB	32,069	31,318	1,265	7.8
Random-1	38,878	14,471	4	9.1
Random-2	48,262	53,931	65,534	26,214

TABLE 2. Phrase statistics when using 1 MB block size. The *max. pairs* column shows the maximum number of pairs formed during the processing of any of the blocks; *av. phr.* is the average number of phrases constructed per block; *longest* is the length in characters of the longest phrase constructed in any of the blocks; and *av. len.* column is the average number of characters in each symbol of the reduced message.

method	*language*	*encoding*	*decoding*
`gzip`	C	40	3
PPM	C	64	70
RE-PAIR	Java	3,181	254
RE-PAIR	C	—	5

TABLE 3. Time required to encode and decode file WSJ-20MB on a 266 MHz Intel Pentium II with 256 MB RAM and 512 kB cache, measured in CPU seconds. For RE-PAIR the times listed include the cost of the entropy coder, which is written in C and requires 6 seconds for encoding and 1 second for decoding. The PPM implementation uses a fifth-order context; `gzip` was used with the `-9` option.

Figure 2 shows the compression rate attained by RE-PAIR on file WSJ-20MB as a function of blocksize. Even with a relatively small blocksize the compression obtained is as good as that of `gzip` (which always operates with a blocksize of 64 kB), and for large block sizes the compression approaches the target set by the fifth-order PPM implementation (here using 32 MB for its model, and escape method D). Because of the memory overheads incurred by the current Java version of the encoder we have been unable to apply RE-PAIR to the entire 20 MB file, but expect, when we are in a position to do so, that the resultant compression will continue to improve.

7. Conclusion

Our compression technique obtains excellent compression, yet is also capable of high decoding rates. Moreover, the amount of memory used by the decoder can be reduced at the cost of slower decoding. The principal drawback of our mechanism is that encoding is relatively expensive. Nevertheless, for a wide range of applications, the combination of attributes that we have achieved is highly desirable.

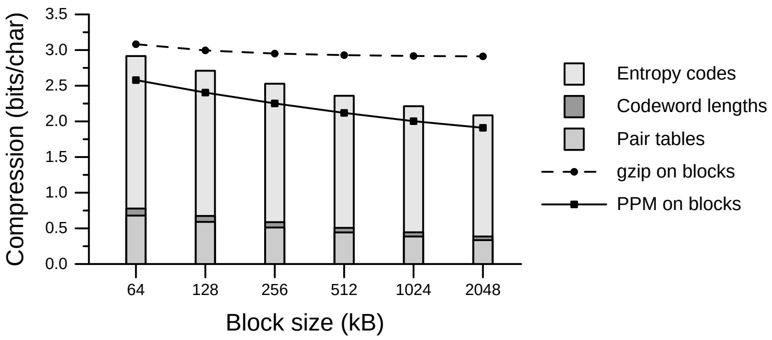

FIGURE 2. Compression as a function of blocksize for file WSJ-20MB. The three components for RE-PAIR are the cost of specifying the pairs; the cost of transmitting the codeword lengths; and the cost of coding the reduced message.

Acknowledgements. Andrew Turpin (University of Melbourne) supplied the entropy coder. This work was supported by the Australian Research Council and the Swedish Research Council for Engineering Sciences.

References

1. Timothy C. Bell and Ian H. Witten, *The relationship between greedy parsing and symbolwise text compression*, Journal of the ACM **41** (1994), no. 4, 708–724.
2. Peter C. Gutmann and Timothy C. Bell, *A hybrid approach to text compression*, Proceedings of the IEEE Data Compression Conference, March 1994, pp. 225–233.
3. Glen G. Langdon, *A note on the Ziv-Lempel model for compressing individual sequences*, IEEE Transactions on Information Theory **IT-29** (1983), no. 2, 284–287.
4. Udi Manber, *A text compression scheme that allows fast searching directly in the compressed file*, ACM Transactions on Information Systems **15** (1997), no. 2, 124–136.
5. Victor S. Miller and Mark N. Wegman, *Variations on a theme by Ziv and Lempel*, Combinatorial Algorithms on Words, Volume 12 (Berlin) (Alberto Apostolico and Zvi Galil, eds.), NATO ASI Series F, Springer-Verlag, 1985.
6. Alistair Moffat and Lang Stuiver, *Exploiting clustering in inverted file compression*, Proceedings of the IEEE Data Compression Conference, April 1996, pp. 82–91.
7. Alistair Moffat and Andrew Turpin, *On the implementation of minimum-redundancy prefix codes*, IEEE Transactions on Communications **45** (1997), no. 10, 1200–1207.
8. Hirofumi Nakamura and Sadayuki Murashima, *Data compression by concatenations of symbol pairs*, Proceedings of the IEEE International Symposium on Information Theory and its Applications (Victoria, BC, Canada), September 1996, pp. 496–499.
9. Craig G. Nevill-Manning and Ian H. Witten, *Compression and explanation using hierarchical grammars*, Computer Journal **40** (1997), no. 2/3, 103–116.
10. Andrew Turpin and Alistair Moffat, *Housekeeping for prefix coding*, IEEE Transactions on Communications, to appear.
11. Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing gigabytes: Compressing and indexing documents and images*, second ed., Morgan Kaufmann, San Francisco, 1999.
12. Jacob Ziv and Abraham Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Transactions on Information Theory **IT-24** (1978), no. 5, 530–536.