Bit Recycling with Prefix Codes*

Danny Dubé† Vincent Beaudoin‡
Université Laval
Canada

Abstract

Many data compression methods cannot remove all redundancy from a file that they compress because it can be encoded to many compressed files. In particular, we consider the redundancy caused by the availability of many equivalent messages. A canonical example is that of the many longest matches that are typically found by the LZ77 derivatives. Most of this redundancy can be removed using a technique previously introduced by the authors as *bit recycling*. Until now, bit recycling has been applied to LZ77 derivatives. In this paper, we consider a more general case: the one when there are multiple equivalent messages available to the compressor. We extend an algorithm, called *resolution*, that allows to include bit recycling in a compressor that proceeds in a stream-like fashion. Finally, we study the efficiency of *proportional* recycling.

1 Introduction

1.1 Redundancy from multiple encodings

Many lossless data compression methods allow some clear-text file to be encoded in the form of any one of many different compressed files. The compressed files are *different* in the sense that they are different sequences of bits. However, they are *equivalent* in the sense that, by decompressing any of them, we recover the original clear-text file exactly. Typically, a compression method that provides multiple encodings for some file also provides multiple encodings for most files, if not all. The existence of multiple encodings tends to increase the size of the compressed files. The severity of the increase depends on the number of the encodings. Note that we consider files here but the point would be similar with streams. We simply believe that the explanations are clearer using files.

Among the data compression methods that allow clear-text files to be encoded many different compressed files, there are the derivatives of the LZ77 method [12], where matches need not be as long as possible and where there may be more than one match of a particular length. A method deriving from LZ78 [13] where the decision of integrating a new word in the dictionary or not would be taken by the compressor

*This work was funded by the Natural Science and Engineering Research Council of Canada.
†Danny.Dube@ift.ulaval.ca
‡Vincent.Beaudoin.1@ulaval.ca

and transmitted to the decompressor would lead to a huge number of compressed files per original file. Also, the technique of selecting one of two predictors for each byte of the original file [8] can legally map a file of length N to 2^N different compressed files.

This paper presents a technique that aims at reducing the expansion of the compressed files that is caused by the multiplicity of encodings. The technique does not try to eliminate the multiplicity itself. Instead, it lets the multiplicity exist and it takes advantage of it by converting it into useful information. We call this process *bit recycling*. The information that is recycled conveys (for free) parts of the compressed file that would normally be produced. A file compressed using bit recycling cannot be decompressed using a normal decompressor as only part of the information is explicitly present and the rest has to be *recycled* from the multiplicity of encodings for the file at hand.

1.2 Redundancy from equivalent messages

While bit recycling could, in principle, be applied to any compression method that features multiple encodings, we concentrate on a particular case of multiplicity: when there exist alternatives to some of the messages that are sent to the decompressor.

Let us describe this case more clearly. From an abstract point of view, the compression and decompression process can be seen as the transmission of a sequence of messages. On the compressor side, an encoding modeler transforms the original data into a sequence of messages and then a statistical encoder transforms these messages into a bit stream. On the decompressor side, the reverse operations are performed by a statistical decoder and a decoding modeler. The nature of the messages themselves can be abstracted away. From this point of view, the multiplicity of encodings means that more than one sequence of messages could be produced by the encoding modeler to describe the original data.

However, in order to have redundancy from *equivalent messages* only, the sequence of messages that is produced by the encoding modeler has to obey a few conditions. First, the number of messages, say N, has to be uniquely determined by the original data. Second, the ith message can be chosen among n_i (≥ 1) *equivalent* messages $M_{i,1}, \ldots, M_{i,n_i}$, for $1 \leq i \leq N$. Each n_i has to be uniquely determined by the original data. By "equivalent", we mean that, at step i, any of the n_i different messages would be interpreted the same by the decoding modeler. Formally, for any sequence of choices $c_1, \ldots, c_N$, where $1 \leq c_i \leq n_i$, for $1 \leq i \leq N$, the sequence of messages $M_{1,c_1} \ldots M_{N,c_N}$ allows the decoding modeler to reconstruct the original data. Note that there are $\prod_N^{i=1} n_i$ different but equivalent encodings for the original data that we abstractly consider here.

1.3 Running example

A good example of a compression method that suffers from redundancy from equivalent messages is LZ77 compression [12] and its derivatives. In its simplest form, LZ77 deals with only two kinds of messages: literal characters and matches. A literal character message $[c]$ explicitly indicates the value of the next character in the

original file. A match message $\langle l, d \rangle$ indicates that the next l characters are copies of those that can be found d characters before the current position. Compression is effectively obtained through matches, provided the length l is sufficient; e.g. $l \geq 3$. If an LZ77-based method requires a match to be selected each time one is available and also requires a *longest* match to be selected, then this method obeys the conditions for redundancy from equivalent messages. Indeed, there typically is redundancy as the longest matches are not unique in general. When the ith message is a match, the available messages $M_{i,1}, \ldots, M_{i,n_i}$ are matches $\langle l, d_1 \rangle, \ldots, \langle l, d_{n_i} \rangle$ that all describe the next l characters by referring to copies located at different distances.

1.4 Getting rid of the equivalent-message redundancy

Our goal is to reduce the adverse effect that redundancy from equivalent messages has on compression efficiency. We intend to do so using bit recycling, which is introduced in the remainder of the paper. But before we present bit recycling itself, we need to mention a naïve way of eliminating the redundancy by modifying the compressor/decompressor pair.

Let us consider some step in the compression process and establish the set of all messages $M_1, \ldots, M_K$ that could possibly (and legally) be emitted by the encoding modeler. Note that this set should not take into account any knowledge about the original data that has not yet been transmitted to the decompressor. Also note that the modeler associates a probability p_i, explicitly or not, to each possible message M_i. Two different messages may happen to be equivalent. For example, messages M_{17} and M_{64} may both describe matches to copies of "`abc`". In order to get rid of the redundancy caused by equivalent messages, the compressor could then partition the set of messages into equivalence classes. Each equivalence class would be given a probability that is the sum of the probabilities of its members. Instead of emitting some message M_i, a modified compressor would emit the equivalence class of M_i. The more numerous the messages equivalent to M_i would be, the higher the probability associated to the equivalent class of M_i would be, and the cheaper the encoding of the latter would be.

On a similarly modified decompressor side, the reception of a message at the corresponding step would proceed like this. Given the current knowledge, the decompressor would establish the set of messages that could possibly be received by the decoding modeler. Note that the set would be $M_1, \ldots, M_K$ also. The decompressor would be able to partition the set of messages into equivalence classes exactly as the compressor would have done. Using these equivalence classes and the associated probabilities, the decompressor would receive the class that would have been emitted by the compressor. Finally, there would remain to select any message among the members of the equivalence class and to have the decoding modeler interpret it.

Clearly such a compressor/decompressor pair would effectively remove the redundancy caused by the presence of equivalent messages. However, computing the equivalence classes at each step would be prohibitively costly. Bit recycling aims at obtaining the same reductions in the redundancy but by working in a more economical way. Instead of eliminating redundancy at the source, bit recycling lets it exist

```
1. while description incomplete do
2.     let C := curr. coding funct.;
3.     let M := select message;
4.     emit C(M);
where
5. procedure emit w:
6.     σ := σ · w;
7.     return;
```

Compressor

```
1. while description incomplete do
2.     let C := curr. coding funct.;
3.     let M := receive C;
4.     interpret M;
where
5. procedure receive C:
6.     let M, σ' s.t. C(M) · σ' = σ;
7.     σ := σ';
8.     return M;
```

Decompressor

Figure 1: Conventional algorithms with prefix codes

and tries to take advantage of it and to extract a compensation from it. As will be apparent in the following sections, bit recycling needs to work with the equivalence class of the single message that was transmitted instead of all equivalence classes.

2 Recycling with prefix codes

2.1 Algorithms for a compressor/decompressor pair

In this section, we present bit recycling applied to a compression method that is based on prefix codes. However, before we explain recycling itself, we present a skeletal compression method. The algorithms for some compressor and the corresponding decompressor are shown in Figure 1. Note that almost all the details are abstracted away. An element that is mentioned explicitly is the fact that the compressed file is the concatenation of a sequence of encoded messages. At each step of the compression (resp. decompression), there is a *current coding function* C that can translate any of the currently possible messages into a *codeword* (a bit sequence). C is a *prefix code* in the sense that, from the front of any infinite string of bits, one can extract the codeword of one and only one message. Also, the bit stream σ that is conceptually the communication channel between the compressor and the decompressor is explicitly mentioned. Finally, the operation "**select message**" identifies a message that correctly describes (part of) the rest of the original data. In case there are many correct message, one of them is selected. For example, *a* longest match is found, if one exists (plausibly, the *closest* longest match or the one with the shortest codeword).

2.2 Recycling versions using non-determinism

Now, we show how to add bit recycling to the skeletal conventional compression method. For the moment, to minimize the extent of the changes to the conventional method, we use non-determinism. The new algorithms are presented in Figure 2.

We intend to describe the compressor's algorithm after that of the decompressor because it is much more complex. Nevertheless, we want the reader to simply note that, at any step during the compression process, the compressor's main program considers all messages that are deemed possible. It has the right to select any one of these but it will do so with the intent of sending a hint to the decompressor.

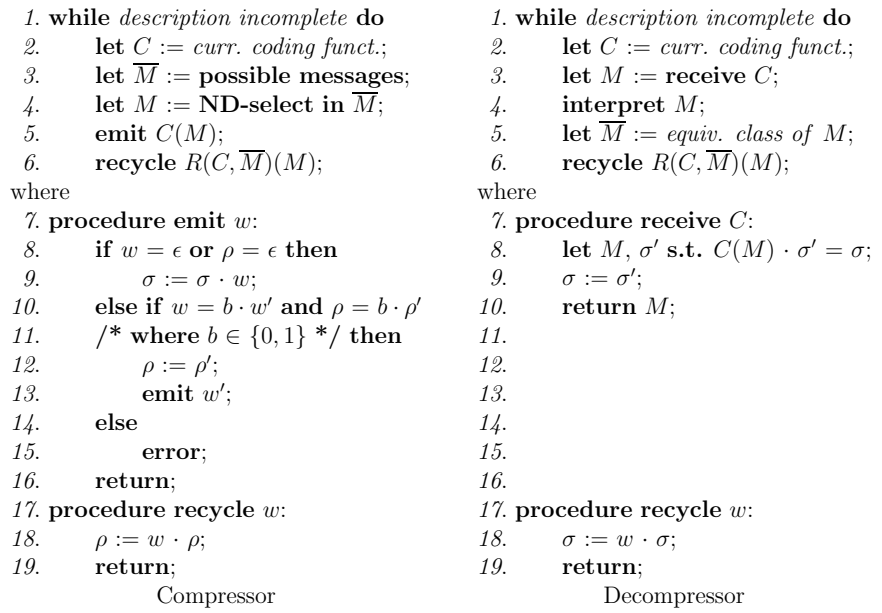

Figure 2: Non-deterministic recycling with prefix codes

Let us describe the decompressor's new algorithm. The first three instructions of the main loop are similar as those of the conventional algorithm. The differences come afterwards. At line 5, given the message M that it just received, it is able to identify all the messages that *could* have been sent in place of M by the compressor; that is, all the messages that are equivalent to M; in other words, the equivalence class $\overline{M}$. At this point, the decompressor is able to observe the extent of the compressor's freedom in its selection of M and it can also identify which one M is among $\overline{M}$. So, it is able to notice the hint from the compressor. The information in the hint is obtained as a bit sequence. Function R is a recycled code construction function. It takes the current coding function C and the current equivalence class $\overline{M}$ as arguments and returns a prefix code $R(C, \overline{M})$. The latter associates to each message $M \in \overline{M}$ a codeword $R(C, \overline{M})(M)$. This is the bit sequence that is recycled thanks to the information carried by the selection of M. As expressed by procedure **recycle**, recycling a bit sequence simply consists in prepending it in front of the input bit stream σ. These very bits then contribute to the encoding of the next message(s).

Seeing how naturally the decompressor can decode messages and recycle bit sequences, it is not difficult to guess that the compressor has to be extremely careful in its preparation of the bit stream. If we inspect the main loop of the compressor's algorithm, the instructions seem to follow a "reasonable" order: considering all messages, selecting the *right* one, emitting it, and recycling the corresponding bit sequence. However, we soon realize that the compressor cannot really recycle a bit sequence since there is yet no remainder of the bit stream to prepend the sequence in front of. Moreover, since the compressor has not yet considered the issue of the following messages at all, how can it select a message in $\overline{M}$ so that the decompressor will recycle in such a way that the decoding of the following messages will proceed

correctly? In any particular step, there is a non-trivial causal effect of the following messages (and their associated recycling) on the decision at hand.

In order to solve this problem, the compressor makes use of non-determinism. In line 4, it non-deterministically selects the right message M, which allows it to emit M and simulate the associated recycling operation. The recycling operation is nothing more than a simulation: guessed bits are pooled in a special guessed bit stream ρ. Bits in ρ are those that will never be transmitted explicitly in the bit stream σ but that will instead be recovered by the decompressor through recycling. During subsequent emissions, procedure **emit** confirms that the non-deterministic choices were correct and consumes the guessed bits. When there are no more guessed bits available, conventional explicit transmission through σ is used. Our interpretation of non-determinism may seem dubious to some readers. From a more operational point of view, we can say that the non-deterministic choice in line 4 causes the execution to fork into $|\overline{M}|$ concurrent executions. Later on, all of these executions except a single one will discover some inconsistency between what has been "guessed" and what needs to be encoded and result in erroneous terminations. The uniqueness of the right option per non-deterministic choice is a direct consequence of the fact that R constructs prefix codes only.

In order to better understand how the compressor and the decompressor collaborate to encode and decode messages with recycling, let us state a few invariants. Let us consider the concurrent executions of the compressor and the decompressor step by step, always making the right non-deterministic choices on the compressor side. The bit stream σ_C on the compressor side monotonically extends to the right, following explicit emissions of bits. The guessed bit stream ρ grows and shrinks in an irregular fashion, following implicit emissions and recycling of bits. The bit stream σ_D on the decompressor side grows and shrinks on the left, following consumption and recycling of bits. Bit stream σ_D can always by split into a recycled prefix σ'_D and an explicitly transmitted suffix σ''_D. At the start of every step, we have that $\rho = \sigma'_D$—that is, the compressor knows exactly about the bits that have been recycled by the decompressor—and that $\sigma_C \cdot \sigma''_D$ is a constant bit stream—that is, the emission and reception of explicit bits remain in synch. This constant bit stream happens to be the complete compressed file. The reader is invited to verify these invariants.[1]

Let us consider a simple example based on LZ77. Suppose that, at some step, there are three longest matches and that $\overline{M} = \{\langle 4, 32\rangle, \langle 4, 67\rangle, \langle 4, 2043\rangle\}$. Suppose also that $R(C, \overline{M})$ associates the recycled bit sequences 10, 0, and 11 to the matches, respectively. If, for the sake of correct transmission of the remainder of the original file, the bit stream faced by the decompressor at the next step has to be 1101100…, then the compressor has to guess that the right match to choose is $\langle 4, 2043\rangle$, so that the appropriate bit sequence gets recycled.

[1]There is a small mistake in the paper by Dubé and Beaudoin [4]. The operands of the concatenation that appears here in the compressor's algorithm on line 18 are reversed in their paper.

2.3 Resolution algorithm

Selecting the right messages cannot be done in a deterministic fashion; at least, if we insist on proceeding *forwards*. We propose a simple and deterministic resolution algorithm to make the right choices. Observe that the successive equivalence classes $\overline{M}_1, \ldots, \overline{M}_N$ can easily be computed forwards in a first phase. Similarly for the successive coding functions $C_1, \ldots, C_N$. Then, the successive bit streams $\sigma_1, \ldots, \sigma_N$ that the decompressor faces at the start of each step can as easily be computed in a second phase, but *backwards*. Let us define σ'_i to be the bit stream at step i between reception and recycling and σ''_i to be the one after recycling. Suppose that, for some step i, σ''_i is known. It is then possible to determine σ'_i since, by prepending the bit sequence associated to some (unique) message $M \in \overline{M}_i$, we obtain σ''_i. In other words, we can compute M and σ'_i from $\overline{M}_i$ and σ''_i using the equation $R(C_i, \overline{M}_i)(M) \cdot \sigma'_i = \sigma''_i$. Since M is the message that is transmitted in step i, we can also determine σ_i as it is $C_i(M) \cdot \sigma'_i$. Observing that $\sigma''_i = \sigma_{i+1}$, for $1 \leq i \leq N-1$, we conclude that all these intermediate bit streams can be determined from σ''_N. It is sufficient to let σ''_N be some arbitrary bit string that is long enough to initiate the resolution process. Indeed, σ''_N is the bit string that remains after the whole original file has been described to the decompressor; so it is meaningless. The final result of the resolution algorithm is the bit stream σ_1, as it is the bit stream that is faced by the decompressor at the start of the first step.

2.4 Greedy resolution algorithm

The resolution algorithm is not very practical as it requires a lot of intermediate information to be kept until completion. Also, it prevents the compressor from working in a stream-like fashion. Of course, it would be possible to perform resolution on a block-by-block basis, for blocks of modest size, but it would cause recycled bits to be lost at every block boundary.

Instead, it is possible to perform the resolution process in a greedy fashion. The simplest way consists in waiting for a step i where there is only one possible message, i.e. where $\overline{M}_i = \{M_i\}$, and detecting whether the concatenation $C_i(M_i) \cdot \sigma'_i$, where σ'_i is yet unknown, provides sufficiently many known bits to trigger a cascade of resolution for the previous, unresolved steps. For example, this would be the case when M_i would be a literal character $[c]$ or a unique longest match $\langle l, d \rangle$. Since many more bits are emitted than recycled, as is typical in practice, unresolved chains do not tend to be long. Moreover, a particular implementation of a compression method may well emit $C_i(M_i)$ in pieces, where the first one happens to be fixed. As observed by Dubé and Beaudoin [4] in their LZ77-based experiments on recycling, where a match is emitted as a length (fixed) followed by a distance (variable), the bit sequence of the length was very often long enough to trigger the resolution of the previous few unresolved steps.

A more involved technique would consist in computing many partially known bit streams backwards from current step i to the most ancient unresolved step j, $j < i$, to determine if uncertainty in step i necessarily implies uncertainty in steps $i-1$, $i-2, \ldots, j$, also.

3 Natural recycling functions

Until now, we did not say much about the recycled code construction function R. We simply mentioned that it has to construct prefix codes. Because it ultimately specifies the bit sequences that get recycled, it definitely has an effect on the improvement in compression that can be expected from recycling. We consider two natural recycling functions.

First, let us consider the *flat* recycling function. That is, R does not use its first argument, the coding function C, and constructs a code $R(C, \overline{M})$ that assigns codewords of length between $\lfloor \log |\overline{M}| \rfloor$ and $\lceil \log |\overline{M}| \rceil$ to the messages. Clearly, the number of recycled bits for a particular message M does not depend on the length of $C(M)$. Note that, with recycling, a message $M \in \overline{M}$ is selected as often as a (almost completely) random bit stream starts with $R(C, \overline{M})(M)$, i.e. with probability $2^{-\lfloor \log |\overline{M}| \rfloor}$ or $2^{-\lceil \log |\overline{M}| \rceil}$, which is independent of the cost of selecting M.

Let us study an example using flat recycling. Suppose that four equivalent messages are available, each costing 6 bits. Then, thanks to bit recycling, the net cost of selecting a message and benefiting from recycling is 4 bits. Note that recycling is not always a winning strategy. If there are still four equivalent messages with the first one costing 5 bits and the others, 8 bits, then the net cost of selecting a message and benefiting from recycling is $29/4 - 2 = 5.25$ bits, on average, compared to 5 bits, if recycling is abandoned and the cheapest option is systematically selected.

Second, let us consider the *proportional* recycling function. In that case, R constructs codes that assign longer recycled codewords to messages that have longer codewords. More precisely, code $R(C, \overline{M})$ is a Huffman code for the messages in $\overline{M}$ built using the probabilities $2^{-|C(M_i)|}$ for the messages M_i, respectively. It is well known that Huffman codes cannot be optimal in general, but proportional recycling tends to keep the difference between $|C(M)|$ and $|R(C, \overline{M})(M)|$ as uniform as possible. Assigning a longer recycled codeword to a more costly message tends to make the selection of that message less probable.

Let us study an example using proportional recycling. Suppose that four equivalent message are available with costs of 4, 4, 5, and 6 bits. Then, for recycling purposes, they are assigned frequencies 2^{-4}, 2^{-4}, 2^{-5}, and 2^{-6}, respectively, and their associated recycled sequences might have lengths 1, 2, 3, and 3, respectively. The net cost of selecting a message and benefiting from recycling would be $3 \cdot 2^{-1} + 2 \cdot 2^{-2} + 2 \cdot 2^{-3} + 3 \cdot 2^{-3} = 21/8 = 2.625$, on average. Note that, using flat recycling, the net cost would be 2.75, on average.

In their experiments, Dubé and Beaudoin obtained the best results using proportional recycling, leading to an average improvement of 2.1% (see experiment 6 [4]). Although we provide no complete proof here, we believe proportional recycling to be close to optimal. In fact, if recycling could be done using fractional bits, then to equivalent messages of probabilities $p_1, \ldots, p_k$, where $p_i = 2^{-l_i}$ and l_i is the length of the codeword associated to message i, we would assign optimal recycled bit sequences of probabilities $f \cdot p_1, \ldots, f \cdot p_k$, for f such that $\sum_k^{i=1} f \cdot p_i = 1$. Now, a recycled bit sequence of probability $f \cdot p_i$ does not necessarily have integer length. To obtain sequences of whole bits, we have to assign frequencies p_i to the equivalent messages

and use Huffman's algorithm. Because of this, the result is suboptimal, in general, because the probabilities of the recycled sequences are not exactly proportional to those of the equivalent messages. For example, equivalent messages of costs 4, 5, and 15, would be assigned recycled sequences `0`, `10`, and `11`, for an average net cost of 5.5 bits, instead of a net cost 3.41 bits if fractional bits could be used. Because sequences of whole bits cause sub-optimality, it may be worthwhile to drop certain equivalent messages whose net contribution is negative. To continue with our example, dropping the message of cost 15 would improve the net cost to 3.5 bits, on average. In order to perform optimal whole-bit recycling, one would need to evaluate the average net cost when dropping from 0 to $k - 1$ of the most costly messages.

4 Related work

While bit recycling itself has first been proposed by the authors, the idea of exploiting the redundancy of some data compression methods, namely the LZ77 derivatives, was presented earlier. The capacity of integrating some amount of information inside of a file compressed using an LZ77 derivative has been used for information hiding (or steganography) [2, 5], authentication [1], and error correction [6, 10]. In these applications, information that is implicitly transmitted comes from an external source, like a document to hide, an electronic signature of the sender or for the original document, or error-correcting codes for the original document. So it is always immediate for the compressor to determine what message should be selected since the bits that the compressor wants the decompressor to guess are known in advance.

In the case of bit recycling, things are more complicated since the information that gets transmitted through the selection of the messages comes from the compressed file itself. In our work [3, 4], this leads to the resolution algorithm. In the work presented by Yokoo *et al.* [11], they choose to split the compressed file in two parts: a first one that is transmitted explicitly and that carries the second part through bit recycling; a second one that is transmitted implicitly inside of the first part and that carries no hidden information. We believe that some space is lost in their technique since the second part could also contribute to bit recycling, even if it is typically shorter than the first part. Except for files that feature high redundancy, we expect the loss to be negligible. Note that all these works about bit recycling could avoid the complications due to the compressed file holding parts of itself by implicitly transmitting bits from the original file instead. However, this approach would not be particularly efficient since bits from the original file have lower informational contents than those from the compressed file and so are less profitable to recycle.

5 Future work

We mention three things that need to be done in the near future. First, the criterion that we present here to determine the optimal way to perform proportional recycling is inefficient. Further investigation is necessary to determine if the proper cut point could be computed in a cheaper way. Second, bit recycling should be adapted to

arithmetic coding. It does not seem hard to do it if arbitrary-precision numbers are used. However, it would become truly useful only by adapting it to the quick, finite-precision methods [9, 7]. Finally, it would be interesting to extend, whenever possible, bit recycling to contexts where the redundancy does not come (only) from equivalent messages. For example, in LZ77 derivatives, when one considers longest matches only, alternative messages can be seen as equivalent, but when one considers matches of varying lengths, the various messages cannot be seen as equivalent anymore.

References

[1] M. J. Atallah and S. Lonardi. Authentication of LZ-77 compressed data. In *Proceedings of the 18th ACM Symposium on Applied Computing*, pages 282–287, Melbourne, Florida, USA, mar 2003.

[2] A. Brown. `gzip-steg`, 1994.

[3] D. Dubé and V. Beaudoin. Recycling bits in LZ77-based compression. In *Proceedings of the Conférence des Sciences Électroniques, Technologies de l'Information et des Télécommunications (SETIT 2005)*, Sousse, Tunisia, mar 2005.

[4] D. Dubé and V. Beaudoin. Improving LZ77 data compression using bit recycling. In *Proceedings of the International Symposium on Information Theory and Applications (ISITA)*, Seoul, South Korea, oct 2006.

[5] S. Lonardi and W. Szpankowski. Joint source-channel LZ'77 coding. In *Proceedings of the IEEE Data Compression Conference*, pages 273–282, Snowbird, Utah, USA, mar 2003.

[6] S. Lonardi, W. Szpankowski, and M. Ward. Error resilient LZ'77 scheme and its analysis. In *Proceedings of IEEE International Symposium on Information Theory (ISIT'04)*, page 56, Chicago, Illinois, USA, 2004.

[7] A. Moffat, R.M. Neal, and I.H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, jul 1998.

[8] P.A.J. Volf and F.M.J. Willems. Switching between two universal source algorithms. In *Proceedings of the IEEE Data Compression Conference*, pages 491–500, mar 1998.

[9] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, jun 1987.

[10] Y. Wu, S. Lonardi, and W. Szpankowski. Error-resilient LZW data compression. In *Proceedings of the IEEE Data Compression Conference*, pages 193–202, Snowbird, Utah, USA, mar 2006.

[11] H. Yokoo. Lossless data compression and lossless data embedding. In *Proceedings of the Asia-Europe Workshop on Concepts in Information Theory*, Jeju, South Korea, oct 2006.

[12] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–342, 1977.

[13] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, sep 1978.