Motifs in Ziv-Lempel-Welch Clef

(Extended Abstract)

Alberto Apostolico[*] Univ. of Padova & Purdue Univ.

Matteo Comin[†] Univ. of Padova

Laxmi Parida [‡] IBM T. J. Watson Center

Abstract

We present variants of classical data compression paradigms by Ziv, Lempel, and Welch in which the phrases used in compression are selected among suitably chosen *motifs*, defined here as strings of intermittently solid and wild characters that recur more or less frequently in the source textstring. This notion emerged primarily in the analysis of biological sequences and molecules. Whereas the number of motifs in a sequence or family may be exponential in the size of the input, a linear-sized *basis* of *irredundant* motifs may be defined such that any other motif can be obtained by the union of a suitable subset from the basis. Previous study has exposed the advantages of using irredundant motifs in lossy as well as lossless off-line compression. In the present paper, we examine adaptations and extensions of classical *incremental* ZL and ZLW paradigms. First, hybrid schemata are proposed along these lines, in which motifs may be discovered and selected off-line, while the parse and encoding is still conducted on-line. The performances thus obtained improve on the one hand over previous off-line implementations of motif-based compression, and on the other, over the traditionally best implementations of ZLW. On the basis of this, both lossy and lossless motif-based schemata are introduced and tested that follow more closely the ZL and ZLW paradigms.

1 Introduction

Ziv and Lempel designed a class of compression methods based on the idea of self reference: while the textfile is scanned, substrings or *phrases* are identified and stored in a *dictionary*, and whenever, later in the process, a phrase or concatenation of

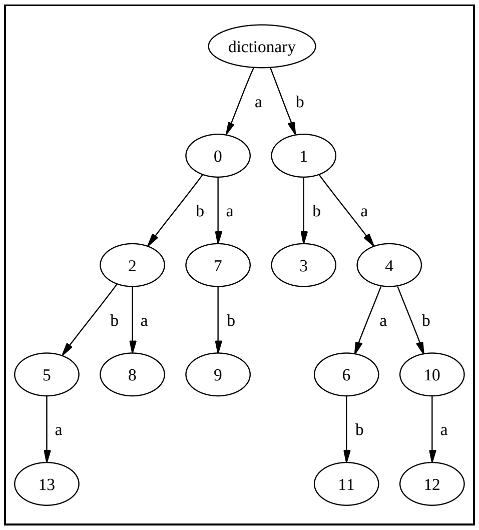
[*]Dipartimento di Ingegneria dell' Informazione, Università di Padova, Padova, Italy *and* Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA. Work Supported in part by an IBM Faculty Partnership Award, by the Italian Ministry of University and Research under the National Projects FIRB RBNE01KNFP, and PRIN "Combinatorial and Algorithmic Methods for Pattern Discovery in Biosequences", and by the Research Program of the University of Padova. `axa@dei.unipd.it`

[†]Dipartimento di Ingegneria dell' Informazione, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy. `ciompin@dei.unipd.it`

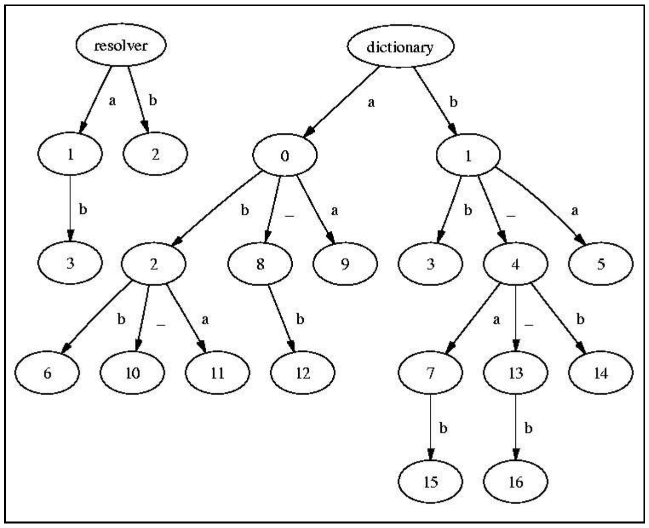
[‡]Yorktown Heights, NY 10598, USA. `parida@us.ibm.com`

phrases is encountered again, this is compactly encoded by suitable pointers or indices [8, 13, 14]. Of the existing versions of the method, we recapture below the parse known as Ziv-Lempel-Welch, which is incarnated in the `compress` of UNIX.

For the encoding, the dictionary is initialized with all the characters of the alphabet. At the generic iteration, we have just read a segment s of the portion of the text still to be encoded. With σ the symbol following this occurrence of s, we now proceed as follows: If $s\sigma$ is in the dictionary we read the next symbol, and repeat with segment $s\sigma$ instead of s. If, on the other hand, $s\sigma$ is not in the dictionary, then we append the dictionary index of s to the output file, and add $s\sigma$ to the dictionary; then reset s to σ and resume processing from the text symbol following σ. Once s is initialized to be the first symbol of the source text, "s belongs to the dictionary" is established as an invariant in the above loop. Note that the resulting set of phrases or codewords obeys the *prefix closure* property, in the sense that if a codeword is in the set, then so is also every one of its prefixes.



Dictionary from original ZLW



Motif and resolver dictionaries

0	1	1	2	4	0	2	7	4	6	10	5	7
a	b	b	ab	ba	a	ab	aa	ba	baa	bab	abb	aa

(0,0)	(1,0)	(1,0)	(2,0)	(4,1)	(0,0)	(2,0)	(8,1)	(4,1)	(7,1)	(4,1)	(13,3)	(7,1)
a	b	b	ab	ba	a	ab	aa	ba	baa	ba	bab	baa

Figure 1: Illustrating ZLW encoding (left) and parse (top) and one of its motif-based variants as applied to the string *abbabbaaabaababaababababbaa*. A pair (i, j) in the bottom parse indicates the $i - th$ phrase of the dictionary, to be completed by the $j - th$ resolver ($j = 0$ means no resolver).

The left part of Figure 1 illustrates this scheme. ZLW is easily implemented in linear time using a trie data structure as the substrate [13, 14], and it requires space linear in the number of phrases at the outset. Another remarkable property of ZLW is that the encoding and decoding are perfectly symmetrical, in particular, the dictionary is recovered while the decompression process runs (except for a special case that is easily taken care of).

In a motif driven ZLW parse, dictionary entries correspond to strings written on the alphabet $\Sigma \cup \{\_\}$, where "_" represents a wildcard or "don't care" symbol that may take up one of several specifications. The right half and bottom parse in Figure 1, for instance, result from the variant described in Figure 3. Allowing don't cares leads to an increase of the average length of phrases hence to a decrease in their number. As the tables in this paper display, savings can be dramatic for inputs such as images and signals, in which the gaps can be filled by interpolation at the receiver with negligible loss (cf. tables). As it turns out, however, improvements are also seen in lossless schemata in which a dictionary of *resolvers* must be added to disambiguate the don't care in the phrases.

To enucleate a rough rule of thumb, we compare encodings under the rosiest possible scenario, that is, we assume that allowing k don't care in each phrase will reduce the total number of phrases by a factor of k. This, however, will come at a cost of doubling the encoding of each phrase. As is known, if p is the number of phrases produced by the standard ZLW for an input of size n, the number L of bits in the corresponding encoding is

$$L \approx (p)\log(p)$$

Under our assumptions, the output size would become

$$L' \approx 2(\frac{p}{k})\log(\frac{p}{k})$$

which induces a *shuffle gain* defined here as

$$\frac{L'}{L} \approx \frac{2}{k}\left(1 - \frac{\log(k)}{\log(p)}\right)$$

which is positive for $p > k$. The last expression gives qualitative insight into the gain that can be expected. In practice, it turns out that the correlations inherent to the input do not extend indefinitely, whence the reduction in the number of phrases tends to saturate at some small value of k. Next, we give some notions and properties that set the framework for picking don't cares. We refer to [4] for more formal treatment.

A *motif* m in a string x is an element of Σ or any string on $\Sigma \cdot (\Sigma \cup \{\_\})^* \cdot \Sigma$ *together* with its list of occurrences $\mathcal{L}_m$ in x. A motif that occurs at least k times in x is a *k-motif*. Motif m is *maximal* if we cannot make it more specific or longer by adding solid characters while retaining (up to a uniform offset $d, |d| \geq 0$) $\mathcal{L}_m$. A maximal motif m is *irredundant* if m and $\mathcal{L}_m$ cannot be deduced by the union of a number of lists of other maximal motifs. Let now x and y be two strings with $m = |x| \leq |y| = n$. The *consensus* of x and y is the string $z_1 z_2 ... z_m$ on $\Sigma \cup$ '_' defined as: $z_i = x_i$ if $x_i = y_i$ and $z_i =$ '_' otherwise $(i = 1, 2, ..., m)$. Deleting all leading and trailing don't care symbols from z yields a (possibly empty) *2-motif* that is called the *meet* of x and y. The following property [4] supports the construction of a linear-sized *basis* of irredundant motifs, where a basis is a set $\mathcal{B}$ of motifs capable of generating all other 2-motifs in a string.

Theorem 1 *Every irredundant 2-motif in s is the meet of two suffixes of s.*

Incremental algorithms are available [4] that find all $O(n)$ irredundant 2-motifs in time $O(n^3)$. They work by iterated update of the sets $\mathcal{B}_i$ of base motifs for the i-th suffix of the input under unit symbol extension.

2 Combining ZLW and Greedy Strategies

In view of Theorem 1 and of the good performance of motif based off-line compression [5], it is natural to inquire into the structure of ZL and ZLW parses which would use these patterns in lieu of exact strings. Possible schemes along these lines include, e.g., adaptations of those in [9], or more radical schemes in which the innovative add-on inherent to ZLW phrase growth is represented not by one symbol alone, but rather by that symbol plus the longest match with the substring that follows some previous occurrence of the phrase. In other words, the task of vocabulary build-up is assigned to the growth of (candidate), perhaps irredundant, 2-motifs.

Before addressing these schemes, we test the power of ZLW encoding on the motifs produced in greedy off-line schemata (cf., e.g., [3, 5, 6, 7]). The latter paradigm consist of repeated selection of a pattern achieving the highest compression in the source strings. This pattern is encoded and the process resumed until no pattern is found capable of producing further compression. We refer to the quoted literature for details. Despite the apparent superiority of such greedy off-line approaches in capturing long range repetitions, one drawback is in the encoding of references, which are bi-directional and thus inherently more expensive than those in ZLW. Our exercise consists thus of using the motifs selected in the greedy off-line to set up an initial vocabulary of motif phrases, but then encode these and their outgrowth while we carry out a parse of the source string similar to that of ZLW. Assuming that we have already selected the motifs to be used, this adaptation of ZLW to motifs requires to address primarily the following problems:

1. We need to modify the parsing in such a way that for every chosen motif, every one of its occurrences is used in the parsing.
2. We need to modify the dictionary in order to accommodate motifs in addition to strings. This is to be done while retaining *prefix closure*.

To ensure that a motif is correctly detected and deployed in the parsing, it has to be stored in the dictionary before its first occurrence is detected. This requires building a small dictionary that needs to be sent over to the decoder together with the encoded string. In order to enforce the prefix closure, all prefixes of a motif are inserted in the dictionary together with that motif.

With the dictionary in place, the parse phase of the algorithm proceeds in much the same way as in the original scheme, with the proviso that once a motif is chosen, then all of its occurrences are to be deployed. For this, the algorithm looks at each stage for the longest substring in the tree that does not interfere with the next motif occurrence already allocated from previous stages. The motif chosen in this way is then concatenated with the symbol following it and the result is inserted in the dictionary. In order to avoid the insertion of undesired don't cares in text regions not encoded by motifs, that symbol is treated as mismatching all other characters at this stage of the search.

Decoding is easier. The recovery follows closely the standard ZLW, except for initialization of the dictionary. The only difference is thus that now the decoder receives, as part of the encoding, also an initial dictionary containing all motifs utilized, which are used to initialize the trie.

The tables below summarize results obtained on gray-scale images (Table 1, 1 pixel = 1 byte), the Calgary Corpus (Table 2), and genetic data (Table 3). For each case, the compression is reported first for lossy encoding with various don't care densities, then also for the respective lossless completions.

source file	orig len	gzip size	lossy size	gzip % % diff	% loss %	lossless size	gzip %diff	'_'/ car
camera	66.336	48.750	34.321	-29.60%	0.00%	34.321	-29.60%	1/6
			34.321	-29.60%	0.06%	34.321	-29.60%	1/5
			32.887	-32.54%	6.16%	35.179	-27.84%	1/4
bridge	66.336	61.657	38.562	-37.46%	0.29%	38.715	-37.21%	1/4
			38.366	-37.78%	5.35%	42.288	-31.41%	1/3
lena	262.944	234.543	120.308	-48.71%	1.36%	123.278	-47.44%	1/4
			123.182	-47.48%	7.32%	135.306	-42.31%	1/3
peppers	262.944	232.334	117.958	-49.23%	1.75%	121.398	-47.75%	1/4
			119.257	-48.67%	4.45%	129.012	-44.47%	1/3

Table 1: Lossy/Lossless compression of gray-scale images. Loss is computed as the number of forfeited pixels as a percentage of the total.

3 Motif-driven ZLW

In so far as we start with a set of irredundant motifs already given, the approach described so far can be regarded as an "external dictionary" scheme. It departs from a traditional such scheme in at least two respects: (1) the dictionary also grows during the parse, and (2) we make no attempt at optimizing the parse in the conventional sense. We examine next adaptations of the two main paradigms of Ziv-Lempel compression, respectively known as ZL77 and ZL78 [13, 14]. For the second one, we consider its subsequent implementation by Welch [12]. Both variants will have lossy and lossless versions, which should be dealt with separately.

At the generic step, ZL77 has parsed and encoded the first $i-1$ symbols of the source x and looks for the longest prefix of the remaining suffix that matches some previously seen substring. With j the starting position of that substring, l its length and σ the character immediately following it, the new phrase in the parse is encoded as (j, l, σ). In our lossy variant, we must assume that a minimum acceptable density d of don't care is prescribed. At the generic step, we thus look for the densest 2-motif beginning at i and at some $j < i$ and encode it by a triplet in much the same way as in

source file	orig len	gzip size	lossy size	gzip % % diff	% loss %	lossless size	gzip %diff	'_'/ car
bib	111.261	35.063	26.679	-23.91%	14.11%	34.174	-2.54%	1/4
			26.679	-23.91%	14.11%	34.174	-2.54%	1/3
geo	102.400	68.493	30.951	-54.81%	19.43%	57.098	-16.64%	1/4
			33.334	-51.33%	20.63%	58.038	-15.26%	1/3
news	377.109	144.840	104.807	-27.64%	11.42%	128.429	-11.33%	1/4
			106.483	-26.48%	19.95%	153.243	5.80%	1/3
obj1	21.504	10.323	8.447	-18.17%	8.38%	9.642	-6.60%	1/4
			7.409	-28.23%	18.03%	9.849	-4.59%	1/3
			6.736	-34.75%	21.61%	8.521	-17.46%	2/5
obj2	246.814	81.631	56.810	-30.41%	12.75%	67.857	-16.87%	1/3
			53.094	-34.96%	19.88%	67.117	-17.78%	1/2
paper1	53.161	18.577	16.047	-13.62%	13.63%	19.411	4.49%	1/3
			15.626	-15.89%	18.22%	19.198	3.34%	1/2
paper2	82.199	29.753	23.736	-20.22%	15.19%	28.743	-3.39%	1/3
			22.519	-24.31%	36.44%	35.390	18.95%	1/2
pic	513.216	56.422	36.491	-35.32%	0.56%	36.599	-35.13%	1/4
progc	39.611	13.275	11.576	-12.80%	6.88%	12.812	-3.49%	1/4
			11.381	-14.27%	8.69%	12.854	-3.17%	1/3
			11.010	-17.06%	24.86%	15.246	14.85%	1/2
progl	71.646	16.273	14.828	-8.88%	2.72%	15.601	-4.13%	1/4
			14.748	-9.37%	6.99%	16.149	-0.76%	1/3
			14.676	-9.81%	7.70%	16.261	-0.07%	2/5
progp	49.379	11.246	10.287	-8.53%	3.83%	10.879	-3.26%	1/4
			10.265	-8.72%	7.43%	11.328	0.73%	1/3
			10.416	-7.38%	7.12%	11.569	2.87%	2/5

Table 2: Lossy/Lossless compression of the Calgary Corpus.

the traditional ZL scheme. For the lossless variant, the threshold d might or might not be defined, but we must encode, in addition to the triplet, the mismatching characters *and their positions*. It is reasonable to anticipate that this overhead will take too high a toll on the encoding, whence this scheme might be viable for lossy compression only. There, don't cares can be left unresolved or copied from an interpolated or erroneous character.

We look next at possible adaptations of ZLW, Welch's implementation of ZL78. As seen, dictionary entries correspond here to strings written on the alphabet $\Sigma \cup \{\_\}$ (the dictionary is still initialized only by characters of Σ, as before). To find the next phrase in the parse, beginning, say, at position i of the text, we look for *a* longest phrase in the dictionary that matches the still un-parsed suffix of the source and *appearing at least twice so far* (to trigger this invariant condition, it suffices to prepend the string of symbols of Σ to the source string x). Let s be this phrase and h its index in the dictionary. We use the reference number h to encode the current phrase. The next step is the augmentation of the dictionary of phrases. For this, we look for a longest match with up to k mismatches (or within a minimum density of d) between the suffix starting at i and the one that begins at the leftmost occurrence of s. (If this string extends beyond i then it is truncated at $i-1$, to secure self-

source file	orig len	gzip size	lossy size	gzip % % diff	% loss %	lossless size	gzip %diff	'_'/ car
Spor EarlyII	25.008	8.008	6.137	-23.36%	16.93%	7.430	-7.22%	1/4
			6.163	-23.04%	12.01%	7.052	-11.94%	1/3
Spor EarlyI	31.039	9.862	7.494	-24.01%	14.66%	8.865	-10.11%	1/4
			7.494	-24.01%	14.66%	8.865	-10.11%	1/3
Helden CGN	32.871	10.379	7.728	-25.54%	15.36%	9.330	-10.11%	1/3
Spor Middle	54.325	16.395	11.565	-29.46%	0.38%	11.672	-28.81%	1/4
			11.555	-29.52%	0.43%	11.703	-28.62%	1/3
Helden All	112.507	33.829	25.873	-23.52%	18.83%	32.029	-5.32%	1/4
			26.010	-23.11%	18.86%	32.182	-4.87%	1/3
Spor All	222.453	68.136	48.035	-29.50%	18.98%	60.042	-11.88%	1/4
			47.896	-29.71%	19.00%	59.955	-12.01%	1/3
All Up 400k	399.615	115.023	90.120	-21.65%	18.62%	110.659	-3.79%	1/3

Table 3: Lossy/Lossless compression of genetic data (fasta).

feeding in decoding.) We may create an extension in the dictionary corresponding to this new word, and give it a suitable index. In the lossless implementation of this scheme, we would also have to append to this node a list of positioned characters that distinguish, and will enable to reconstruct, the new occurrence from the old one. Like for the LZ77 scheme, this might be bulky in general. To mitigate this problem, a new phrase could be described by the pair of indices of two past phrases. Still, the encoding of a single phrase doubles in format and probably in size in this way. As seen next, a more conservative incremental variant stems from a closer adherence to the original parse.

The generic stage of ZLW may be considered as consisting of two parts, as follows. In the first part (hereafter, *seek phrase*), a longest matching phrase from the dictionary is found, and its index is appended to the output. In the second, the one-symbol extension of the *current occurrence* of the phrase is added to the dictionary for possible future reference. In our adaptation, Part 1 is identical, except that the output now must contain both the reference to the phrase and also the characters for its disambiguation. This is achieved through the use of two dictionaries that grow hand-in-hand, whereby a phrase is resolved in the shuffle of a word from the main dictionary and one from the auxiliary one. Looking for a *best* phrase, e.g., the one minimizing mismatches is feasible but time consuming, hence our implementation of seek-phrase greedily pursues matches over mismatches. If we assume $\Sigma \cup \{\_\}$ to be sorted with $\{\_\}$ its maximum element, then seek-phrase finds the lexicographically least phrase occurring at the current position.

The pseudo-code of Fig.3 describes the algorithm that produces the parse of Fig.1. At the outset, each phrase in the parse is decomposed in a pair consisting of a primary phrase s over $\Sigma \cup \{\_\}$ and an auxiliary resolver s' over Σ. A suitable shuffle of s and s' reconstructs the actual phrase over Σ. The information needed for the shuffle is the set of positions of s occupied by don't cares. However, this does not need to be supplied explicitly. Rather, the gaps are filled with the characters from s', in exact succession, and the mechanics of the encoding and decoding, respectively, takes care

```
Initializations
Initialize dictionary trie and resolver trie with the single characters from $\Sigma$
Initialize phrase $s$ to first input character, resolver $s'$ to the empty word $\lambda$
Set $output \leftarrow < code(s), code(s') >$
Body  Repeat until no more input characters
    $\sigma \leftarrow$ read the next input character
    if $s\sigma$ is in the table then set $s = s\sigma$ (seek-phrase continues)
    else, if $s_-$ is in the dictionary and $s'\sigma$ is in resolvers
        then set $s = s_-$ ,  $s' = s'\sigma$ (seek-phrase continues)
        else (the end of a stored phrase has been reached)
            1- $output \leftarrow output \cdot < code(s), code(s') >$
            2- if $s\sigma'$ for some $\sigma' \neq \sigma$ is in the dictionary then
                2a- add $s_-$ to dictionary
                2b- if $s'\sigma$ is not already in it add $s'\sigma$ to resolvers
                else add $s\sigma$ to dictionary
            3- $s \leftarrow \sigma$
end Body
```

Figure 2: Motif-driven ZLW

source file	orig len	zlw phrases	max d.c.	lossless phrases	phrases % diff	lossy size	gzip size	gzip % diff
camera	66,336	31,575	3	22,145	-29.87%	29,620	48,750	-39.24%
	66,336	31,575	5	21,685	-31.32%	29,616	48,750	-39.25%
	66,336	31,575	20	21,667	-31.38%	29,606	48,750	-39.27%
bridge	66,336	39,457	3	26,095	-33.86%	34,166	61,657	-44.59%
	66,336	39,457	5	26,094	-33.87%	34,166	61,657	-44.59%
	66,336	39,457	20	26,094	-33.87%	34,166	61,657	-44.59%
lena	262,944	121,054	20	84,890	-29.87%	117,978	121,054	-49.70%
peppers	262,944	117,515	20	82,994	-29.38%	114,010	117,715	-50.93%

Table 4: Results from gray-scale images.

of consistency.

As is easy to see, each phrase is a substring of a meet although not necessarily a maximal or irredundant motif. There are $O(n^3)$ possible such motifs, but it can be shown that the strategy learns a sublinear subset of this set.

Note that the dictionary of phrases is a superset of the one produced by the ZLW parse, in the sense that for each phrase in that parse there is a matching motif in this one. That scheme may be indeed incorporated by few fixes to steps 1-2-4, but this would lead to doubling the size of the dictionary. It is easy to further modify the algorithm in such a way that a maximum number k of errors is allowed in each phrase. Correspondingly, the length of an entry in the auxiliary table becomes bounded and may be block-encoded, which might be profitable in some cases. The last three tables show results obtained with this method. The reduction in the number of phrases is general and dramatic in some cases, but the toll exacted by heavier phrase encodings tends to offset this gain in lossless encodings as implemented here.

source file	orig len	zlw phrases	max d.c.	lossless phrases	phrases % diff	lossy size	gzip size	gzip % diff
bib	111,261	26,861	3	25,956	-3.37%	29,592	35,063	-15.60%
	111,261	26,861	20	21,795	-18.86%	29,559	35,063	-15.70%
geo	102,400	42,839	5	27,708	-35.32%	40,432	68,493	-40.97%
news	377,109	90,905	5	78,048	-14.14%	99,779	144,640	-31.11%
obj1	21,504	9,068	4	6,584	-27.39%	9,174	10,323	-11.13%
	21,504	9,068	5	6,596	-27.26%	9,154	10,323	-11.32%
obj2	246,814	68,091	4	58,518	-14.06%	81,942	81,631	0.38%
paper1	53,161	15,370	4	13,175	-14.28%	15,741	18,577	-15.27%
	53,161	15,370	5	12,689	-17.44%	15,696	18,577	-15.51%
paper2	82,199	21,332	4	19,011	-10.88%	20,545	29,753	-30.95%
	82,199	21,332	5	18,089	-15.20%	20,935	29,753	-29.64%
progc	39,611	11,979	3	10,938	-8.69%	12,846	13,275	-3.23%
	39,611	11,979	20	9,658	-19.38%	12,699	13,275	-4.34%
progl	71,646	16,525	3	17,550	6.20%	17,166	16,273	5.49%
	71,646	16,525	20	13,926	-15.73%	18,047	16,273	10.90%
progp	49,379	12,017	3	12,659	5.34%	13,345	11,246	18.66%
	49,379	12,017	20	10,120	-15.79%	13,691	11,246	21.74%

Table 5: Results from Calgary Corpus.

4 Conclusions and Plans for Future Work

Off-line lossy and lossless motif based compression and grammatical inference for documents of various nature has been proposed and studied previously in [5, 2]. In this paper, we address the versatility of the basis of irredundant motifs in schemes that emulate the classical ones by Lempel and Ziv. As is well known, that family of constructors of information-lossless encoders rely on an incremental parse that creates a new phrase as soon as a prefix of the still unparsed section of the string differs from all preceding phrases. The differences among the various parses rests on the choice of the repertoire of phrases, and this in turn depends on a subtle tradeoff between the number of phrases and the implied size of their encoding. Incremental lossy schemata that operate in this way were addressed in [9], along with their information-theoretic asymptotic optimality. In our derivations, the basis of motifs can be similarly used as an efficient lossy scheme as well as completed into lossless encodings in which motifs offer an efficient bilateral context for prediction. Experiments conducted using vocabularies of motifs resulting from off-line greedy strategies show that directional pointers bring about significant savings in encoding. The final part of this paper addresses schemata in which vocabulary build-up and parse are intertwined in much the same way as in traditional ZLW. Their further refinement represents the most promising focus of future work.

References

[1] A. Apostolico and Z. Galil (Eds.), *Pattern Matching Algorithms*, Oxford University Press, New York (1997).

source file	orig len	zlw phrases	max d.c.	lossless phrases	phrases % diff	lossy size	gzip size	gzip % diff
Spor EarlyII	25,008	5,356	4	4,587	-14.36%	4,397	8,008	-45.09%
	25,008	5,356	5	4,192	-21.73%	4,559	8,008	-43.07%
	25,008	5,356	20	4,098	-23.49%	4,673	8,008	-41.65%
Spor EarlyI	31,039	6,446	4	5,662	-12.16%	5,301	9,862	-46.25%
	31,039	6,446	5	5,083	-21.14%	5,471	9,862	-44.52%
	31,039	6,446	20	4,945	-23.29%	5,586	9,862	-43.36%
Spor Middle	54,325	10,409	20	7,887	-24.23%	9,528	16,395	-41.88%
Spor All	222,453	35,920	4	37,759	5.12%	31,477	68,136	-53.80%

Table 6: Results from DNA sequences.

[2] A. Apostolico, M.Comin and L. Parida, "Gapped Codes and their Induced Grammars: Bridging Lossy and Lossless Data Compression by Motif Pattern Discovery", submitted for publication.

[3] A. Apostolico and S. Lonardi, "Off-line Compression by Greedy Textual Substitution", *Proceedings of the IEEE* , **88**(11): 1733–1744 (2000).

[4] A. Apostolico and L. Parida, "Incremental Paradigms of Motif Discovery", (2001) *Journal of Computational Biology*, to appear (2003).

[5] A. Apostolico and L. Parida, "Compression and the Wheel of Fortune", *Proceedings of DCC 2003*, IEEE Computer Society Press, 143–152 (2003).

[6] A. Apostolico and F.P. Preparata, "Data Structures and Algorithms for the String Statistics Problem", *Algorithmica*, **15**, pp. 481–494 (1996).

[7] E. Lehman and A. Shelat, "Approximation Algorithms for Grammar Based Compression", *Proceedings of the eleventh ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, January 2002.

[8] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Trans. on Inform. Theory*, vol. 22, pp. 75–81, 1976.

[9] I. Sadeh "On Approximate String Matching", *Proceedings of DCC 1993*, IEEE COmputer Society Press, 148–157 (1993).

[10] J. A. Storer, *Data Compression: Methods and Theory*, Computer Science Press (1988).

[11] J. Wang, B. Shapiro, and D. Shasha (Eds.), *Pattern Discovery in Biomolecular Data: Tools, Techniques, and Applications* Oxford University Press (1999).

[12] T. A. Welch, "A Technique for High-Performance Data Compression", *IEEE Computer* **17**:6, 8–19, (1984).

[13] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Inform. Theory*, vol. IT-23, no. 3, pp. 337-343, (1977).

[14] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. on Inform. Theory*, vol. 24, no. 5, pp 530-536 (1978).