Compression of Biological Sequences by Greedy Off-line Textual Substitution *

Alberto Apostolico Stefano Lonardi

Purdue University and Università di Padova

1. Introduction and Summary

For some notable classes of data, the two tasks of compression and analysis or interpretation are often and subtly intertwined. Biological sequences, specially DNA, have been long been regarded in this spirit (see, e.g., [6]). The deoxyribonucleic acid (DNA) constitutes the physical medium in which all properties of living organisms are encoded. The knowledge of its sequence is fundamental in molecular biology. Molecular sequence databases (e.g., EMBL, Genbank, DDJB, Entrez, SwissProt, etc.) currently collect hundreds of thousand of sequences of nucleotides and amino-acids from biological laboratories all over the world, reaching into the thousands of gigabytes, and are under continuous expansion.

DNA compression by standard methods such as, e.g., the Lempel-Ziv family of schemes does not seem to fully exploit the redundancies inherent to those sequences. The design of *ad hoc* methods for the compression of genetic sequences constitutes, therefore, an interesting and worthwhile task. Along these lines, a corpus of specialized approaches to DNA compression has been developed in the recent past. As highlighted above, pendant notions of information content and structure have been associated with the compressibility of a sequence. From such a perspective, the amount of compression achievable on genetic sequences has been used in the detection of fragments carrying biological significance, or in assessing the relatedness of fragments and sequences. We refer to, e.g., [2, 6, 7, 8, 10, 11, 12, 15, 16] and references therein for a sampler of the rich literature existing on these subjects.

In bio-sequence repositories and other applications, like for instance in the production of a CD-ROM or magnetic disk for massive data dissemination, one could afford the extra cost of performing compression *off-line* in exchange for some gain in compression [5]. In view of the intractability of optimal off-line macro schemes

*Corresponding Address: Department of Computer Sciences, Purdue University, 1398 Computer Sciences Building, West Lafayette, IN 47907, USA. {axa,stelo}@cs.purdue.edu Work supported in part by NSF Grant CCR-9700276, by Purdue Research Foundation Grant 690-1398-3145, and by the Italian Ministry of University and Research.

[18], various approximate schemes have been considered. Here we follow one of the simplest possible *steepest descent* paradigms. This will consist of performing repeated stages in each one of which we identify a substring of the current version of the text yielding the maximum compression, and then replace all those occurrences except one with a pair of pointers to the untouched occurrence. This is somewhat dual with respect to the bottom up vocabulary buildup scheme considered by Rubin [17] and, more recently, in [9]. As seen in [3], this simple scheme already poses some interesting algorithmic problems. In terms of performance, the method does outperform current Lempel-Ziv implementations in most of the cases. Here we show that, on biological sequences, it beats all other generic compression methods and approaches the performance of methods specifically built around some peculiar regularities of DNA sequences, such as tandem repeats and palindromes, that are neither distinguished nor treated selectively here. The most interesting performances, however, are obtained in the compression of entire groups of genetic sequences forming *families* with similar characteristics. This is becoming a standard and useful way to group sequences in a growing number of important specialized databases. On such inputs, the approach presented here yields scores that are not only better than those of any other method, but also improve increasingly with increasing input size. This is to be attributed to a certain ability to capture distant relationships among the sequences in a family, a feature the merits of which were dramatically exposed in the recent paper [4].

2. Overview of the Method

The basic structure of greedy off-line compression was detailed in [3]. At the generic step, a word is selected such that its encoding would yield the highest contraction in the text. To estimate such a contraction without actually carrying out the encoding, we need to know, for each substring, its maximum possible number of nonoverlapping occurrences in the text. For example, $w = \texttt{aba}$ occurs 11 times in $x = \texttt{abaababaabaababaababababababaa}$, with starting positions in the set $\{1, 4, 6, 9, 12, 14, 17, 19, 21, 23, 25\}$, but only 7 mutually disjoint occurrences can be chosen at most.

Figure 1 displays a linked implementation of the *augmented suffix tree* index structure that supports such a statistics. With respect to a standard statistical index, a twofold modification is needed. On the one hand, the weight (statistics) in each node does no longer coincide with the number of leaves in the subtree rooted at that node, as it would be the case with a standard suffix tree. On the other, auxiliary unary nodes are needed in order to pin down changes in the statistics that occur in the middle of arcs. The efficient construction of this augmented index in minimal form (i.e., with the minimum possible number of unary nodes) is quite elaborate and takes $O(n \log^2 n)$ time and $O(n \log n)$ space. (refer to [3] and references therein). The result is called the *Minimal Augmented Suffix Tree* of x. Alternatively, it is not difficult to build $\hat{T}_x$ in $O(n \log n)$ expected time by straightforward iterated suffix insertions.

We discuss now some encodings and related gain measures. It is not easy to define precisely an a-priori measure of the gain function G that drives the substring selection at each stage, since at the time when we need to estimate the contraction potentially

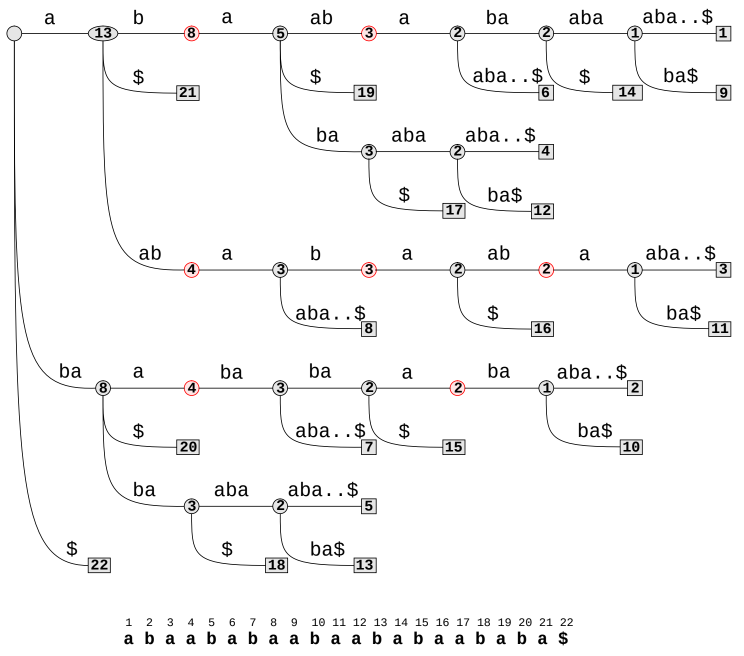

Figure 1: Linked implementation of the minimal augmented suffix tree for abaababaabaababaababa$. The label of a leaf is the starting position of the suffix described on the path from the root to that leaf. Each node is weighed by the maximum possible number of nonoverlapping occurrences for a string terminating at any of the symbols on the preceding arc. Beginning with a standard tree, insertions of unary nodes and weight changes are needed in order to produce this type of index.

induced by a particular substring, we do not have all the costs in place. Letting $l(i)$ be the number of bits needed to encode integer i, we fix tentatively $l(i) = \lceil \log i \rceil$ at the time the gain is computed. We consider three alternative measures of gain, assuming w is the current candidate word.

In Scheme 1, we assume that *all* the f_w nonoverlapping occurrences of the string w are removed from the text, while w itself is saved in an auxiliary data structure that thus contains: (1) the string w, which is Bm_w bits long, where $m_w = |w|$ and $B = \log |\Sigma|$; (2) the length m_w of w, at a cost of $l(m_w)$ bits; and (3) the f_w positions of w in x, at a global cost bounded by $f_w l(n)$ bits. The corresponding gain is given by $G_1(w) = Bf_w m_w - Bm_w - l(m_w) - l(f_w) - f_w l(n) = (f_w - 1)Bm_w - l(m_w) - l(f_w) - f_w l(n)$.

In Scheme 2, we assume that one of the f_w copies of w is kept in the original text, marked by a "literal identification" bit, while the remaining $f_w - 1$ copies are encoded by pointers, each pointer being preceded by a suitable identification bit. For this, we need an auxiliary bit vector. In summary, we need $(B+1)m_w$ bits for the original copy of w, and $(f_w - 1)(l(n) + l(m_w) + 1)$ bits for the $f_w - 1$ pointers. The difference between these expressions yields $G_2(w) = (B+1)f_w m_w - (B+1)m_w - (f_w - 1)(l(n) + l(m_w) + 1)(f_w - 1)(B+1)m_w - (f_w - 1)(l(n) + l(m_w) + 1) = (f_w - 1)((B+1)m_w - l(n) - l(m_w) - 1)$.

In Scheme 3, words in the textfile are replaced by pointers to their corresponding entries in an external dictionary. Thus, following the selection of w at the generic iteration, w is added as a new entry into the dictionary and *all* of its occurrences become pointers to that entry. This still requires an auxiliary bit-vector except possibly for the the words in the dictionary. The plain text representation of all the occurrences of w

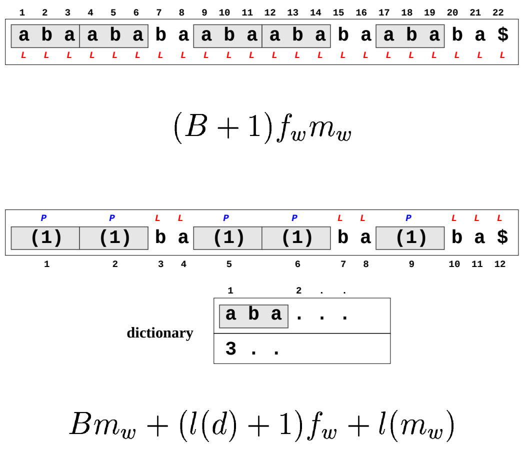
a b a a b a b a a b a a b a b a a b a b a $
L L L L L L L L L L L L L L L L L L L L L L
$(B+1)f_w m_w$
(1) (1) b a (1) (1) b a (1) b a $
P P L L P P L L P L L L
dictionary
a b a . . .
3 . .
$Bm_w + (l(d)+1)f_w + l(m_w)$

Figure 2: Illustrating Scheme 3 for $w =$ aba: d is the size of the dictionary, "(1)" denotes a pointer to the first entry in the dictionary, L and P mark literals and pointers, respectively.

	paper2 (82,199)		mito (78,521)	
encoder	*size*	$time_{[\text{min}]}$	*size*	$time_{[\text{min}]}$
Off-Line$_1$	30,848	3.21	16,426	1.66
Off-Line$_2$	33,757	3.01	17,741	2.24
Off-Line$_3$	**30,219**	2.38	**16,086**	2.38

Figure 3: A first glance at the three Off-Line encoders' performances on a 300Mhz Solaris machine.

encoder	*size*	*bpc*
GZip	91,827	2.33
Pack	86,281	2.19
Compress	86,009	2.18
BZip2	85,705	2.17
BZip	84,809	2.15
Off-Line3	77,764	1.97
CDna [10]	76,471	1.94
BioCompress2 [8]	75,682	1.92
AED [1]	75,407	1.913

Figure 4: Comparing Off-Line with DNA-specific compression programs on the third chromosome (chrIII) of the yeast (315,344 bps). The parameter *bpc* represents the average number of bits per character in the compressed representation (some final sizes are extrapolated from Table 1 of [1]).

requires $(B+1)f_w m_w$ bits. The costs of the pointer-based representation are now (see Figure 2): Bm_w bits for the string w in the dictionary, $l(m_w)$ to store the length m_w, $l(d)f_w$ for the f_w pointers inside the text, where d is the size of the dictionary. The resulting expression for $G_3(w)$ is $G_3(w) = (B+1)f_w m_w - Bm_w - (l(d)+1)f_w - l(m_w) = B(f_w - 1)m_w + f_w m_w - (l(d)+1)f_w - l(m_w)$.

It is important to observe that, for any of the above specifications of G and any word w in x, $G(w)$ is a monotone increasing function of m_w. Moreover, the maximum number of nonoverlapping occurrences of w in x does not change in the middle of an arc of $\hat{T}_x$. Therefore, the word maximizing the gain at each stage *always* ends on a node of $\hat{T}_x$. If now w is this word, then its occurrences are suitably encoded, and the whole process is repeated until the gain becomes zero or negligible, according to some predetermined threshold.

3. Results

The three schemes just described were embedded in as many encoders, respectively called Off-Line$_1$, Off-Line$_2$ and Off-Line$_3$, implemented in C++, and extensively tested. Table 3 offers a first glance of the performances of the three encoders on two typical inputs, namely, paper2 from the Calgary Corpus and mito, the mitochondrial DNA sequence of the yeast (*Saccharomyces Cerevisiae*). Running times are in the order of 2-3 minutes for files of about 80 KB on a 300 Mhz machine running under Solaris. In terms of compression, the best encoder is Off-Line$_3$, followed by Off-Line$_1$ and, at some distance, Off-Line$_2$.

Testing performances among textual substitution methods over the entire Calgary Corpus, Off-Line$_3$ outperforms the other two encoders on most inputs. As a whole, Off-Line encoders perform better than the rest on most inputs, and loose marginally to GZip where they do. Crossing the boundary of textual substitution methods, the block-sorting techniques Bzip and Bzip2 outperform Gzip and Off-Line on the whole Calgary Corpus. As is seen next, a different scenario is displayed when we turn to biological data sets.

We compare the performance of Off-Line encoders with those of standard compression programs in the Table 1. The encoder Off-Line$_3$ outperforms each and every general purpose encoder on the fourteen chromosomes and the mitochondrial DNA of the yeast. It should be noted that the actual compressions are very small and sometimes negative. In fact, raw biological sequences (notably, those coming from coding regions [13]) However, even comparing our encoders with programs specifically designed to compress DNA, the difference in performance is not large, as shown in the Figure 4.

It is worthwhile to highlight such DNA-specific analyzers and compressors. As mentioned, information theoretic analyses of biological sequences mingle with the very dawn of bioinformatics studies (see, e.g., [6]), but this area has known recently a considerable revival of interest in view of the massive production of genomic sequences of various kinds. In this context, the detection of redundancy serves not only the purpose of achieving more compact descriptors, but also, and perhaps more importantly, may act as a filter of possibly relevant biological functions. The tenet there is that an incompressible string is more random and thus less likely than a repetitive one to carry some biological function.

Due to mutations, errors in the sequencing process, and other biological events, a substantial part of the redundancy present in DNA manifests itself in form of consecutive (*tandem*) repeats of the same word or *motif*, and palindromes. However, such tandem repeats and palindromes are not exact. Rather, they may occur with substitutions, insertions or deletions of symbols. Moreover, palindromes are actually *complemented*, meaning that in the reverse half of the word the base `A` is mirrored by a `T` (and vice-versa), while `C` is mirrored by a `G` (and vice-versa).

Among the recent dedicated approaches to DNA compression, the one by Grumbach and Tahi [7, 8], called BioCompress2, extends LZ-77 to catch very distant repeats and complementary palindromes.

Loewenstern and Yianilos [10] consider the problem of computing good estimates of the entropy of DNA sequences by building a Ppm-like predictive model. With respect to the original Ppm, they extend the context model by allowing mismatches. Their algorithm estimates the parameters of the model, called Cdna, via a learning process that tries to optimize a complex objective function. The general problem is known to be $\mathcal{NP}$-complete, but they devise more realistic approximation schemes.

Allison, Edgoose and Dix propose the most computationally intensive approach to DNA compression [1]. They search for both approximate repeats and approximate palindromes. Their primary purpose is not to compress the text, but rather to model the statistical properties of the data as accurately as possible and to find patterns and structures within them. They build a model with parameters such as the probability of repeats, of the length of repeats, and of mismatches within repeats. The parameters of the model are estimated by an expectation maximization algorithm that takes time $O(n^2)$ at each iteration. Their results may well be taken to represent the current "state of the art", but as said the algorithm is extremely slow.

File	*Size* (bytes)	*Huffman* PACK	*LZ-78* COMPRESS	*LZ-77* GZIP	*BWT* BZIP	*BWT* BZIP2	OFF-LINE3
chrI	230,195	63,144	62,935	66,264	61,674	62,373	**56,915**
chrII	813,137	222,597	219,845	236,837	218,463	221,032	**201,180**
chrIII	315,344	86,281	86,009	91,827	84,809	85,705	**77,764**
chrIV	1,522,191	416,516	409,957	440,056	407,799	411,250	**370,796**
chrV	574,860	157,415	155,944	167,749	154,580	155,731	**141,919**
chrVI	270,148	74,077	73,873	78,925	72,838	73,651	**67,391**
chrVII	1,090,936	298,680	294,417	317,282	293,079	296,245	**269,265**
chrVIII	562,638	154,110	152,265	163,135	151,240	152,992	**139,271**
chrIX	439,885	120,669	118,965	127,805	118,182	119,553	**109,303**
chrX	745,443	204,152	201,783	216,148	200,325	202,223	**184,287**
chrXI	666,448	182,377	180,100	194,119	179,306	180,901	**165,478**
chrXII	1,078,171	295,441	291,754	305,653	288,112	290,800	**259,898**
chrXIII	924,430	253,176	249,099	267,127	248,450	250,735	**227,610**
chrXIV	784,328	215,020	212,219	228,757	210,988	212,816	**194,947**
chrXV	1,091,282	298,762	294,921	317,971	293,838	297,279	**269,921**
chrXVI	948,061	286,579	264,113	278,651	254,947	257,590	**233,150**
mito	78,521	18,149	17,890	19,369	17,962	18,094	**16,086**

Table 1: Comparing OFF-LINE with other compression programs on the chromosomes of the yeast.

Finally, we run OFF-LINE3 on families of related and unrelated genetic sequences. Entries in most genetic databases are flat text files containing one or more sequences that are usually functionally related, with some annotations. The `fasta` format is the most commonly used standard for storing and exchanging genetic files. The generic `fasta` file contains one or more blocks. Each block is composed by one or more annotation lines each starting with the symbol `>`, followed by the genetic sequence.

Table 2 shows the results of running OFF-LINE3 on several families of randomly chosen sequences of the yeast genome. The complete dataset is available at `http://www.cs.purdue.edu/homes/stelo/Off-line/`. The file `Spor_All_2x.fasta` is artificially obtained by concatenating `Spor_All.fasta` with itself, in an attempt to probe into extreme cases of inter-sequence correlation [4]. The last two families (8 and 9) are a segment of *all* the upstream regions of the yeast and thus not functionally related. Table 2 shows that not only the absolute performance of OFF-LINE, but also its relative advantage over the other methods improves as the input size increases. Likewise, as soon as the input files contain sequences not as strongly related, the improvements, while still present, decay immediately, as shown for files 8 and 9 in the table. The ability to capture distant relationships is enhanced in the comparison with GZIP and BZIP2 as we move from their default window sizes (900Kb in BZIP2) to smaller sizes. The results, shown in Figure 7, suggest that the relative advantage of OFF-LINE will increase as it will be applied to larger and larger families.

4. Fine Tunings

The most time-consuming activity of the compression phase is the construction of the augmented suffix tree and its annotation with the values of the gain. We employed three heuristics to overcome the high computational demands of a "full-fledged" version of the compressor.

Table 5 shows the results achieved by one of these heuristics on the basic algorithm, in which more than just one substring selection and substitution is performed between

	`paper2`		`mito`	
Q	*size*	$time_{[\text{min}]}$	*size*	$time_{[\text{min}]}$
1	30,773	19.70	16,326	7.06
2	30,780	10.36	16,367	4.06
5	30,785	5.06	16,405	2.24
10	30,787	3.21	16,446	1.66
20	30,826	2.39	16,476	1.36
50	30,904	1.97	16,632	1.28
100	30,923	1.86	16,702	1.37
1,000	30,923	1.98	16,702	1.47

Figure 5: Performances of Off-Line_1 for different sizes of the candidates heap. We fixed `min_occ` $= 2$, `min_length` $= 2$, $l = 100$.

two consecutive updates of the statistical index. Of course, such an approach saves time on one hand, but it risks blurring the perception of the best candidates for substitution. In our implementation, a heap is maintained with the statistical index, containing at each step the Q best words in terms of G, for some chosen value of the parameter Q. Between any two consecutive index reconstructions, the Q strings in the heap are retrieved and used in succession in a contraction step for the text. It is possible at some point that a string from the heap will no longer be found in the contracted text. In fact, part of the words in the heap turn out to be useless in general. In any case, as soon as all words in the heap have been considered, a new augmented suffix tree is built on the contracted text.

As the Table displays, the number of individual substring substitution passes over the text grows with the maximum allowed size of the heap. On the other hand, we spend less and less time building weighted suffix trees. The overall result is, within a wide interval, a considerable speed up with respect to the eager version of Off-Line without substantial penalty in compression performance. When the size of the heap becomes too large (approximately $Q > 100$ in our experiments) only a small subset of the words in the heap is used: most of the computational effort is spent in pattern searching, which results in deterioration of both speed and compression.

Whenever one can assume it as being highly unlikely that very long words occur frequently in a text, then building the statistics for *all* the substrings can be a waste of resources. Pruning the tree speeds up considerably the implementation and saves large amounts of memory. Pruning the tree does not mean that we could completely miss the word involved in a long substitution. If the current best substitution is a word w longer than the threshold l, then the encoder will eventually choose some substring of w of length l because that substring occurs without overlap at least as many times as w.

Table 6 shows that the pruned version of Off-Line_1 at $l = 100$ performs almost ten time faster and achieves exactly the same compression as the version that builds the complete tree.

The collective speed-up gained from these heuristics combined is significant: our original implementation took several hours to compress those files while it would process afterwards in few minutes. What is even better, the corresponding loss of efficiency in terms of compression is almost negligible.

As documented in some additional tables, a few hundred iterations of the word selection loop of Off-Line suffice on inputs of the order of 100,000 symbols. This sug-

	paper2		mito	
l	*size*	*time*$_{[\text{min}]}$	*size*	*time*$_{[\text{min}]}$
10	30,986	2.58	17,044	0.29
50	30,664	2.62	16,491	1.32
100	30,636	2.68	16,470	1.38
∞	30,636	19.39	16,470	10.34

Figure 6: Comparing the performance of OFF-LINE$_1$ for different choices of the maximum allowed length of a candidate for substitution. We fixed min_occ $= 4, l = 4, Q = 10$.

Family	*LZ-77*	*BWT*	
	GZIP -1	BZIP2 -1	OFF-LINE$_3$
(6)	76,629$_{(29.1\%)}$	63,332$_{(14.2\%)}$	**54,317**
(7)	153,103$_{(57.0\%)}$	126,314$_{(47.8\%)}$	**65,891**

Figure 7: Constraining the competitors to work on small windows enhances the gain of OFF-LINE. Here the input strings 6 and 7 correspond, respectively, to the families of Spor_All.fasta, Spor_All_2x.fasta (cf. table 2 for their respective statistics).

File	*Size*	OFF-LINE$_1$	OFF-LINE$_2$	OFF-LINE$_3$
chrI	230,195	78	603	80
chrII	813,137	112	474	128
chrIII	315,344	61	309	68
chrIV	1,522,191	383	1297	441
chrV	574,860	109	276	118
chrVI	270,148	22	226	30
chrVII	1,090,936	144	1009	162
chrVIII	562,638	91	264	102
chrIX	439,885	54	543	63
chrX	745,443	108	376	123
chrXI	666,448	49	302	58
chrXII	1,078,171	444	1443	499
chrXIII	924,430	187	706	212
chrXIV	784,328	24	441	72
chrXV	1,091,282	128	924	147
chrXVI	948,061	193	755	217

Figure 8: Iterations of the main loop of OFF-LINE on the chromosomes of the yeast.

Family	*Total size*		*Huffman*	*LZ-78*	*LZ-77*	*BWT*	
	(bytes)	*k*	PACK	COMPRESS	GZIP	BZIP2 -9	OFF-LINE$_3$
(1)	25,008	29	7,996$_{(11.0\%)}$	7,875$_{(9.6\%)}$	8,008$_{(11.1\%)}$	7,300$_{(2.5\%)}$	**7,119**
(2)	31,039	36	9,937$_{(12.5\%)}$	9,646$_{(9.8\%)}$	9,862$_{(11.8\%)}$	9,045$_{(3.8\%)}$	**8,697**
(3)	32,871	38	10,590$_{(12.2\%)}$	10,223$_{(9.0\%)}$	10,379$_{(10.4\%)}$	9,530$_{(2.4\%)}$	**9,301**
(4)	54,325	63	17,295$_{(14.6\%)}$	16,395$_{(9.9\%)}$	16,961$_{(12.9\%)}$	15,490$_{(4.6\%)}$	**14,778**
(5)	112,507	130	36,172$_{(17.7\%)}$	33,440$_{(11.0\%)}$	33,829$_{(12.0\%)}$	31,793$_{(6.4\%)}$	**29,758**
(6)	222,453	258	70,755$_{(23.2\%)}$	63,939$_{(15.0\%)}$	68,136$_{(20.3\%)}$	61,674$_{(11.9\%)}$	**54,317**
(7)	444,906	516	141,431$_{(53.4\%)}$	124,637$_{(47.1\%)}$	135,816$_{(51.5\%)}$	85,142$_{(22.6\%)}$	**65,891**
(8)	399,615	191	121,700$_{(12.3\%)}$	115,029$_{(7.22\%)}$	115,023$_{(7.22\%)}$	112,363$_{(5.0\%)}$	**106,722**
(9)	1,001,002	477	305,054$_{(11.9\%)}$	286,971$_{(6.4\%)}$	285,064$_{(5.8\%)}$	280,334$_{(4.1\%)}$	**268,612**

Table 2: Comparing OFF-LINE$_3$ with other compression programs on families of sequences of the yeast. The figures in parentheses report percentage gains achieved by OFF-LINE$_3$, k is the number of upstream sequences in each family, individual sequence length is 800 bps except in the last two rows, where it is 2,000. The alphabet consists of about 50 symbols. The input strings 1-9 correspond, in this order to the families of Spor_EarlyII.fasta, Spor_EarlyI.fasta, Helden_GCN.fasta, Spor_Middle.fasta, Helden_All.fasta, Spor_All.fasta, Spor_All_2x.fasta, All_Up_400k.fasta, All_Up_1M.fasta.

gests that dedicated fine-grained parallel architectures of this kind would implement virtually instantaneous encoders for biosequences and general inputs alike. Table 8 shows the modest number of iterations of the main loop performed by OFF-LINE on our inputs. The experiments that subtend Figures 5 and 6 suggest that such a figure might become negligible in practical cases. Therefore, the most expensive tasks, represented by the tree constructions, can be limited considerably in a parallel implementation, turning the method into an on-line, even real-time application.

Since the number of iterations performed determines the size of the vocabulary, whence ultimately of pointers, this generates "quantization" phenomena in the neighborhood of certain values that play critical roles in a computer program. Figure 9 displays the sensitivity of the current implementations to pointer encodings at the crossing of one byte. The two curves plot the sizes of the compressed strings mito and paper2, respectively, at all consecutive stages of the iterated substitutions performed by OFF-LINE$_3$. Following a steady increase until iteration 256, the compression starts decreasing as soon as OFF-LINE$_3$ must employ more than one byte to represent a

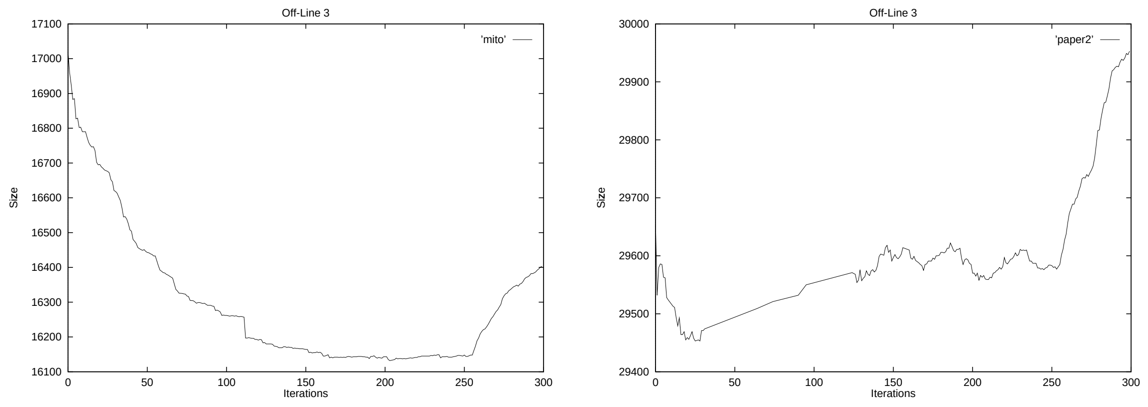

Figure 9: Compressed sizes of `mito` (left) and `paper2` (right) versus number of iterations of OFF-LINE3.

pointer. In addition to this, the erratic shape of the plot for `paper2` suggests, with its several local minima, that it is hard at run time to pin down precisely the best moment when to stop the iterations.

5. Concluding Remarks

We have presented a small battery of compressors that perform well on all data but especially well on biological data. The basic paradigm is uncluttered, relatively easy to program, and acceptably fast in comparison to ad-hoc, considerably slower and more involved methods.

Besides the obvious challenge of developing versions specifically tailored to biological sequence data, a number of interesting questions emerged in the course of the experiments which shall warrant further study and experimentation. These include analysis of allowing variable window sizes, better approximations of the gain, fine-tuning of the number of iterations and of the encoding at the outset. As already noted in [3], OFF-LINE may be usefully regarded also as a paradigm for inferring hierarchical grammatical structures in sequences, along the lines of [14], which appears to yield interesting insights into the structure of biological and general sequences alike.

Acknowledgements. We are thankful to I. Rivals, J. Storer and F. Tahi for helpful discussions.

References

[1] L. Allison, T. Edgoose, and T. I. Dix. Compression of strings with approximate repeats. *Intell. Sys. in Mol. Biol.*, pages 8–16, 1998.

[2] L. Allison, D. Powell, and T. I. Dix. Compression and Approximate Matching, *Computer Journal, 42*, vol.1, pages 1–10, 1999.

[3] A. Apostolico and S. Lonardi. Some theory and practice of greedy off-line textual substitution. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 119–128, Snowbird, Utah, 1998.

[4] J. Bentley and D. McIrlroy. Data compression using long common strings. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 287–295, Snowbird, Utah, 1999.

[5] S. DeAgostino and J. A. Storer. On-line versus off-line computation in dynamic text compression. *Inform. Process. Lett.*, vol.59, no.3, pages 169–174, 1996.

[6] L. Gatlin. *Information Theory and the Living Systems.* Columbia University Press, 1972.

[7] S. Grumbach and F. Tahi. Compression of DNA sequences. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 340–350, Snowbird, Utah, 1993.

[8] S. Grumbach and F. Tahi. A new challenge for compression algorithms: genetic sequences. *Inform. Proc. and Mngm.*, vol.30, no.6, pages 875–886, 1994.

[9] N. J. Larsson and A. Moffat. Offline dictionary-based compression, In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 296–305, Snowbird, Utah, 1999.

[10] D. M. Loewenstern and P. N. Yianilos. Significant lower entropy estimates for natural DNA sequences. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 151–160, Snowbird, Utah, 1997. Also, *Journal of Computational Biology*, vol.6, no.1, 1999.

[11] D. M. Loewenstern, H. M. Berman, and H. Hirsch. Maximum a posteriori classification of DNA structure from sequence information. *Pacific Symp. Biocomputing*, pages 667–678, 1998.

[12] A. Milosavljevic and J. Jurka. Discovery by minimal length encoding: a case study in molecular evolution. *Machine Learning*, vol.12, pages 69–87, 1993.

[13] C. Nevill-Manning, and I. H. Witten. Protein is incompressible. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 257–266, Snowbird, Utah, 1999.

[14] C. Nevill-Manning, I. H. Witten, and D. Maulsby. Compression by induction of hierarchical grammars. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, pages 244–253, Snowbird, Utah, 1994.

[15] E. Rivals, J. P. Delahaye, M. Dauchet, and O. Delgrange. A guaranteed compression scheme for repetitive DNA sequences. In J. A. Storer and M. Cohn, eds., *Data Compression Conference*, page 453, Snowbird, Utah, 1996.

[16] E. Rivals, O. Delgrange, J. P. Delahaye, M. Dauchet, M. O. Delorme, A. Henaut, and E. Ollivier. Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences. *CABIOS*, vol.13, no.2, pages 131–136, 1997.

[17] F. Rubin. Experiments in text file compression. *Communications of the ACM*, vol.19, no.11, pages 617–623, Nov. 1976.

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