1999

Compression of Biological Sequences by Greedy Off-Line Textual Substitution

Alberto Apostolico

Stefano Lonardi

Report Number:
99-037
COMPRESSION OF BIOLOGICAL SEQUENCES BY
GREEDY OFF-LINE TEXTUAL SUBSTITUTION

Alberto Apostolico
Stefano Lonardi

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #99-037

November 1999
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, especially 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 thousands 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
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, \( w = \text{aba} \) occurs 11 times in \( x = \text{abaababaabaabababababababaa} \), 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 \( T_x \) 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
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 \( B m_w \) 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 \( G_1(w) = B f_w m_w - B m_w - l(m_w) - l(f_w) - f_w l(n) = (f_w - 1)B m_w - l(m_w) - l(f_w) - f_w l(n) \).

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 \( G_2(w) = (B+1)f_w m_w - (B+1)m_w - (f_w - 1)(l(n) + l(m_w) + 1) = (f_w - 1)((B+1)m_w - l(n) - l(m_w) - 1) \).

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 words in the dictionary. The plain text representation of all the occurrences of \( w \)

![Figure 1: Linked implementation of the minimal augmented suffix tree for ababababababababababa$. 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.](image-url)
requires \((B+1)f_wm_w\) 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 

\[
G_3(w) = (B + 1)f_wm_w - Bm_w - l(m_w) = B(f_w - 1)m_w + f_wm_w - l(d) + 1\]

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 \(T_x\). Therefore, the word maximizing the gain at each stage always ends on a node of \(T_x\). 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-LINE1, OFF-LINE2 and OFF-LINE3, 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-LINE3, followed by OFF-LINE1 and, at some distance, OFF-LINE2.

Testing performances among textual substitution methods over the entire Calgary Corpus, OFF-LINE1 outperforms the other two encoders on most inputs. As a whole, OFF-LINE encoders perform better than the rest on most inputs, and lose 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-LINE3 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]) are known to be hard to compress. 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.
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 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 strongly 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 index trie 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

<table>
<thead>
<tr>
<th>File</th>
<th>Size (bytes)</th>
<th>Huffman Pack</th>
<th>LZ-78 Compass</th>
<th>LZ-77 GZIP</th>
<th>BWT BZIP</th>
<th>BWT BZIP2</th>
<th>OFF-LINE3</th>
</tr>
</thead>
<tbody>
<tr>
<td>chrI</td>
<td>230,195</td>
<td>63,144</td>
<td>62,535</td>
<td>66,264</td>
<td>61,674</td>
<td>62,373</td>
<td>56,015</td>
</tr>
<tr>
<td>chrII</td>
<td>813,137</td>
<td>222,597</td>
<td>219,845</td>
<td>236,837</td>
<td>218,463</td>
<td>231,032</td>
<td>201,130</td>
</tr>
<tr>
<td>chrIII</td>
<td>315,344</td>
<td>86,281</td>
<td>86,009</td>
<td>91,627</td>
<td>84,809</td>
<td>88,705</td>
<td>77,798</td>
</tr>
<tr>
<td>chrIV</td>
<td>1,221,191</td>
<td>416,316</td>
<td>409,597</td>
<td>440,050</td>
<td>407,799</td>
<td>411,260</td>
<td>370,799</td>
</tr>
<tr>
<td>chrV</td>
<td>874,660</td>
<td>157,415</td>
<td>155,946</td>
<td>167,749</td>
<td>154,880</td>
<td>168,731</td>
<td>141,911</td>
</tr>
<tr>
<td>chrVI</td>
<td>740,148</td>
<td>74,077</td>
<td>73,873</td>
<td>78,925</td>
<td>72,383</td>
<td>73,651</td>
<td>67,391</td>
</tr>
<tr>
<td>chrVII</td>
<td>1,090,936</td>
<td>298,680</td>
<td>294,417</td>
<td>317,282</td>
<td>293,079</td>
<td>296,245</td>
<td>269,265</td>
</tr>
<tr>
<td>chrVIII</td>
<td>562,638</td>
<td>154,114</td>
<td>152,265</td>
<td>163,135</td>
<td>151,240</td>
<td>152,992</td>
<td>139,271</td>
</tr>
<tr>
<td>chrIX</td>
<td>439,885</td>
<td>120,669</td>
<td>116,956</td>
<td>127,605</td>
<td>116,182</td>
<td>119,563</td>
<td>105,303</td>
</tr>
<tr>
<td>chrX</td>
<td>745,443</td>
<td>204,152</td>
<td>201,783</td>
<td>216,148</td>
<td>200,325</td>
<td>202,223</td>
<td>184,287</td>
</tr>
<tr>
<td>chrXI</td>
<td>666,448</td>
<td>182,377</td>
<td>180,100</td>
<td>194,119</td>
<td>179,306</td>
<td>180,961</td>
<td>165,478</td>
</tr>
<tr>
<td>chrXII</td>
<td>1,078,171</td>
<td>295,441</td>
<td>291,754</td>
<td>305,693</td>
<td>288,112</td>
<td>290,290</td>
<td>255,958</td>
</tr>
<tr>
<td>chrXIII</td>
<td>924,450</td>
<td>253,316</td>
<td>240,099</td>
<td>267,127</td>
<td>246,450</td>
<td>256,735</td>
<td>227,610</td>
</tr>
<tr>
<td>chrXIV</td>
<td>784,328</td>
<td>215,202</td>
<td>212,219</td>
<td>228,757</td>
<td>210,988</td>
<td>212,316</td>
<td>194,947</td>
</tr>
<tr>
<td>chrXV</td>
<td>1,090,232</td>
<td>298,762</td>
<td>294,921</td>
<td>317,971</td>
<td>293,838</td>
<td>297,279</td>
<td>269,921</td>
</tr>
<tr>
<td>chrXVI</td>
<td>948,061</td>
<td>286,579</td>
<td>284,113</td>
<td>278,651</td>
<td>254,947</td>
<td>257,590</td>
<td>233,150</td>
</tr>
<tr>
<td>mito</td>
<td>78,521</td>
<td>18,149</td>
<td>17,890</td>
<td>19,369</td>
<td>17,652</td>
<td>18,094</td>
<td>16,086</td>
</tr>
</tbody>
</table>
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 trie 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 tries. 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 $w$. The table in Figure 6 shows that the pruned version of Off-Line
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 afterwards
it would complete 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
gests that dedicated fine-grained parallel architectures of this kind would implement
virtually instantaneous encoders for biosequences and general inputs alike. Figure 8 shows the modest number of iterations of the main loop performed by Off-Line on our inputs. The experiments reported in Figures 5 and 6 show that such a number of iterations is negligible in a parallel context. Therefore, the most expensive tasks, represented by the tree constructions, can be limited considerably in a parallel implementation, turning the method into an online, 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. Following a steady increase until iteration 256, the compression starts decreasing as soon as Off-Line must employ more than one byte to represent a pointer. In addition to this, the erratic shape of the plot for paper2 suggests, with...
Figure 9: Compressed sizes of mito (left) and paper2 (right) versus number of iterations of OPP-LINE2.

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], OPP-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 E. Rivals, J. Storer and F. Tahi for helpful discussions.

References


