We are IntechOpen, the world's leading publisher of Open Access books
Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

154
Countries delivered to

Our authors are among the

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS
BOOK CITATION INDEX
INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

6

Finite-context models for DNA coding[*]

Armando J. Pinho, António J. R. Neves, Daniel A. Martins,
Carlos A. C. Bastos and Paulo J. S. G. Ferreira
Signal Processing Lab, DETI/IEETA, University of Aveiro
Portugal

1. Introduction

Usually, the purpose of studying data compression algorithms is twofold. The need for efficient storage and transmission is often the main motivation, but underlying every compression technique there is a model that tries to reproduce as closely as possible the information source to be compressed. This model may be interesting on its own, as it can shed light on the statistical properties of the source. DNA data are no exception. We urge to find out efficient methods able to reduce the storage space taken by the impressive amount of genomic data that are continuously being generated. Nevertheless, we also desire to know how the code of life works and what is its structure. Creating good (compression) models for DNA is one of the ways to achieve these goals.

Recently, and with the completion of the human genome sequencing, the development of efficient lossless compression methods for DNA sequences gained considerable interest (Behzadi and Le Fessant, 2005; Cao et al., 2007; Chen et al., 2001; Grumbach and Tahi, 1993; Korodi and Tabus, 2005; 2007; Manzini and Rastero, 2004; Matsumoto et al., 2000; Pinho et al., 2006; 2009; 2008; Rivals et al., 1996). For example, the human genome is determined by approximately 3 000 million base pairs (Rowen et al., 1997), whereas the genome of wheat has about 16 000 million (Dennis and Surridge, 2000). Since DNA is based on an alphabet of four different symbols (usually known as nucleotides or bases), namely, Adenine (*A*), Cytosine (*C*), Guanine (*G*), and Thymine (*T*), without compression it takes approximately 750 MBytes to store the human genome (using $\log_2 4 = 2$ bits per symbol) and 4 GBytes to store the genome of wheat.

In this chapter, we address the problem of DNA data modeling and coding. We review the main approaches proposed in the literature over the last fifteen years and we present some recent advances attained with finite-context models (Pinho et al., 2006; 2009; 2008). Low-order finite-context models have been used for DNA compression as a secondary, fall back method. However, we have shown that models of orders higher than four are indeed able to attain significant compression performance.

Initially, we proposed a three-state finite-context model for DNA protein-coding regions, i.e., for the parts of the DNA that carry information regarding how proteins are synthesized (Ferreira et al., 2006; Pinho et al., 2006). This three-state model proved to be better than a single-state model, giving additional evidence of a phenomenon that is common in these protein-coding regions, the periodicity of period three.

More recently (Pinho et al., 2008), we investigated the performance of finite-context models for unrestricted DNA, i.e., DNA including coding and non-coding parts. In that work, we have shown that a characteristic usually found in DNA sequences, the occurrence of inverted repeats, which is used by most of the DNA coding methods (see, for example, Korodi and Tabus (2005); Manzini and Rastero (2004); Matsumoto et al. (2000)), could also be successfully integrated in finite-context models. Inverted repeats are copies of DNA sub-sequences that appear reversed and complemented ($A \leftrightarrow T$, $C \leftrightarrow G$) in some parts of the DNA.

Further studies have shown that multiple competing finite-context models, working on a block basis, could be more effective in capturing the statistical information along the sequence (Pinho et al., 2009). For each block, the best of the models is chosen, i.e., the one that requires less bits for representing the block. In fact, DNA is non-stationary, with regions of low information content (low entropy) alternating with regions with average entropy close to two bits per base. This alternation is modeled by most DNA compression algorithms by using a low-order finite-context model for the high entropy regions and a Lempel-Ziv dictionary based approach for the repetitive, low entropy regions. In this work, we rely only on finite-context models for representing both regions.

Modeling DNA data using only finite-context models has advantages over the typical DNA compression approaches that mix purely statistical (for example, finite-context models) with substitutional models (such as Lempel-Ziv based algorithms): (1) finite-context models lead to much faster performance, a characteristic of paramount importance for long sequences (for example, some human chromosomes have more than 200 million bases); (2) the overall model might be easier to interpret, because it is made of sub-models of the same type.

This chapter is organized as follows. In Section 2 we provide an overview of the DNA compression methods that have been proposed. Section 3 describes the finite-context models used in this work. These models collect the statistical information needed by the arithmetic coding. In Section 4 we provide some experimental results. Finally, in Section 5 we draw some conclusions.

2. DNA compression methods

The interest in DNA coding has been growing with the increasing availability of extensive genomic databases. Although only two bits are sufficient to encode the four DNA bases, efficient lossless compression methods are still needed due to the large size of DNA sequences and because standard compression algorithms do not perform well on DNA sequences. As a result, several specific coding methods have been proposed. Most of these methods are based on searching procedures for finding exact or approximate repeats.

The first method designed specifically for compressing DNA sequences was proposed by Grumbach and Tahi (1993) and was named *Biocompress*. This technique is based on the sliding window algorithm proposed by Ziv and Lempel, also known as LZ77 (Ziv and Lempel, 1977). According to this universal data compression technique, a sub-sequence is encoded using a reference to an identical sub-sequence that occurred in the past. *Biocompress* uses a characteristic usually found in DNA sequences which is the occurrence of inverted repeats. These are sub-sequences that are both reversed and complemented ($A \leftrightarrow T$, $C \leftrightarrow G$). The second version of *Biocompress*, *Biocompress-2*, introduced an additional mode of operation, based on an order-2 finite-context arithmetic encoder (Grumbach and Tahi, 1994).

Rivals et al. (1995; 1996) proposed another compression technique based on exact repetitions, *Cfact*, which relies on a two-pass strategy. In the first pass, the complete sequence is parsed using a suffix tree, producing a list of the longest repeating sub-sequences that have a potential

coding gain. In the second pass, those sub-sequences are encoded using references to the past, whereas the rest of the symbols are left uncompressed.

The idea of using repeating sub-sequences was also exploited by Chen et al. (1999; 2001). The authors proposed a generalization of this strategy such that approximate repeats of sub-sequences and of inverted repeats could also be handled. In order to reproduce the original sequence, the algorithm, named *GenCompress*, uses operations such as replacements, insertions and deletions. As in *Biocompress*, *GenCompress* includes a mechanism for deciding if it is worthwhile to encode the sub-sequence under evaluation using the substitution-based model. If not, it falls back to a mode of operation based on an order-2 finite-context arithmetic encoder. A further modification of *GenCompress* led to a two-pass algorithm, *DNACompress*, relying on a separated tool for approximate repeat searching, *PatternHunter*, (Chen et al., 2002). Besides providing additional compression gains, *DNACompress* is considerably faster than *GenCompress*.

Before the publication of *DNACompress*, a technique based on context tree weighting (CTW) and LZ-based compression, *CTW+LZ*, was proposed by Matsumoto et al. (2000). Basically, long repeating sub-sequences or inverted repeats, exact or approximate, are encoded by a LZ-type algorithm, whereas short sub-sequences are compressed using CTW.

One of the main problems of techniques based on sub-sequence matching is the time taken by the search operation. Manzini and Rastero (2004) addressed this problem and proposed a fast, although competitive, DNA encoder, based on fingerprints. Basically, in this approach small sub-sequences are not considered for matching. Instead, the algorithm focus on finding long matching sub-sequences (or inverted repeats). Like most of the other methods, this technique also uses fall back mechanisms for the regions where matching fails, in this case, finite-context arithmetic coding of order-2 (*DNA2*) or order-3 (*DNA3*).

Tabus et al. (2003) proposed a sophisticated DNA sequence compression method based on normalized maximum likelihood discrete regression for approximate block matching. This work, later improved for compression performance and speed (Korodi and Tabus (2005), *GeNML*), encodes fixed-size blocks by referencing a previously encoded sub-sequence with minimum Hamming distance. Only replacement operations are allowed for editing the reference sub-sequence which, therefore, always have the same size as the block, although may be located in an arbitrary position inside the already encoded sequence. Fall back modes of operation are also considered, namely, a finite-context arithmetic encoder of order-1 and a transparent mode in which the block passes uncompressed.

Behzadi and Le Fessant (2005) proposed the *DNAPack* algorithm, which uses the Hamming distance (i.e., it relies only on substitutions) for the repeats and inverted repeats, and either CTW or order-2 arithmetic coding for non-repeating regions. Moreover, *DNAPack* uses dynamic programming techniques for choosing the repeats, instead of greedy approaches as others do.

More recently, two other methods have been proposed (Cao et al., 2007; Korodi and Tabus, 2007). One of them (Korodi and Tabus, 2007), is an evolution of the normalized maximum likelihood model introduced by Tabus et al. (2003) and improved by Korodi and Tabus (2005). This new version, *NML-1*, is built on the *GeNML* framework and aims at finding the best regressor block using first-order dependencies (these dependencies were not considered in the previous approach).

The other method, proposed by Cao et al. (2007) and called *XM*, relies on a mixture of experts for providing symbol by symbol probability estimates which are then used for driving an arithmetic encoder. The algorithm comprises three types of experts: (1) order-2

Markov models; (2) order-1 context Markov models, i.e., Markov models that use statistical information only of a recent past (typically, the 512 previous symbols); (3) the copy expert, that considers the next symbol as part of a copied region from a particular offset. The probability estimates provided by the set of experts are them combined using Bayesian averaging and sent to the arithmetic encoder. Currently, this seems to be the method that provides the highest compression on the April 14, 2003 release of the human genome (see results in `ftp://ftp.infotech.monash.edu.au/software/DNAcompress-XM/XMCompress/humanGenome.html`). However, both *NML-1* and *XM* are computationally intensive techniques.

3. Finite-context models

Consider an information source that generates symbols, s, from an alphabet $\mathcal{A}$. At time t, the sequence of outcomes generated by the source is $x^t = x_1 x_2 \ldots x_t$. A finite-context model of an information source (see Fig. 1) assigns probability estimates to the symbols of the alphabet, according to a conditioning context computed over a finite and fixed number, M, of past outcomes (order-M finite-context model) (Bell et al., 1990; Salomon, 2007; Sayood, 2006). At time t, we represent these conditioning outcomes by $c^t = x_{t-M+1}, \ldots, x_{t-1}, x_t$. The number of conditioning states of the model is $|\mathcal{A}|^M$, dictating the model complexity or cost. In the case of DNA, since $|\mathcal{A}| = 4$, an order-M model implies 4^M conditioning states.
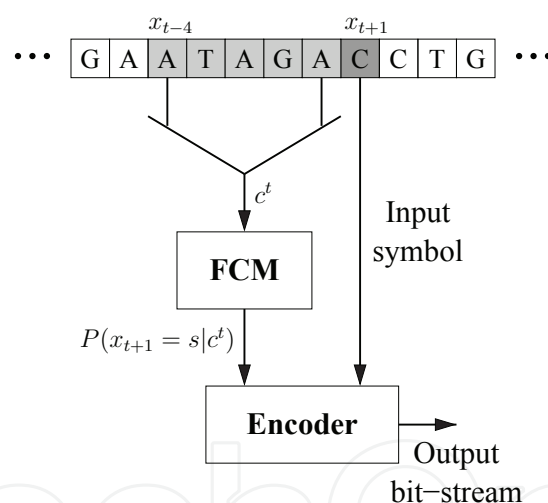

Fig. 1. Finite-context model: the probability of the next outcome, x_{t+1}, is conditioned by the M last outcomes. In this example, $M = 5$.

In practice, the probability that the next outcome, x_{t+1}, is s, where $s \in \mathcal{A} = \{A, C, G, T\}$, is obtained using the Lidstone estimator (Lidstone, 1920)

$$P(x_{t+1} = s|c^t) = \frac{n_s^t + \delta}{\sum\limits_{a \in \mathcal{A}} n_a^t + 4\delta}, \tag{1}$$

where n_s^t represents the number of times that, in the past, the information source generated symbol s having c^t as the conditioning context. The parameter δ controls how much probability is assigned to unseen (but possible) events, and plays a key role in the case of high-order

Context, c^t	n_A^t	n_C^t	n_G^t	n_T^t	$\sum\limits_{a \in \mathcal{A}} n_a^t$
$AAAAA$	23	41	3	12	79
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$ATAGA$	16	6	21	15	58
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$GTCTA$	19	30	10	4	63
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$TTTTT$	8	2	18	11	39

Table 1. Simple example illustrating how finite-context models are implemented. The rows of the table represent probability models at a given instant t. In this example, the particular model that is chosen for encoding a symbol depends on the last five encoded symbols (order-5 context).

models.[1] Note that Lidstone's estimator reduces to Laplace's estimator for $\delta = 1$ (Laplace, 1814) and to the frequently used Jeffreys (1946) / Krichevsky and Trofimov (1981) estimator when $\delta = 1/2$. In our work, we found out experimentally that the probability estimates calculated for the higher-order models lead to better compression results when smaller values of δ are used.

Note that, initially, when all counters are zero, the symbols have probability $1/4$, i.e., they are assumed equally probable. The counters are updated each time a symbol is encoded. Since the context template is causal, the decoder is able to reproduce the same probability estimates without needing additional information.

Table 1 shows an example of how a finite-context model is typically implemented. In this example, an order-5 finite-context model is presented (as that of Fig. 1). Each row represents a probability model that is used to encode a given symbol according to the last encoded symbols (five in this example). Therefore, if the last symbols were "$ATAGA$", i.e., $c^t = ATAGA$, then the model communicates the following probability estimates to the arithmetic encoder:

$$P(A|ATAGA) = (16 + \delta)/(58 + 4\delta),$$

$$P(C|ATAGA) = (6 + \delta)/(58 + 4\delta),$$

$$P(G|ATAGA) = (21 + \delta)/(58 + 4\delta)$$

and

$$P(T|ATAGA) = (15 + \delta)/(58 + 4\delta).$$

The block denoted "Encoder" in Fig. 1 is an arithmetic encoder. It is well known that practical arithmetic coding generates output bit-streams with average bitrates almost identical to the entropy of the model (Bell et al., 1990; Salomon, 2007; Sayood, 2006). The theoretical bitrate average (entropy) of the finite-context model after encoding N symbols is given by

$$H_N = -\frac{1}{N} \sum_{t=0}^{N-1} \log_2 P(x_{t+1} = s|c^t) \quad \text{bps,} \tag{2}$$

---

[1] When M is large, the number of conditioning states, 4^M, is high, which implies that statistics have to be estimated using only a few observations.

Context, c^t	n_A^t	n_C^t	n_G^t	n_T^t	$\sum_{a \in \mathcal{A}} n_a^t$
$AAAAA$	23	41	3	12	79
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$ATAGA$	16	$\boxed{7}$	21	15	59
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$GTCTA$	19	30	10	4	63
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$TTTTT$	8	2	18	11	39

Table 2. Table 1 updated after encoding symbol "C", according to context "$ATAGA$".

where "bps" stands for "bits per symbol". When dealing with DNA bases, the generic acronym "bps" is sometimes replaced with "bpb", which stands for "bits per base". Recall that the entropy of any sequence of four symbols is, at most, two bps, a value that is achieved when the symbols are independent and equally likely.

Referring to the example of Table 1, and supposing that the next symbol to encode is "C", it would require, theoretically, $-\log_2((6 + \delta)/(58 + 4\delta))$ bits to encode it. For $\delta = 1$, this is approximately 3.15 bits. Note that this is more than two bits because, in this example, "C" is the least probable symbol and, therefore, needs more bits to be encoded than the more probable ones. After encoding this symbol, the counters will be updated according to Table 2.

3.1 Inverted repeats

As previously mentioned, DNA sequences frequently contain sub-sequences that are reversed and complemented copies of some other sub-sequences. These sub-sequences are named "inverted repeats". As described in Section 2, this characteristic of DNA is used by most of the DNA compression methods that rely on the sliding window searching paradigm.

For exploring the inverted repeats of a DNA sequence, besides updating the corresponding counter after encoding a symbol, we also update another counter that we determine in the following way. Consider the example given in Fig. 1, where the context is the string "$ATAGA$" and the symbol to encode is "C". Reversing the string obtained by concatenating the context string and the symbol, i.e., "$ATAGAC$", we obtain the string "$CAGATA$". Complementing this string ($A \leftrightarrow T$, $C \leftrightarrow G$), we get "$GTCTAT$". Now we consider the prefix "$GTCTA$" as the context and the suffix "T" as the symbol that determines which counter should be updated. Therefore, according to this procedure, for taking into consideration the inverted repeats, after encoding symbol "C" of the example in Fig. 1, the counters should be updated according to Table 3.

3.2 Competing finite-context models

Because DNA data are non-stationary, alternating between regions of low and high entropy, using two models with different orders allows a better handling both of DNA regions that are best represented by low-order models and regions where higher-order models are advantageous. Although both models are continuously been updated, only the best one is used for

Context, c^t	n_A^t	n_C^t	n_G^t	n_T^t	$\sum\limits_{a \in \mathcal{A}} n_a^t$
$AAAAA$	23	41	3	12	79
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$ATAGA$	16	7	21	15	59
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$GTCTA$	19	30	10	5	64
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$TTTTT$	8	2	18	11	39

Table 3. Table 1 updated after encoding symbol "C" according to context "$ATAGA$" (see example of Fig. 1) and taking the inverted repeats property into account.

encoding a given region. To cope with this characteristic, we proposed a DNA lossless compression method that is based on two finite-context models of different orders that compete for encoding the data (see Fig. 2).

For convenience, the DNA sequence is partitioned into non-overlapping blocks of fixed size (we have used one hundred DNA bases), which are then encoded by one (the best one) of the two competing finite-context models. This requires only the addition of a single bit per data block to the bit-stream in order to inform the decoder of which of the two finite-context models was used. Each model collects statistical information from a context of depth $M_i, i = 1, 2, M_1 \neq M_2$. At time t, we represent the two conditioning outcomes by $c_1^t = x_{t-M_1+1}, \ldots, x_{t-1}, x_t$ and by $c_2^t = x_{t-M_2+1}, \ldots, x_{t-1}, x_t$.
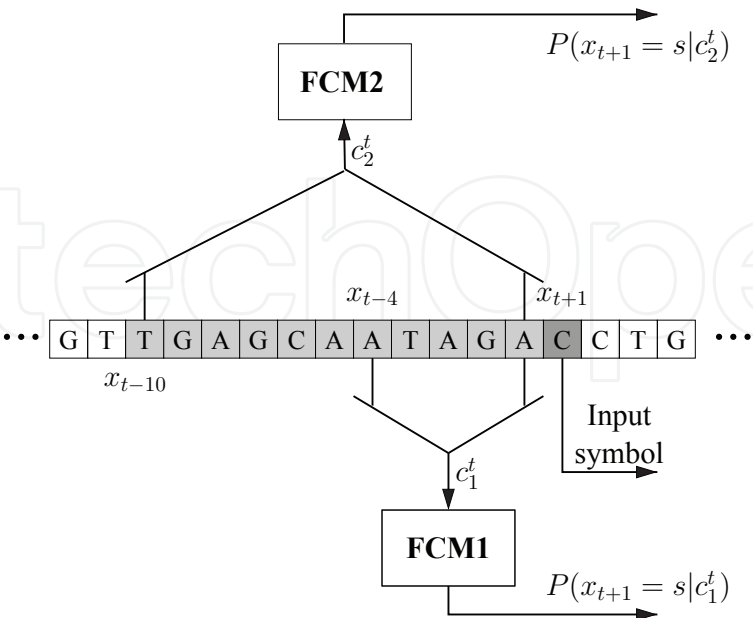

Fig. 2. Proposed model for estimating the probabilities: the probability of the next outcome, x_{t+1}, is conditioned by the M_1 or M_2 last outcomes, depending on the finite-context model chosen for encoding that particular DNA block. In this example, $M_1 = 5$ and $M_2 = 11$.

Using higher-order context models leads to a practical problem: the memory needed to represent all of the possible combinations of the symbols related to the context might be too large. In fact, as we mentioned, each DNA model of order-M implies 4^M different states of the Markov chain. Because each of these states needs to collect statistical data that is necessary to the encoding process, a large amount of memory might be required as the model order grows. For example, an order-16 model might imply a total of $4\,294\,967\,296$ different states.
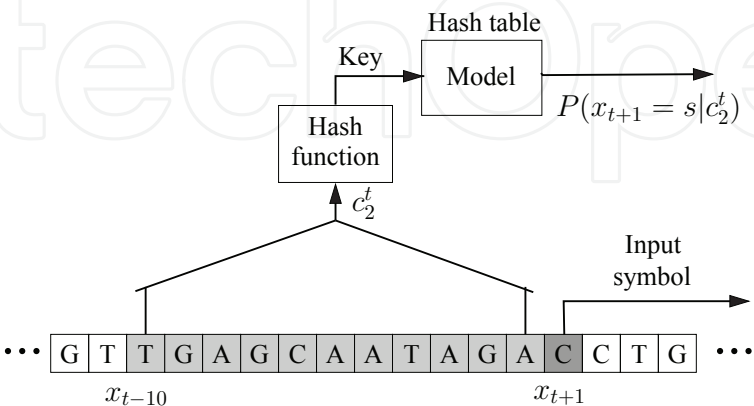
Fig. 3. The context model using hash tables. The hash table representation is shown in Fig. 4.

In order to overcome this problem, we implemented the higher-order context models using hash tables. With this solution, we only need to create counters if the context formed by the M last symbols appears at least once. In practice, for very high-order contexts, we are limited by the length of the sequence. In the current implementation we are able to use models of orders up to 32. However, as we will present later, the best value of M for the higher-order models is 16. This can be explained by the well known problem of context dilution. Moreover, for higher-order models, a large number of contexts occur only once and, therefore, the model cannot take advantage of them.

For each symbol, a key is generated according to the context formed by the previous symbols (see Fig. 3). For that key, the related linked-list if traversed and, if the node containing the context exists, its statistical information is used to encode the current symbol. If the context never appeared before, a new node is created and the symbol is encoded using an uniform probability distribution. A graphical representation of the hash table is presented in Fig. 4.
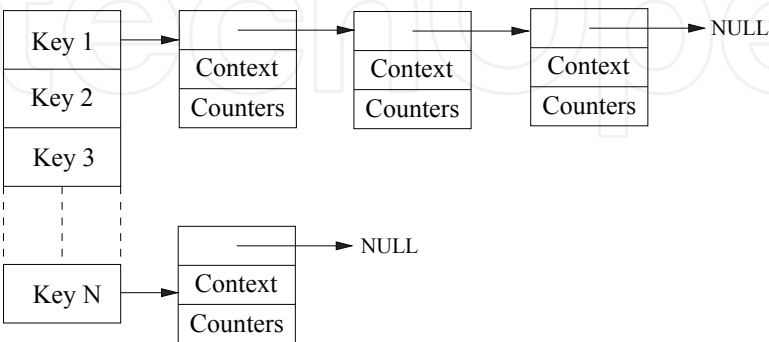

Fig. 4. Graphical representation of the hash table used to represent higher-order models. Each node stores the information of the context found (`Context`) and the counters associated to that context (`Counters`), four in the case of DNA sequences.

4. Experimental results

For the evaluation of the methods described in the previous section, we used the same DNA sequences used by Manzini and Rastero (2004), which are available from `www.mfn.unipmn.it/~manzini/dnacorpus`. This corpus contains sequences from four organisms: yeast (*Saccharomyces cerevisiae*, chromosomes 1, 4, 14 and the mitochondrial DNA), mouse (*Mus musculus*, chromosomes 7, 11, 19, x and y), arabidopsis (*Arabidopsis thaliana*, chromosomes 1, 3 and 4) and human (*Homo sapiens*, chromosomes 2, 13, 22, x and y).

First, we present results that show the effectiveness of the proposed inverted repeats updating mechanism for finite-context modeling. Next, we show the advantages of using multiple (in this case, two) competing finite-context models for compression.

4.1 Inverted repeats

Regarding the inverted repeats updating mechanism, each of the sequences was encoded using finite-context models with orders ranging from four to thirteen, with and without the inverted repeats updating mechanism. As in most of the other DNA encoding techniques, we also provided a fall back method that is used if the main method produces worse results. This is checked on a block by block basis, where each block is composed of one hundred DNA bases. As in the *DNA3* version of Manzini's encoder, we used an order-3 finite-context model as fall back method (Manzini and Rastero, 2004). Note that, in our case, both the main and fall back methods rely on finite-context models.

Table 4 presents the results of compressing the DNA sequences with the "normal" finite-context model (FCM) and with the model that takes into account the inverted repeats (FCM-IR). The bitrate and the order of the model that provided the best results are indicated. For comparison, we also included the results of the *DNA3* compressor of Manzini and Rastero (2004).

As can be seen from the results presented in Table 4, the bitrates obtained with the finite-context models using the updating mechanism for inverted repeats (FCM-IR) are always better than those obtained with the "normal" finite-context models (FCM). This confirms that the finite-context models can be modified according to the proposed scheme to exploit inverted repeats. Figure 5 shows how the finite-context models perform for various model orders, from order-4 to order-13, for the case of the "y-1" and "h-y" sequences.

4.2 Competing finite-context models

Each of the DNA sequences used by Manzini was encoded using two competing finite-context models with orders $M_1, M_2, 3 \leq M_1 \leq 8$ and $9 \leq M_2 \leq 18$. For each DNA sequence, the pair M_1, M_2 leading to the lowest bitrate was chosen. The inverted repeats updating mechanism was used, as well as $\delta = 1$ for the lower-order model and $\delta = 1/30$ for the higher-order model. All information needed for correct decoding is included in the bit-stream and, therefore, the compression results presented in Table 5 take into account that information. The columns of Table 5 labeled "M_1" and "M_2" represent the orders of the used models and the columns labeled with the percent sign show the percentage of use of each finite-context model.

As can be seen from the results presented in Table 5, the method using two competing finite-context models always provides better results than the *DNA3* compressor. This confirms that the finite-context models may be successfully used as the only coding method for DNA sequences. Although we do not include here a comprehensive study of the impact of the δ parameter in the performance of the method, nevertheless we show an example to illustrate its influence on the compression results of the finite-context models. For example, using $\delta = 1$

Name	Size	DNA3	FCM		FCM-IR	
		bpb	Order	bpb	Order	bpb
y-1	230 203	1.871	10	1.935	11	1.909
y-4	1 531 929	1.881	12	1.920	12	1.910
y-14	784 328	1.926	9	1.945	12	1.938
y-mit	85 779	1.523	6	1.494	7	1.479
Average		–	–	1.915	–	1.904
		1.882				
m-7	5 114 647	1.835	11	1.849	12	1.835
m-11	49 909 125	1.790	13	1.794	13	1.778
m-19	703 729	1.888	10	1.883	10	1.873
m-x	17 430 763	1.703	12	1.715	13	1.692
m-y	711 108	1.707	10	1.794	11	1.741
Average		1.772	–	1.780	–	1.762
at-1	29 830 437	1.844	13	1.887	13	1.878
at-3	23 465 336	1.843	13	1.884	13	1.873
at-4	17 550 033	1.851	13	1.887	13	1.878
Average		1.845	–	1.886	–	1.876
h-2	236 268 154	1.790	13	1.748	13	1.734
h-13	95 206 001	1.818	13	1.773	13	1.759
h-22	33 821 688	1.767	12	1.728	12	1.710
h-x	144 793 946	1.732	13	1.689	13	1.666
h-y	22 668 225	1.411	13	1.676	13	1.579
Average		1.762	–	1.732	–	1.712

Table 4. Compression values, in bits per base (bpb), for several DNA sequences. The "DNA3" column shows the results obtained by Manzini and Rastero (2004). Columns "FCM" and "FCM-IR" contain the results, respectively, of the "normal" finite-context models and of the finite-context models equipped with the inverted repeats updating mechanism. The order of the model that provided the best result is indicated under the columns labeled "Order".

for both models would lead to bitrates of 1.869, 1.865 and 1.872, respectively for the "at-1", "at-3" and "at-4" sequences, i.e., approximately 2% worse than when using $\delta = 1/30$ for the higher-order model.

Finally, it is interesting to note that the lower-order model is generally the one that is most frequently used along the sequence and also the one associated with the highest bitrates. In fact, the bitrates provided by the higher-order finite-context models suggest that these are chosen in regions where the entropy is low, whereas the lower-order models operate in the higher entropy regions.

5. Conclusion

Finite-context models have been used by most DNA compression algorithms as a secondary, fall back method. In this work, we have studied the potential of this statistical modeling paradigm as the main and only approach for DNA compression. Several aspects have been addressed, such as the inclusion of mechanisms for handling inverted repeats and the use

Average bitrate for sequence "y-1"
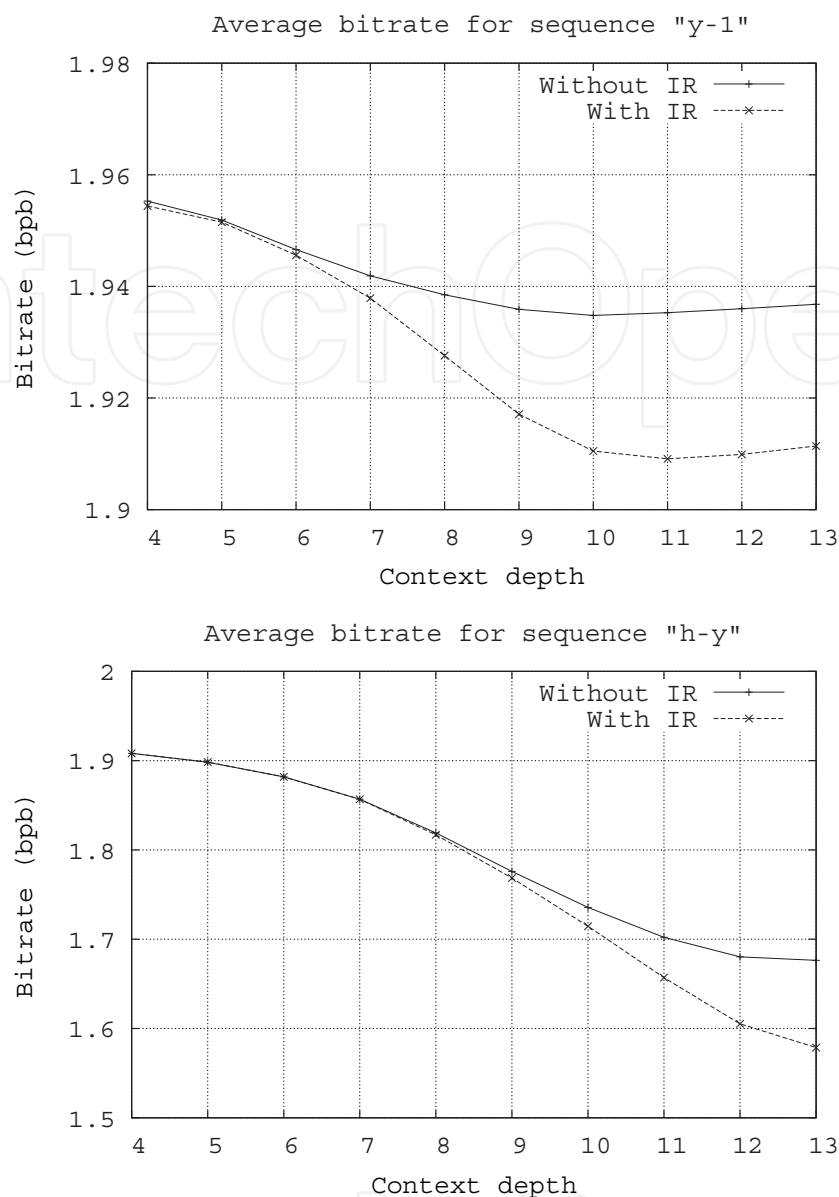

Average bitrate for sequence "h-y"

Fig. 5. Performance of the finite-context model as a function of the order of the model, with and without the updating mechanism for inverted repeats (IR), for sequences "y-1" and "h-y".

of multiple finite-context models that compete for encoding the data. This study allowed us to conclude that DNA models relying only on Markovian principles can provide significant results, although not as expressive as those provided by methods such as *MNL-1* or *XM*. Nevertheless, the experimental results show that the proposed approach can outperform methods of similar computational complexity, such as the *DNA3* coding method (Manzini and Rastero, 2004).

One of the key advantages of DNA compression based on finite-context models is that the encoders are fast and have $\mathcal{O}(n)$ time complexity. In fact, most of the computing time needed by previous DNA compressors is spent on the task of finding exact or approximate repeats of sub-sequences or of their inverted complements. No doubt, this approach has proved to give good returns in terms of compression gains, but normally at the cost of long compression

Name	Size	DNA3	FCM1			FCM2			FCM
		bps	M_1	%	bps	M_2	%	bps	bps
y-1	230 203	1.871	3	82	1.939	12	18	1.462	1.860
y-4	1 531 929	1.881	4	88	1.930	14	12	1.470	1.879
y-14	784 328	1.926	3	90	1.938	13	10	1.716	1.923
y-mit	85 779	1.523	5	83	1.533	9	17	1.178	1.484
Average	–	1.882	–	–	1.920	–	–	1.533	1.877
m-7	5 114 647	1.835	6	81	1.907	14	19	1.353	1.811
m-11	49 909 125	1.790	4	76	1.917	16	24	1.230	1.758
m-19	703 729	1.888	4	83	1.920	13	17	1.582	1.870
m-x	17 430 763	1.703	6	70	1.896	15	30	1.081	1.656
m-y	711 108	1.707	3	66	1.896	13	34	1.199	1.670
Average	–	1.772	–	–	1.911	–	–	1.206	1.738
at-1	29 830 437	1.844	6	82	1.898	16	18	1.475	1.831
at-3	23 465 336	1.843	6	80	1.901	16	20	1.495	1.826
at-4	17 550 033	1.851	6	80	1.897	15	20	1.560	1.838
Average		1.845	–	–	1.899	–	–	1.503	1.831
h-2	236 268 154	1.790	4	76	1.905	16	24	1.212	1.755
h-13	95 206 001	1.818	5	80	1.895	15	20	1.279	1.723
h-22	33 821 688	1.767	3	68	1.925	15	32	1.180	1.696
h-x	144 793 946	1.732	5	66	1.901	16	34	1.217	1.686
h-y	22 668 225	1.411	4	47	1.901	16	53	0.941	1.397
Average	–	1.762	–	–	1.903	–	–	1.212	1.711

Table 5. Compression values, in bits per symbol (bps), for several of DNA sequences. The "DNA3" column shows the results obtained by Manzini and Rastero (2004). Column "FCM" contains the results of the two combined finite-context models. The orders of the two models that provided the best result for each sequence are indicated under the columns labeled "M_1" and "M_2".

times. Although slow encoders could be tolerated for storage purposes (compression could be ran in batch mode), for interactive applications such as those involving the computation of complexity profiles (Dix et al., 2007) they are certainly not the most appropriate; faster methods, such as those examined in this chapter, could be particularly useful in those cases.

6. References

Behzadi, B. and F. Le Fessant (2005, June). DNA compression challenge revisited. In *Combinatorial Pattern Matching: Proc. of CPM-2005*, LNCS, Jeju Island, Korea. Springer-Verlag.

Bell, T. C., J. G. Cleary, and I. H. Witten (1990). *Text compression*. Prentice Hall.

Cao, M. D., T. I. Dix, L. Allison, and C. Mears (2007). A simple statistical algorithm for biological sequence compression. In *Proc. of the Data Compression Conf., DCC-2007*, Snowbird, Utah.

Chen, X., S. Kwong, and M. Li (1999). A compression algorithm for DNA sequences and its applications in genome comparison. In K. Asai, S. Miyano, and T. Takagi (Eds.), *Genome Informatics 1999: Proc. of the 10th Workshop*, Tokyo, Japan, pp. 51–61.

Chen, X., S. Kwong, and M. Li (2001). A compression algorithm for DNA sequences. *IEEE Engineering in Medicine and Biology Magazine 20*, 61–66.

Chen, X., M. Li, B. Ma, and J. Tromp (2002). DNACompress: fast and effective DNA sequence compression. *Bioinformatics 18*(12), 1696–1698.

Dennis, C. and C. Surridge (2000, December). *A. thaliana* genome. *Nature 408*, 791.

Dix, T. I., D. R. Powell, L. Allison, J. Bernal, S. Jaeger, and L. Stern (2007). Comparative analysis of long DNA sequences by per element information content using different contexts. *BMC Bioinformatics 8*(1471-2105-8-S2-S10).

Ferreira, P. J. S. G., A. J. R. Neves, V. Afreixo, and A. J. Pinho (2006, May). Exploring three-base periodicity for DNA compression and modeling. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, ICASSP-2006*, Volume 5, Toulouse, France, pp. 877–880.

Grumbach, S. and F. Tahi (1993). Compression of DNA sequences. In *Proc. of the Data Compression Conf., DCC-93*, Snowbird, Utah, pp. 340–350.

Grumbach, S. and F. Tahi (1994). A new challenge for compression algorithms: genetic sequences. *Information Processing & Management 30*(6), 875–886.

Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proc. of the Royal Society (London) A 186*, 453–461.

Korodi, G. and I. Tabus (2005, January). An efficient normalized maximum likelihood algorithm for DNA sequence compression. *ACM Trans. on Information Systems 23*(1), 3–34.

Korodi, G. and I. Tabus (2007). Normalized maximum likelihood model of order-1 for the compression of DNA sequences. In *Proc. of the Data Compression Conf., DCC-2007*, Snowbird, Utah.

Krichevsky, R. E. and V. K. Trofimov (1981, March). The performance of universal encoding. *IEEE Trans. on Information Theory 27*(2), 199–207.

Laplace, P. S. (1814). *Essai philosophique sur les probabilités (A philosophical essay on probabilities)*. New York: John Wiley & Sons. Translated from the sixth French edition by F. W. Truscott and F. L. Emory, 1902.

Lidstone, G. (1920). Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Trans. of the Faculty of Actuaries 8*, 182–192.

Manzini, G. and M. Rastero (2004). A simple and fast DNA compressor. *Software—Practice and Experience 34*, 1397–1411.

Matsumoto, T., K. Sadakane, and H. Imai (2000). Biological sequence compression algorithms. In A. K. Dunker, A. Konagaya, S. Miyano, and T. Takagi (Eds.), *Genome Informatics 2000: Proc. of the 11th Workshop*, Tokyo, Japan, pp. 43–52.

Pinho, A. J., A. J. R. Neves, V. Afreixo, C. A. C. Bastos, and P. J. S. G. Ferreira (2006, November). A three-state model for DNA protein-coding regions. *IEEE Trans. on Biomedical Engineering 53*(11), 2148–2155.

Pinho, A. J., A. J. R. Neves, C. A. C. Bastos, and P. J. S. G. Ferreira (2009, April). DNA coding using finite-context models and arithmetic coding. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, ICASSP-2009*, Taipei, Taiwan.

Pinho, A. J., A. J. R. Neves, and P. J. S. G. Ferreira (2008, August). Inverted-repeats-aware finite-context models for DNA coding. In *Proc. of the 16th European Signal Processing Conf., EUSIPCO-2008*, Lausanne, Switzerland.

Rivals, E., J.-P. Delahaye, M. Dauchet, and O. Delgrange (1995, November). A guaranteed compression scheme for repetitive DNA sequences. Technical Report IT–95–285, LIFL, Université des Sciences et Technologies de Lille.

Rivals, E., J.-P. Delahaye, M. Dauchet, and O. Delgrange (1996). A guaranteed compression scheme for repetitive DNA sequences. In *Proc. of the Data Compression Conf., DCC-96*, Snowbird, Utah, pp. 453.

Rowen, L., G. Mahairas, and L. Hood (1997, October). Sequencing the human genome. *Science 278*, 605–607.

Salomon, D. (2007). *Data compression - The complete reference* (4th ed.). Springer.

Sayood, K. (2006). *Introduction to data compression* (3rd ed.). Morgan Kaufmann.

Tabus, I., G. Korodi, and J. Rissanen (2003). DNA sequence compression using the normalized maximum likelihood model for discrete regression. In *Proc. of the Data Compression Conf., DCC-2003*, Snowbird, Utah, pp. 253–262.

Ziv, J. and A. Lempel (1977). A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory 23*, 337–343.
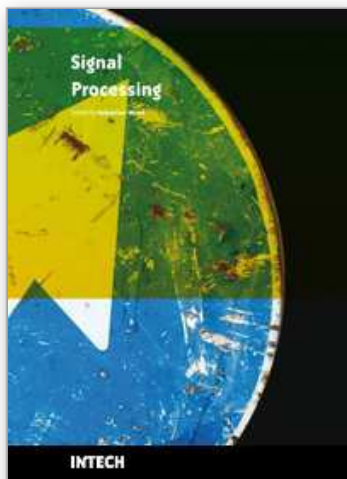
Signal Processing

Edited by Sebastian Miron

This book intends to provide highlights of the current research in signal processing area and to offer a snapshot of the recent advances in this field. This work is mainly destined to researchers in the signal processing related areas but it is also accessible to anyone with a scientific background desiring to have an up-to-date overview of this domain. The twenty-five chapters present methodological advances and recent applications of signal processing algorithms in various domains as telecommunications, array processing, biology, cryptography, image and speech processing. The methodologies illustrated in this book, such as sparse signal recovery, are hot topics in the signal processing community at this moment. The editor would like to thank all the authors for their excellent contributions in different areas of signal processing and hopes that this book will be of valuable help to the readers.

INTECH
open science | open minds