BIOINFORMATICS APPLICATIONS NOTE Vol. 18 no. 12 2002
Pages 1696–1698

DNACompress: fast and effective DNA sequence compression

Xin Chen[1,*], Ming Li[1], Bin Ma[2] and John Tromp[3]

[1]Bioinformatics Laboratory, Computer Science Department, University of California, Santa Barbara, CA 99106, USA, [2]Computer Science Department, University of Western Ontario, London N6A 5B8, Canada and [3]Bioinformatics Solutions Inc., 145 Columbia Street West, Waterloo, Ontario N2L 3L2, Canada

Received on March 14, 2002; revised on May 8, 2002; accepted on June 17, 2002

ABSTRACT

Summary: While achieving the best compression ratios for DNA sequences, our new *DNACompress* program significantly improves the running time of all previous DNA compression programs.

Availability: http://dna.cs.ucsb.edu/DNACompress

Contact: chxin@cs.ucsb.edu mli@cs.ucsb.edu

INTRODUCTION

Biological sequence compression is a useful tool to recover information from biological sequences. This was demonstrated for example in the construction of whole genome phylogenies in Li *et al.* (2001). Better compression implies better understanding. While the ultimate compression, i.e. the Kolmogorov complexity, is not computable (Li and Vitányi, 1997), it has been shown that even some heuristically reasonable compression helps to uncover enough similarities and differences among the mammalian mitochondrial genomes (Li *et al.*, 2001) to construct a correct phylogeny. Similar studies were performed for English documents, programs, languages (Bennett *et al.*, 2000; Chen *et al.*, 2002; Benedetto *et al.*, 2002).

Compression of DNA sequences is a very challenging task. This can be seen by the fact that no commercial file-compression program achieves any compression on benchmark DNA sequences we use in this paper. Several compression algorithms specialized for DNA sequences have been developed in the past ten years. Notably, these include *Biocompress-2*, *GenCompress* and *CTW+LZ* (Grumbach and Tahi, 1994; Chen *et al.*, 2001; Matsumoto *et al.*, 2000). Note that in this paper, we consider only pure compression software, and we do not consider and do not survey entropy estimation software and algorithms that do not produce final encoding of the input sequence.

Two essential pieces of structural information for compression are present in DNA sequences: complemented palindromes and approximate repeats. However, searching for all approximate repeats in a very long DNA sequence is not a trivial task. All such algorithms take a long time (essentially a quadratic time search or even more) in order to find approximate repeats that are optimal for compression. For example, the *CTW+LZ* algorithm (Matsumoto *et al.*, 2000) takes several hours to compress a sequence HEMCMVCG of 227 kb. Simultaneously achieving high speed and best compression ratio remains to be a challenging task. In this paper, we settle this problem—our *DNACompress* achieves a better compression ratio and runs significantly faster than any existing compression program for benchmark DNA sequences, simultaneously.

DNACompress employs the Lempel–Ziv compression scheme as *Biocompress-2* and *GenCompress* do. It consists of two phases: find all approximate repeats including complemented palindromes; and encode approximate repeat regions and non-repeat regions. We utilize a software tool *PatternHunter*, which we have developed for fast and sensitive homology search (Ma *et al.*, 2002), as our approximate repeat search engine. *DNACompress* is available via our web server. It runs on any platform with JVM.

METHODS

Consider a finite sequence *s* over the DNA alphabet {a, c, g, t}. An approximate repeat is a substring in *s* that can be transformed from another substring in *s* with not too many edit operations (substitution, insertion and deletion). The quantity of 'not too many' is dictated by the encoding procedure. We only encode those approximate repeats that provide profits on overall compression.

Searching for approximate repeats

Searching approximate repeats that are optimal for compression is very time-consuming. Greedy search method may miss longer approximate repeats to prevent the pro-

*To whom correspondence should be addressed.

Table 1. Comparison of compression ratios

Sequence	Size	Compress	GenCompress	CTW+LZ	DNACompress
CHMPXX	121024	2.09	1.673	1.6690	1.6716
CHNTXX	155844	2.19	1.6146	1.6129	1.6127
HEHCMVCG	229354	2.20	1.847	1.8414	1.8492
HUMDYSTROP	38770	2.23	1.9231	1.9175	1.9116
HUMGHCSA	66495	2.19	1.0969	1.0972	1.0272
HUMHBB	73323	2.20	1.8204	1.8082	1.7897
HUMHDABCD	58864	2.21	1.8192	1.8218	1.7951
HUMHPRTB	56737	2.23	1.8466	1.8433	1.8165
MPOMTCG	186608	2.20	1.9058	1.9000	1.8920
PANMTPACGA	100314	2.12	1.8624	1.8555	1.8556
VACCG	191737	2.14	1.7614	1.7616	1.7580
Average	—	2.18	1.7428	1.7389	1.7254

Table 2. Running time, on a Pentium III 700 MHz

		GenCompress		CTW+LZ		DNACompress	
Sequence	Size	bits	time	bits	time	bits	time
humdystrop	38 770	1.9231	7s	1.9175	8 minutes	1.9116	3s
hehcmvcg	229 354	1.8472	53s	1.8414	hours	1.8492	4s
H. influenza	1 830 029	1.8785	1591s	—	—	1.8766	22s
E. coli	4 630 230	1.9208	1595s	—	—	1.9172	58s

gram from obtaining higher compression ratio. Even using a lazy evaluation mechanism (Matsumoto *et al.*, 2000) the same problem still exists (see Figure 1a). Here we employ a new search engine *PatternHunter* (Ma *et al.*, 2002) in our *DNACompress* program. *PatternHunter* does homology search like Blastn, but with many innovations improving sensitivity, alignments, memory use, and speed. At the same sensitivity levels, *PatternHunter* is over two orders of magnitudes faster than Blastn (Ma *et al.*, 2002).

Since *PatternHunter* provides us all approximate repeats with highest score including complemented palindromes, the selection of which repeats are more optimal for sequence compression can be deferred at the end of *PatternHunter* homology search. *DNACompress* is as below.

1. Run *PatternHunter* and output all approximate repeats (and approximate reverse complements) into a list A in the order of descending scores;
2. Extract a repeat r with highest score from list A and add r into another repeat list B;
3. Process each repeat in A so that there's no overlap with the extracted repeat r;
4. Goto step 2 if the highest score of repeats in A is still higher than a pre-defined threshold; otherwise exit.

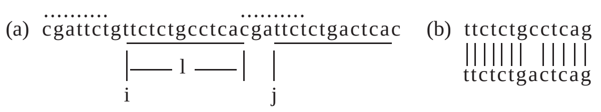

Fig. 1. (a) A repeat (l, i, j) may be missed by a lazy evaluation mechanism if its location offset to another repeat (dotted line) exceeds M (Matsumoto *et al.*, 2000). (b) A substitute operation $(e, o, b) = (\textit{substitute}, 7, \text{'}a\text{'})$ in the repeat (l, i, j).

All those extracted repeats in list B then parse a DNA sequence into a mixture of regions with little structure and repeat regions each of which can be replaced by a substring previously located.

Encoding repeats

An approximate repeat can be presented as two kinds of triples. One is (l, i, j), where l means the repeat length, i and j show the starting positions of two substrings in a repeat, respectively. The other kind is (e, o, b) representing an edit operation, where e indicates which kind of edit operation it is, o means its location offset in the repeat and b a base character that will be used by a substitute or insert edit operation, see Figure 1(b). In order to recover an approximate repeat correctly the following information must be encoded in the output data stream:

1. One bit to show which kind of repeat it is, forward repeat or complemented palindrome.
2. A triple (l, i, j). It is used to copy a previous substring of length l starting at i to the current position j;
3. The total number of edit operations contained in this approximate repeat;
4. All triples (e, o, b). They are used to edit the copied substring. Instead of encoding each edit operation separately, a consecutive region of the same edit operation (or say a block edit operation) can alternatively employ a more efficient encoding method.

DNACompress checks each repeat to see whether it saves bits to encode. If not, it will be discarded. At the end, all the remaining regions other than repeats are concatenated together and then sent as input to a two-order arithmetic coder.

EXPERIMENTS

DNACompress uses the following parameters for *PatternHunter* input: reward for a base match is 1, penalty for a base mismatch −4, open gap penalty −4, and gap extension penalty −3. These parameters were set and optimized based on the number of bits needed to encode a mismatching base as well as to encode a match. We compared the results of *DNACompress* to the best DNA compression algorithms *GenCompress* and *CTW+LZ*. Table 1 shows the compression ratios (the number of bits per base) of these algorithms on standard benchmark sequences. *DNACompress* achieves the best average compression ratio. Table 2 compares running times of these algorithms. Instead of about 8 minutes for sequence HUMDYSTROP and some hours for HEHCMVCG by *CTW+LZ*, *DNACompress* takes only 3 and 4 seconds, respectively.

ACKNOWLEDGEMENTS

XC and ML was supported by NSF ITR grant 0085801. JT was supported by BSI.

REFERENCES

Bennett,C., Li,M. and Ma,B. (2000) Linking chain letters. *Scientific American*, to appear

Benedetto,D., Caglioti,E. and Loreto,V. (2002) Language trees and zipping. *Phys. Rev. Lett.*, **88**, 048702.

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

Chen,X., Li,M., Mckinnon,B. and Seker,A. (2002) A theory of uncheatable program plagiarism detection and its practical implementation. SID website at http://dna.cs.ucsb.edu/SID/.

Grumbach,S. and Tahi,F. (1994) A new challenge for compression algorithms: genetic sequences. *J. Inform. Process. Management*, **30**, 875–866.

Li,M., Badger,J., Chen,X., Kwong,S., Kearney,P. and Zhang,H. (2001) An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, **17**, 149–154.

Li,M. and Vitányi,P. (1997) *An Introduction to Kolmogorov Complexity and its Applications*, 2nd edn, Springer, New York.

Ma,B., Tromp,J. and Li,M. (2002) PatternHunter—faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.

Matsumoto,T., Sadakane,K. and Imai,H. (2000) *Biological sequence compression algorithms*, Genome Informatics Workshop, Universal Academy Press, pp. 43–52.