An Improved 64-bit Cyclic Redundancy Check for Protein Sequences

David T. Jones

Department of Computer Science
Bioinformatics Unit
University College London
Gower Street
London
WC1E 6BT

ABSTRACT

Summary
The CRC-64 (64-bit Cyclic Redundancy Check) algorithm employed in the SWISS-PROT and TrEMBL data banks is shown to have a flaw which greatly increases the likelihood of duplicate key values being generated for pairs of sequences differing in only 2, 3 or 4 positions. A new CRC function has been implemented which behaves with better statistical properties when applied to a large set of similar but distinct protein sequences.

Availability

C source code for the improved CRC function can be downloaded from http://bioinf.cs.ucl.ac.uk/downloads/crc64

Contact

d.jones@cs.ucl.ac.uk

With the ever increasing growth of sequence data banks, and the lack of any unified scheme for allocating reliable identifiers to sequences, some means is needed for tracking individual sequences and also to detect undocumented changes to sequences resulting from either deliberate corrections or from human or computer error. Early sequence analysis tools made use of simple checksums to ensure the validity of sequence entries, but of course such a simple error detection scheme is not only easy to fool, but also does not provide a useful probably-unique identifier for a given sequence. More recently, the SWISS-PROT and TrEMBL data banks (Bairoch & Apweiler, 1996) have included a 64-bit Cyclic Redundancy Check (CRC) value alongside each sequence to act as both an error detecting device and also as a means of tagging the sequence data itself. The use of CRCs to provide a probably-unique hash key value is commonplace in computing applications, though many other hashing functions have been proposed, particularly for cryptographic applications. Where keys are destined to be used in a single database, it is possible to formulate an algorithm which guarantees uniqueness, though at the expense of large computational cost. However, absolute guarantees of uniqueness are not possible when the key is smaller than the hashed data element and where there is no available record of previously generated keys.

There are of course many different ways one might assess the "quality" of a hashing function, but in this particular case it is stipulated as a minimum requirement, that a good hashing function should not produce an identical key for a sequence and any closely related but distinct sequences more often than one would expect for keys generated purely at random. This is a reasonable requirement in that one of the main aims of placing a hashing or checksum value in a sequence data bank is to track minor changes in sequences (deliberately made or otherwise).

A Cyclic Redundancy Check or CRC value of length M bits is designed with error detection in serial communications in mind (Tanenbaum, 1996). The mathematical properties of a CRC with M bits are such that errors in M or fewer consecutive bits are guaranteed to be detected. This is a useful property for serial communications, as noise in a communication system frequently occurs in bursts e.g. from electrical interference. Two issues come to mind here for biological sequences. Firstly, when represented as an ASCII string, biological sequences cannot be considered to be a random bit string. Secondly, the requirement for guaranteed detection of errors in M consecutive bits may or may not be relevant for biological sequences.

An M-bit CRC function can be described as division of polynomials of degree M over the integers modulo 2. A binary message such as 10101 can be represented as a polynomial with coefficients 0 or 1 as follows:

$$x^4 + x^2 + 1$$

CRC functions make use of a generator polynomial which is primitive i.e. cannot be factorised into simpler polynomials (modulo 2). For example, a common 16-bit CRC polynomial is as follows:

$$x^{16} + x^{12} + x^5 + 1$$

For an *M*-bit CRC, the binary message is left shifted *M* times and then divided by the generator polynomial (without carries) giving a remainder, which is the final CRC value.

To guarantee the consecutive bit property of a CRC, the polynomial used must be primitive. Furthermore, any primitive polynomial of similar order is guaranteed to produce a CRC with this property. So for the problem of detecting consecutive errors in serial communications, all primitive polynomials are born equal. However, the choice of primitive polynomial can affect other properties of the CRC function, and if the function is destined to be used for other purposes, such as generating hash keys, the choice of generator polynomial can be critical, depending on any patterns evident in the data to be hashed.

In the current SWISS-PROT and TrEMBL data banks, the following CRC polynomial is implemented:

$$x^{64} + x^4 + x^3 + x + 1$$

This polynomial is actually the smallest primitive polynomial of order 64, and is not necessarily a good choice for hashing. Applying this function to the current NCBI NR data bank (September 24th 2002 release), comprising 1,169,485 sequences, reveals two pairs of different sequences with identical SP-TrEMBL CRC-64 values. Given that for a uniformly distributed random 64-bit key, the probability of finding a duplicate in a data bank of this size should be ~10^{-8}, finding two unrelated duplicates is either a case of extreme bad luck or an indication of bias in the key generator.

As a more stringent test of hashing similar sequences, 1000 simulated mutant sequences were generated for every entry in SWISS-PROT Release 40. Six sequence sets were created in all, covering the range from 1 to 6 mutations per duplicated sequence. Duplicate keys were only sought within each set of mutants i.e. to check if any of the artificially generated mutants produced identical hash keys to their parent sequence. Table 1 shows the results of applying the standard SP-TrEMBL CRC function to these artificial sequence variants compared to this much denser generating polynomial:

$$x^{64} + x^{63} + x^{61} + x^{59} + x^{58} + x^{56} + x^{55} + x^{52} + x^{49} + x^{48} + x^{47} + x^{46} + x^{44} + x^{41} + x^{37} + x^{36} + x^{34} + x^{32} + x^{31} + x^{28} + x^{26} + x^{23} + x^{22} + x^{19} + x^{16} + x^{13} + x^{12} + x^{10} + x^8 + x^7 + x^5 + x^3 + 1$$

Table 1 shows clearly that the current CRC algorithm employed in SWISS-PROT and TrEMBL behaves poorly for pairs of sequences differing in just 2, 3 or 4 positions. Looking at the generating polynomial being used ($x^{64} + x^4 + x^3 + x + 1$) it is plain that it is extremely sparse, with bits 6-64 all set to zero. This evidently creates a subtle "blind spot" for certain characters at positions i and i+8. Both of the sequence pairs in the NR data bank which hash to identical keys differ in two locations with a sequence separation of 8, and so this flaw is already apparent with real sequences. There is no single pattern evident for the 8 collisions involving 3 mutations, but these are not as might be expected limited to positions i, i+8 and i+16. However, as shown in Table 1, none of the sequence pairs with multiple mutations produced key collisions with the new polynomial as would be expected.

References

Bairoch A., Apweiler R. (1996) The SWISS-PROT protein sequence data bank and its new supplement TrEMBL. Nucl. Acids Res.24:21-25.

Tanenbaum, A.S. *Computer Networks*. Chapter 4. Prentice Hall, 1996.

Table 1

Numbers of hash key collisions found for the proposed CRC function compared to the current SP-TrEMBL CRC function for 111,052,000 sequence pairs with 1-6 differing amino acids. Note that the expected number of matches in each case for a 64-bit key is ~ 6×10^{-12}.

	Number of Simulated Mutations					
	1	2	3	4	5	6
CRC-64 (SP-TrEMBL)	0	967	8	1	0	0
CRC-64 (Improved)	0	0	0	0	0	0