Fingerprint for sequence similarity search

The recent literature suggests an increasing interest in the use of alignment-free methods for the comparison of sequences; these methods can be faster avoiding the computational cost of find the best alignment before the calculation of the actual percentage of identity. At the moment several approaches are available in the literature. Most of them make use of an approach based on the use of short words, i.e. contiguous subsets of the full sequences including a small number of aminoacids or nucleotides. The short words are a used also as pre-filter for the psi-blast program that uses the so-called seeds having fixed lengths to eliminate low similarity sequences.

Here I present a method to implement the use of binary fingerprint to protein sequences in a way similar to that described for the chemical fingerprints used for small molecules comparison.

The number of possible short words increase exponentially with its length (e.g.: the number of 5-aminoacids long short words is 20^5, i.e. 3,200,000). For this reason, create a hash table accounting for all the possibilities is unfeasible. The use of fingerprints can eliminate the need of using predefined patterns. A fingerprint is a Boolean array or a bitmap, however, the meaning of each bit is not associated to any particular short word. The algorithm enumerates all the possible short-words in a sequence, then each one is used to seed a pseudo-random number generator to obtain a set of bits (2 to 10, a configurable parameter) to be turned on. There is the possibility that a single bit (or more) set by a short-word is shared with those the ones set by other short-words.

Since the fingerprints are binary array they can be easily compared using simple Boolean operators. In particular, the similarity between two fingerprints can be estimated by using the Tanimoto coefficients.

$$T = \frac{\sum_i (X_i \land Y_i)}{\sum_i (X_i \lor Y_i)}$$

The Tanimoto similarity coefficient range from 0 to 1, two identical sequences will have a value of 1.

The binary fingerprint can be easily stored on the disk and used in further studies.

Implementation

The code used to generate a fingerprints database and to screen a sequence against it has been written in **Python** and **C++**, the first programming language has been used mainly for the I/O operations and for the command line interface, while the second for the most computational demanding routines.

The generation of a fingerprint database requires the use of a set of sequences in the FASTA format but, in the future, other formats can be added. The program enumerates all the short-words from a length of 2 aminoacids to a maximum of 4, the frequency of each word is used to calculate the number of bits to be turned on using the formula (1):

$$numOfBits = \begin{cases} 2 \; for \; frequency \leq 2 \\ \quad frequency \\ 5 \; for \; frequency \geq 5 \end{cases}$$

The use of this formula accounts for the highest relative frequency of very shorts words.

Each short-word is hashed using the MD5 message-digest algorithm resulting in 4 32bit integers that can be used to seed the pseudo-random number generator. The pseudo-random number generator is the XORSHIFT128, having a period of 128bit that can be sufficient to avoid collisions between different short-words. During the calculation the fingerprint is stored into a bitset object and is then converted and returned as an array of 32bits integers; for example a bit set with a length of 4096 bit is converted to and array of 512 integers. The fingerprints are stored into an HDF5 file as a dataset with shape [N,M], with N equal to the number of the sequences and M equal to the number of integers representing the fingerprint. In the HDF5 file will be stored also two vectors one containing the number of on bits for each fingerprint and one containing the IDs of each sequence.

The comparison of a reference sequence to the pre-calculated database starts with the calculation of the fingerprint for the reference that is then compared with the fingerprints stores in the HDF5 database. Two measures have been adopted to reduce the computation time. The first one was the use of the built-in popcnt function to count the number of on bit in each 32bit integer, this function tells the compiler to use a specific hardware instruction to perform the operation. The second measure is the use of a slight modified formula for the calculation of the Tanimoto coefficient; indicating with “a” the number of on bits on the first fingerprint, with “b” the number of on bits in the second and with “c” the number of on bits in the intersection of the two of them, the Tanimoto coefficient became:

$$T = \frac{c}{a + b - c}$$

Using this formula “a” and “b” can be calculated at the start of stored in the HDF5 database while the “c” value is the only one to be calculate at each iteration of the search.

Results

This approach requires further studies and assessment but from early test it appears to be up to 3 times faster than the PSI-BLAST program and return similarity values that correlates with the percentage of identity that can be calculate with both clustal Ω and PSI-BLAST.

Availability

The source code is available for download at https://github.com/giorgiomaccari/seqFP.