PDF link

Code link

Kmer counting algorithm using perfect hash function: C++ pseudo-code integration into R using Rcpp API

MACHERKI M. E.

Laboratory of Analysis, Treatment and Valorization of Pollutants of the Environment and Products (LATVPEP), Faculty of Pharmacy, University of Monastir, Monastir 5000, Tunisia

Abstract

Counting kmers (substrings of length k in DNA sequence data) is an essential component of many methods in bioinformatics, including data preprocessing for de novo assembly, repeat detection, and sequencing coverage estimation. We proposed a simple algorithm to calculate the kmer count using perfect hash table implemented in C++ and using Rcpp API to be able exported into R.

Keywords: kmer, counting, DNA, hash function, R, RCPP

Introduction

The goal of kmer counting is to find the number of occurrences for each fixed-length word of length k in a DNA data set [1]. Efficient kmer counting plays an important role in many bioinformatics approaches especially the alignment-free methods. Usually, a hash table is used to find kmers with low complexity after sequence transformation into digital signal [2]. There is many approach using parallel counting, disc memory and unique counting elimination was already optimized for big data and large k [3].We proposed in this paper a simple injective hash function, given directly the index of each sequence into the table of all possible ranged kmers lexicographically.

Methods

A DNA sequence S can be represented as an index table of character of size n (1).

$$S=\{S_0,S_1,..S_{n-1}\},n,\forall x\in S,x\in\{A,C,G,T\} \quad (1)$$

Let's X a subsequence of size k and V a function such:

$$V(S_i)=\begin{cases}V(S="A")=0\\V(S="C")=1\\V(S="G")=2\\V(S="T")=3\end{cases} \quad (2)$$

The index in the table lexicographically can be found used the base 10 computing starting from a base β (=4 in the example (3)) annotation of the subsequence.

$$\begin{aligned}&k=11\ \beta=4\\&ATGGGGACCTT\\&03222201133 \quad : \text{number in base 4} \qquad (3)\\&X=\sum_{1}^{k}V_{si}\times\beta^{i-1}\\&X=960607 \quad : \text{number in base 10}\end{aligned}$$

The algorithm (4) is iterative, requiring a mutable value X and O (N) time complexity. Rcpp implementation is available in supplement data. We used three functions:

- V(x): calculate the digital value of each char in the DNA sequence based in a switch structure.
- X0: calculate the first index value or X0
- Kmer4: return a vector P of lexicographically ordered kmers.

Kmer4 had three arguments:

- A string 'A' given the DNA sequence
- The size of the kmer k
- The length of the DNA sequence

$$
\begin{gathered}
S=\{\ S_0,S_1,..S_{n-1}\ \},n,\forall x\in S,x\in\{A,C,G,T\ \} \\
P=\{P_0,P_1,P_2,..P_{\beta^k-1}\}\beta^k,\forall x\in \boldsymbol{P},\boldsymbol{x}=0 \\
X_0=S(n-k,n-1)=\sum_{n-k}^{n-1}V_{si}\times\beta^{i-1} \\
p[X_0]=p[X_0]+1 \\
FOR(i=n-k-1,\text{while}:i>0;i=i-1) \qquad (4) \\
x=V(S_i) \\
X_i=\beta^{k-1}x+\frac{X_{i-1}-X_{i-1}\bmod\beta}{\beta} \\
P[X_i]=P[X_i]+1
\end{gathered}
$$

Result

The kmer4 function cannot support masked DNA sequence (generally for eukaryote genome). Therefore, it is possible to use base five to overcome none identified values in this type of sequences. With the same context, the same approach is applicable for RNA and protein sequence as kmers, unique identifier with the hash table itself or as basic step in translation. Although its simplicity, the benchmark kmer4 function is competitive to the *oligonucleotideFrequency* function in the *Biostrings* package [4].This result is improved by the fast memory accession using the hash function.

References

1. Deorowicz, S., A. Debudaj-Grabysz, and S. Grabowski, *Disk-based k-mer counting on a PC.* BMC bioinformatics, 2013. **14**(1): p. 1.
2. Melsted, P. and J.K. Pritchard, *Efficient counting of k-mers in DNA sequences using a bloom filter.* BMC bioinformatics, 2011. **12**(1): p. 1.
3. Zhang, Q., et al., *These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure.* PloS one, 2014. **9**(7): p. e101271.
4. Pages, H., et al., *String objects representing biological sequences, and matching algorithms.* R package version, 2009. **2**(2).