CBMF 4761 Computational Genomics
Final Project Report

Sequence Analysis

Tandem Repeats Browser

Corresponding to Chuchu Ding[1,*], Dejia Tang[1,2,*]

[1]Department of Computer Science, Columbia University, the Fu Foundation School of Engineering and Applied Science, [2]Department of Computer Science, Smith College.

*To whom correspondence should be addressed.

Received on 05/06/2017; revised on 05/06/2017; accepted on 05/06/2017

Abstract

Summary: The Tandem Repeats Browser is a program designed to search tandem repeats occurring in DNA sequences such as human genome with user-defined parameters including mismatch tolerance and pattern length's lower boundary.

Availability: The Tandem Repeats Browser is freely available to all users interactively when the server is up running on Amazon Cloud Service. For testing locally, application code is available for cloning on Github public repository. Please refer to the supplementary information section for more details.

Contact: c.ding@columbia.edu; dt2559@columbia.edu

Supplementary information: Supplementary data and code are available at Github: (https://github.com/chuuu/tandem-repeats-browser.git).

1 Introduction

Tandem repeats are patterns of one or more nucleotides that repeat in DNA, when the repetitions are adjacent to each other. A pattern that repeats for a fraction of a time is also counted in as part of the current repeat, as long as the this pattern is repeated directly adjacently for more than two times already.

Tandem repeats occurring in the genome are important markers for disease diagnosis, mapping studies, and human identity testing. For instance, as mentioned by Benson [1], diseases such as fragile-X mental retardation, Huntington's disease, myotonic dystrophy, spinal and bulbar muscular atrophy and Friedreich's ataxia, are shown to be the result of a dramatic increase in the number of occurrence of a trinucleotide pattern. Another application is to use part of a tandem repeats as a guiding sequence when using a multiplexed labeling sequence to identify various genomic loci. For example, Ma et al. have shown a way to use multiplexed labeling of genomic loci with dcas9 and engineered sgrnas using CRISPRainbow [2]. Finally, tandem repeats play a variety of regulatory and evolutionary roles, as is shown by many previous studies.

The analytic and clinical importance of tandem repeats call for efficient algorithms to find them in huge gene sequences accurately. Most if not all tandem repeats finders freely available to public users depend on dynamic programming to handle all cases of repeats, which gives accurate results but relies on $O(L^2)$ computation time. This can be unacceptable when the input sequence has the length of a whole human genome or even a full chromosome, and is certainly not necessary when the user only aims to find some short repeats to support clinical or analytical use (as is in the case for trinucleotide pattern analysis).

2 Methods

In order to improve performance over $O(L^2)$ for the human genome sequence, we separate all tandem repeats to be found into two categories: (1) short repeats and (2) long repeats, depending on the length of the repeat pattern, and handle them using separate methods. The boundary for this separation is defined to be $\log_4 w$, with w = the window size. Here we rely on the assumption that long repeats defined this way are exponentially unlikely to occur than short repeats, and therefore can be handled with more computationally expensive method, to be described in more details in this section. If the input data fails to meet this assumption, performance might suffer to an extent, but the repeats found would still be accurate for analysis.

2.1 Sliding Window Approach

We separate the whole input sequence into separate windows of size w, find repeats in each window, and stitch together repeats that are cut apart by the window boundaries.

Window size w is an integer parameter defined by user input and the windows are overlapped by ¼ w to prevent missing the repeats that cross windows. Even though the searching process still has quadratic time complexity within each window and the overlapped parts add some repetitive work, the sliding window method still improve time complexity. If we apply a quadratic dynamic programing algorithm to the whole genome sequence with length L to find all tandem repeats, the time complexity would be $O(L^2)$. Given L is extremely large, the algorithm becomes borderline infeasible. By performing search within each window, the time complexity becomes $kw^2 = (kw)*w = L*w$ where k = number of windows. Since w is much smaller than L, the overall performance is improved.

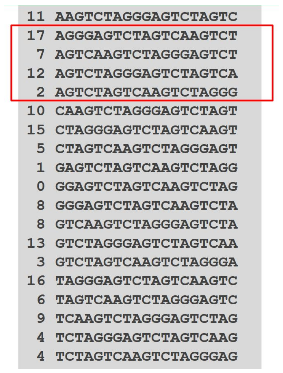
After repeats are found within each window, they need to be stitched together and combine the repeats that are crossing windows. We do this by comparing repeats within a window with the ones from the previous window. The following quarteira need to be met for a pair of repeats to be stitched together: the two repeats have the same pattern length and they overlap for at least one pattern. Using this approach, patterns with length greater than ¼ w can still be missed, yet this situation is very rare give w is usually very large (for instance, 10,000.)

2.2 Short repeats

Within each window, short repeats with length <= $\log_4 w$, w = the window size, are simply checked with enumeration. We enumerate all possible combinations of the given alphabet (default: ATCG) of length <= $\log_4 w$, and matches each window to it using a linear scan.

2.3 Long repeats

For long repeats with length > $\log_4 w$, w = the window size, a Burrows Wheeler transform matrix [3] is used as the first step to locate a potential repeat.



```
11 AAGTCTAGGGAGTCTAGTC
17 AGGGAGTCTAGTCAAGTCT
 7 AGTCAAGTCTAGGGAGTCT
12 AGTCTAGGGAGTCTAGTCA
 2 AGTCTAGTCAAGTCTAGGG
10 CAAGTCTAGGGAGTCTAGT
15 CTAGGGAGTCTAGTCAAGT
 5 CTAGTCAAGTCTAGGGAGT
 1 GAGTCTAGTCAAGTCTAGG
 0 GGAGTCTAGTCAAGTCTAG
 8 GGGAGTCTAGTCAAGTCTA
 8 GTCAAGTCTAGGGAGTCTA
13 GTCTAGGGAGTCTAGTCAA
 3 GTCTAGTCAAGTCTAGGGA
16 TAGGGAGTCTAGTCAAGTC
 6 TAGTCAAGTCTAGGGAGTC
 9 TCAAGTCTAGGGAGTCTAG
 4 TCTAGGGAGTCTAGTCAAG
 4 TCTAGTCAAGTCTAGGGAG
```

Fig. 1. Sorted BWT matrix for string GG AGTCT AGTCA AGTCT AG.

Fig. 1 shows an example of sorted BWT matrix with input "GG AGTCT AGTCA AGTCT AG". After sorting the BWT matrix, tandem repeat patterns are inclined to be adjacent to each other, especially for longer repeats. Mismatches/insertion/deletion may cause the original patterns' sequence not being reserved in the sorted BWT matrix. As we are using the indices to decide the portions to do string comparison, changing the patterns' sequence may lead to incomplete repeats. For the example shown in Fig. 1, the matrix can only help us to locate the repeat patterns (7)AGTCA (12)AGTCT. To solve this issue, we extend forwards and backwards along the original sequence after locating potential repeats with at least two repeating patterns to get the complete repeats including partial ones in the end.

Since the patterns can be changing along the whole repeat, we keep a dynamic consensus as a reference pattern for future pattern search. The consensus is updated every time the repeat extends to represent the repeat pattern. This can address the effect of changing pattern in a degree, but is still sub-optimal in certain situations. For instance, the pattern ATCG ATCA ATCG get rejected for mismatch tolerance 10%. This can be improved by applying more complicated scoring mechanism to compare a pattern considering not only a single reference pattern but the whole repeat. This is a potential area for future work.

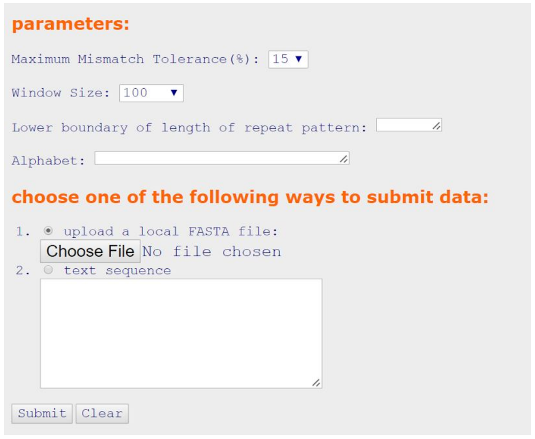
2.4 Miscellaneous

Our program avoids redundantly including cyclic shifts of found repeats, by doing a linear check of all repeats found in the current window, upon insertion of a new repeat just detected. Since this linear scan only involves with the found items in the sublist in the result list that corresponds to the current window, it has negligible effects over the performance.

The program determines whether two strings match or not by using the pairwise alignment algorithm in the pairwise2 module in package Biopython. Although this part of the algorithm uses dynamic programming (theoretically with a performance cost of $O(l^2)$), since we're only comparing two repeat patterns, l is very small here, making the DP here no longer hinder the overall performance. In our testing of the program, we observe that the number of repeats drops significantly even only after pattern length >= 5 nucleotides, which agrees with this assumption.

3 Result

The algorithm is implemented with Python and can be run in two versions: as a command line tool or as a web server. For more specific information about the code and data please refer to the supplementary information part of the header page.



parameters:

Maximum Mismatch Tolerance(%): 15

Window Size: 100

Lower boundary of length of repeat pattern:

Alphabet:

choose one of the following ways to submit data:

1. [✓ upload a local FASTA file:] Choose File No file chosen
2. text sequence

Fig. 2. Web server version GUI.

Both version can take sequence input in either FASTA file format or a text string. If the program is run from command line, output is in the format of a CSV file saved to the current directory. The CSV file has each row representing a repeat found and three columns representing three indices of it: the starting index of the repeat, the ending index of the first pattern, and the ending index of the whole repeat. Parameters including mismatch tolerance, window size, alphabet, text sequence, input filename, output filename and lower boundary of pattern can be set using command line flags if needed.

The web server is implemented intending to provide a graphical user interface (Fig. 2) for users to browse the tandem repeats found in a sequence. The user can still upload a FASTA file or copy a text sequence as input and set parameters including mismatch tolerance, window size and alphabet. The three indices mentioned above for each tandem repeat are printed on the result page. The user can check the repeats in string format by clicking any index or searching using the result number.

Acknowledgements

This program is designed under the guidance of Professor Itshack G Pe'er of Columbia University.

References

[1] Benson, Gary. "Tandem repeats finder: a program to analyze DNA sequences." *Nucleic acids research* 27.2 (1999): 573-580.
[2] Ma H, Tu L C, Naseri A, et al. Multiplexed labeling of genomic loci with dCas9 and engineered sgRNAs using CRISPRainbow[J]. Nature biotechnology, 2016.
[3] Burrows, Michael, and David J. Wheeler. "A block-sorting lossless data compression algorithm." (1994).