LtrDetector: De-novo Discovery of Complete LTR Retrotransposons Using K-mer Counts and Structural Features

Hani Z Girgis*[1]

[1]Tandy School of Computer Science, University of Tulsa, 800 South Tucker Drive, Tulsa, OK 74104

Email: Hani Z Girgis*- hani-girgis@utulsa.edu;

*Corresponding author

Abstract

Text for the Abstract.

1 Introduction

Combines de-novo and knowledge based methods. Combines structural features and the repatitive nature of the LTR Retrotransposones.

I will survey the available tools to detect LTR transposon. These tools detect LTR according to the structure: LTRharvest (Ellinghaus et al., 2008), LTR_par (Kalyanaraman and Aluru, 2006), LTR_STRUC (McCarthy, E. M. and McDonald, J. F., 2003), Find_LTR (Rho et al., 2007), LTR_FINDER (Xu and Wang, 2007)

2 Methods

Overview: The goal of this study is use computational methods to accurately and efficiently identify LTR transposon candidates in whole genomes. My method consists of the following four steps:

- Indexing the closest copy of a k-mer in the input sequence,
- Finding candidates based on the structure of the long terminal repeats (LTR),
- Filtering candidates based on the count of their k-mer's in the genome, and
- Delineating candidates based on the target site duplication (TSD).

The first and the third steps utilize a special hash table, which is a data structure, specifically designed for DNA sequences. To describe the method, I start with the data used in the development of the method. Then, I illustrate the hashing algorithm. Subsequently, I outline the algorithms used in each of the four steps.

Data: During the development of the method, I used the sequence of the X chromosome of the *D. Melanogater*. In addition, I obtained the genomic coordinates of the LTR transposons found on the X chromosome from (Lerat, 2010). The sequence of the *D. Melanogater* and its LTR transposons were obtained from FlyBase release 5.48 (McQuilton et al., 2012).

Data Structure: The algorithm depends on a data structure known as a hash table, which is conceptually similar to a dictionary or a phone book. Each row in the table consists of a key-value pair. Keys are unique. A value associated with a given key can be retrieved from the table using this key. In practice, an array is used to implement the hash table. The hash table uses a function known as the hashing function to convert a key to an index of the array. A value associated with a key is stored in the array at the index obtained by the hashing function. In this work, I designed a hashing function specific to DNA sequences. To start, nucleotides (A, C, G, and T) in a DNA sequence are converted to digits (0, 1, 2, and 3). The idea of the hashing function is to consider a k-mer as a quaternary, base-4, number, which is converted to a decimal number. This decimal number is the k-mer index in the underlying array. The Horner's rule (Cormen et al., 2001) is used to efficiently convert the quaternary number to the decimal number.

Step 1 - indexing the input sequence: The method depends on assembling short k-mer's, 10-15 bp long, found in the sequence into LTRs. To start, a sequence of indexes is generated from the input nucleotide sequence. The goal is to mark the location of the nearest exact copy of a k-mer. In the index sequence, the index associated with the start of a k-mer points to the index of the start of its copy. The distance between the k-mer and its copy has to be within a specific range due to the length property of the LTR transposons (Lerat, 2010). Figure 1 shows a simplified example. A pseudo code of the algorithm is given in Algorithm 1 in the Appendix.

Step 2 - finding candidates based on the structure of the LTRs: Once the index sequence is obtained, it is scanned for a subsequence of consecutive, increasing, possibly gaped indexes. Recall that an entry in the index sequence contains the index of the closest copy of the word starting at this entry. Therefore, this subsequence represents one of the LTRs, and its entries points to the other LTR. The scanning of the sequence results in two detections for each LTR transposon: the forward detection and the backward detection. Each detection includes two LTRs. The forward detection occurs when the 5' LTR is encountered. Entries of the 5' LTR point to the 3' LTR. Similarly, the backward detection occurs when the 3' LTR is encountered. Entries of the 3' LTR point the 5' LTR. For example, suppose that the region 100-200 represents the 5' LTR of a transposon, and the region 5100-5200 represents the 3' LTR. In the forward detection, the 100-200 region (5' LTR) points to the 5100-5200 region (3' LTR). Whereas, in the case of the backward detection, the 5100-5200 region (3' LTR) points to the 100-200 region (5' LTR). Next, a stack-like data structure is used to unify the two detections into one transposon candidate. As the index sequence is scanned, the forward detection is constructed first. Then, it is pushed to the top of the stack. Afterwards, the backward detection is constructed when the 3' LTR is encountered. The top of the stack is searched for a matching forward detection. When a match is found, it is removed from the stack. Then, the forward and backward detections are unified. Each unified detection, which consists of two LTRs and the sequence in between, is a potential transposon. However, each of the LTRs has to be 100 bp or longer; and both have to be at least 97% identical. I used the Smith-Waterman algorithm to calculate the identity score between the two LTRs (Smith and Waterman, 1981). In addition, the length of the sequence in between the two LTRs has to meet a user-specified threshold (at least 1 kbp).

Step 3 - filtering candidates based on the count of their k-mer's in the whole genome: Each LTR transposon has two LTRs. Consequently, the count of the k-mer's of the LTRs should be higher than the count of the k-mer's of the core sequence in between the two LTRs. This ratio should be 2:1 under perfect, but rare, conditions. Scanning the whole genome to count an LTR (100-1000 bp long) or the core sequence in between two LTRs (1000-15000 bp long (Ellinghaus et al., 2008)) is time-consuming and resources-intensive process. Therefore, LtrDetector uses the count of short k-mer's (about 14 bp) comprising the transposon in stead of using the count of the whole sequence. Then, the ratio of the median of the k-mer's of the LTRs to the median of the k-mer's of the core sequence in between the LTRs is calculated for each candidate. I call this ratio as the terminal-core ratio (Equation 1).

$$\text{Terminal-Core Ratio} = \frac{\text{Median of the k-mer's of the LTRs}}{\text{Median of the k-mer's of the core sequence}} \tag{1}$$

There are several copies of an LTR transposon in a genome. Therefore, the repetitive nature of the LTR transposons is a potential filtering criterion. It was observed that the count of k-mer's of the majority of transposons are higher than that of k-mer's of non-repeats sequences (Price et al., 2005; Kurtz et al., 2008). However, I decided not to use this principle as a basis for filtering because a threshold based on k-mer counts is species-specific (Kurtz et al., 2008). In addition, this threshold may be biased against transposons that have low copy number.

To calculate the terminal-core ratio, the DNA hash table, described earlier, is used for storing the count of a k-mer in the genome. The table holds the counts of 4^k k-mer's. The value associated with each k-mer in the table represents the count of this k-mer in the whole genome. Once the table is built, the input nucleotide sequence, which is usually a chromosome sequence, is converted into a sequence of counts. A nucleotide is assigned the count of the k-mer starting at this nucleotide.

Next, I trained a decision tree (DT) (Mitchell, 1997) to determine the terminal-core ratio from experimental data. A DT is an instance of supervised learning, which requires labeled data. To generate labeled data, I used the LtrDetector to process the sequence of the X chromosome of the *D. Melanogater* as described in the previous step. Candidates detected by LtrDetector were divided into two groups. The first group consisted of detections overlapping with known LTR

transposons found in the X chromosome, i.e. these detections are true positives. Candidates that did not overlap with known LTR transposons comprised the second group. The majority of detections in the second group are likely false positives; however, some of these detections can be unknown LTR transposons. Accordingly, the task is to remove the false positives but to keep the new transposons. The DT is trained and tested using the 10-fold cross validation. Cross validation is usually used when the size of the labeled data is small. To apply the cross validation, the labeled data are divided into 10 partitions. Then, training and testing are repeated 10 times. At each iteration, the DT is trained on 9 partitions; then, the DT is used to predict the labels of the held out partition. Each partition is held out once while the other nine partitions are used for training.

The DT found that the median of the k-mer counts in the LTRs is 1.5-5 times greater than the median of those found in the sequence between the two LTRs. As a result, candidates that have terminal-core ratio less than 1.5 or greater than 5.0 are unlikely to be LTR transposons.

After that, I wanted to ensure that this property is applicable to all species specifically the human.

Step 4 - delineating candidates based on the TSD: The final step is to determine the boundaries of the potential transposon accurately. To this end, LtrDetector searches for the TSD. TSD are two identical, short sequences that precede the 5' LTR and follow the 3' LTR. Computationally, the identification of the TSD are known as the problem of the longest common substring where two sequences are searched for the longest common sub-sequence without gaps. There is a well-known algorithm based on dynamic programming to solve this problem. I applied this algorithms to the 100 bp preceding the 5' LTR and the 100 bp following the 3' LTR. The common substring, TSD, has to be at least 4 bp long; otherwise, this candidate transposon is rejected. If more than one common substring is found, the one that has the closest average distance to the LTRs is reported.

3 Results

- Show results on the k-order random chromosome. This should be a one paragraph to show that the method did not work when it was not supposed to work.

Software availability:

(Chor et al., 2009)

Appendix

Algorithm 1 Index copies of K-mers

INPUT: C {A sequence of alphabet $\{A, C, G, T\}$}
INPUT: MAX {Maximum distance}
INPUT: MIN {Minimum distance}
OUTPUT: I {A sequence of indexes}
$S = \text{encode}(C)$ {Convert A, C, G, and T to 0, 1, 2, and 3, respectively}
$X = \text{size}(S)$
$I = \text{array}(X)$ {Construct an array of size X}
$T = \text{table}(S)$ {Construct a hash table of all words of length K. All values are set to $\emptyset$}
for $G = 0$ to $X - 1$ **do**
  $W = S[G, G + K - 1]$ {A K-mer starting at index G}
  $L = T[W]$ {The last index of the K-mer W before index G}
  if $L \neq \emptyset$ **then**
    $D = |G - L|$
    {Check if the distance between G and L is within the specified range}
    if $D \geq MIN$ and $D \leq MAX$ **then**
      $I[G] = L$
      $N = I[L]$
      {Check to update the index associated with index L}
      if $|L - N| > D$ **then**
        $I[L] = G$
      end if
    end if
  end if
  $T[W] = G$
end for

Figures

Figure 1 - Indexing Example

In this example the input sequence is indexed according to words of length 4 bp, i.e. 4-mer (the actual results are based on longer words). To allow efficient computations, the sequence of

nucleotides is converted to a sequence of digits, S. The letters: A, C, G, and T are converted to the digits: 0, 1, 2, and 3, respectively. Initially, an empty array, I, of the same size as the input sequence is allocated; in addition, a hash table, T, of all possible words of length four is allocated. An entry in the array represents the index of the closest copy of the word starting at the corresponding index of this entry. The distance between the word and its copy has to be within a predetermined range. The table is used to store the index of the last copy of a word. The algorithm scans the sequence from left to right. Assume that the first copy of "3333" starts at index i. Once index i of the input sequence is reached, the value associated with the word "3333" in the table is set to i. The second copy of "3333" starts at index j. Then, the index of the last occurrence of this word is retrieved from the table. Currently, it is i. If the distance between i and j is within the desired range, the i^{th} and the j^{th} entries of the array are set to j and i, respectively. In other words, i is pointing to j and j is pointing to i. After that, the value associated with "3333" in the table is updated to j. The third copy is encountered at index h. Again, the table is used to retrieve the index of the last occurrence, which is j at this time. If the distance between h and j is within the specified distance range, the h^{th} entry of the array is assigned to j. The j^{th} entry of the array, which currently holds i, is updated to h if h is closer to j than i. Then, the value associated with the word "3333" is updated to h in the table.

Figure 2 - Sample figure title

Figure legend text.

Author's contributions

Text for this section ...

Acknowledgements

Text for this section ...

References

Chor, B., Horn, D., Goldman, N., Levy, Y., and Massingham, T., 2009. Genomic DNA k-mer spectra: models and modalities. *Genome Biol*, **10**(10):R108.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E., 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Ellinghaus, D., Kurtz, S., and Willhoeft, U., 2008. Ltrharvest, an efficient and flexible software for de novo detection of ltr retrotransposons. *BMC Bioinformatics*, **9**(1):18.

Kalyanaraman, A. and Aluru, S., 2006. Efficient algorithms and software for detection of full-length LTR retrotransposons. *J Bioinform Comput Biol*, **4**(2):197–216.

Kurtz, S., Narechania, A., Stein, J., and Ware, D., 2008. A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, **9**(1):517.

Lerat, E., 2010. Identifying repeats and transposable elements in sequenced genomes: how to find your way through the dense forest of programs. *Heredity*, **104**(6):520.

McCarthy, E. M. and McDonald, J. F., 2003. LTR_STRUC: a novel search and identification program for LTR retrotransposons. *Bioinformatics*, **19**(3):362–367.

McQuilton, P., St. Pierre, S. E., Thurmond, J., and the FlyBase Consortium, 2012. FlyBase 101 – the basics of navigating FlyBase. *Nucleic Acids Res*, **40**(D1):D706–D714.

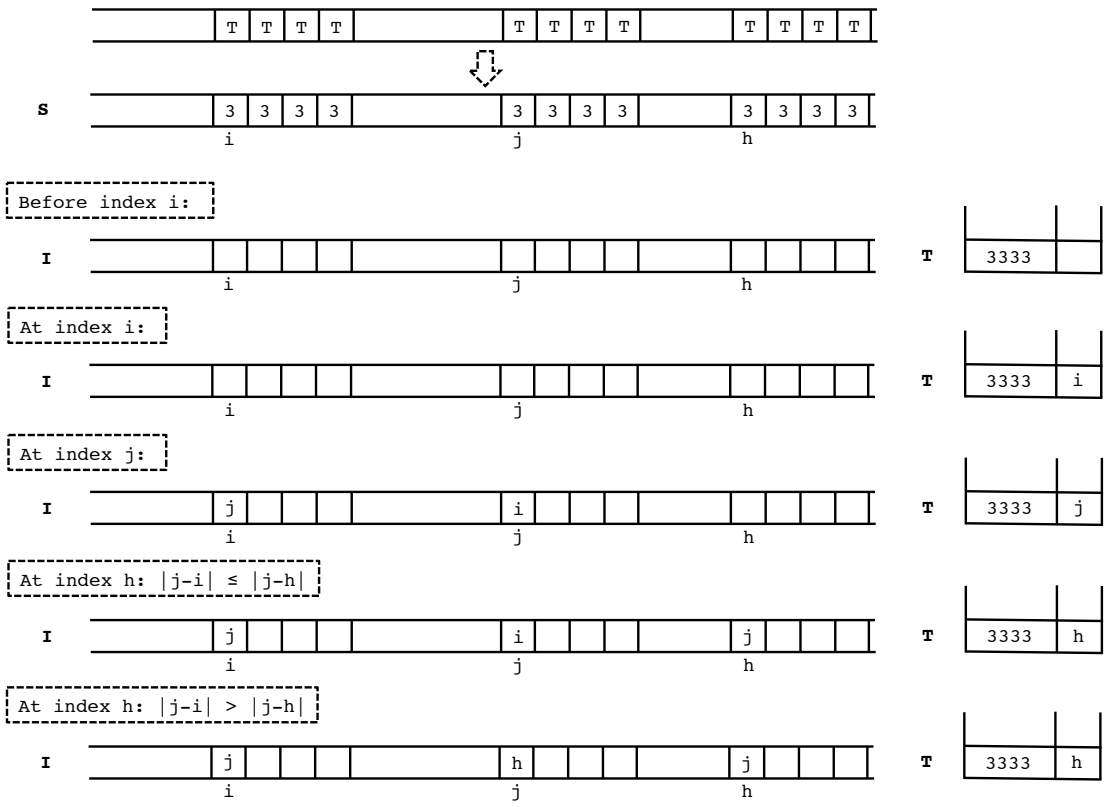

Figure 1: Indexing

Mitchell, T. M., 1997. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Price, A. L., Jones, N. C., and Pevzner, P. A., 2005. De novo identification of repeat families in large genomes. *Bioinformatics*, **21**(suppl 1):i351–i358.

Rho, M., Choi, J.-H., Kim, S., Lynch, M., and Tang, H., 2007. De novo identification of LTR retrotransposons in eukaryotic genomes. *BMC Genomics*, **8**:90.

Smith, F. T. and Waterman, M. S., 1981. Identification of common molecular subsequences. *J Mol Biol*, **147**(1):195–197.

Xu, Z. and Wang, H., 2007. LTR_FINDER: an efficient tool for the prediction of full-length LTR retrotransposons. *Nucleic Acids Res*, **35**(suppl 2):W265–W268.

Tables

Table 1 - Sample table title

Here is an example of a *small* table in LaTeX using `\tabular{...}`. This is where the description of the table should go.

My Table		
A1	B2	C3
A2	...	..
A3	..	.

Additional Files

Additional file 1 — Sample additional file title

Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title

Additional file descriptions text.