Journal of Bioinformatics and Computational Biology
© Imperial College Press

EFFICIENT ALGORITHMS AND SOFTWARE FOR DETECTION OF FULL-LENGTH LTR RETROTRANSPOSONS*

ANANTHARAMAN KALYANARAMAN
Department of Electrical and Computer Engineering, Iowa State University
Ames, Iowa 50011, United States of America
ananthk@iastate.edu

SRINIVAS ALURU
Department of Electrical and Computer Engineering, Iowa State University
Ames, Iowa 50011, United States of America
aluru@iastate.edu

Received (15 September 2005)
Revised (1 December 2005)
Accepted (3 January 2006)

LTR retrotransposons constitute one of the most abundant classes of repetitive elements in eukaryotic genomes. In this paper, we present a new algorithm for detection of full-length LTR retrotransposons in genomic sequences. The algorithm identifies regions in a genomic sequence that show structural characteristics of LTR retrotransposons. Three key components distinguish our algorithm from that of current software — (i) a novel method that preprocesses the entire genomic sequence in linear time and produces high quality pairs of LTR candidates in run-time that is constant per pair, (ii) a thorough alignment-based evaluation of candidate pairs to ensure high quality prediction, and (iii) a robust parameter set encompassing both structural constraints and quality controls providing users with a high degree of flexibility. Validation of both our serial and parallel implementations of the algorithm against the yeast genome indicates both superior quality and performance results when compared to existing software. Additional validations on rice BACs and the newly sequenced chimpanzee genome also show encouraging results.

Keywords: Full-length LTR retrotransposons; retroelements; reverse transcriptase; *de novo* repeat identification.

1. Introduction

Retrotransposons are DNA sequences that reside within cells of a host organism, copying and inserting themselves into the host genome. Studies have revealed their ubiquity in many eukaryotic organisms, both plants and animals — they constitute more than 50% of the maize genome[32,39,40], up to 90% of the wheat genome[9] and

*In press for publication in June 2006

at least 45% of the human genome[7]. *LTR retrotransposons* form a special class of retroelements that are typically characterized by two long terminal identical repeat sequences, one at the $5'$ end and the other at the $3'$ end of the inserted retrotransposon; these terminal repeats are referred to as *Long Terminal Repeats* or *LTRs*. LTR retrotransposons were originally discovered in maize and tobacco[14,18,43], and are now known to be abundant in numerous complex eukaryotic plant (e.g., barley, rice, maize, wheat, etc.) and animal (e.g., Drosophila, human, mouse, etc.) genomes.

Ever since their discovery, LTR retrotransposons have been a topic of great research interest to biologists. Understanding the behavior of these retroelements has been key to many significant advances in molecular genetics and functional genomes[5,8,12,33,40]. Because of their mobile nature, retrotransposons play key roles in genomic rearrangements[3,8,24] and in the evolution of genes and genomes[12,19,44,45,46]. LTR retrotransposons have also been identified to be sources of spontaneous and induced mutations and are an important subject in studies relating to mutations and genetic variations[17,22,43]. The transposition mechanism by which LTR retrotransposons copy and relocate involves an RNA-intermediary — a copy of the retrotransposon is made into an RNA molecule, which is then inserted back as a DNA molecule in another location of the host genome, with the aid of an enzyme called reverse transcriptase. This mechanism being highly similar to the transposition mechanism of retroviruses such as the *HIV* has contributed to a continued interest in retrotransposon research[4,6]. The structural attributes of LTR retrotransposons provides significant insights into species evolution because of the following property: the $5'$ and $3'$ LTRs of a retrotransposon are completely identical when the retrotransposon inserts itself, but can undergo mutations and become increasingly divergent with time[36,47].

The aforementioned applications and many others have been contributing to a sustained research interest in LTR retrotransposons. Also with the continued advancement in sequencing technology and with various new large-scale genome sequencing projects of complex eukaryotic organisms either currently underway or finished, understanding retrotransposons and their biological role in all these genomes has become imperative in furthering research for functional and molecular genomics.

In this paper, we propose an efficient algorithm for *de novo* identification of full-length LTR retrotransposons with key emphasis on quality and performance. The main contributions of this research are the following:

- an efficient algorithm for quickly generating high-quality candidates, significantly reducing the search space. The algorithm has a run-time complexity proportional to the input size plus the number of candidates (i.e., amortized constant time per candidate);
- a thorough alignment-based evaluation of candidates using standard dynamic programming techniques[35] that guarantees optimality in the alignment score;

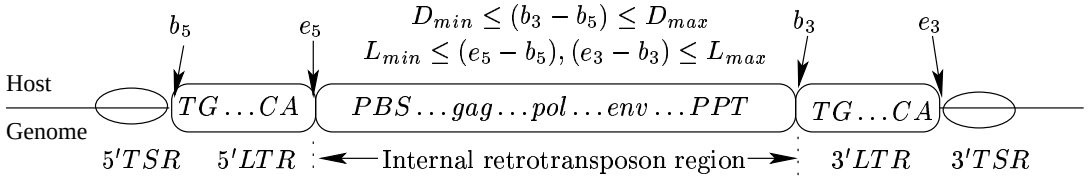

Fig. 1. A full-length LTR retrotransposon is characterized by an internal region containing the retrotransposon that is flanked by two identical repeats (5′ and 3′ *Long Terminal Repeats* or *LTRs*), and two identical short repeats 5–6 bp long (5′ and 3′ *Target Site Repeats* or *TSRs*) that are a result of a target site duplication event when the retroelement inserts itself into the host genome. The internal retrotransposon region contains the following sequences from 5′ to 3′: *Primer Binding Site* (*PBS*) is a region complementary to a tRNA 3′ terminal sequence used during reverse transcription at a later stage of the retroelement's life cycle; the *gag* region is a gene that codes for a capsid-like protein; the *pol* region contains genes coding for protease, integrase and reverse transcriptase enzymes; the *env* region contains the gene coding for an envelope protein; and the 3′ end of the retroelement contains a purine-rich sequence called the *Poly-Purine Tract* (*PPT*).

- support for a robust parameter set encompassing both structural constraints and quality controls; and
- an implementation of our algorithm that can run on both serial and parallel computers.

Preliminary validations of our software indicate better quality results and significantly faster run-times than currently available software. For example, our software took 10 minutes on the yeast genome (on a 1.1 GHz Pentium III) and made better predictions than *LTR_STRUC*[30], which took 210 minutes (on a 1 GHz Pentium III) despite not computing rigorous alignments. Furthermore, the parallel implementation of our algorithm can be used to further reduce the run-time in proportion to the number of processors used. Our software also provides a flexible framework to incorporate more LTR-specific improvements with minimal changes to the algorithmic core.

2. Problem Description and Related Work

The typical structure of a full-length LTR retrotransposon has been well characterized in the literature, and is illustrated in Figure 1. For computational detection, these structural attributes can be modeled as follows:

- **Similarity Constraint:** The 5′ and 3′ LTRs show a good sequence homology that can be demonstrated by a high-scoring global alignment between them. While the LTRs are identical when a retroelement inserts into a host genome, they may accumulate mutational variations — hence the need for computing an alignment.
- **Distance Constraint:** The number of nucleotides separating the starting positions of the two LTRs is bounded within the range $[D_{min}, D_{max}]$ — this range is

determined by the minimum and the maximum expected lengths of an individual LTR and the internal retrotransposon region; a range of $[100, 15000]$ is sufficient in practice.

- **TSRs:** The regions 5–6 bp long immediately upstream and downstream of a $5'$ and $3'$ LTR respectively have a high sequence identity (this definition accounts for mutational variations in the TSR regions as well).
- **LTR motif:** Most LTR sequences start and end with a conserved motif such as $TG \ldots CA$.
- **Other Signals:** The region between a $5'$ and $3'$ LTR pair contains a series of special purpose genes and sequences: PBS, *gag*, *pol*, *env*, and *PPT*.

While the sequence identity expected between $5'$ and $3'$ LTRs of a retrotransposon could vary across different retroelement families, typically ranging between 70%–100%[24], a high identity (>90%) has been observed in most cases[24,37]. Because of the strong homology expected between $5'$ and $3'$ LTRs, they are also expected to contain long exact matches. Thus, identification of exact matching repeats serves as a good starting point for LTR retrotransposon detection. Repeat detection is a well studied problem and a number of excellent programs are already available. These include *RepeatMasker*[42], *REPuter*[26,27] and *RECON*[2]. LTR retrotransposons, on the other hand, are uniquely characterized by distance constraints. Therefore, the repeats identified by general purpose repeat identification software must be screened to eliminate repeats that do not satisfy the distance constraint. For instance, the *SMaRTFinder* program[34] designed for retrotransposon detection utilizes *REPuter* for repeat detection prior to screening for additional LTR features. The problem with this approach is the extra run-time cost incurred in initially generating repeats that are either too close or too far apart to be part of any valid LTR retrotransposon — an issue for highly repetitive genomes. Even on genomes with abundant LTR retrotransposon content, there could be an LTR sequence that is common across numerous members of the same retrotransposon family, and generic repeat finding tools will generate all pairs of these LTRs before invalid pairs are sieved out.

A more efficient solution is to build software that is specifically designed for LTR retrotransposon detection, and *LTR_STRUC*[30] is the only available program that is so designed. It has been successfully used for detection of full-length LTR retrotransposons in *Oryza sativa*[29], *Mus musculus*[31] and *Drosophila melanogaster*[10]. The underlying algorithm, however, is a brute-force approach that results in unnecessarily long run-times, which could be problematic for large genomic sequences. A more efficient algorithm will significantly reduce the cost of identifying potential LTR pairs, and the resulting time savings could be utilized to improve prediction quality.

The underlying algorithm in *LTR_STRUC* can be viewed as a two-step procedure: (i) detect all pairs of genomic locations that both satisfy the distance constraint and are starting positions of two "highly similar" (say 70% identity) substrings (or "seeds") of a particular fixed length ω (say 40 bp) — each such pair can

be considered a "candidate pair"; (ii) for each candidate pair generated, extend the seeds in either direction as long as the alignment continues to satisfy the similarity constraint. The resulting aligning regions are reported as a full-length LTR retrotransposon. Alignment of an extension is computed by a simple greedy strategy that aligns longer exact matches before aligning the remainder of the region with shorter matches. This method does not guarantee a best possible alignment of the predicted LTRs, and therefore has the potential danger of missing some LTR pairs. Ideally, an alignment method that computes a combinatorially optimal alignment score is desirable to ensure that no such genuine LTR pairs are missed.

Candidate pairs are generated by the following brute-force approach: Let s denote the input genomic sequence of length n. Walk along s and for each position i, $1 \leq i \leq n$, scan all positions j such that $(i + D_{min}) \leq j \leq (i + D_{max})$. For each (i, j)-pair, compute the percentage identity of the two $\omega-$length substrings starting at i and j. If the identity is above the similarity threshold (say 70%), then the pair (i, j) is reported as a "candidate pair" and is further evaluated for alignment as described above. The algorithm has a worst-case run-time complexity of $O(n \times (D_{max} - D_{min}) \times \omega)$. In practice, D_{max} could be as high as 10,000 - 15,000 and D_{min} could be as low as 100. In an attempt to save run-time, the algorithm's implementation resorts to a technique of sampling the search interval, i.e., the value of i is incremented by some $\Delta i > 1$ instead of 1. This would reduce the run-time cost by a factor of Δi, but also at the expense of prediction accuracy. Moreover, this algorithm will consider many redundant or "duplicate" pairs of locations corresponding to the same matching pair of regions. To see this, note that if a $5'$-$3'$ LTR pair share a long exact match of length $l > \omega$ bp, then there are $(l - \omega + 1)$ pairs of $\omega-$length identical substrings and the algorithm will generate all these pairs of locations even though they correspond to the same longer exact match. Ideally, generation of such"duplicate" pairs should be completely avoided in the interest of run-time. Also note that the run-time complexity is independent of the repetitive nature of the genome, i.e., while at long stretches of the genome that have no LTRs, this algorithm would search for an entire $(D_{max} - D_{min})$-length interval only to result in more wasted effort.

In a pilot study on a Windows machine with 1 GHz Pentium III processor conducted by one of our colleagues[11], *LTR_STRUC* took 3.5 hours on the entire yeast genome (over 12 Mbp) and over 15 hours on chromosome 1 of *Arabidopsis thaliana* genome (over 30 Mbp). These high run-times are likely to be a major limiting factor in the applicability of the *LTR_STRUC* software on larger genomes such as the human, maize, etc., mainly because a biologist would like to run a *de novo* prediction tool such as *LTR_STRUC* multiple times under different parameter settings before arriving at a high-quality repository of predictions.

3. Notation

Let s denote the input DNA sequence comprising of n nucleotides. For computational purposes, we view s as a string of n characters in alphabet $\Sigma = \{A, C, G, T, N\}$, where '$N$' may denote either a low-quality or masked base in the input sequence. Let $s[i]$ denote the character at position i in s ($1 \leq i \leq n$). Let $s[i..j]$ denote the substring $s[i]s[i+1]\ldots s[j]$. Let $left(i) = s[i-1]$, if $i > 1$, and 'N' otherwise; similarly, let $right(i) = s[i+1]$, if $i < n$, and 'N' otherwise. Two identical substrings $s[i_1..(i_1+k)] = s[i_2..(i_2+k)]$ are said to be *left-maximal* (respectively *right-maximal*) if and only if $left(i_1) \neq left(i_2)$ (respectively $right(i_1+k) \neq right(i_2+k)$). They are said to be a *maximal matching pair* if they are both right- and left-maximal. We will assume that aligning 'N' with any other character should be treated as a mismatch.

4. Our Approach

The main idea of our approach is to have an efficient linear time preprocessing of the entire input sequence, followed by an algorithm that provides a direct mechanism (as opposed to a searching mechanism) for generation of "candidate pairs". Our definition of "candidate pairs" is based on maximal matches subject to LTR retrotransposon length constraints. Each candidate pair is then subjected to a rigorous alignment test that guarantees an alignment with the combinatorially best score for testing against the similarity constraint.

The advantage of generating candidate pairs based on maximal matches instead of fixed-length matches is that it provides a direct means of detecting a "long" exact match rather than as a chain of smaller fixed-length exact matches. While the detection of maximal matches is well studied in literature using the suffix tree data structure[16], our pair generation algorithm follows a related strategy using the suffix array and Longest Common Prefix (LCP) array[16] data structures, taking into account the distance constraints. The suffix array of a string is the lexicographically sorted array of all its suffixes, and the following property is key for our pair generation algorithm: any two identical substrings starting at a pair of positions can be represented as a common prefix shared by the two suffixes starting at these positions.

4.1. *The Sequential Algorithm*

Let L_{min} (L_{max}) denote the minimum (maximum) allowed length of an LTR (as shown in Figure 1). Let L_{ex} denote a length such that any $5'$-$3'$ pair of LTRs will share at least one exact match of that length. This user-specified parameter can even be analytically computed as follows: if ψ, the rate of mutation (as a fraction) in the host genome is known, a reasonable value is $\frac{L_{min}}{(\psi \times L_{min})+1}$.

Definition 1. Candidate Pair: A pair of genomic positions (i_1, i_2) ($1 \leq i_1, i_2 \leq n$) is defined to be a *candidate pair* if and only if it satisfies the following properties:

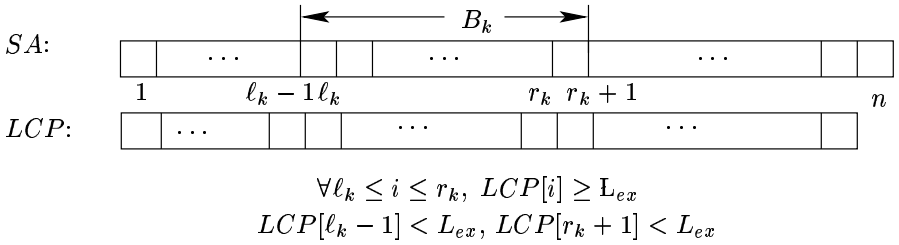

Fig. 2. Illustration of the process of creating a bucket B_k during preprocessing. Each bucket corresponds to a set of suffixes covered by a unique maximal interval within the LCP array with values greater than or equal L_{ex}. In the above example, $B_k = \{SA[l_k], SA[l_k + 1], \ldots, SA[r_k + 1]\}$ corresponding to the maximal interval: $LCP[l_k..r_k]$.

(1) the positions satisfy the distance constraint, i.e., $(i_1 + D_{min}) \leq i_2 \leq (i_1 + D_{max})$.
(2) the substrings $s[i_1 \,..\, (i_1 + L_{ex} - 1)]$ and $s[i_2 \,..\, (i_2 + L_{ex} - 1)]$ are left-maximal.

Note that there is a one-to-one correspondence between the set of maximal matching pairs of minimum length L_{ex} and left-maximal pairs of length L_{ex}. Our algorithm comprises of three phases: a preprocessing phase, a candidate pair generation phase, and an alignment phase.

4.1.1. *Preprocessing*

The goal of preprocessing is to "arrange" the positions $\{1, 2, \ldots, n\}$ in s in a manner that allows quick generation of candidate pairs as per Definition 1. This is achieved in two steps — (i) partition the positions based on their their L_{ex}-length substrings and then internally subpartition them based on the character preceding each position so that any two left-maximal substrings are in different subsets, and (ii) sort the positions within each subset so that the distance constraint check can be quickly performed. The algorithm is as follows.

In the first step, construct a suffix array (denoted by SA) data structure[28] on s in linear time[20,23,25] and also the corresponding longest common prefix array (denoted by LCP)[21]. As a result, $SA[i]$ is the i^{th} lexicographically smallest suffix in s ($\forall 1 \leq i \leq n$), and $LCP[i]$ is the length of the longest common prefix between suffixes $SA[i]$ and $SA[i + 1]$ ($\forall 1 \leq i \leq n - 1$). Next, a set $B = \{B_1, B_2, \ldots, B_m\}$ of m buckets is generated such that $\forall i, j \in B_k$, $\forall 1 \leq k \leq m$, $s[i \,..\, (i + L_{ex} - 1)] = s[j \,..\, (j + L_{ex} - 1)]$. This is achieved by linearly scanning the $LCP[1..n - 1]$ array and recording all maximal intervals in which the LCP values are all greater than or equal to L_{ex}. The value of m is therefore the number of such maximal intervals. For each maximal interval the set of all suffix entries that it covers in the array $SA[1..n]$ is then assigned to a unique bucket in B. See Figure 2 for an illustration. Because every LCP entry covers two consecutive suffix entries in SA, each resulting bucket contains at least two suffix entries, i.e., $0 \leq m \leq \lfloor \frac{n}{2} \rfloor$. Choosing maximal

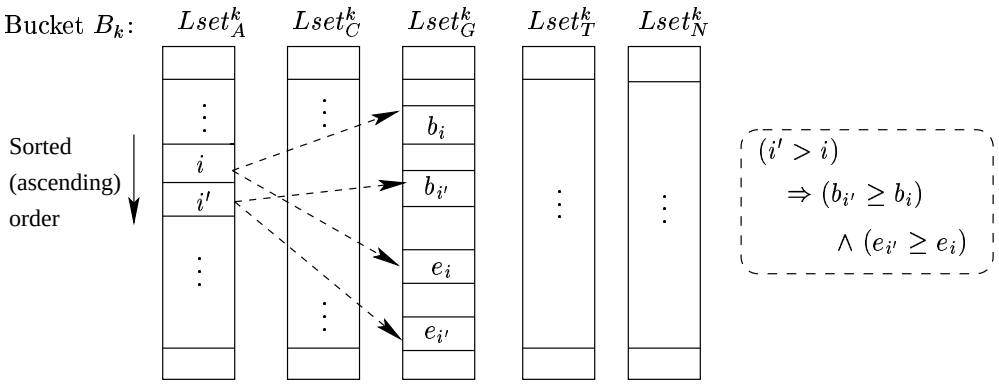

Fig. 3. Illustration of the candidate pair generation algorithm. Shown are the five *Lsets* for a given bucket B_k. The entry $i \in Lset_A^k$ is paired with all entries satisfying the distance constraint, denoted by the interval $[b_i \ldots e_i]$, in $Lset_G^k$. For the next entry $i' \in Lset_A^k$, the corresponding interval $[b_{i'} \ldots e_{i'}]$ is such that $b_{i'} \geq b_i$ and $e_{i'} \geq e_i$.

intervals in the LCP array with values $\geq L_{ex}$ ensures that $\forall 1 \leq k \leq m$, $\forall i \in B_k$, all substrings $s[i..(i + L_{ex} - 1)]$ are identical.

The next step is to sort each bucket in the ascending order of the position numbers. This is done once for all buckets through a stable integer sort. Each bucket B_k is then further partitioned into $|\Sigma|$ ordered sets called *Lsets*: $\forall c \in \Sigma$, $Lset_c^k = \{i \mid left(i) = c, i \in B_k\}$. It is easy to see that one can partition every B_k into these individual *Lsets* still maintaining the internal sorted order within each *Lset*. Maintaining the sorted order is critical for efficient generation of candidate pairs, as will soon become evident.

4.1.2. *Candidate Pair Generation*

Once the input sequence is preprocessed, candidate pairs can be generated from within each bucket. The algorithm for generating candidate pairs is presented in Figure 4 and an illustration to help understand the algorithm is provided in Figure 3.

For each bucket B_k, all *Lsets* are scanned in the ascending order of the position number. A position i in $Lset_{c_1}^k$ is paired with a position j if and only if $j \in Lset_{c_2}^k$ such that $c_2 \neq c_1$ or $c_2 =$'N' (i.e., the substrings $s[i..(i+L_{ex}-1)]$ and $s[j..(j+L_{ex}-1)]$ are left-maximal), and $(i + D_{min}) \leq j \leq (i + D_{max})$ (i.e., the pair (i, j) satisfies the distance constraints). This guarantees that a pair (i, j) is generated only if it is a candidate pair by Definition 1. Enumerating all j (and only those j) that should be paired with i is achieved in the following manner. Since each *Lset* is internally sorted by position numbers, the valid entries for j for a given value of i will be placed consecutively in $Lset_{c_2}^k$, defined by a range, say $[b_i, \ldots, e_i]$. If i is the first entry of the ordered set $Lset_{c_1}^k$, b_i can be located in $Lset_{c_2}^k$ by performing a linear scan until a value that is $\geq (i + D_{min})$ is encountered. Once b_i is located, we can

Algorithm 1. Candidate Pair Generation

Input: Bucket B_k
L_1: FOR EACH $c_1 \in \Sigma$ DO
L_2: FOR EACH $i \in Lset^k_{c_1}$ DO
L_3: FOR EACH $c_2 \in \Sigma$ and ($c_1 \neq c_2$ or $c_1 = c_2 =$ 'N') DO
S_1: $b_i \leftarrow min\{j | j \in Lset^k_{c_2}, D_{min} \leq (j-i) \leq D_{max}\}$
S_2: $e_i \leftarrow max\{j | j \in Lset^k_{c_2}, D_{min} \leq (j-i) \leq D_{max}\}$
S_3: Generate pairs $(i,j), \forall j \in Lset^k_{c_2}, b_i \leq j \leq e_i$

Fig. 4. Algorithm to generate candidate pairs from a given bucket B_k.

continue pairing i with all subsequent elements from b_i in $Lset^k_{c_2}$ until $(i + D_{max})$ is exceeded or the $Lset$ is exhausted. The last element to be paired is e_i. Henceforth, in advancing each i to its next position say i' in $Lset^k_{c_1}$, it is sufficient to start searching for $b_{i'}$ from b_i onwards, since $b_{i'} \geq b_i$ as $i' > i$. Even better, one can record the location of $b_{i'}$ if it is found before e_i, while generating pairs for i, and directly start from $b_{i'}$ for i'.

Since the algorithm ensures every entry in each bucket is considered for i, and that for each such i all valid entries for j from the same bucket are considered, it can be seen that our candidate pair generation does not miss any candidate pair satisfying Definition 1. Moreover, since each entry in a bucket is considered for i exactly once it is also easy to see that each candidate pair is generated exactly once.

Lemma 1. *Let* $s[i_1..(i_1+k-1)]$ *and* $s[i_2..(i_2+k-1)]$ *be two maximal matching substrings, for some* $k \geq L_{ex}$*, and* $(i_1 + D_{min}) \leq i_2 \leq (i_1 + D_{max})$*. Then* (i_1, i_2) *is generated exactly once.*

Proof. If $s[i_1..(i_1+k-1)]$ and $s[i_2..(i_2+k-1)]$ are two maximal matching substrings of length $\geq L_{ex}$ then there is exactly one pair of left-maximal L_{ex}-long substrings: $s[i_1..(i_1 + L_{ex} - 1)]$ and $s[i_2..(i_2 + L_{ex} - 1)]$. Therefore (i_1, i_2) is a candidate pair by Definition 1 and is generated exactly once by the algorithm. □

4.1.3. *Run-time Analysis*

For the preprocessing phase, the construction of suffix array[20,23,25] and LCP array[21] take $O(n)$ time. Generating the set of buckets also takes $O(n)$ time because the algorithm performs a linear scan of the arrays. Sorting each bucket by position numbers and generating all $Lsets$ for all buckets are integer sorting operations. The outermost loops, L_1 and L_2 of Algorithm 1, over all iterations visits each position in $\{1, \ldots, n\}$ at most once, although in an arbitrary order. By Lemma 1, the cost of Step S_3 over all iterations is proportional to the number of candidate pairs generated. For steps S_1 and S_2, note that at worst case, locating a particular b_i

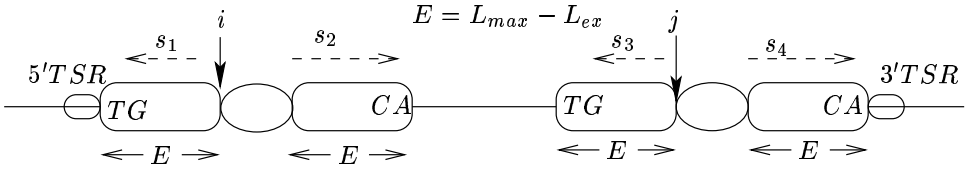

Fig. 5. Two alignments are performed for each candidate pair (i, j): s_1 vs. s_3 and s_2 vs. s_4. Dotted arrows indicate the directions of the alignments, and the two ovals indicate the anchoring match.

may take $O(n)$ if it is the first entry in its $Lset$. However, the amortized worst case total cost is still $O(n)$ because each entry is considered exactly once for choice of i and at most $2\times|\Sigma|$ times for j, implying a run-time cost of $O((2\times|\Sigma|+1)\times n) = O(n)$ (taking $|\Sigma| = 5$ to be a constant). Thus the pair generation algorithm has an optimal run-time, i.e., $O(n)$ plus the number of candidates pairs in s.

4.1.4. *Alignment and LTR Prediction*

Once a candidate pair is reported, the regions flanking the corresponding match are evaluated to check if the aggregate region indeed has an expected LTR structure. This is achieved by computing an alignment as follows: For each candidate pair (i, j), four substrings each of length $L_{max} - L_{ex}$ are extracted as indicated in Figure 5, following which two alignments are computed — one alignment between $s[i+L_{ex} \,..\, i+L_{max}-1]$ and $s[j+L_{ex} \,..\, j+L_{max}-1]$, and another alignment between the reverse of $s[i-(L_{max}-L_{ex}) \,..\, i-1]$ and the reverse of $s[j-(L_{max}-L_{ex}) \,..\, j-1]$. We use standard dynamic programming techniques for computing an optimal global alignment score between two sequences using affine gap penalties[13]. In order to save run-time, alignment computation is restricted over a band of diagonals while ensuring the optimality of score. Once the alignments are computed, an aggregate alignment score is calculated by adding the scores of the best aligning prefixes in the two computed alignments plus the matching score of the anchored match in the middle. If this aggregate score satisfies the specified similarity constraint, the boundaries of the two aligning regions is reported as a *predicted pair of LTRs*.

As part of the above outlined alignment method, we also account for the presence of TSRs and LTR motifs. TSRs are detected by looking for an exact match of length 5-6 bp in the left and right vicinity of the predicted $5'$ and $3'$ LTRs, respectively (as shown in Figure 5). Also, the $5'$ and $3'$ ends of each of the two LTRs and their vicinity are searched for a presence of the motifs TG and CA respectively. Along the process of this search for motifs and TSRs, the alignment boundaries are adjusted between the predicted pair of LTRs.

In order to be able to distinguish among the predictions made by our algorithm based on the presence and absence of LTR structural signals, we associate a "confidence level" and report it as part of each prediction. The confidence level for each

prediction is given by the following formula:

$$Confidence\ Level = Weight_{TSR} * TSR_{code} + Weight_{motif} * Motif_{code}$$

where $0 \leq Weight_{TSR}, Weight_{motif} \leq 1$ are weights assigned by the user to specify the relative importance of the presence of identical TSRs and motifs; note that $Weight_{TSR} + Weight_{motif} = 1$. For example, if the presence of the motifs in both LTRs is only half as important as the presence of identical TSRs, then the values can be: $Weight_{motif} = 0.33$ and $Weight_{TSR} = 0.67$. For a given prediction: TSR_{code} is set to 1 if the two predicted TSRs are identical, and 0 otherwise; and $Motif_{code}$ is set to 1 if both $5'$ and $3'$ LTRs start and end with TG and CA respectively, 0.5 if only one motif is found, and 0 otherwise. Given weights of 0.5 for both $Weight_{TSR}$ and $Weight_{motif}$, Table 1 shows the different confidence levels and their meanings.

Table 1. Confidence levels for different scenarios depending on the presence or absence of TSRs and motifs.

TSRs	$5'$ Motif (TG)	$3'$ Motif (CA)	Confidence Level
Identical	Present	Present	1.0
Identical	Present	Absent	0.75
Identical	Absent	Present	0.75
Identical	Absent	Absent	0.5
Not Identical	Present	Present	0.5
Not Identical	Present	Absent	0.25
Not Identical	Absent	Present	0.25
Not Identical	Absent	Absent	0.0

4.2. *Parallelization*

The sequential algorithm presented in the previous section is parallelized in the following manner: The input sequence can be distributed evenly across processors. To ensure that no pairs are missed, the last $D_{max} - 1$ characters in the local portion of the input are duplicated as a prefix in the local portion of the next processor, resulting in at most $\frac{n}{p} + D_{max} - 1$ characters per processor. Once distributed, each processor can run the serial algorithm on its local portion of the input without needing to further communicate. The speedup of the preprocessing phase is proportional to $\frac{n}{\frac{n}{p}+D_{max}}$, thereby achieving a linear speedup as long as $D_{max} << \frac{n}{p}$. The speedup of the candidate pair generation and alignment phases are however highly dependent on the input, and the distribution of the repetitive elements among processors. Although one can dynamically balance this workload, our current implementation does not support such features.

Table 2. Parameter set for our program with default values.

Parameter Name	Default Value	Comment
D_{min},D_{max}	100, 15000	Distance constraint for 5′-3′ LTR pair
L_{min}, L_{max}	100, 1000	Length constraints for 5′-3′ LTR pair
L_{ex}	20	Exact match length requirement for 5′-3′ LTR pair
τ	75%	Similarity threshold of a 5′-3′ LTR pair
match	2	Match score
mismatch	-5	Mismatch score
open_gap	6	Gap opening penalty
continuation_gap	1	Gap continuation penalty
$Weight_{TSR}$	0.5	Weight for presence/absence of TSR
$Weight_{motif}$	0.5	Weight for presence/absence of the motif $TG \ldots CA$

4.3. *Software Availability*

We have developed a software program called *LTR_par* that implements our algorithms. The implementation is in C and can be run either serially or on multiprocessor computers and clusters with support for MPI (e.g., MPICH[15]). The software can be obtained by contacting the authors.

5. Results

5.1. *Quality Validation*

Validation of our software was performed by running the program on the entire yeast genome and comparing the results against a "benchmark" of known LTR retrotransposon locations ([24], see the URL http://www.public.iastate.edu/~voytas for more details). The list of parameters and their values used while running our *LTR_par* software is shown in Table 2.

The yeast genome has 16 chromosomes, and the benchmark has a total of 50 known full-length LTR retrotransposons. *LTR_par* predicted a total of 191 elements with different confidence levels: 49 with a confidence level of 1, 11 with a level of 0.75, 44 with a level of 0.5, 57 with a level of 0.25, and 30 with a level of 0. We extracted the 49 predictions with confidence level 1, and evaluated them against the benchmark entries as follows. Each prediction made by *LTR_par* is categorized as a "true positive" if the prediction is part of the benchmark, and a "false positive" otherwise. Those retroelements that were not part of our prediction are labeled "false negative". The results are shown in Table 3, listed by each chromosome.

All the 44 true positives accurately predicted the LTR boundaries along with their TSRs and motif locations. Of the 6 false negatives, 3 were not identified because they do not have identical TSRs; all these 3 were however accurately reported at a lower confidence level (0.5). Of the remaining three, one was not identified because of a low LTR sequence identity (69%) and another was identified with a lower confidence (0.5) because of boundary mispredictions resulting from its computed optimal alignment not matching the "biologically-preferred" LTR boundaries

Table 3. Quality validation of running *LTR_par* and *LTR_STRUC* programs on the entire yeast genome.

Chromosome	*LTR_par*			*LTR_STRUC*		
	TP	FP	FN	TP	FP	FN
1	1	0	0	1	0	0
2	3	0	0	2	0	1
3	1	0	1	1	0	1
4	7	0	1	1	2	7
5	1	1	1	1	0	1
6	1	0	0	1	0	0
7	6	3	0	5	0	1
8	2	0	0	2	0	0
9	1	0	0	1	0	0
10	2	0	0	2	0	0
11	0	0	0	0	0	0
12	5	0	1	4	0	2
13	3	1	1	3	0	1
14	3	0	0	2	0	1
15	3	0	1	3	0	1
16	5	0	0	5	0	0
Total	44	5	6	34	2	16

in the benchmark entry. *LTR_par* identified the last false negative entry although with its predicted LTR boundaries inconsistent with that of the benchmark entry; however, this approximate prediction was made with a confidence of 1.

Of the 5 false positives, 3 of them were LTRs part of other full-length retroelements but reported because they were similar and proximate along the genome. We invalidated these candidates by ensuring that there is no known reverse transcriptase coding sequences intervening the predicted $5'$ and $3'$ sequences (by running tblastx[1]). Of the remaining two false positives, one shares its $3'$ LTR with that of a true positive prediction, while the $5'$ is different. We speculate that this is a nested retrotransposon, although further investigation is required to validate this claim. The last false positive is same as the last false negative case we discussed above — the $3'$ LTR (334 bp long) matches accurately with that of the benchmark; however, instead of the $5'$ LTR (140 bp long) reported in the benchmark, our prediction has a longer $5'$ prediction that is 338 bp long, covering the benchmark's $5'$ region. The fact that there is a $5'$ LTR that matches closely in length and sequence identity with that of the $3'$ LTR (along with a pair of identical TSRs and motifs as predicted by *LTR_par*) suggests that the length discrepancy between this LTR pair in the benchmark record is probably outdated with respect to the current sequence in GenBank.

For comparison purposes, we also ran the *LTR_STRUC* program on the yeast genome and compared its results against the benchmark. The program was run in its default parameter settings (which has a similarity threshold of 75%), and at the highest level of "thoroughness" permitted by the program[30]. These results are

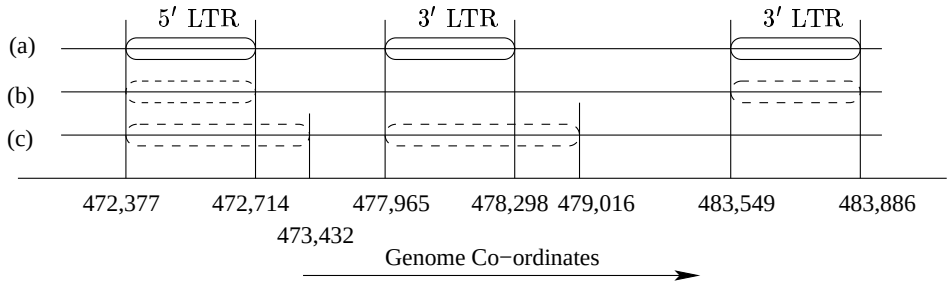

Fig. 6. A case of nested retrotransposons in chromosome 10 of *S. cerevisiae* with 3 LTRs. The bottom-most line indicates the genome (not to scale). Part (a) shows the benchmark co-ordinates for the LTRs. Parts (b) and (c) show the two predictions made by our software.

also shown in Table 3. Of the total 16 false negatives, the program misses all the three elements with unidentical TSRs (note that *LTR_par* identifies these as low confident predictions). As expected, the program also misses the retroelement with 69% LTR sequence identity. We could not ascertain the reason(s) for missing of the remaining 12 LTR pairs. It is likely that the program either failed to generate candidate pairs because of jumping by Δi characters as a means to save run-time or that an alignment that was inferior to a best alignment was computed on aligning them.

The above results show that *LTR_par* has a better sensitivity than *LTR_STRUC*, while *LTR_STRUC* has a better specificity than *LTR_par*. For a *de novo* prediction program, while it is important to keep the number of false predictions low in the interest of saving further validation efforts, it is even more important to high sensitivity in the interest of saving quality — for the simple reason that a missed prediction cannot be found through post-processing the program's output. As for the false predictions made by *LTR_par*, we observed that most of these predictions are due to generic repeats that have both high sequence identities and genomic proximities. In addition to offering a higher sensitivity, the scheme of predicting at different confidence levels provides additional flexibility in handling false predictions. For instance, in case of the above results on the yeast genome, even though a total of 191 predictions were reported by *LTR_par*, 44 of the total 50 retroelements were predicted with a confidence of 1, while a majority of these false predictions were reported at lower confidence levels. This allows a user to evaluate the predictions in the order of confidence reported.

There was also a case of "nested" retrotransposon in the benchmark data set along chromosome 10. Figure 6 shows this case, where one 5′ LTR is shared between two full-length retrotransposons. As illustrated in the figure, our software also predicted the two retrotransposons, one of which with consistent boundary and TSR predictions as well.

Besides the yeast genome, we also ran *LTR_par* on a collection of 9 rice BAC sequences, randomly selected from a larger set of rice BACs analyzed using

LTR_STRUC by McCarthy *et al*[29]. Both the programs detected 8 full-length LTR retrotransposons in common. However, *LTR_par* detected 4 predictions that were absent in the *LTR_STRUC*'s list of predictions. On the other hand, *LTR_STRUC* predicted 2 solo-LTRs (i.e., non full-length elements) which were not predicted by *LTR_par*.

5.2. *Performance Results*

Table 4. Run-time results of LTR_par on different genomes.

Organism	Genome Size (in bp)	Number of processors	Total Time (in minutes)
Saccharomyces cerevisiae	12,070,811	8	1.2
Arabidopsis thaliana	119,186,497	32	67
Drosophila melanogaster	118,357,599	32	33
Pan troglodytes	3,084,092,060	50	491

As for run-time on the yeast genome (over 12 Mbp), *LTR_STRUC* took about 210 minutes on a Windows Intel Pentium III 1 GHz machine, while *LTR_par* took 10 minutes on a single Intel Pentium III 1.1 GHz processor. While *LTR_par* spends much less time on candidate pair generation than *LTR_STRUC*, it spends most of its time in performing alignments simply because it does more work to guarantee optimality. For example, on the yeast genome, *LTR_par* spent only 8% of the time in preprocessing and generating pairs, while the remaining 92% was spent in aligning the LTR candidates. This extra effort spent in ensuring a thorough alignment is supported by a better sensitivity of our software when compared to *LTR_STRUC*, as was seen in the above validation studies. We also studied the performance of *LTR_par* on a Linux cluster of 25 nodes, each with 2 Intel Xeon 3.06 GHz processors and 2 GB RAM. The parallel run-times taken by *LTR_par* for genomes of different sizes are shown in Table 4.

In order to assess the parallel scalability of our current implementation, we ran *LTR_par* on different number of processors by keeping the input size fixed. Table 5 shows these results on the entire yeast genome (11 Mbp) and on the chromosome 3R (27 Mbp) of the Drosophila genome. As can be observed in both cases, the parallel efficiency decreases with increase in the number of processors. This is expected because the current implementation does not distribute alignment workload across processors, i.e., a candidate pair generated at a given processor is aligned at the same processor, regardless of the repetitive extent of its portion of the input. Thus the parallel bottleneck is the processor with the maximum alignment work. Asymptotically, we expect the role of this imbalance in the workload to diminish. This is corroborated by our experiments on the entire chimpanzee genome ($>$ 3 billion bp) — on a single processor *LTR_par* 161 hours, while on 50 such processors it finished in 491 minutes, implying a speedup of over 19.

Table 5. Parallel run-times (in seconds) of *LTR_par* on the yeast genome and the chromosome 3R of Drosophila.

Input	Number of processors					
	1	2	4	8	16	32
Yeast Genome	226	150	96	71	53	40
Chromosome 3R	820	598	455	294	270	255

5.3. *A Large Scale Application*

In order to validate the applicability of the software on newly sequenced genomes, we ran *LTR_par* on the entire chimpanzee (*Pan troglodytes*) genome[41]. The chimpanzee genome comprises of 23 pairs of autosomal chromosomes and 2 pairs of sex chromosomes. The sequence data downloaded from GenBank as of September 2005 has over 3 billion bp. On 50 processors of the Linux cluster described above the program took under 8.5 hours to complete on the entire genome. On the longest chromosome (∼229 Mbp long chromosome 1) the program took about 27 minutes, while the longest run-time was 58 minutes on chromosome 4 (∼209 Mbp long). Running on this genome takes a week to 10 days using LTR_STRUC[38].

As part of an ongoing research initiative, a team at Georgia Institute of Technology is working on identification of full-length LTR retrotransposons in the chimpanzee genome[38]. Recently, the team identified a set of full-length retroelements using a custom-developed framework that performs an extensive search for LTRs accompanied by other important intervening patterns such as known reverse transcriptase sequences, primer binding site, poly-purine tract etc[38]. Due to its elaborate treatment, the full-length elements detected by this procedure are expected to be highly accurate and conservative; so we used the resulting set as a benchmark for our studies and performed a case study on a randomly chosen chromosome (chromosome 12 which is ∼135 Mbp long).

The benchmark set for chromosome 12 comprised of 19 full-length elements. Under the default parameter settings in Table 2, our program predicted only 7 of the 19. When the similarity threshold (τ) was decreased to 70% and L_{max} was increased 2,000 bp, 12 of the 19 predicted correctly. Note that only 3 of these 12 were at confidence level 1. Of the remaining 7 not predicted by *LTR_par*, 5 were predicted when the similarity threshold was further reduced from 70% to 60%. The remaining two were not predicted because one has two LTRs of largely differing lengths (658 bp and 960 bp) and another has less than 50% sequence similarity between the LTRs. In addition to the benchmark hits, *LTR_par* predicted a total of 895 elements, including 38 at confidence level 1. Upon investigation of randomly chosen predictions, we found that many candidates do not contain any known reverse transcriptase sequences. However, the 38 predictions with confidence 1 show promise and need further validation.

6. Discussion

The results of validating our software are encouraging. The sensitivity on the yeast genome is better than that of the *LTR_STRUC* because our algorithm more accurately models mutation events in LTR and TSR regions. Moreover, *LTR_par* offers a good flexibility by providing the user with a better control — the user can assign weights to the presence/absence of TSRs and $TG \ldots CA$ motifs, and the software can output its predictions at different confidence levels reflecting the weights specified by the user. Also, if a user is searching a newly sequenced genome for LTR retroelements, the user can try different combinations of weights and scoring parameter values and observe changes in the predictions before deciding on an appropriate set of parameters. The speed of our software plays a critical role in facilitating multiple such experiments under different parameter settings. The software can also be used to identify nested retroelements; cases that correspond to multiple nested insertions can be detected by running our software iteratively on the genomic sequence with all the full-length elements found in previous iterations excised out.

Given that the current version of the software accounts only for the structural attributes such as TSRs and motifs, we recommend using our software for *de novo* full-length LTR retrotransposon prediction on genomes in which the two LTRs of each retrotransposon are expected to be highly conserved. The specificity of the current state of our software can be extended to incorporate other structural attributes typical of an LTR retrotransposon: The genomic region between a pair of LTR sequences typically contains special-purpose sequences such as PPT, PBS, *gag*, *pol*, and *env*, and detecting these patterns is important in confirming the biological identity of each prediction. Poly-Purine Tract can be detected by searching for a purine-rich (bases A/G) region of an approximate length of 10 bp immediately upstream of the predicted $3'$ LTR boundary. Similarly, the region immediately downstream of the predicted $5'$ LTR sequence can be searched for presence of Primer Binding Site. PBS is usually a complement of a known tRNA $3'$ terminal sequence — a pattern that can be input by the user. The genes in the *gag* and *env* regions can be detected by looking for sequences that encode retroviral capsid and envelope proteins respectively. The genes in the *pol* region can be searched for sequences that encode for protease, integrase and reverse transcriptase enzymes.

7. Conclusion

In this paper, we provided efficient algorithms and software towards detection of full-length LTR transposons. One salient feature of our method is a quick and efficient algorithm for generating candidate LTR pairs, which facilitates use of rigorous methods for aligning the candidates in order to ensure high quality LTR predictions. The software has been designed with the intent of giving a high degree of flexibility to the user. The various planned functional improvements to the software, such as incorporation of detection strategies for PPT, PBS, *gag, pol* genes in the structure finding procedure, should strengthen the specificity of the software. Due to

the ubiquity of LTR retroelements in complex eukaryotic genomes, we believe that the utility of such a highly accurate and yet fast and scalable LTR retrotransposon discovery tool is important for the advancement of biological understanding of these genomes.

8. Acknowledgments

We are thankful to the reviewers for useful comments both on improving the quality of our software and on the readability of this document. We are grateful to Xiaowu Gai for introducing us to the retrotransposon identification problem and for providing some useful biological insights during the validation of our software, and for sharing some preliminary results of running *LTR_STRUC* on the yeast and *Arabidopsis* genomes. We thank Nalini Polavarapu and other members of the Georgia Institute of Technology retrotransposon team for sharing with us their preliminary results on the chimpanzee genome and for subsequently helping us validate our results. Many thanks to the reviewers of an earlier conference publication for their useful comments that helped us identify and plan further quality improvements. We thank Daniel Voytas and Scott Emrich for comments on earlier drafts of our paper. We also thank the *LTR_STRUC* team for providing the tool. A. Kalyanaraman was supported by an IBM Ph.D. Research Fellowship. This research was partially supported by CREST grant HRD-0420407.

References

1. S.F. Altschul and W. Gish and W. Miller *et al.* Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
2. Z. Bao and S.R. Eddy. Automated *de novo* identification of repeat sequence families in sequenced genomes. *Genome Research*, 12(8):1269–1279, 2002.
3. J.L. Bennetzen. The contributions of retroelements to plant genome organization, function and evolution. *Trends in Microbiology*, 4(9):347–353, 1996.
4. F.D. Bushman. Targeting survival: integration site selection by retroviruses and LTR-retrotransposons. *Cell*, 115(2):135–138, 2003.
5. B. Charlesworth, P. Sniegowski, and W. Stephan. The evolutionary dynamics of repetitive DNA in eukaryotes. *Nature*, 371:215–220, 1994.
6. J.M. Coffin, S.H. Hughes, and H.E. Varmus. Retroviruses. Cold Spring Harbor Laboratory Press, Plainview, New York, 1997.
7. E.S. Lander *et al.* Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.
8. C. Feschotte, N. Jiang, and S.R. Wessler. Plant transposable elements: Where genetics meets genomics. *Nature Reviews (Genetics)*, 3:329–341, 2002.
9. R.B. Flavell. Repetitive DNA and chromosome evolution in plants. *Philosophical Transactions of the Royal Society of London. B.*, 312(1154):227–242, 1986.
10. L.F. Franchini, E.W. Ganko, and J.F. McDonald. Retrotransposon-gene associations are wide-spread among *D.melanogaster* populations. *Molecular Biology and Evolution*, 21(7):1323–1331, 2004.
11. X. Gai. *Personal Communication*, 2005.
12. E.W. Ganko, V. Bhattacharjee, P. Schliekelman, and J.F. McDonald. Evidence for the

contribution of LTR retrotransposons to *C. elegans* gene evolution. *Molecular Biology and Genetics*, 20:1925–1931, 2003.
13. O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.
14. M. Grandbastien, A. Spielmann, and M. Caboche. Tnt1, a mobile retroviral-like transposable element of tobacco isolated by plant cell genetics. *Nature*, 337:376–380, 1989.
15. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22:789–828, 1996.
16. D. Gusfield. *Algorithms on strings, trees and sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, London, 1997.
17. H. Hirochika, K. Sugimoto, Y. Otsuki, H. Tsugawa, and M. Kanda. Retrotransposons of rice involved in mutations induced by tissue culture. *Proceedings of the National Academy of Sciences USA*, 93(15):7783–7788, 1996.
18. M.A. Johns, J. Mottinger, and M. Freeling. A low copy number, copia-like transposon in maize. *The EMBO Journal*, 4(5):1093-1101, 1985.
19. I.K. Jordan and J.F. McDonald. Comparative genomics and evolutionary dynamics of *Saccharomyces cerevisiae* Ty elements. *Genetica*, 107(1-3):3–13, 1999.
20. J. Kärkkäinen and P. Sander. Simple linear work suffix array construction. In *Proc. International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, 2719:943–955, 2003.
21. T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. Combinatorial Pattern Matching*, Lecture Notes in Computer Science, 2089:181–192, 2001.
22. M.G. Kidwell and D. Lisch. Transposable elements as sources of variation in animals and plants. *Proceedings of the National Academy of Sciences USA*, 94(15):7704–7711, 1997.
23. D.K. Kim, J.S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. In *Proc. Combinatorial Pattern Matching*, Lecture Notes in Computer Science, 2676:186–199, 2003.
24. J.M. Kim, S. Vanguri, J.D. Boeke, A. Gabriel, and D.F. Voytas. Transposable Elements and Genome Organization: A Comprehensive Survey of Retrotransposons Revealed by the Complete *Saccharomyces cerevisiae* Genome Sequence. *Genome Research*, 8(5):464–478, 1998.
25. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In *Proc. Combinatorial Pattern Matching*, Lecture Notes in Computer Science, 2676:200–210, 2003.
26. S. Kurtz, J.V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Research*, 29(22):4633–4642, 2001.
27. S. Kurtz and C. Schleiermacher. REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5):426–427, 1999.
28. U. Manber and G. Myers. Suffix arrays: A new method for on-line search. *SIAM Journal of Computing*, 22:935–948, 1993.
29. E.M. McCarthy, L. Liu, G. Lizhi, and J.F. McDonald. Long terminal repeat retrotransposons of *Oryza sativa*. *Genome Biology*, 3(10):0053.1–0053.11, 2002.
30. E.M. McCarthy and J.F. McDonald. LTR_STRUC: a novel search and identification program for LTR retrotransposons. *Bioinformatics*, 19(3):362–367, 2003.
31. E.M. McCarthy and J.F. McDonald. LTR Retrotransposons of *Mus musculus*. *Genome Biology*, 5:R14, 2004.

32. B.C. Meyers, S.V. Tingey, and M. Morgante. Abundance, distribution, and transcriptional activity of repetitive elements in the maize genome. *Proceedings of the National Academy of Sciences USA*, 11(10):1660–1676, 2001.
33. J.T. Miller, F. Dong, S.A. Jackson, J. Song, and J. Jiang. Retrotransposon-related DNA sequences in the centromeres of grass chromosomes. *Genetics*, 150:1615–1623, 1998.
34. M. Morgante, A. Policriti, N. Vitacolonna, and A. Zuccolo. Automated search for LTR retrotransposons. *http://citeseer.ist.psu.edu/644336.html*, 2002.
35. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.
36. B.D. Peterson-Burch, D. Nettleton, and D.F. Voytas. Genomic Neighborhoods for *Arabidopsis* retrotransposons: a role for targeted integration in the distribution of the Metaviridae. *Genome Biology*, 5:R78, 2004.
37. D.E.L. Promislow, I.K. Jordan, and J.F. McDonald. Genomic demography: a life-history analysis of transposable element evolution. *Proc. Royal Society of London B: Biological Sciences*, 266(1428):1555-1560, 1999.
38. N. Polavarapu. *Personal Communication*, 2005.
39. P. SanMiguel, B.S. Gaut, A. Tikhonov, Y. Nakajima, and J.L. Bennetzen. The paleontology of intergene retrotransposons of maize. *Nature Genetics*, 20:43–45, 1998.
40. P. SanMiguel, A. Tikhonov, Y.K. Jin, N. Motchoulskaia, D. Zakharov, A. Melake-Berhan, P.S. Springer, K.J. Edwards, M. Lee, Z. Avramova, and J.L. Bennetzen. Nested retrotransposons in the intergenic regions of the maize genome. *Science*, 274(5288):765–768, 1996.
41. The Chimpanzee Sequencing and Analysis Consortium. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437:69–87, 2005.
42. A.F.A. Smit and P. Green. RepeatMasker. *http://ftp.genome.washington.edu/RM/RepeatMasker.html*, 1999.
43. M.J. Varagona, M. Purugganan, and S.R. Wessler. Alternative splicing induced by insertion of retrotransposons into the maize *waxy* gene. *Plant Cell*, 4:811–820, 1992.
44. C.M. Vicient, A. Suoniemi, K. Anamthawat-Jnsson, J. Tanskanen, A. Beharav, E. Nevo, and A.H. Schulman. Retrotransposon BARE-1 and Its Role in Genome Evolution in the Genus Hordeum. *Plant Cell*, 11:1769–1784, 1999.
45. S.R. Wessler, T.E. Bureau, and S.E. White. LTR-retrotransposons and MITEs: important players in the evolution of plant genomes. *Current Opinion In Genetics and Development*, 5:814–821, 1995.
46. S.E. White, L.F. Habera, and S.R. Wessler. Retrotransposons in the Flanking Region of Normal Plant Genes: A Role for Copia-Like Elements in the Evolution of Gene Structure and Expression. *Proceedings of the National Academy of Sciences USA*, 91:11792–11796, 1994.
47. Y. Xiong and T.H. Eickbush. Origin and evolution of retroelements based upon their reverse transcriptase sequences. *EMBO Journal*, 9:3353–3362, 1990.