Efficiently Enumerating Substrings with Statistically Significant Frequencies of Locally Optimal Occurrences in Gigantic String

Atsuyoshi Nakamura,[1] **Ichigaku Takigawa,**[2,3] **Hiroshi Mamitsuka**[4,5]
[1]Graduate School of Information Science and Technology, Hokkaido University, Japan
[2]RIKEN Center for Advanced Intelligence Project. Japan
[3]Institute for Chemical Reaction Design and Discovery, Hokkaido University, Japan
[4]Bioinformatics Center, Institute for Chemical Research, Kyoto University, Japan
[5]Department of Computer Science, Aalto University, Finland
atsu@ist.hokudai.ac.jp, ichigaku.takigawa@riken.jp, mami@kuicr.kyoto-u.ac.jp

Abstract

We propose new frequent substring pattern mining which can enumerate all substrings with statistically significant frequencies of their locally optimal occurrences from a given single sequence. Our target application is genome sequences, around a half being said to be covered by interspersed and consecutive (tandem) repeats, and detecting these repeats is an important task in molecular life sciences. We evaluate the statistical significance of frequent substrings by using a string generation model with a memoryless stationary information source. We combine this idea with an existing algorithm, ESFLOO-0G.C (Nakamura et al. 2016), to enumerate all statistically significant substrings with locally optimal occurrences. We further develop a parallelized version of our algorithm. Experimental results using synthetic datasets showed the proposed algorithm achieved far higher F-measure in extracting substrings (with various lengths and frequencies) embedded in a randomly generated string with noise, than conventional algorithms. The large-scale experiment using the whole human genome sequence with 3,095,677,412 bases (letters) showed that our parallel algorithm covers 75% of the whole positions analyzed, around 4% and 24% higher than the recent report and the current cutting-edge knowledge, implying a biologically unique finding.

1 Introduction

Genome sequences are known to be strings that contain many consecutive (tandem) and interspersed repeats, and the percentage of these repeats reaches 30-50% in mammalian genomes (Faulkner, Kimura, and et al. 2009). These interspersed repeats are created by transposition of the so-called *retrotransposons*, where typical examples are SINEs (short interspersed nuclear elements), LINEs (long interspersed nuclear elements) and LTRs (long terminal repeats). More importantly retrotransposons affect genomes directly, by which retrotransposons are involved with a lot of genome related biological activities, such as promoting genome evolution, supporting genome structure and so on (Faulkner, Kimura, and et al. 2009). A lot of efforts are being made to identify repeats in genomes, and the number of repetitive substrings in the most well-recognized database, Repbase (Bao, Kojima, and Kohany 2015), is steadily increasing.

Currently we can easily access genome sequences of various species. A widely conducted approach of finding repeats out of these sequences is the so-called *de novo* discovery, which finds repeats in genome sequences without using prior information about structures or similarity of the known repeats (Bergman and Quesneville 2007). Most major *de novo* approaches can be classified into three types: 1) clustering after pairwise alignment (Bao and Eddy 2002; Quesneville, Nouaud, and Anxolabéhère 2003; Edgar and Myers 2005; Kurtz et al. 2001; Volfovsky, Haas, and Salzberg 2001), 2) frequent k-mer extension (Li et al. 2005; Price, Jones, and Pevzner 2005; Gu et al. 2008) and 3) clustering after the second approach (Ghodsi, Liu, and Pop 2011).

Frequent pattern mining has been well investigated and matured in the past 30 years, and frequent subsequence mining (Agrawal and Srikant 1995) would be a possible approach for genome sequences. There are two types of sequence patterns: *subsequences* and *substrings* (contiguous subsequences). In general a *subsequence* is, given multiple sequences, evaluated by the number (called *support*) of sequences (occurrences) with this pattern, while a *substring* is, given a single sequence, by the number of occurrences of this substring in this sequence. The former idea of subsequences is more standard in data mining, while the latter idea of substrings would be more reasonable for finding repetitions in genomes. In fact there are studies on frequent substring mining to find interspersed repeats as occurrences of patterns in a given string (Zhu et al. 2007; Nakamura et al. 2016). Frequent pattern mining is to enumerate all patterns that satisfy a prefixed minimum number of supports which is called *min_sup*. In other words, each obtained pattern has a reason of why being a pattern, making frequent pattern mining more advantageous than finding patterns by heuristics or clustering. However, there are two problems when applying frequent pattern mining to huge genome sequences: 1) determining the min_sup, because a low min_sup causes many garbage small patterns and a high min_sup might lose important large patterns with low supports. 2) avoiding heavy computation of enumerating patterns with low supports.

The first problem can be addressed by enumerating *statistically significant* patterns (Terada et al. 2013). Statistical significance in mining frequent *subsequences* can be considered under an independence (memoryless) model (Low-Kam et al. 2013), which can compute the expected number of sequences with a candidate pattern in given multiple sequences and then the statistical significance of the real appearances of the pattern is tested by multiple hypothesis testing as to if a given family-wise error rate is achieved. This approach is for subsequences and not for substrings, and cannot be directly applied to mining frequent substrings in a given single string. However statistical significance under the memoryless model has been considered for genome sequence already (Ewens and Grant 2005) though not necessarily for mining patterns. Recently a method for enumerating the longest substring pairs appearing many times in a genome is proposed (Jelovic et al. 2018), checking the statistical significance under the assumption of randomly generated sequences. This method considers exactly the same strings, resulting in only short strings captured as statistically significant repeats.

We propose novel frequent substring pattern mining which, given a string, enumerates all substrings that occur in statistically significant frequencies under a string generation model with a memoryless stationary information source. Into this new framework of pattern mining, we incorporate the idea of frequent locally optimal occurrences (Nakamura et al. 2016), where pattern occurrences need local optimality in alignment with the pattern. This combination would be appropriate for considering the occurrences of interspersed repeats. For computational efficiency, we consider the simplest 0-gaped settings, in which occurrences must have a pattern with the same length. We say that, given p-value threshold α, pattern q with length m has *statistically significant frequency*, if the p-value on frequency $f(q)$ of locally optimal occurrences of pattern q is at most α, i.e. $f(q) \geq \sigma_q$ for $\sigma_q = \min\{\sigma \mid \mathbb{P}\{f(q) \geq \sigma\} \leq \alpha\}$, where the probability is calculated using the assumed stochastic model of string generation. We use the lower bound $\bar{\sigma}[m] = \min_{q:|q|=m} \sigma_q$ instead of σ_q for all length-m patterns q for computational efficiency. Frequency threshold $\bar{\sigma}[m]$ for length-m patterns in a length-n string is approximated by $\min\{\sigma \in \mathbb{N} \mid \mathbb{P}\{X \geq \sigma \mid X \geq 1\} \leq \alpha\}$ for Poisson random variable $X \sim \mathrm{Po}(\lambda)$ with $\lambda = (n-m+1)W_*(m)$, where $W_*(m)$ is the maximum value among occurrence probabilities of all length-m patterns in a random length-m string. Probability $W_*(m)$ cannot be computed trivially, because many different strings can be locally optimal occurrences of a length-m pattern and $W_*(m)$ is the sum of those probabilities. We show an efficient algorithm of calculating $W_*(m)$, based on dynamic programming. We develop an efficient algorithm, E3SFLOO (Enumerate Substring patterns with Statistically Significant Frequencies of Locally Optimal Occurrences) incorporating the above calculation of $\bar{\sigma}[m]$ $(m = 1, \ldots, n)$ into ESFLOO-0G.C (Nakamura et al. 2016) and further develop a parallelized version of E3SFLOO.

Effectiveness of E3SFLOO was demonstrated by using synthetic data of random strings in which substrings with various lengths, frequencies and noise are embedded. Without noise, E3SFLOO could extract all embedded substrings (F-measure: 0.997), which was hard for ESFLOO-0G.C with any min_sup (F-measure: $\leq$0.688). In particular, E3SFLOO was robust, regarding approximate matching, achieving F-measure of 0.876 for strings, containing substrings with statistically significant frequency and noise of 4%. Also E3SFLOO ran fast enough for huge strings, for example, spending only 19.4 seconds for a string with 10,000,000 letters and 9,996 embedded substrings (75 letters at the longest). We found that parallelization (64 threads) made 36.34 times faster than that with only the main thread.

We applied E3SFLOO to the whole human genome sequence (around 3.1 billion letters) and examined the occurrences of the obtained patterns after postprocessing. The result shows that the occurrences covered 75% of the whole sequences, around 4% larger than the recent report by using P-clouds (de Koning et al. 2011). We further compared our occurrences with the positions annotated by RepeatMasker (Smit, Hubley, and Green 2017) basically as substrings similar to those which are already registered in Repbase (Bao, Kojima, and Kohany 2015), or annotated by Tandem Repeats Finder (TRF) (Benson 1999). These positions covered only around 51% of the entire genome, 24% lower than our coverage rate, and 87.5% of these positions were covered by our occurrences. Finally detected repeats with the length of $> 2,000$ showed the discovery of unknown repeats, some being distributed over different chromosomes. Overall these results imply that E3SFLOO can be a promising tool to detect unknown repeats in genome sequences, which might be covered by various types of repeats far more than those already found.

2 Problem Setting

Let Σ be a finite *alphabet* with *letters* as elements. For arbitrary two letters $x, y \in \Sigma$, real-valued function $w(x, y)$, called a *score function*, is defined to satisfy the following three conditions:

SF1 $w(x, y) = w(y, x)$ for all $x, y \in \Sigma$

SF2 $w(x, x) > 0$ for all $x \in \Sigma$

SF3 $w(x, y) < 0$ for some $x, y \in \Sigma$

Positive $w(x, y)$ implies that two letters x and y are similar. Sequence $s = s[1] \cdots s[n]$ that is composed of letters $s[1], \ldots, s[n] \in \Sigma$, is a *string* and n is the *length* of string s. For two strings $s = s[1] \cdots s[n]$, $t = t[1] \cdots t[n]$ with the same length of n, *similarity score* S is defined as $S(s, t) = \sum_{i=1}^{n} w(s[i], t[i])$.

For $1 \leq i \leq j \leq n$, consecutive part $s[i] \cdots s[j]$ of string $s = s[1] \cdots s[n]$ is called a *substring* of s and denoted as $s[i..j]$. Note that $s[i+1..i]$ is also used to denote a *null string*.

We consider approximate pattern $p[1..m]$ that frequently occurs in string $s[1..n]$. As occurrences of pattern $p[1..m]$ in string $s[1..n]$, we consider *minimal locally optimal occurrences* defined as follows.

Definition 1 (Minimal locally optimal occurrences[1]) *Substrings $s[i..i+m-1]$ of $s[1..n]$ that satisfy the following two conditions are called* minimal locally optimal occurrences *of $p[1..m]$.*

CS1 $S(p[1..h], s[i..i+h-1]) > 0$ *for all* $0 < h \leq m$
CS2 $S(p[1..m], s[i..i+m-1]) > S(p[1..h], s[i..i+h-1])$ *for all* $0 \leq h < m$

Example 1 *For* $s = s[1..12] = abcaacbabbba$, $p[1..5] = abcba$ *and* $w(x, y)$ *that is* 1 *for* $x = y$ *and* -1 *for* $x \neq y$, $s[8..12]$ *is a minimal locally optimal occurrence of* p *but* $s[1..5]$ *and* $s[4..8]$ *do not satisfy CS2 and CS1, respectively.*

The following problem is the frequent pattern mining problem considered in (Nakamura et al. 2016).

Problem 1 *Given string* s, *score function* w *and natural number* σ, *enumerate substrings* $s[i..j]$ *of* s *for which minimal locally optimal occurrences appear in* s *at least* σ *times.*

In Problem 1, frequent patterns are defined by fixed threshold σ. Since short string patterns occur more frequently than long string patterns, statistically significant long patterns might be missed by using a large threshold while a lot of garbage short patterns might be extracted by using a small threshold.

To overcome this problem, we propose new frequent pattern mining, in which we give[2] p-value threshold α, instead of fixed threshold σ, assuming that a given sequence is generated by a memoryless stationary information source. Consider memoryless stationary information source $S_0(P_0)$ that generates letter x in Σ with probability $P_0(x)$. Let s' be a length-n random sequence generated from $S_0(P_0)$. Let $\sigma(n, p, P_0, \alpha)$ be the minimum natural number σ such that minimal locally optimal occurrences of pattern p appear in s' at least σ times with the probability of at most α under the condition that p appears in s' at least once[3]. Then, our frequent pattern mining can be described as follows.

Problem 2 *Given string* $s[1..n]$, *score function* w, *memoryless information source* $S_0(P_0)$ *and positive real value* $\alpha \in (0, 1)$, *enumerate substrings* $s[i..j]$ *of* s, *for which minimal locally optimal occurrences appear in* s *at least* $\sigma(n, s[i..j], P_0, \alpha)$ *times.*

3 Computing $\sigma(n, p, P_0, \alpha)$

3.1 Approximation by Poisson Distribution

Since a naive exact computation of $\sigma(n, p, P_0, \alpha)$ would be very costly, we propose an approximate way of computing σ. Let $s'[1..n]$ be a random sequence generated by a given memoryless stationary information source $S_0(P_0)$. Then, $s'[1..m], s'[2..m+1], \ldots, s'[n-m+1..n]$ can be seen as a state transition sequence of a Markov information source with Σ^m states. Note that given pattern $p[1..m]$ is one of the Σ^m states. Let $\{w_q \mid q \in \Sigma^m\}$ be the stationary distribution over the state space Σ^m. Let $Q(p)$ be the subset of Σ^m so that every member $t[1..m]$ of $Q(p)$ is a minimal locally optimal occurrence of $p[1..m]$, and let W_p denote $\sum_{q \in Q(p)} w_q$. Let X be the number of member state occurrences in $Q(p)$ in s'. Then, $\sigma(n, p, P_0, \alpha)$ is expressed as

$$\sigma(n, p, P_0, \alpha) = \min\{\sigma \in \mathbb{N} \mid \mathbb{P}\{X \geq \sigma \mid X \geq 1\} \leq \alpha\}.$$

We approximate[4] the distribution of X by Poisson distribution $\mathrm{Po}((n-m+1)W_p)$. Let $X' \sim \mathrm{Po}(\lambda)$, where $\lambda = (n-m+1)W_p$. Then,

$$\begin{aligned}\mathbb{P}\{X \geq \sigma \mid X \geq 1\} \approx & \mathbb{P}\{X' \geq \sigma \mid X' \geq 1\} \\ = & \frac{1}{1-\mathrm{e}^{-\lambda}}\left(1 - \mathrm{e}^{-\lambda}\sum_{i=0}^{\sigma-1}\frac{\lambda^i}{i!}\right)\end{aligned}$$

holds. Thus the following $\tilde{\sigma}(n, p, P_0, \alpha)$ can be regarded as an approximate value of $\sigma(n, p, P_0, \alpha)$:

$$\begin{aligned}&\tilde{\sigma}(n, p, P_0, \alpha) \\ &= \min\left\{\sigma \in \mathbb{N} \,\middle|\, \frac{1}{1-\mathrm{e}^{-\lambda}}\left(1 - \mathrm{e}^{-\lambda}\sum_{i=0}^{\sigma-1}\frac{\lambda^i}{i!}\right) \leq \alpha\right\} \\ &\quad \text{where } \lambda = (n-m+1)W_p\end{aligned}$$

To avoid computing $\tilde{\sigma}(n, p, P_0, \alpha)$ for each pattern p, we use the maximum value $\bar{\sigma}(n, m, P_0, \alpha)$ of $\tilde{\sigma}(n, p, P_0, \alpha)$ over all patterns p with the length of m that can be calculated using $W_*(m) = \max_{p \in \Sigma^m} W_p$:

$$\bar{\sigma}(n, m, P_0, \alpha) = \tilde{\sigma}(n, \arg\max_{p \in \Sigma^m} W_p, P_0, \alpha) \qquad (1)$$

3.2 Computing $W_*(m)$

Consider the following simple score function w:

$$w(x, y) = \begin{cases} a & (x = y) \\ -b & (x \neq y), \end{cases} \qquad (2)$$

where a and b are positive natural numbers with $b > a > 0$. Then similarity score $S(p, t)$ is determined only by the number of unmatched positions. Also only unmatched positions determines if t is a minimal locally optimal occurrence of p. Let $F(m)$ denote the family of unmatched position sets of minimal locally optimal occurrences of any $p[1..m]$, i.e. $F(m) = \{\{i \mid t[i] \neq p[i]\} \mid t \in Q(p)\}$. Then, $F(m)$ does not depend on p and is fixed for any p of length m. Thus by using the above score function w, W_p for pattern $p \in \Sigma^m$ can be computed as $W_p = \sum_{t \in Q(p)} \prod_{i=1}^{m} P_0(t[i]) = \sum_{S \in F(m)} \prod_{i \notin S} P_0(p[i]) \prod_{i \in S}(1 - P_0(p[i]))$. Let $c_* = \arg\max_{x \in \Sigma} P_0(x)$. Then the following relation holds between $W_*(m)$ and c_*.

Theorem 1 *Let* $p_*[1..m] = c_* \cdots c_*$. *Then, for score function* w *defined as (2),*

$$W_*(m) = W_{p_*} = \sum_{k=0}^{\lfloor \frac{a}{a+b} m \rfloor} \begin{array}{l} |\{S \in F(m) \mid |S| = k\}| \\ \quad \times P_0(c_*)^{m-k}(1 - P_0(c_*))^k. \end{array}$$

[1] In (Nakamura et al. 2016), occurrences that satisfies this condition are called *minimal locally optimal* 0*-gap occurrences*.

[2] p-value threshold α corresponds to family-wise error rate $\alpha n(n+1)/2$.

[3] We need this condition because patterns are restricted to substrings of a given string.

[4] The distribution of X can be approximated by binomial distribution $B(n-m+1, W_p)$ for $n \gg m$ due to the ergodicity of memoryless stationary source. Poison approximation is adequate for $p[1..m]$ with small $(n-m+1)W_p$ and rare overlapping occurrences. For nonsmall $(n-m+1)W_p$, a normal distribution can approximate the distribution of X, and for nonrare overlapping occurrences, *compound Poisson distribution* can be used as an approximation under $Q(p) = \{p\}$ (Ewens and Grant 2005).

(Proof) The second equality is easily derived as

$$W_{p_*} = \sum_{S\in F(m)} \prod_{i\notin S} P_0(c_*) \prod_{i\in S}(1-P_0(c_*))$$
$$= \sum_{k=0}^{\lfloor \frac{a}{a+b}m \rfloor} |\{S \in F(m) \mid |S| = k\}| \times P_0(c_*)^{m-k}(1-P_0(c_*))^k.$$

The first inequality $W_p \le W_{p_*}$ for any $p \in \Sigma^m$, that is,

$$\sum_{S\in F(m)} \prod_{i\notin S} P_0(p[i]) \prod_{i\in S}(1-P_0(p[i]))$$
$$\le \sum_{S\in F(m)} \prod_{i\notin S} P_0(c_*) \prod_{i\in S}(1-P_0(c_*)).$$

can be proved by induction on $|\{i \mid P_0(p[i]) < P_0(c_*)\}|$. □

We now describe a way to compute $|\{S \in F(m) \mid |S| = k\}|$ efficiently for any m and k, as follows. Let $S \in F(m)$ with $|S| = k$. Consider a one-dimensional walk along a number line that starts from 0 at time 0 and goes right by a if $i+1 \notin S$ and left by b if $i+1 \in S$ at time $i+1$ from the position at time i. The walks for $S \in F(m)$ with $|S| = k$ reaches $(m-k)a - kb = ma - (a+b)k$ at time m without reaching either $(-\infty, 0]$ or $[ma-(a+b)k, \infty)$ at time $1 < i < m$. The number of such walks is just $|\{S \in F(m) \mid |S| = k\}|$ and can be computed by dynamic programming as follows.

For h and i ($h \ge i \ge 0$), where difference $h-i$ is a multiple of a, let $\gamma_{h,i,k}$ denote the number of different walks that start from i at time 0 and reach $h-(a+b)k$ at time $(h-i)/a$ without reaching either $(-\infty, 0]$ or $[h-(a+b)k, \infty)$ at time $1 < i < (h-i)/a$. Then,

$$\gamma_{ma,0,k} = |\{S \in F(m) \mid |S| = k\}|$$

holds. As for $\gamma_{h,i,k}$, the following theorem holds.

Theorem 2 *The following recursive formulas hold for non-negative integers h, i, k satisfying that $h \ge i$ and difference $h-i$ is a multiple of a.*

$$\gamma_{h,i,k} = \begin{cases} 1 & (k = 0) \\ \gamma_{h,i+a,k} & (i \le b) \\ \gamma_{h,i+a,k} + \gamma_{h-(a+b),i-b,k-1} & \\ \qquad (b+1 \le i \le h-(a+b)k-a-1) & \\ \gamma_{h-(a+b),i-b,k-1} & (i \ge h-(a+b)k-a) \end{cases}$$

(Proof) When $k = 0$, only right moves must be taken to reach h, and the number of different walks is one. Assume $k \ge 1$. When $i \le b$, the walk reaches $(-\infty, 0]$ if the next move is left. Thus the number of different walks is equal to $\gamma_{h,i+a,k}$. When $i \ge h-(a+b)k-a$, if the next move is right, then the walk reaches $[h-(a+b)k, \infty)$. Thus the number of different walks is equal to $\gamma_{h-(a+b),i-b,k-1}$. Otherwise, that is, when $b+1 \le i \le h-(a+b)k-a-1$, the next move can be either right or left. Thus the number of different walks is $\gamma_{h,i+a,k} + \gamma_{h-(a+b),i-b,k-1}$. □

Given a natural number M, Algorithm 1 shows a pseudocode of an algorithm for computing $\gamma_{ma,0,k}$ for $1 \le m \le M$ and $0 \le k \le \frac{ma}{a+b}$. This algorithm is an $O(M^3)$ time algorithm.

Algorithm 1 Computing $\gamma_{ma,0,k}$

Require: M: maximum pattern length,
a, b: parameters of score function (2)
Ensure: $\gamma_{ma,0,k}$ for $1 \le m \le M$ and $0 \le k \le \frac{ma}{a+b}$.
1: $\gamma_{h,i,0} \leftarrow 1$ for all $1 \le h \le Ma$ and $0 \le i \le h$ where difference $h-i$ is a multiple of a.
2: Output $\gamma_{ma,0,0}$ for $1 \le m \le M$.
3: **for** $k = 1$ to $\frac{Ma}{a+b}$ **do**
4: $\gamma_{h,i,k} \leftarrow \gamma_{h-(a+b),i-b,k-1}$ for $(a+b)k + a \le h \le Ma$ and $i = h-(a+b)k-a$.
5: **for** $h = Ma$ to $(a+b)(k+1)+2$ **step** -1 **do**
6: {The same variable can be used for $\gamma_{h,i,k}$ and $\gamma_{h,i,k-1}$ by processing in the decreasing order of h.}
7: **for** $i = h-(a+b)k-a-1$ to $b+1$ **step** -1 **do**
8: $\gamma_{h,i,k} \leftarrow \gamma_{h,i+a,k} + \gamma_{h-(a+b),i-b,k-1}$.
9: **end for**
10: **end for**
11: **for** $i = b$ to 0 **step** -1 **do**
12: $\gamma_{h,i,k} \leftarrow \gamma_{h,i+a,k}$ for $i + a \le h \le Ma$.
13: **end for**
14: Output $\gamma_{ma,0,k}$ for $1 \le m \le M$.
15: **end for**

Using $\gamma_{ma,0,k}$ ($k = 0, \ldots, \lfloor \frac{a}{a+b}m \rfloor$) computed by Algorithm 1, $W_*(m)$ can be computed as

$$W_*(m) = \sum_{k=0}^{\lfloor \frac{a}{a+b}m \rfloor} \gamma_{ma,0,k} P_0(c_*)^{m-k}(1-P_0(c_*))^k. \quad (3)$$

4 Algorithm

4.1 Algorithm E3SFLOO

We propose an algorithm, which we call E3SFLOO (Enumerate Substring patterns with Statistically Significant Frequencies of Locally Optimal Occurrences), developed from ESFLOO-0G.C (Nakamura et al. 2016) for Problem 1, as an approximation algorithm (for Problem 2) of using $\bar{\sigma}(n, m, P_0, \alpha)$ instead of $\sigma(n, s[i..i+m-1], P_0, \alpha)$ for $i = 1, \ldots, n-m+1$.

Algorithm 2 shows a pseudocode of E3SFLOO. Given string $s[1..n]$ and maximum p-value (for frequent patterns), E3SFLOO first checks the occurrence rate of each letter c in $s[1..n]$ and set it to $P_0(c)$ (Line 1). Using n, α and P_0, E3SFLOO computes $\bar{\sigma}(n, m, P_0, \alpha)$ for length-m patterns in the increasing order of m until $m = m_0$ with $\bar{\sigma}(n, m_0, P_0, \alpha) < 3$ and $\bar{\sigma}(n, m, P_0, \alpha)$ is stored as $\bar{\sigma}[m]$ (Lines 2-5). Note that $\bar{\sigma}[m]$ is set to 2 for $m \ge m_0$. The rest part of the algorithm are the same as ESFLOO-0G.C (Nakamura et al. 2016) except using $\bar{\sigma}[m]$ for length-m patterns, instead of a simple cut-off against frequencies (min_sup).

E3SFLOO is an $O(n^3)$-time and $O(n^2)$-space algorithm (Nakamura et al. 2016) except computing $\bar{\sigma}[m]$ ($m = 1, \ldots, n$), which is $O(M^3)$ for $M = \arg\min\{m \mid \bar{\sigma}(n, m, P_0, \alpha) < 3\}$ for given $s[1..n]$ and $0 < \alpha < 1$.

Algorithm 2 E3SFLOO(s, α)

Require: $s[1..n]$: string, α: maximum p-value

1: $P_0(c) \leftarrow |\{i \mid s[i] = c\}|/n$ for $c \in \{s[i] \mid i = 1, \ldots, n\}$.
2: **for** $m = 1$ to n **do**
3: Set $\bar{\sigma}[m]$ to $\bar{\sigma}(n, m, P_0, \alpha)$ computed by Eq. (1) and (3).
4: **if** $\bar{\sigma}[m] < 3$ **then** $\bar{\sigma}[i] \leftarrow 2$ for $i = m+1, \ldots, n$ and **break**
5: **end for**
6: Run ESFLOO-0G.C(s) that is modified so as to use min. sup. $\bar{\sigma}[m]$ for len-m pats ($m = 1, \ldots, n$).

4.2 Parallelization of E3SFLOO

The most time consuming part of E3SFLOO is subroutine PatGen in ESFLOO-0G.C, in which every substring $p[1..m]$ of given string s is generated by traversing the suffix tree of s and checked whether its frequency of minimal locally optimal occurrences is at least $\bar{\sigma}[m]$. PatGen can be parallelized by traversing the suffix tree in parallel, i.e. parallelly processing different child nodes at the same time. Each process parallelized in this way needs no communication with each other, and all processes can be executed independently.

Practically the above parallelization was implemented by using multi-thread programming. In parallelization of PatGen, the task for each child node of the current node in the suffix tree is separated as an independent subroutine and assigned to a waiting thread if there is, and otherwise continuously processed in the same thread.

5 Experiments Using Synthetic Datasets

5.1 Datasets

For each pair of $\alpha = 0, 1, 2, 4, 8$ and $\beta = 1, 2$, we generated dataset $D_\beta(\alpha)$ that has 10 strings, each with the length of 10^7. Given a set of pairs of length and frequency $\{(\ell_1, f_1), \ldots, (\ell_k, f_k)\}$, each string in $D_\beta(\alpha)$ is generated as follows: 1) we randomly generated string s by simulated quaternary memoryless information source with the occurrence probability of $1/4$ for each letter, 2) for each $i = 1, \ldots, k$, we generated f_i occurrences of a length-ℓ_i pattern by copying $s[10^3 j..10^3 j + \ell_i - 1]$ to $s[10^3(j + a)..10^3(j + a) + \ell_i - 1]$ ($a = 1, \ldots, f_i - 1$) for some natural number j so as not to overlap the occurrence positions. We repeated this procedure $\lfloor 10^4 / \sum_{i=1}^{k} f_i \rfloor$ times, and 3) for each position $i = 1, \ldots, 10^7$, we selected 'replace' with the probability of $\alpha/100$, and then replaced letter $s[i]$ with one of the other three letters uniformly and randomly.

As a set of pairs of length and frequency, in $D_1(\alpha)$, we used $\{(15, 14), (20, 8), (25, 6), (30, 5), (35, 4), (55, 3), (75, 2)\}$, in which frequency f_i for length ℓ_i is selected so as to satisfy $f_i = \bar{\sigma}(10^7, \ell_i, 1/4, 1.0 \times 10^{-20})$. In $D_2(\alpha)$, we used $\{(15, 28), (20, 16), (25, 12), (30, 10), (35, 8), (55, 6), (75, 4)\}$, where frequency f_i for length ℓ_i is twice larger than the corresponding frequency in $D_1(\alpha)$.

Table 1: Performance comparison between E3SFLOO and ESFLOO-0G.C using $D_1(0)$. Performances (in percentage) are averaged over 10 strings.

algorithm	parameter	prec.	recall	F-m.
E3SFLOO	p-value: 10^{-20}	99.4	100.0	99.7
ESFLOO-0G.C	min_sup: 14	98.8	18.6	31.3
	min_sup: 8	91.3	35.2	50.8
	min_sup: 6	74.5	57.7	65.0
	min_sup: 5	65.9	72.0	68.8
	min_sup: 4	35.5	86.9	50.4
	min_sup: 3	6.3	97.8	11.9
	min_sup: 2	1.3	100.0	2.5

Table 2: Performance [%] of E3SFLOO for noisy datasets. Performances are averaged over the 10 strings.

α	$D_1(\alpha)$			$D_2(\alpha)$		
	prec.	rec.	F-m.	prec.	rec.	F-m.
0	99.4	100.0	99.7	82.4	100.0	90.4
1	99.3	70.7	82.6	85.4	97.8	91.2
2	99.4	53.6	69.6	87.3	94.8	90,9
4	99.5	29.8	45.9	89.5	85.9	87.6
8	99.2	6.7	12.6	88.7	42.8	57.7

5.2 Effectiveness of Length-Dependent Supports

We compared the performance of E3SFLOO with ESFLOO-0G.C under various minimum supports using dataset $D_1(0)$. In all experiments, we used score function (2) with $a = 1$ and $b = 4$. First, we enumerated patterns using E3SFLOO or ESFLOO-0G.C, filtered out non-*closed patterns* and then extracted all occurrences of the enumerated patterns. A *closed pattern* is defined as substring $s[i..j]$ for which any extension $s[i'..,j']$ ($i' \leq i$, $j \leq j'$ and $(i', j') \neq (i, j)$) always has a smaller number of minimal locally optimal occurrences. We evaluated the ability of extracting occurrences by using the following three measures, precision (prec.), recall and F-measure (F-m.): let X and E be the sets of extracted and embedded occurrence positions, respectively, and precision, recall and F-measure can be defined as $\frac{|\mathrm{X} \cap \mathrm{E}|}{|\mathrm{X}|}$, $\frac{|\mathrm{X} \cap \mathrm{E}|}{|\mathrm{E}|}$ and $\frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$, respectively.

Table 1 shows the obtained results, in which ESFLOO-0G.C has always a trade-off between precision and recall, while E3SFLOO achieved almost perfect results for both precision and recall; E3SFLOO achieved F-measure of 99.7%, while $\leq$68.8% for ESFLOO-0G.C.

5.3 Robustness

We ran E3SFLOO using noisy datasets $D_1(\alpha)$ and $D_2(\alpha)$ for $\alpha = 1, 2, 4$ and 8 to check the robustness of E3SFLOO. Table 2 shows the result. For $D_1(\alpha)$, precision was almost 100% for all α but recall was drastically reduced for larger α. In $D_1(\alpha)$, pattern frequency f_i was decided to be $\bar{\sigma}(10^7, \ell_i, 1/4, 10^{-20})$, and so patterns are undetected as frequent substrings if the pattern occurrence is undetected. As for $D_2(\alpha)$, recall was not reduced like that of $D_1(\alpha)$ for larger α, because pattern frequency f_i was twice larger than

Table 3: Elapsed time [sec] of E3SFLOO averaged over 10 strings in $D_1(0)$.

#threads	1	2	4	8	16	32	64
parallel	161.4	82.3	41.2	23.3	12.6	8.1	4.7
(speedup)	(1.00)	(1.96)	(3.92)	(6.93)	(12.81)	(19.93)	(34.34)
total	176.5	98.2	56.0	38.2	27.2	22.8	19.4

that of $D_1(\alpha)$. Thus, almost all patterns were detected as frequent substrings, except the data with $\alpha = 8$, even if some occurrences cannot be detected. Note that an occurrence can be detected if any prefix and suffix of the occurrence contains less than 20% replacement positions for the score function (2) with $a = 1$ and $b = 4$.

5.4 Parallelization

We measured the elapsed time for each string in dataset $D_1(0)$, particularly the parallel computation part as well as the total elapsed time. Table 3 shows the averaged elapsed time over 10 strings in $D_1(0)$, changing the number of threads. From this table, we can see that our parallel implementation worked well, particularly for a larger number of threads. This table reveals that our parallel implementation achieved a speed-up factor of 34.34 for using 64 threads.

6 Experiments on Human Genome

6.1 Human Genome Data

We used Homo_sapiens.GRCh37.75.dna.chromosome.x.fa for $x = 1, \ldots, 22, X, Y$ in the human assembly GRCh37 (release 75) [ftp.ensembl.org/pub/release-75/fasta/homo_sapien/ dna/]. The total length of the human chromosomes is 3,095,677,412, while this string includes many long substrings with consecutive 'N'[5], because of ambiguity caused by incomplete experiments. We excluded substrings with at least 10 'N's consecutively. As a result, 24 chromosomes are divided into 281 strings. Finally the length of the data we used is around 2.86 billion and the occurrence rates of letters 'A', 'C', 'G', 'T' and 'N' of the data are around 0.295, 0.204, 0.205, 0.296 and 2.24×10^{-8}, respectively.

6.2 Four Steps for Repeats from Human Genome

We detected repeats by E3SFLOO and subsequent postprocessing, which are summarized in the following four steps:

[Step 1] Running E3SFLOO

For 281 strings $s_1, \ldots, s_{281}$ derived from the original DNA sequences of human chromosomes, we ran a multiple-string version of E3SFLOO that uses a generalized suffix tree and counting all occurrences in all input strings. Parameter settings are: $\alpha = 1.0 \times 10^{-20}$ and $a = 1$ and $b = 4$ for score function (2). We used 64 threads (plus the main thread) for parallel computation. The machine has 80 core CPUs (Intel(R) Xeon(R) CPU E7-2850@2.00GHz), 3 TB memory and CentOS release 6.7. The elapsed computational time was 63 days 2 hours 1 minute and 21 seconds, and the maximum resident memory size was 335,624,444KB. The output of Step 1 was 2,011,166,454 patterns[6].

[5] 'N' means one of 'A', 'C'. 'G' and 'T'.

Table 4: Cover rates (%) of positions by O-E3SFLOO and by RepeatMasker using Repbase (RM+R) and TRF.

chro. no.	O-E3S-FLOO	RM+R &TRF	chro. no.	O-E3S-FLOO	RM+R &TRF	chro. no.	O-E3S-FLOO	RM+R &TRF
1	74.3	51.1	9	75.8	50.4	17	73.2	49.8
2	74.0	48.6	10	73.9	49.0	18	73.3	47.3
3	74.6	50.2	11	73.6	51.2	19	77.7	58.7
4	76.6	50.7	12	75.0	51.9	20	70.8	51.0
5	75.0	50.2	13	75.3	47.8	21	75.1	48.2
6	74.9	49.4	14	74.6	50.4	22	72.1	50.2
7	75.8	50.4	15	74.3	49.6	X	79.8	61.2
8	74.4	50.5	16	73.6	51.0	Y	93.7	62.4
						total	75.1	50.9

[Step 2] Removing Non-Closed Patterns

In order to remove redundancy, we extracted only *closed* patterns. The output of Step 2 was 1,276,455,962 patterns.

[Step 3] Removing Similar Patterns

Detected patterns may be the occurrences of other patterns, resulting in all these patterns are similar to each other. In order to remove such similar patterns, we run greedy covering algorithms as follows: Let S be the set of the extracted patterns (substrings). Define S_p as $S_p = \{q \in S \mid q \text{ is a minimal locally optimal occurrence of } p\}$ for $p \in S$. Then, we consider a set cover problem of S by S_p ($p \in S$). That is, we find a smallest-sized $T \subseteq S$ with $\bigcup_{p\in T} S_p = S$. An approximation algorithm for this problem is a greedy covering algorithm, which starts from $T = \emptyset$ and repeats adding S_p with $p = \arg\max_{q\in S\setminus T} |(S \setminus \bigcup_{r\in T} S_r) \cap S_q|$ to T until $\bigcup_{p\in T} S_p = S$. Since the reverse complementary sequence of sequence p is the same in DNA as p, we used $S_p = \{q \in S \mid q$ is a minimal locally optimal occurrence of either p or its reverse complementary sequence$\}$. The output of Step 3 was 204,236,850 patterns.

[Step 4] Covering by Maximal Occurrences

For each component string $s_i[1..n_i]$ ($i = 1, \ldots, 281$), we computed set O_i of the minimal locally optimal occurrences of the extracted patterns and removed non-maximal occurrences from O_i. That is, we removed occurrence $s_i[h..j]$ from O_i if $s_i[h'..j']$ in O_i satisfies $h' \leq h$, $j \leq j'$ and $(h', j') \neq (h, j)$. Let O be the set of maximal occurrences $O = \bigcup_{i=1}^{281} O_i$. The size of O was 182,122,859. Then, we generated the final set of the extracted patterns, for which the occurrences of the patterns are in any of $O_i (i = 1, \ldots 281)$. The output of Step 4 was 42,721,486 patterns. The shortest and longest pattern lengths were 5 and 127,353, respectively. The most frequent length was 20 with 2,505,216 patterns. We write the set of final occurrences as O-E3SFLOO.

6.3 Analysis

[6] Some non-closed patterns for right extension are already removed to reduce the number of outputted patterns.

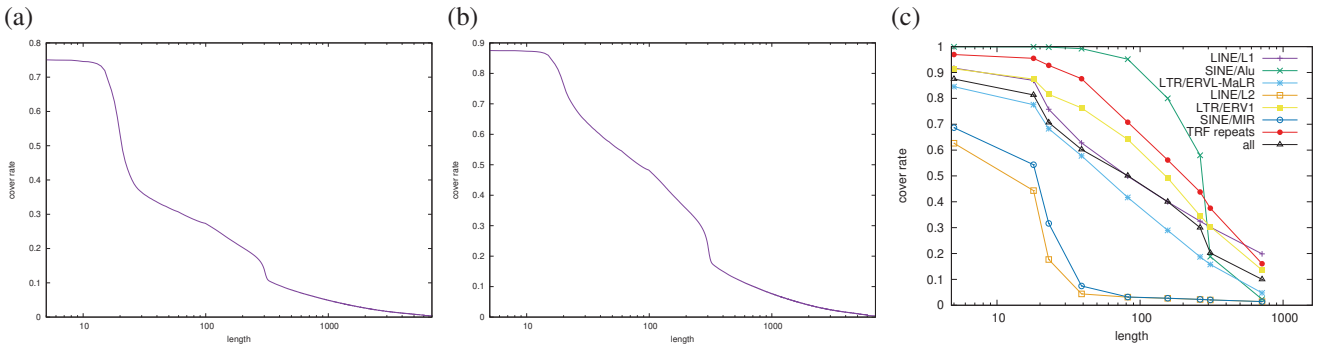

Figure 1: Cover rate vs. pattern length x: Cover rate against (a) all or (b) (RM+R and TRF)-annotated positions by minimal locally optimal occurrences of the obtained patterns with the length of at least x. (c) For each repeat category, cover rate against (RM+R and TRF)-annotated positions by the same occurrences as (a) and (b).

Table 5: Cover rate (CR) [%] of O-E3SFLOO to positions annotated by RM+R and TRF for each of ten largest repeat categories.

repeat category	rate	CR	repeat category	rate	CR
LINE/L1	34.8	91.7	SINE/MIR	5.4	68.7
SINE/Alu	21.0	99.9	TRF repeats	4.9	96.9
LTR/ERVL-MaLR	7.5	84.5	LTR/ERVL	3.9	69.3
LINE/L2	6.9	62.7	DNA/hAT-Charlie	3.1	76.0
LTR/ERV1	5.7	91.4	DNA/TcMar-Tigger	2.5	88.1
others	7.9	83.9	all	100.0	87.5

Table 6: Four patterns with the longest cover lengths which are not in the annotations by RM +R and TRF and at least two occurrences in O-E3SFLOO.

chro. no.:pos.	len.	freq.	cov. len.	occ. pos. ((c): rev. compl.))
X:151868211	2431	4	9724	X:151868211,X:151884496,
15:30437979	2597	3	7791	15:23265766,15:30437979, 15:32682178(c)
22:21473896	3235	2	6470	22:21473896,22:21641963(c)
21:9856253	2090	3	6270	14:19765953(c),21:9856253, 22:16074924

Cover Rate Table 4 shows the (cover) rate of the number of positions covered by O-E3SFLOO (by Step 4) to the number of all positions, i.e. the entire length (which is not the whole length but those analyzed), for each chromosome. From Table 4, we can see that the cover rate of E3SFLOO reached more than 75%, which is larger than 71%, reported by an existing method, P-clouds (de Koning et al. 2011).

Figure 1 (a) shows the cover rate by the occurrences with the length of at least m. This figure shows that around a half (46.6%) of the covered positions are by the patterns (occurrences) with the length of less than 25. In our model, we may say that many short strings can be occurrences with statically significant frequencies, while these short strings might be parts of longer repetitive structures. Repetitive strings detected by P-clouds have the length of at least 25 (de Koning et al. 2011). On the other hand, the covered rate of our method is 40.1% by the occurrences with the length of at least 25. This result implies that the cover rate of our method by long repetitive strings might not be so high as P-clouds.

Regarding the difference among chromosomes, the cover rate of chromosome Y is significantly higher than those of other chromosomes. This result is consistent with the report that chromosome Y has a high value of a certain repetitive index (Haubold and Wiehe 2006).

We compared O-E3SFLOO with those annotated by RepeatMasker 4.0.7 (Smit, Hubley, and Green 2017) (search engine: NCBI/RMBLAST ver.2.2.27+), a well-accepted similarity search tool, as occurrences of known repetitive strings registered in Dfam_Consensus (release: 20171107) [www.dfam-consensus.org] and Repbase (release: 20170127) [www.girinst.org/repbase]. RepeatMasker does not annotate long approximate tandem repeats, and so repeats found by Tandem Repeat Finder (TRF) (Benson 1999) were also added as annotations.

Table 4 shows that the total cover rate by repeat occurrences found by RepeatMasker with Repbase (RM+R) and TRF was only 50.9%, around 24% lower than O-E3SFLOO. Particularly the cover rate of chromosome Y was not so high as O-E3SFLOO.

Table 5 shows the cover rate by O-E3SFLOO against annotations by RM+R and TRF, indicating the total cover rate of 87.5%. Figure 1 (b) shows the cover rate by the occurrences with the length of at least x against the annotations by RM+R and TRF. Comparing with Figure 1 (a), we can see that the cover rate was significantly improved at the range from 20 to 300 of the x-axis.

Table 5 shows the cover rate of each of the ten repeat categories with the largest numbers of annotated positions. This table shows that 99.9% and 96.9% of positions annotated as SINE/Alu and TRF repeats, respectively, were detected by O-E3SFLOO. Also LINE/L1 and LTR/ERV1 were well covered by O-E3SFLOO (91.7% and 91.4%, respectively).

Figure 1 (c) shows the cover rate for each repeat category by the occurrences with the pattern length of at least x, against the annotations by RM+R and TRF. From this figure, we can see that the significant improvement of the total cover rate in the range of 20 to 300 in Figure 1 (b) was

caused by the significant cover rate increases of LINE/L2 and SIN/MIR (around 20), and SINE/Alu (around 300).

Discovery of Unknown Repeats Table 6 shows four patterns with the longest cover lengths among those with at least two occurrences in O-E3SFLOO and no overlapped covered positions with the positions annotated by RP+R and TRF. These four repeats are all interspersed repeats longer than 2,000 with the frequency of 2 to 4 including reverse complementary occurrences. The fourth pattern has occurrences in three different chromosomes, while the occurrences of each of others are in the same chromosome. According to UCSC Genome Browser [genome.ucsc.edu], the occurrences of these four patterns are parts, particularly pairs, of known segmental duplications (Bailey et al. 2001), defined as block pairs ranging in size from 1,000 to 200,000 with the similarity of at least 90%. We emphasize that the found patterns are significantly shorter than known segmental duplications, and at least three patterns are not pairs but patterns.

7 Conclusion and Future Work

We have proposed new frequent substring mining that can enumerate patterns with statistically significant frequencies of locally optimal occurrences, where a memoryless stationary information source was used as a null stochastic model to evaluate the statistical significance of the generated patterns. Possible future work is to consider Markov information sources, which might be more appropriate than the current memoryless source. Also entirely understanding repeat patterns with various lengths would be also interesting future work, since statistically significant short patterns might take important roles as part of longer significant patterns.

Acknowledgments

This research has been partially supported by JSPS KAKENHI Grant numbers 19H04161, 19H04169, 18H05413, 17H01783, 17K19953, 16H02868, 15H05711, JST CREST Grant numbers JPMJCR18K3 & JPMJCR1662, JST ACCEL Grant number JPMJAC1503, FiDiPro by Tekes (now Business Finland) and AIPSE (Grant number: 315896) by Academy of Finland.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *Proc. of ICDE 1995*, 3–14.

Bailey, J. A.; Yavor, A. M.; Massa, H. F.; Trask, B. J.; and Eichler, E. E. 2001. Segmental duplications: organization and impact within the current human genome project assembly. *Genome Research* 11(6):1005–1017.

Bao, Z., and Eddy, S. R. 2002. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Research* 12(8):1269–1276.

Bao, W.; Kojima, K.; and Kohany, O. 2015. Repbase update, a database of repetitive elements in eukaryotic genomes. *Mob DNA*. 6:11.

Benson, G. 1999. Tandem repeats finder: a program to analyze dna sequences. *Nucleic Acids Research* 27(2):573–580.

Bergman, C. M., and Quesneville, H. 2007. Discovering and detecting transposable elements in genome sequences. *Brief Bioinform* 8(6):382–392.

de Koning, A. P. J.; Gu, W.; Castoe, T. A.; Batzer, M. A.; and Pollock, D. D. 2011. Repetitive elements may comprise over two-thirds of the human genome. *Plos Genetics* 7(12). e1002384.

Edgar, R. C., and Myers, E. W. 2005. Piler: identification and classification of genomic repeats. *Bioinformatics* 21(suppl 1):i152–i158.

Ewens, W. J., and Grant, G. R. 2005. *Statistical Methods in Bioinformatics: An Introduction (2nd Ed.)*. Springer Science+Business Media, Inc.

Faulkner, G. J.; Kimura, Y.; and et al., C. O. D. 2009. The regulated retrotransposon transcriptome of mammalian cells. *Nature Genetics* 41:563–571.

Ghodsi, M.; Liu, B.; and Pop, M. 2011. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC Bioinformatics*. 12:271.

Gu, W.; Castoe, T. A.; Hedges, D. J.; Batzer, M. A.; and Pollock, D. D. 2008. Identification of repeat structure in large genomes using repeat probability clouds. *Analytical Biochemistry* 380:77–83.

Haubold, B., and Wiehe, T. 2006. How repetitive are genomes? *BMC Bioinformatics*. 7:541.

Jelovic, A. M.; Mitic, N. S.; Eshafah, S.; and Beljanski, M. V. 2018. Finding statistically significant repeats in nucleic acids and proteins. *Journal of Computational Biology* 25(4):375–387.

Kurtz, S.; Choudhuri, J. V.; Ohlebusch, E.; Schleiermacher, C.; Stoye, J.; and Giegerich, R. 2001. Reputer: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Research* 29(22):4633–4642.

Li, R.; Ye, J.; Li, S.; Wang, J.; Han, Y.; Ye, C.; Wang, J.; Yang, H.; Yu, J.; Wong, G. K.-S.; and Wang, J. 2005. Reas: Recovery of ancestral sequences for transposable elements from the unassembled reads of a whole genome shotgun. *PLOS Computational Biology* 1(4):e43.

Low-Kam, C.; Raïssi, C.; Kaytoue, M.; and Pei, J. 2013. Mining Statistically Significant Sequential Patterns. In *IEEE International Conference on Data Mining*.

Nakamura, A.; Takigawa, I.; Tosaka, H.; Kudo, M.; and Mamitsuka, H. 2016. Mining approximate patterns with frequent locally optimal occurrences. *Discrete Applied Mathematics* 200:123–152.

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.

Quesneville, H.; Nouaud, D.; and Anxolabéhère, D. 2003. Detection of new transposable element families in drosophila melanogaster and anopheles gambiae genomes. *Journal of Molecular Evolution* 57(1):S50–S59.

Smit, A.; Hubley, R.; and Green, P. 2017. Repeatmasker open-4.0.7. http://www.repeatmasker.org/.

Terada, A.; Okada-Hatakeyama, M.; Tsuda, K.; and Sese, J. 2013. Statistical significance of combinatorial regulations. *Proceedings of the National Academy of Sciences* 110(32):12996–13001.

Volfovsky, N.; Haas, B. J.; and Salzberg, S. L. 2001. A clustering method for repeat analysis in dna sequences. *Genome biology* 2(8):research0027.1–research0027.11.

Zhu, F.; Yan, X.; Han, J.; and Yu, P. S. 2007. Efficient discovery of frequent approximate sequential patterns. In *Proc. of ICDM 2007*, 751–756.