Finding Approximate Tandem Repeats in Genomic Sequences

Ydo Wexler[*] Zohar Yakhini[†] Yechezkel Kashi[‡] Dan Geiger[§]

ABSTRACT

An efficient algorithm is presented for detecting approximate tandem repeats in genomic sequences. The algorithm is based on a flexible statistical model which allows a wide range of definitions of approximate tandem repeats. The ideas and methods underlying the algorithm are described and its effectiveness on genomic data is demonstrated.

Categories and Subject Descriptors

J.3 [**Computer Applications**]: Life and medical sciences; G.3.8 [**Mathematics of Computing**]: Probability and statistics—*Probabilistic algorithms*

General Terms

Algorithms, Experimentation

Keywords

tandem repeats, TRF, TEIRESIAS, ATR, pattern discovery, DNA repeats, DNA satellites

1. INTRODUCTION

Genomic sequences often contain consecutive copies of patterns known as *tandem repeats* (*TR*s). The origin of these repeats, as well as their biological function, is not fully understood. Nevertheless, they are believed to play an important role in genome organization and evolution [35, 26]. Replication slippage, unequal crossing-over and evolutionary pressures generate a high degree of polymorphism in the number of repeats [9, 14, 41]. TRs are therefore useful as genetic markers, such as for DNA fingerprinting [5, 22, 24], mapping genes, comparative genomics and for evolution studies. Several studies have shown that tandem repeat polymorphism plays an important role in the adaptation of pathogenic bacteria to their host [16] and may also have pharmacological effects in humans [37].

Unusual numbers of triplet tandem repeats are known to cause disorders such as Fragile-X syndrome and Huntington's disease [44, 17]. The SCA10 and Baltic myoclonus disorders are caused when nucleotide repeats of length 5 and 12 respectively expand to produce hundreds of copies [18, 45]. Studies have shown that tandem repeats located in regulatory regions can cause disease by influencing gene expression [8, 27]; for example, a tandem repeat polymorphism in the dopamine transporter (DAT) gene has been associated with Parkinson's disease in the Korean population [48]. Other disorders, such as colorectal and ovarian cancer, are being investigated for similar connections [13, 23].

A *perfect tandem repeat* in a genomic sequence is a string of nucleotides consisting of multiple consecutive occurrences of a substring called a motif. For example, ATTGAATTGA is a perfect tandem repeat of a motif of length 5. Finding perfect tandem repeats with two repeating units in a sequence of length n is a well-studied problem for which several $O(n \cdot \log n)$ algorithms have been presented [2, 10, 36]. This is an asymptotically optimal bound because the maximum number of occurrences of such repeats is $\Omega(n \cdot \log n)$ [10]. In the case of tandem repeats for which adding a symbol to their left or right yields a string with a greater motif, tighter bounds have been obtained [29].

However, perfect tandem repeats are of limited biological interest, since events such as mutations, translocations and reversal events will often render the copies imperfect. The result is an *approximate tandem repeat (ATR)*, defined as a string of nucleotides repeated consecutively at least twice with small differences between the instances. Finding ATRs in a sequence is a harder task than finding perfect repeats and has been addressed by several papers during recent years. The scope of ATRs discovered by some of the algorithmic approaches are limited by constraints on the input data, search parameters, the type of allowed mutations and the number of such mutations [12, 19, 30, 31, 32, 33, 34, 39]. In others, time requirements render the algorithm infeasible for the analysis of whole genomes containing mil-

[*]Computer Science Department, Technion, Haifa, 32000, Israel (ywex@cs.technion.ac.il).

[†]Agilent Laboratories and Computer Science Department, Technion (zohar_yakhini@agilent.com).

[‡]Food and Biotechnology Department, Technion, Haifa, 32000, Israel (kashi@tx.technion.ac.il).

[§]Computer Science Department, Technion, Haifa, 32000, Israel (dang@cs.technion.ac.il).

lions of base pairs [25]. More widely applicable algorithms have been proposed in [6, 40, 42]. Other related algorithms are [28, 20, 38, 43].

In this paper we present a new approach to finding ATRs in genomic sequences. Similarly to [6] and [42], we employ a two-phased algorithm, which consists of a screening phase, followed by a candidate verification phase. Our main contribution is an innovative flexible screening phase, which generates a list of candidate regions which may contain ATRs that are subsequently verified (Section 3). The flexibility is achieved by using a variable size sliding window along with a suitable similarity metric, as well as a novel statistical model which captures well the behavior of the distributions involved (Section 4). The result is an algorithm and software that detects significantly more ATRs than previous methods under a variety of definitions and counting methods. The thoroughly tested algorithm (Section 5) and executables are available at `http://bioinfo.cs.technion.ac.il/ATRHunter`.

2. MODELS OF ATRS

Consider sequences over a finite alphabet Σ. A *genomic sequence* is a sequence of symbols from the alphabet $\Sigma = \{A, C, G, T\}$ and a *proteomic sequence* is a sequence over the alphabet representing the 20 amino acids. An alignment of two sequences Ψ_1 and Ψ_2 over Σ is defined as a pair of sequences Ψ'_1 and Ψ'_2 over $\Sigma \cup \{-\}$ of equal length such that the removal of the symbols '$-$' from Ψ'_i yields Ψ_i for $i = 1, 2$. An *optimal alignment* is an alignment for which $\varphi(\Psi'_1, \Psi'_2)$ is maximal where φ is a given scoring function. The score of the optimal alignment indicates the similarity of the two original sequences; the higher the score, the more similar the two sequences are. The scoring function used reflects an underlying biological model. Common scoring functions are additive, namely, functions φ for which $\varphi(\Psi'_1, \Psi'_2) = \sum_\alpha \varphi(\Psi'_1[\alpha], \Psi'_2[\alpha])$, where $\Psi'_i[\alpha]$ is the symbol at position α in Ψ'_i. Standard scoring functions for proteomic sequences are PAM and BLOSUM [11, 21]. For a wider discussion of scoring functions and alignments, consult [47].

We define several types of *Approximate Tandem Repeats (ATRs)* with respect to a scoring function φ. A *simple ATR* is a concatenation of sequences $T = T_1T_2 \cdots T_r$ for which there exists a sequence T_* such that $\varphi(T_i, T_*) \geq \eta$ for every $i = 1, \ldots, r$. In other words, T consists of r mutated copies of a *consensus motif* T_* with a limited amount of diversity dictated by $\varphi(T_i, T_*) \geq \eta$. In this definition T_* may be different from each and every T_i, alternatively we may require that T_* be equal to at least one T_i. A *neighboring ATR* is a concatenation of sequences $T = T_1T_2 \cdots T_r$ for which $\varphi(T_i, T_{i+1}) \geq \eta$ for every $i = 1, \ldots, r-1$. According to this definition, the similarity between distant copies can become quite small. A *pairwise ATR*, which generalizes the latter definition, is a concatenation of sequences $T = T_1T_2 \ldots T_r$ for which the similarity $\varphi(T_i, T_j)$ of every pair T_i and T_j is higher than η_{ij}. The threshold η_{ij} limits the dispersal of distant sequences and is usually set to be a monotonically decreasing function of $|i - j|$.

3. ALGORITHM FOR FINDING ATRS

Our goal is to efficiently detect ATRs in large genomes given an arbitrary scoring function. The proposed algorithm has two phases, *screening* and *verification*. The screening phase quickly identifies candidate ATRs of one of the abovementioned types. These candidates are subsequently accepted or rejected by the *verification* phase which is tailored to the type of ATR under study.

3.1 Screening phase

The screening phase identifies substrings which have an unusually high probability of being an ATR. Such candidate ATRs are those that pass three *similarity criteria* developed below. Setting useful thresholds for these similarity criteria requires a delicate balance; while strict similarity criteria limit the number of false candidates and hence reduce verification time, lenient criteria exclude fewer substrings that are in fact ATRs. In Section 4 we develop a statistical framework to determine these thresholds. In this section we assume they are given.

For a substring to qualify as any type of ATR, every pair of adjacent copies in the repeat must be similar. Thus, a basic step in screening is to determine whether a substring of length t is similar to the following substring of approximately the same length. Producing an alignment for each pair of consecutive substrings to assess their similarity is computationally expensive, so an alternative approach is required. The similarity between two adjacent substrings of length t is tested by comparing segments of length l of these two substrings. More explicitly, every segment of length l, called an l-window, in the first substring is compared with appropriate l-windows in the second substring. The outcome of a comparison of two l-windows is a vector of length l, in which each entry indicates a match or a mismatch between an aligned pair of symbols. Given $0 \leq q \leq 1$, such a vector is said to be a *q-quality vector* if the number of matches is at least $q \cdot l$. Given l and q, for every position i in the input sequence, we define two quantities: score and gap. The *score* $S_t(i)$ is the number of q-quality vectors produced by the comparisons of the l-windows in the substring of length t starting at position i. There are at most $t - l + 1$ possible q-quality vectors in a substring of length t. Attaining this maximum of $S_t(i)$ for two contiguous substrings of length t indicate a certain similarity between these substrings but does not necessarily imply that they are identical. This is due to the fact that q could be less than 1 and so a low density of mismatches between aligned symbols is not accounted for. The *gap* $\Delta_t(i)$ is the maximal number of consecutive l-windows in the substring of length t starting at position i that produce vectors which are not q-quality. The quantities $S_t(i)$ and $\Delta_t(i)$ and their distributions play a key role in the screening phase.

A substring has to satisfy three similarity criteria in order to be a candidate ATR. These criteria depend on the motif length t and on two parameters, P_M and P_I, which we now define. The parameter P_M is the probability of a match between aligned symbols. For the alphabet $\{A, C, G, T\}$, we set $P_M = P_A^2 + P_C^2 + P_G^2 + P_T^2$, where P_A, P_C, P_G, P_T are the respective frequencies of the symbols A, C, G, T. When the four symbols are distributed uniformly $P_M = 0.25$. The parameter P_I is the percentage of insertion and deletion expected between adjacent copies in an ATR, and is determined by a biological model.

A substring of length t starting at position i is a *candidate ATR* if and only if it passes the following three similarity criteria:

Score criterion: $S_t(i) \geq \sigma_t$.

Continuity criterion: $\Delta_t(i) \leq \delta_t$.

Distance criterion: Every q-quality vector counted in $S_t(i)$ is the result of a comparison between two l-windows whose *offset*, namely, the distance over t positions between them, is at least 0 but no more than d^t_{max}. The distance criterion is similar to the one in [6].

The screening phase employs an iterative algorithm with up to t_{max} iterations. In each iteration, candidate ATRs with motif length t are sought. For each such t the algorithm selects parameters l and q. These choices are later explained. Initially, two windows of length l are positioned at locations 1 and $t+1$. We refer to these windows as the first and second respectively. In each iteration, the two l-windows slide towards the end of the sequence, and their contents are compared at each step to produce a vector of length l. While the first window is moved by a single position in each and every sliding step, the algorithm advances the second window by 0,1 or 2 positions so as to greedily maximize the number of q-quality vectors. Namely, if advancing the second window by a single position, which is the default choice, does not produce a q-quality vector, the algorithm considers moving it by either none or two positions if such a move would produce a q-quality vector without violating the distance criterion. At the end of the i^{th} sliding step, for $i > t - l$, the algorithm counts the number of q-quality vectors and the maximum number of consecutive vectors that are not q-quality within the last $t-l+1$ vectors, and sets these quantities as $S_t(i-t+l)$ and $\Delta_t(i-t+l)$ respectively. The computation of $S_t(i)$, which takes $O(1)$ time per sliding step, is facilitated by the recursive formula $S_t(i) = S_t(i-1) + \Lambda_i - \Lambda_{i-(t-l+1)}$, where Λ_i is a boolean variable indicating whether the vector produced in the i^{th} sliding step is q-quality. To compute the gaps $\Delta_t(i)$ in $O(1)$ time per sliding step, the algorithm maintains a bi-directional linked list, in which each element contains the size and location of a set of consecutive vectors that are not q-quality, generated in the last $t-l+1$ steps. The first and last nodes are updated in every sliding step, in the obvious way. A description of the algorithm is given in Figure 1.

The choice of the parameters q and l adjusts the screening phase to the given scoring function. We set upper bounds q_{max} and l_{max} for q and l to be the minimal values such that any alignment that contains $(1-q_{max})l$ mismatches and $q_{max}l$ matches scores more than η, as defined for pairwise ATRs, so a q-quality vector always complies with the definition of pairwise ATRs. Similarly, any alignment with $t - l_{max}$ insertions and deletions (indels) and l_{max} matches scores more than η, ensuring that the screening algorithm can account for ATRs with any amount of indels. Recall that no alignment is performed in the screening phase in order to save computations. Consequently, multiple indels may yield vectors that are not q-quality even if the first and second windows are quite similar. To restrict the effects of this phenomenon we limit l to be smaller than $\frac{1}{P_I}$. This choice guarantees that the expected number of indels in a window of length l is no more than 1. A symbol in a substring of length t appears in the first window an average of $[\frac{(t-l+1)}{t}]l$ times when the first window is fully contained in that substring. Setting the window size l to $min\{\lceil \frac{t}{2} \rceil, l_{max}, \frac{1}{P_I}\}$ maximizes this expression under the two previous constraints. We consider a simple statistical model, in which the given sequence is i.i.d. with probability p_w for the occurrence of symbol w. Let $P_q = \sum_{i=\lceil ql \rceil}^{l} \binom{l}{i} P_M^i (1-P_M)^{l-i}$. This is the probability for a comparison to produce a q-quality vector. We set q sufficiently high such that $2P_q(1-P_q) \leq \frac{1}{l}$; as we shall see, this choice of q yields a linear expected time complexity. Whenever $2P_{q_{max}}(1-P_{q_{max}}) > \frac{1}{l}$, we decrease the value of l and set q_{max} accordingly. Regardless of the value P_M, there always exist values l and q_{max} to satisfy this constraint. Actual values for l and q are provided in a web supplement [46].

We now analyze the algorithm's time complexity, showing that the expected run time is linear in the sequence length n for any motif length. A naive implementation, where in each sliding step a vector of length l is generated afresh, takes $O(n \cdot l)$ time for each motif length. However, generating such a vector afresh is not necessary in all sliding steps. Whenever the two windows are advanced by a single position, namely, the default choice is made, comparison between the two l-windows takes constant time, since $l-1$ entries in the resulting vector have already been computed in the previous step. Otherwise, when the chosen step is not the default one, comparing the two windows of length l to produce a vector of that length takes time $O(l)$. The probability that advancing the second window by a single position does not produce a q-quality vector is $1-P_q$, and the probability that advancing the second window by two positions or none produces a q-quality vector without violating the distance criterion is less than or equal to $2P_q$. Consequently, it can be shown that the probability for both events to occur, and hence for the algorithm to choose a sliding step other than the default, is no more than $2P_q(1-P_q)$. Since q is chosen to satisfy $2P_q(1-P_q) \leq \frac{1}{l}$, the additional comparisons, originating from choosing a step which is not the default, contribute an average time of $O(1)$ to each step. Consequently, and since computing the score and gap at the end of each sliding step takes constant time, the expected time for each motif length is $O(n)$. Hence, the average time complexity for motif lengths $t = 1 \ldots t_{max}$ is $O(t_{max} \cdot n)$. The performance of this algorithm on various genomic sequences is demonstrated in Section 5.

3.2 Verification phase for pairwise ATRs

Given a list of candidate ATRs and their lengths, the verification phase determines which of these candidates are in fact ATRs of the desired type. Clearly, different types of ATRs lead to different verification procedures that correspond to the definition of that type. For a pairwise ATR a two-phased verification procedure is performed.

Firstly, candidate ATRs with motif length t are aligned with the following substring of length t, to test whether the alignment score passes the given threshold. This stage is skipped in cases where a previous alignment already indicated the required level of similarity.

Secondly, these two-repeat ATRs are compounded into a single ATR containing more repeats. Let $A_t(i)$ denote an ATR with motif length t such that its last repeat starts at position i. The ATR may contain k repeats, $k \geq 2$. To extend this ATR, the algorithm aligns each of the first $k-1$ repeats against the second copy of an ATR $A_{t'}(i+t')$ containing two copies, where $t' \in [t - d^t_{max}, t + d^t_{max}]$, if such an ATR exists. When several such ATRs exist, the one with motif length t' closest to t is chosen. The two ATRs are combined into one if each of the $k-1$ alignments scores above the given thresholds. The motif length of the combined ATR is set to be the most common motif length among the original two-repeats ATRs that generated it.

```
ATRHUNTER
Input: Sequence $Seq$, maximum motif length $t_{max}$, scoring function $\varphi$, alignment threshold $\eta$.
Output: A list of approximate tandem repeats in $Seq$ (stored in REPEATS).
REPEATS $\leftarrow \emptyset$
For all $t$ up to $t_{max}$ do
    Set $l$ and $q$ to be the window size and the fraction of matches as a function of $t$, $\varphi$ and $\eta$ so that $2P_q(1-P_q) \leq \frac{1}{l}$
    {as described in Section 3.1, where $P_q$ is the probability for $q$ matches in a vector}
    Set the thresholds $\sigma_t$, $\delta_t$ and $d^t_{max}$ as a function of $t$, $l$ and $q$
    {as described in Section 4}
    CANDIDATES $\leftarrow$ Candidate_Selection($t, l, q, \sigma_t, \delta_t, d^t_{max}, Seq$)
    {The set CANDIDATES contains all candidate ATRs of length $t$}
    For each candidate in CANDIDATES
        If verification(candidate)= TRUE, then REPEATS$\leftarrow$ REPEATS $\cup$ {candidate}
```

```
Candidate_Selection($t$, $l$, $q$, $\sigma_t$, $\delta_t$, $d^t_{max}$, $Seq$)
Input: Sequence $Seq$, motif length $t$, window size $l$, $q \in [0,1]$, thresholds for the similarity criteria.
Output: A list of the candidate ATRs of length $t$ in $Seq$ (stored in C).
C $\leftarrow \emptyset$ ; $dist \leftarrow t$
Initialize a binary array $A$ of length $|Seq|$ to 0                              {A success array}
Place two windows of length $l$, called $w_1$ and $w_2$, at positions 1 and $t+1$ in $Seq$
For $i = 1$ up to $|Seq| - t - l + 1$                        {every such iteration is called a sliding step}
    Advance $w_1$ by a single position
    If advancing $w_2$ by a single position produces a $q$-quality vector
        Advance $w_2$ by a single position ; $A[i] \leftarrow 1$
    Else:
        If $dist > t$ and matching $w_1$ and $w_2$ produces a $q$-quality vector
            $A[i] \leftarrow 1$ ; $dist \leftarrow dist - 1$
        Else if: $dist < t + d^t_{max}$ and advancing $w_2$ by two positions produces a $q$-quality vector
            Advance $w_2$ by two positions ; $A[i] \leftarrow 1$ ; $dist \leftarrow dist + 1$
        Else:
            Advance $w_2$ by a single position ; $A[i] \leftarrow 0$            {the default sliding step}
    If $i > t - l$ do
        $S_t(i-t+l) = \sum_{j=i-t+l}^{i} A[j]$
        $\Delta_t(i-t+l) =$ maximum number of consecutive 0's in $A[i-t+l, \ldots, i]$
        If $S_t(i-t+l) \geq \sigma_t$ and $\Delta_t(i-t+l) \leq \delta_t$, then C$\leftarrow$ C $\cup \{i-t+l\}$
Return C
```

Figure 1: The screening algorithm

From an implementation point of view, it is important to note that alignment is done via dynamic programming with quadratic complexity. Therefore, for motif lengths over 20 only a central band of the alignment matrix is used of width no more than $2 \cdot d^t_{max}$, twice the maximum insertion and deletions allowed by the distance criterion. Since all candidate ATRs have passed the screening phase, it is unlikely that a better alignment exists outside these bounds.

The verification procedure does not allow overlapping ATRs of the same motif length. Still, for a specific position i, the algorithm may report ATRs with various motif lengths starting at that position. To prevent cluttering the output, when two ATRs which overlap have motif lengths m_1 and m_2, where m_1 divides m_2, the ATR with motif length m_1 is reported only if it scores significantly higher than the other ATR or if it continues more than m_2 positions to the non-overlapping region. In practice, no more than 3 different motif lengths are reported, and most of the time reporting different motif lengths indicates a certain hierarchy of ATRs. Other implementation details, such as default values for all parameters, are given on ATRHUNTER's web site [46].

For other types of ATRs, the same screening phase is used, and only minor changes in verification are needed, according to the definition of that specific type.

4. STATISTICAL FRAMEWORK

This section develops a statistical framework for determining the thresholds σ_t, δ_t and d^t_{max} for the score, gap and offset respectively. In Section 4.1 we briefly describe how our algorithm sets the threshold d^t_{max} for the offset. This method is identical to Benson's method [6]. In Sections 4.2 and 4.3 we explain in detail our approach for determining the thresholds σ_t and δ_t using a novel approximation for the distributions of the random variables describing scores and gaps.

The threshold σ_t used for the score criterion is set so that only in a fraction ϵ_t of the binary sequences of length t, the number of substrings of length l that contain at least $q \cdot l$ 1's is greater than or equal to σ_t, where q is a constant $q \in [0, 1]$. In the software ATRHUNTER we fine tuned ϵ_t to be $max\{0.05 \cdot 10^{-6}, min\{0.05, 0.6^{t-l}\}\}$. When this fraction can not be achieved, we set σ_t to $t-l+1$, the maximum possible value. The threshold δ_t used in the continuity criterion is set so that the gaps of 95% of the substrings of length t that pass the score criterion are not over this threshold. Setting these thresholds requires the distributions for the score and gap. Unfortunately, these distributions are not available, hence, we introduce a method to approximate them.

4.1 Setting bounds for the distance criterion

Consider an optimal alignment (Ψ'_1, Ψ'_2), with respect to a scoring function φ, between two adjacent sequences Ψ_1 and Ψ_2, both of which contain t symbols over the same finite alphabet. The distance between matching symbols $\Psi'_1[i]$ and $\Psi'_2[i]$, corresponding to positions i_1 and i_2 in Ψ_1 and Ψ_2 respectively, is $t - i_1 + i_2$. The maximum of these distances minus t over all positions i, $1 \leq i \leq t$, is called the *offset* and is denoted by Z. Let P_I denote the average percentage of insertions and deletions in an optimal alignment that scores more than a threshold η using a scoring function φ. This probability is hard to compute and is estimated based on a biological model or set to a default value of, say, 0.03. Given P_I, it has been shown (in [15], pp. 342-347) that 95% of the time, the offset Z ranges between $\pm 2.3\sqrt{P_I \cdot t}$. The proof of this claim is based on viewing the random variable Z as a one-dimensional random walk of $P_I \cdot t$ expected steps where each step moves a unit to left or right with equal probability. The idea of setting bounds on Z in the screening tests is due to Benson [6]. The screening phase in our algorithm finds windows with high similarity which are at most $t + d^t_{max}$ symbols apart. Among substrings with sufficient number and density of such windows the verification phase finds adjacent sequences of length t with alignment scores more than η. To enhance the success of the screening phase we set $d^t_{max} = \lfloor 2.3\sqrt{P_I \cdot t} \rfloor$, following Benson [6].

4.2 Approximation of the score distribution

Let the sequence $\omega = \omega_1\omega_2 \dots \omega_t$ be drawn from $\{0, 1\}^t$ where each element ω_i is 1 with probability p and 0 with probability $1-p$, and all elements are independent. Denote $\omega^k_j = \omega_j\omega_{j+1} \dots \omega_k$. The substring ω^{j+l-1}_j is called a *q-quality run of length l* or a *(q, l)-run* if the fraction of 1's in it is at least q where $0 \leq q \leq 1$. A 1-quality run of length l is also simply called an *l-run*. We study the distribution of the number S of q-quality runs of length l appearing in a random sequence ω drawn as above. This distribution of the random variable S, which depends on q, l and t, is used to determine the threshold σ_t in our algorithm.

Distributions related to such runs have been studied quite extensively. In particular, a method to compute the variance of the sum of 1's in all l-runs is given in [7]. The distribution of the number of l-runs appearing in a random sequence before the first appearance of a k-run is discussed in [1]. An approximation to the distribution of the number of l-runs in a random binary sequence is given in [4]. Finally, an approximation to the distribution of the longest q-quality run in a random sequence of length t for q larger than a given parameter p is studied in [3] where rigorous bounds on the approximation error are derived.

Since the distribution of S is not known and all known bounds are not sufficiently tight for our application, we resort to a novel heuristic approximation. Our approximation uses a graph that describes sequences of length l according to the number of 1's contained, namely, according to their *Hamming weight*. The experiments in Section 5 show that in practice our approximation is quite tight, especially when l is of the same order as t, the length of the original sequence.

To define the relevant graph some definitions are required. A sequence u of length l is said to *follow* a sequence v of length l if the first $l-1$ elements of u are identical to the last $l-1$ elements of v. Each sequence of length l can be followed by exactly two sequences, one that terminates with 0 and the other that terminates with 1. Two consecutive substrings ω^{j+l-1}_j and ω^{j+l}_{j+1} share $l-1$ elements, so they differ in their Hamming weights if and only if $\omega_j \neq \omega_{j+l}$, in which case the difference is exactly 1. Furthermore, when $\omega_j = 1$, it follows that $h_l(j+1) = h_l(j)$ with probability p, where $h_l(j)$ is the Hamming weight of ω^{j+l-1}_j, and $h_l(j+1) = h_l(j) - 1$ with probability $1-p$. When $\omega_j = 0$, it follows that $h_l(j+1) = h_l(j) + 1$ with probability p and $h_l(j+1) = h_l(j)$ with probability $1-p$. Thus, knowing the Hamming weight of a substring and its first element determines the probability that the following substring's weight is w.

Let $G = (V \cup V', E)$ be an edge-weighted directed graph with $2l$ vertices. Each vertex $v_i \in V$, $0 \leq i \leq l-1$, represents the set of sequences of length l whose Hamming weight is i that start with 0. Similarly, each vertex $v'_i \in V'$, for $1 \leq i \leq l$, represents the set of sequences of length l whose Hamming weight is i that start with 1. Given q, vertices representing q-quality sequences are called *q-vertices*. An edge $e = (u, v)$ is in E iff there exists a sequence represented by v that follows a sequence represented by u. For example, (v_i, v_{i+1}) is an edge in E whenever $l > 1$, and the pair (v'_i, v_{i+1}) is not an edge in E because all sequences represented by v'_i start with 1, so the Hamming weight of any possible sequence that follows is no more than i. The weight p_e on each edge $e = (u, v)$ is a number satisfying $0 < p_e \leq 1$ which is interpreted as the *transition probability* of moving from u to v in one step of a random walk on G. The graph G is shown in Figure 2.

To determine the value of p_e for every edge in G, we employ the following reasoning. Every vertex v_i represents $\binom{l-1}{i}$ sequences of length l; in $\binom{l-2}{i}$ of these sequences the 2^{nd} position is 0 and in the other $\binom{l-2}{i-1}$ sequences the 2^{nd} position is 1. Consequently, the probability that a random sequence represented by v_i is followed by a random sequence represented by a vertex $u \in V$ is $\frac{l-i-1}{l-1}$, whereas the probability it is followed by a vertex $u \in V'$ is $\frac{i}{l-1}$. Analogously, the probability that a random sequence represented by v'_i is followed by a random sequence represented by a vertex $u \in V$ or $u \in V'$ is $\frac{l-i}{l-1}$ and $\frac{i-1}{l-1}$, respectively. The transition probability p_e associated with an edge $e = (u, v)$ is defined to be the probability that a random sequence of length l represented by the vertex u is followed by a random sequence represented by v. The eight possible transition probabilities are shown in Figure 3. These are generated using the abovementioned formulae multiplied by either p or $1-p$. We multiply by p when $u \in V'$ and $\beta = 0$ or when $u \in V$ and $\beta \neq 0$, where β is the difference in the Hamming weights of sequences represented by u and v, and multiply by $1-p$ otherwise. For example, the probability associated with a self looped edge $e=(v_i,v_i)$, corresponding to a sequence with Hamming weight i, which is followed by a sequence with identical Hamming weight that starts with 0, is $p_e=(1-p)\cdot\frac{l-i-1}{l-1}$.

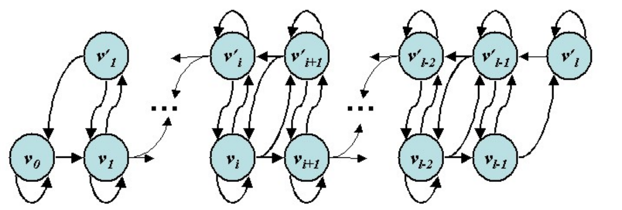


Figure 2: The graph reflecting all possible windows of length l.

We present a dynamic programming algorithm to compute the probability distribution of the number of q-vertices visited by a random walk of length $t-l$ on the graph G, whose initial vertex is chosen according to the distribution $\pi(v_i)=\binom{l-1}{i}p^i(1-p)^{l-i}$, $\pi(v'_i)=\binom{l-1}{i-1}p^i(1-p)^{l-i}$. The j^{th} iteration computes D^j_k, the probability that a random walk of length j visits exactly k q-vertices, for $0\le k\le j+1$. As we will show, this quantity approximates the probability distribution of the number of q-quality runs of length l over a random sequence of length $l+j$. Let $D^j_k(u)$ stand for the probability that a random walk of length j visits k q-vertices and ends at vertex u. In this notation, the sum $D^{t-l}_k=\sum_{u\in V\cup V'}D^{t-l}_k(u)$, for $0\le k\le t-l$, is the output of the dynamic programming algorithm.

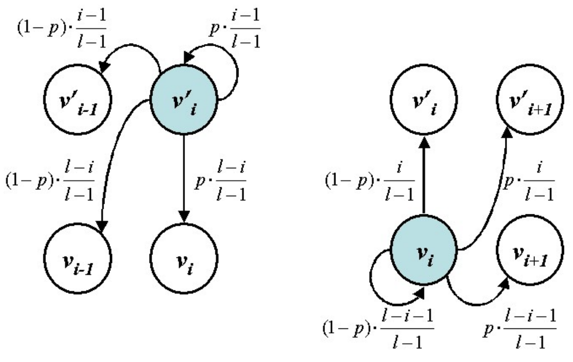
Let Q be the set of all q-vertices in G, and let $\overline{Q}$ be the set of all vertices in G that are not q-vertices. We set $D^0_c(u)=\pi(u)$ for each $u\in V\cup V'$, where $c=0$ if $u\in\overline{Q}$ and $c=1$ otherwise, setting all other values D^0_k to 0 for $k\ne c$. The following recursive formula is used in the $j^{\,th}$ iteration, to compute the values $D^j_k(v)$ for each $v\in V\cup V'$,

$$D^{j+1}_k(v)=\begin{cases}\sum_{u\in\{V\cup V'\}}p_{(u,v)}\cdot D^j_k(u) & \text{if } v\in\overline{Q}\\ \sum_{u\in\{V\cup V'\}}p_{(u,v)}\cdot D^j_{k-1}(u) & \text{if } v\in Q\end{cases}\qquad(1)$$

where $p_{(u,v)}=0$ if $(u,v)\notin E$.

We claim that the value $D^{t-l}_k(u)$ computed by this algorithm is indeed the probability that a random walk of length $t-l$ on the graph G, with an initial vertex chosen according to the distribution π, visits k q-vertices and ends at vertex u. First, consider a random walk of length 0 with its only vertex chosen according to the distribution π. By definition, for every vertex $u\in V\cup V'$ and $0\le k\le t-l$, the value $D^0_k(u)$ is the probability that such a random walk ends at vertex u after visiting k q-vertices. Now, consider a random walk of length j that visits k q-vertices and ends at a vertex v. The probability of such a random walk equals the probability that a random walk of length $j-1$, with an initial vertex chosen according to the same distribution, visits k q-vertices if $v\in\hat{Q}$ and $k-1$ q-vertices if $v\in Q$, and ends at some vertex u, multiplied by the probability it traverses (u,v) in the $j^{\,th}$ step. Hence, from Eq.(1), using the induction hypothesis we obtain, as claimed, that $D^j_k(v)$ is the probability that a random walk of length j visits k q-vertices and ends at a vertex v.

The probability distribution of the number of q-vertices visited by a random walk of length $t-l$ on G is not identical to the probability distribution of the number of (q,l)-runs in a binary sequence of length t for the following reason. Any binary sequence of length $l+j$ corresponds to a single walk of length j in the graph G. The number of (q,l)-runs in such a sequence equals the number of q-vertices visited by the corresponding walk. However, the process of moving between vertices in $V\cup V'$ is Markovian by definition while the result of mapping ℓ-long substrings of a uniformly drawn longer sequence into the appropriate vertices in $V\cup V'$ is not a Markovian process. More explicitly, the probability



$(1-p)\cdot\frac{i-1}{l-1}$
$p\cdot\frac{i-1}{l-1}$
v'_{i-1}
v'_i
$(1-p)\cdot\frac{l-i}{l-1}$
$p\cdot\frac{l-i}{l-1}$
v_{i-1}
v_i
v'_i
v'_{i+1}
$(1-p)\cdot\frac{i}{l-1}$
$p\cdot\frac{i}{l-1}$
v_i
v_{i+1}
$(1-p)\cdot\frac{l-i-1}{l-1}$
$p\cdot\frac{l-i-1}{l-1}$

Figure 3: An illustration of the edges with a source in vertex v_i (right) and v'_i (left) with their probabilities.

that a substring ω^{j+l}_{j+1} in a random sequence is represented by v, given that the substring ω^{j+l-1}_j is represented by u, often depends on the number of previous (q,l)-runs, while in a random walk on G, the edge $e=(u,v)$ in the $j^{\,th}$ step is selected with probability p_e regardless of the number of q-vertices visited in previous steps. For example, given $l=3$, $q=\frac{2}{3}$ and $p=\frac{1}{2}$, the probability that substring ω^5_3 in a random sequence is represented by v'_1, given that ω^4_2 is represented by v_1, is $\frac{1}{4}$, which equals p_e for the edge $e=(v_1,v'_1)$, however, the probability that ω^5_3 in a random sequence, in which exactly one of the first two substrings of length l is q-quality, given that ω^4_2 is represented by v_1, is $\frac{1}{2}$. The experiments in Section 5 show that the Markovian assumption yields an approximation sufficiently useful for setting σ_t appropriately.

Our dynamic programming algorithm has a polynomial time complexity. In each iteration, for each vertex, the algorithm performs no more than $t-l+1$ constant time computations, each involving the summation of at most 4 values multiplied by a known factor. Since there are $t-l$ iterations and $2l$ vertices, the overall time complexity of the algorithm is $O(t^2l)$.

4.3 Approximate bounds for the gap distribution

It remains to study the distribution of the random variable M, defined as the maximum number of consecutive substrings of length l that are not (q,l)-runs in a random binary sequence of length t that contains at least σ (q,l)-runs. The distribution of M is needed for setting the threshold δ_t, but it can not be computed exactly sufficiently fast.

We approximate this distribution using the expressions D_k^j computed via Eq.(1). Consider a path r of length $a = t-l$ in graph G (defined in Section 4.2) that contains at least σ vertices in Q. Any such path r can be regarded as a concatenation of three possibly-empty paths r_1, r_2 and r_3, where r_2 is the longest sub-path in r that contains only vertices in $\overline{Q}$, namely, vertices that are not q-vertices. Let $\hat{M}$ be a random variable denoting the length of r_2 and let $\hat{S}$ be a random variable denoting the number of q-vertices in r, which clearly equals the number of q-vertices in r_1 and r_3. We are interested in an upper bound for the probabilities $P(\hat{M} \geq \delta \mid \hat{S} \geq \sigma)$ for every $0 \leq \delta \leq a - \sigma$.

We note that if r_1 is not empty, then its last vertex is $v'_{q\cdot l}$ (To simplify notation we use v_{ql} to mean $v_{\lceil ql \rceil}$). Similarly if r_3 is not empty, its first vertex is $v'_{q\cdot l}$ with probability $\frac{ql-1}{l-1}$ or $v_{q\cdot l}$ with probability $\frac{(1-q)l}{l-1}$. Hence, the concatenation $r_1 r_3$, with one additional edge, $(v'_{q\cdot l}, v_{q\cdot l})$ or $(v'_{q\cdot l}, v'_{q\cdot l})$, is a path of length $L = a - \hat{M}$ in G. The probability that a path of length i which starts and ends in Q contains at least σ vertices in Q is less than or equal to $\sum_{k=\sigma}^{i} D_{k-2}^{i-2}$. Thus,

$$P(L \leq a-\delta, \hat{S} \geq \sigma) \leq \sum_{i=\sigma}^{a-\delta} P(L=i) \sum_{k=\sigma}^{i} D_{k-2}^{i-2}$$

and since $D_k^{i+1} \geq D_k^i$ for every i and k, we get

$$P(L \leq a-\delta, \hat{S} \geq \sigma) \leq \sum_{i=\sigma}^{a-\delta} P(L=i) \sum_{k=\sigma}^{a-\delta} D_{k-2}^{a-\delta-2}$$

Therefore, since $\sum_{i=\sigma}^{a-\delta} P(L=i) \leq 1$, an upper bound on the probability that r contains a path of length greater than or equal to δ which consists entirely of vertices in $\overline{Q}$ is given by:

$$P(\hat{M} \geq \delta \mid \hat{S} \geq \sigma) = \frac{P(L \leq a-\delta, \hat{S} \geq \sigma)}{P(\hat{S} \geq \sigma)} \leq \frac{\sum_{k \geq \sigma} D_{k-2}^{a-\delta-2}}{\sum_{k \geq \sigma} D_k^a}$$

where $\sum_{k \geq \sigma} D_k^a$ is the probability that r visits at least σ vertices in Q.

The additional time required to compute this formula for each value of δ is $O(t \cdot l)$ since in each iteration of the dynamic programming algorithm presented in Section 4.2, we sum the intermediate expressions $D_k^j(u)$ for every vertex $u \in V \cup V'$ and every $k \geq \sigma$. Since $\delta \leq t-l+1$ it follows that the overall additional time required to set δ_t is $O(t^2 l)$ which does not exceed the time complexity of the dynamic programming algorithm.

Our experiments show that setting the threshold δ_t in ATRHUNTER according to the distribution of $\hat{M}$ is almost as good as setting it according to the distribution of M.

5. EXPERIMENTAL RESULTS

The main goal of the experiments performed is to test the quality of our algorithm's output, as implemented in ATRHUNTER. This quality is measured both by the total number of ATRs found and the number of ATRs found for different motif lengths. We evaluate the quality for both real-world and simulated data. The real-world data sets, for which the complete list of ATRs is not known, consisted of three genomes: chromosome I of yeast (230,203 bp) and the complete genomes of two types of E. coli: K–12 (4,639,221 bp) and O157:H7 (5,498,450 bp). The output quality is compared against TRF described in [6], which we ran on all data sets and the published results of TEIRESIAS on chromosome I of yeast as reported in [42]. The simulated data consisted of ATRs planted into a synthetic sequence of length 100,000 bp, for which we plot the percentage of planted ATRs recovered by ATRHUNTER and, for comparison by TRF, for each motif length. For the simulated data we also compare the run times of the different programs.

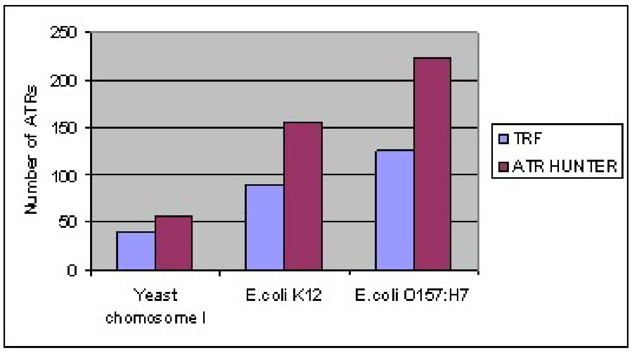


Figure 4: Number of ATRs found by ATRHUNTER and TRF on real data using TRF's definition of an ATR.

Figure 4 shows that ATRHUNTER found an average of 61% more ATRs than TRF in the three genomic sequences, using TRF's definition of an ATR. Two examples for ATRs uniquely detected by ATRHUNTER in these sequences are: a motif of length 116 which repeats twice starting at position 111,400 in the E.coli–K12 genome, and 2.7 copies of a motif of length 33 in the E.coli–O157:H7 genome starting at position 608,919. ATRHUNTER also detected 1189 ATRs in the first chromosome of yeast, using TEIRESIAS's definition of an ATR, compared to 172 found by TEIRESIAS. The differences from other programs are due to the strength of our algorithm rather than a specific counting method. Overlapping ATRs are only counted when reflecting a hierarchy of repeats in a specific region. Even in this case, our counting method applies a cutoff, which limits the number of overlapping ATRs reported per position. The default value of this cutoff used in our experiments is 3, while Figure 5 shows that changing it has negligible effect. ATRHUNTER finds 1.6 times more ATRs than TRF in the three genomic sequences, when comparing the reports of the two programs for non-overlapping regions (*cutoff*=1). Complete outputs from running ATRHUNTER and TRF on the three genomic sequences using TRF's definition of an ATR and the list of ATRs detected by ATRHUNTER using TEIRESIAS definition are provided in the web supplement [46], including an indication of the ATRs that were counted for the purpose of comparison, and an indication of how clustered repeats are counted.

We note that the ATRs found by the other algorithms were almost always ($\sim 99\%$) found by ATRHUNTER as well. It is interesting to examine a few exceptions. For example, in the E.coli–K12 genome, TRF found two copies of a motif of length 18 starting at position 1,253,217 while ATRHUNTER missed this ATR because of its method for choosing the window size l. We further note that the definitions of an ATR

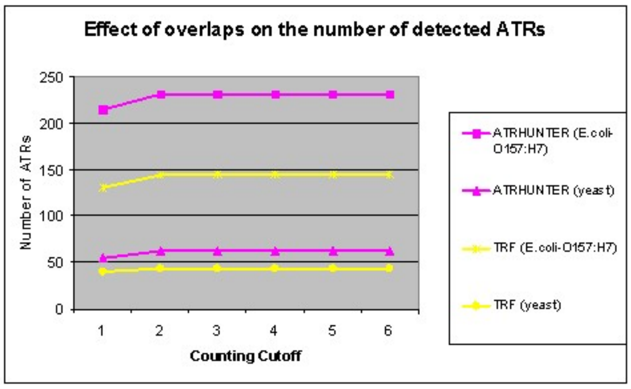


Figure 5: Changes in the number of detected ATRs as a function of the counting cutoff.

in TRF and TEIRESIAS are distinct and different from ours. Therefore, we performed the comparative experiments based on the other programs' ATR definitions. For the comparison with TRF the same experiments were repeated using the four possible setups for parameters allowed by TRF. Similar results are obtained and are available in the web supplement. One important strength of ATRHUNTER is the ability to quickly detect similarity between large regions, even when small regions within are different. This enables the program to find more ATRs with long motifs than other programs. Figure 6 shows the number of ATRs found by ATRHUNTER and TRF as a function of the motif length when searching the E.coli–K12 genome.

On the three genomic sequences shown ATRHUNTER runs in 3, 40, and 45 seconds, respectively, on a PC with 1GB RAM. This is comparable to the TRF running times of 2.5, 22, and 25 seconds. Overall, ATRHUNTER spends 5% less time per ATR found. The second evaluation method uses

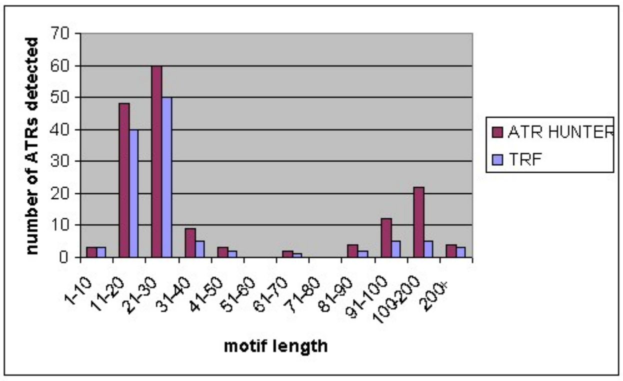


Figure 6: Number of ATRs found by ATRHUNTER and TRF as a function of the motif length on E.coli–K12.

ATRs planted into 10 synthetic i.i.d. sequences of length 100,000 in which each symbol had probability 0.25 of appearing in each position, and plots the percentage of ATRs recovered by ATRHUNTER. We planted 100 ATRs in each sequence with location, motif length and level of similarity randomly chosen, and the number of copies geometrically distributed with parameter $p = 0.5$. The average score of an ATR over all sequences was 238 with a standard deviation 116. We report the success rate in finding the planted ATRs, using TRF's definition of an ATR. Some non-planted

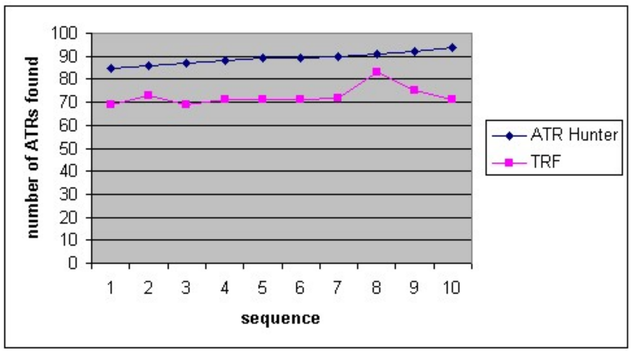


Figure 7: Number of synthetic ATRs found by ATRHUNTER and TRF in 10 random sequences.

ATRs were also detected, but were omitted from the count. When running ATRHUNTER and TRF with the same alignment parameters, the following facts emerge:

- ATRHUNTER found an average of 89% of the planted ATRs with standard deviation 2.6%. TRF found 72% of them with standard deviation 4.6%.
- Almost all the ATRs uniquely found by ATRHUNTER had motif length over 35.
- The running time of both programs was less than 3 seconds for each sequence. TRF is 25% faster.

The number of planted ATRs found by both programs on each sequence is shown in Figure 7, while Figure 8 shows this as a function of motif length. More details of these experiments and further experiments with planted ATRs, where other alignment parameters are used and planted ATRs have different score distributions, are available in the web supplement. We obtained similar results when planting ATRs in genuine genomic sequences rather than the abovementioned random sequences.

We examined the screening precision and the success rate of ATRHUNTER in finding planted ATRs as a function of P_I. The screening precision is the percentage of candidate ATRs that passed verification, representing the efficiency of the screening phase. The success rate is the percentage of planted ATRs reported by ATRHUNTER. The input sequences for these tests were synthetic sequences of length 500,000 each containing 100 planted ATRs, using the default values of the program's parameters. The success rate for detecting ATRs ranges between 74% and 90%, and increases monotonically with P_I. While the success rate is sensitive to P_I, the screening precision is almost oblivious to P_I, remaining at 6% for all $P_I \geq 0.02$. Hence, the run time of verification is almost independent of P_I.

Finally, we tested the accuracy of our approximation of the distribution of the number of q-quality runs of length l in a random binary sequence of length t. The exact distributions, which have been computed exhaustively, are plotted in Figure 9 against the approximated ones, for several sets of

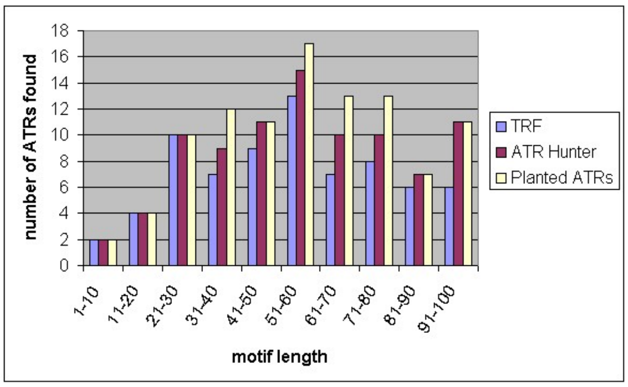


Figure 8: Average number of planted ATRs found by ATRHUNTER and TRF as a function of the motif length in synthetic sequences.

parameters t ,l and q used by the algorithm. The total variation between the distributions is very small (e.g., 0.0059 for $l = 10$) and their similarity is evident.

The penalty in output quality from using the approximated score distributions is quite minor, as can be seen in Figure 10 where the thresholds σ_t obtained from the approximation are compared to those derived from the exact distributions, as a function of l.

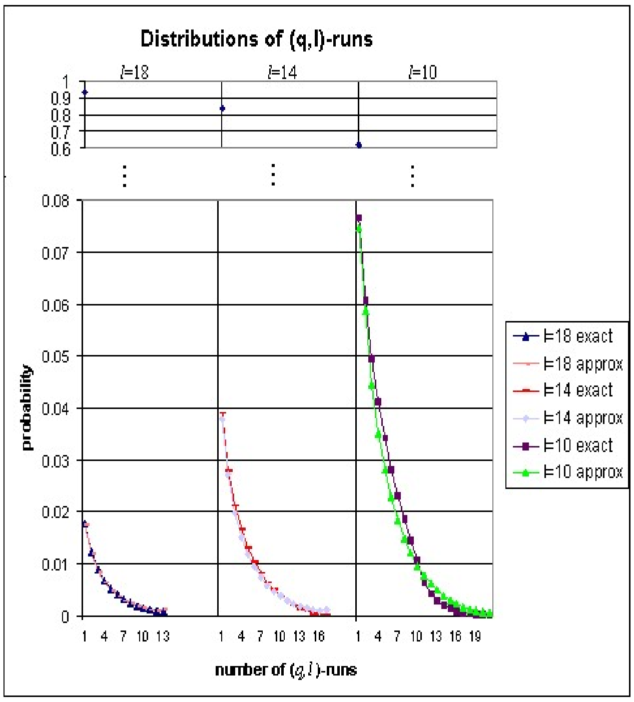


Figure 9: The distribution of (q, l)-runs in a random binary sequence of length $t = 30$, and our approximation of it, for $q = 0.5$ and $l = 18, 14, 10$.

Acknowledgements

We thank Cyril Cohen for his advice at the early stages of this research and Gideon Greenspan for helpful comments.

6. REFERENCES

[1] S. Aki. Waiting time problems for a sequence of discrete random variables. *Annals of the Institute of Statistical Mathematics*, 44:363–378, 1992.

[2] A. Apostolico and F. Prefarata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22(3):297–315, 1983.

[3] R. Arratia, M. Gordon, and S. Waterman. The Erdos-Renyi law in distribution, for coin tossing and sequence matching. *Annals of Statistics*, 18:539–570, 1990.

[4] A. Barbour, O. Chryssafinou, and E. Vaggelatou. *Applications of compound Poisson approximation.* Chapman and Hall, Boca Raton, 2001.

[5] J. Beckman and M. Soller. Toward a unified approach to genetic mapping of eukaryotes based on sequence tagged microsatellite sites. *Biotechnology*, 8:930–932, 1990.

[6] G. Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acid Research*, 27:573–580, 1999.

[7] G. Benson and X. Su. On the distribution of k-tuple matches for sequence homology: a constant time exact calculation of the variance. *Journal of Computational Biology*, 5:87–100, 1998.

[8] P. Bois and A. Jeffreys. Minisatellite instability and germline mutation. *Cellular and Molecular Life Science*, 55:1636–1648, 1999.

[9] A. Bowcock, A. Ruiz-Linares, J. Tomfohrde, E. Minch, J. Kidd, and L. Cavalli-Sforza. High resolution of human evolutionary trees with polymorphic microsatellites. *Nature*, 368:455–457, 1994.

[10] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12:244–250, 1981.

[11] M. Dayhoff, R. Schwartz, and B. Orcutt. A model for evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:345–352, 1978.

[12] O. Delgrange, and E. Rivals. STAR: an algorithm to Search for Tandem Approximate Repeats. *Bioinformatics*, 20(16):2812–2820, 2004.

[13] W. Dietmaier, W. Riedlinger, A. Köhler, P. Wegele, K. Beyser, G. Sagner, R. Wartbichler, and J. Rüschoff. Detection of microsatellite instability (msi) and loss of heterozygosity (loh) in colorectal tumors by fluorescence-based multiplex microsatellite PCR. *Biochemica*, 2:42–45, 1999.

[14] N. Dokhylyan, S. Buldyrev, S. Halvin, and H. Stanley. Model of unequal chromosomal crossing over in DNA sequences. *Physica A*, 249:594–599, 1998.

[15] W. Feller. *An introduction to probability: Theory and its applications*, volume 1. John Wiley & Sons, New York, NY, 1968.

[16] P. L. Flèche, Y. Hauck, L. Onteniente, A. Prieur, F. Denoeud, V. Ramisse, P. Sylvestre, G. Benson, F. Ramisse, and G. Vergnaud. A tandem repeats database for bacterial genomes: application to the genotyping of yersinia pestis and bacillus anthracis. *BMC Microbiology*, 1:2, 2001.

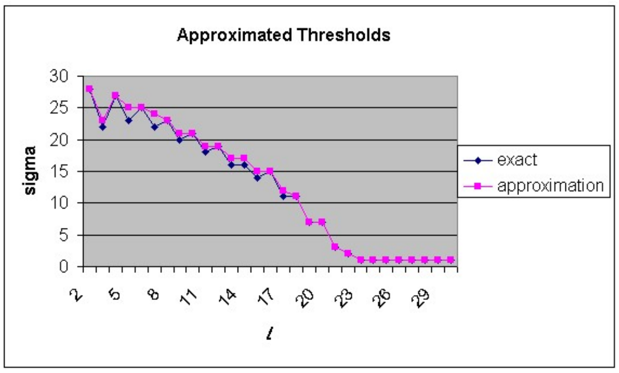


Figure 10: The difference in σ_{30} calculated from the exact and approximated distributions.

[17] H. D. C. R. Group. A novel gene containing a trinucleotide repeat that is expanded and unstable on huntington's disease chromosomes. *Cell*, 72:971–983, 1993.

[18] R.P. Grewal, M. Achari, T. Matsuura, A. Durazo, E. Tayag, L. Zu, S. Pulst, T. Ashizawa. Clinical features and ATTCT repeat expansion in spinocerebellar ataxia type 10. *Arch. Neurol.* 59:1285–1290, 2002.

[19] X. Guan and E. Uberbacher. A fast look-up algorithm for detecting repetitive DNA sequences. *Proceedings of the Pacific Symposium on Biocomputing 1996*, pages 718–719, 1996.

[20] A. Hauth and D. Joseph. Beyond tandem repeats: complex pattern structures and distant regions of similarity. *Bioinformatics*, 18:s31–s37, 2002.

[21] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences (PNAS)*, 89(22):10915–10919, 1992.

[22] K. Inman and N. Rudin. *An introduction to forensic DNA analysis*. CRC press, Boca Raton, Florida, 1997.

[23] Y. Ionov, M. Peinado, S. Malkhosyan, D. Shibata, and M. Perucho. Ubiquitous somatic mutations in simple repeated sequences reveal a new mechanism for clonic carcinogenesis. *Nature*, 363:558–561, 1993.

[24] A. Jeffreys, D. Monckton, K. Tamaki, D. Neil, J. Armour, A. MacLeod, A. Collick, M. Allen, and M. Jobling. Minisatellite variant repeat mapping: application to DNA typing and mutation analysis. In *DNA Fingerprinting: State of the Science*, pages 125–139, Basel, 1993. Birkhauser.

[25] S. Kannan and E. Myers. An algorithm for locating non-overlapping regions of maximum alignment score. *SIAM Journal on Computing*, 25(3):648–662, 1996.

[26] Y. Kashi, D. King, and M. Soller. Simple sequence repeats as a source of quantitative genetic variation. *Trends in Genetics*, 13:74–78, 1997.

[27] D. King, M. Soller, and Y. Kashi. Evolutionary turning knobs. *Endeavour*, 21:36–40, 1997.

[28] R. Kolpakov, G. Bana, and G. Kucherov. mreps: efficient and flexible detection of tandem repeats in DNA sequences. *Nucleic Acid Research*, 31:3672–3678, 2003.

[29] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. *FOCS 1999*, pages 596–604, 1999.

[30] R. Kolpakov and G. Kucherov. Finding repeats with fixed gap. *Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, pages 162–168, 2000.

[31] R. Kolpakov and G. Kucherov. Finding approximate repetitions under hamming distance. *ESA 2001*, pages 170–181, 2001. Also in *LNCS*, 1(303):135–156, 2003.

[32] A. Krishnan, and F. Tang. Exhaustive whole-genome tandem repeats search. *Bioinformatics*, 20(16):2702–2710, 2004.

[33] S. Kurtz, J. Choudhuri, E. Ohlebush, C. Chleiermacher, J. Stoye, and R. Giegerich. Reputer: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acid Research*, 29:4633–4642, 2001.

[34] G. Landau, J. Schmidt, and D. Sokol. An algorithm for approximate tandem repeats. *Journal of Computational Biology*, 8:1–18, 2001.

[35] Y. Li, A. Korol, T. Fahima, A. Beiles, and E. Nevo. Microsatellites: genomic distribution, putative functions and mutational mechanisms. *Molecular Ecology*, 11:2453–2465, 2002.

[36] M. Main and R. Lorentz. An o(n log n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.

[37] D. Nebert and E. Bingham. Pharmacogenomics: out of the lab and into the community. *Journal of Korean Medical Science*, 19(12), 2001.

[38] V. Parisi, V. D. Fonzo, and F. Aluffi-Pentini. String: finding tandem repeats in DNA sequences. *Bioinformatics*, 19:1733–1738, 2003.

[39] E. Rivals and O. Delgrange. A first step toward chromosome analysis by compression algorithms. *First International IEEE Symposium on Intelligence in Neural and Biological Systems, IEEE Computer Society Press*, pages 233–239, 1995.

[40] M. Sagot and E. Myers. Identifying satellites and periodic repetitions in biological sequences. *Journal of Computational Biology*, 5(3):539–554, 1998.

[41] M. Stewart and R. Denell. The drosophila ribosomal protein s6 gene includes a 3' triplication that arose by unequal crossing-over. *Molecular Biology and Evolution*, 10(5):1041–1047, 1993.

[42] G. Stolovitzky, Y. Gao, A. Floratos, and I. Rigoutsos. Tandem repeat detection using pattern discovery, with applications to identification of yeast satellites. *IBM T.J.Watson Research Center*, 1999.

[43] J. Stoye, and D. Gusfield. Simple and Flexible Detection of Contiguous Repeats Using a Suffix Tree. *Theoretical Computer Science*, 270:843–856, 2002.

[44] A. Verkerk, M. Pieretti, J. Sutcliffe, Y. Fu, D. Kuhl, A. Pizzuti, O. Reiner, S. Richards, M. Victoria, F. Zhang, B. Eussen, G. Vanommen, L. Blonden, G. Riggins, C. Kunst, H. Galjaard, C. Caskey, D. Nelson, B. Oostra, and S. Warren. Identification of a gene (fmr-1) containing a cgg repeat coincident with a breakpoint cluster region exhibiting length variation

in fragile x syndrome. *Cell*, 65:905–914, 1991.

[45] K. Virtaneva, E. D'Amato, J. Miao, M. Koskiniemi, R. Norio, G. Avanzini, S. Franceschetti, R. Michelucci, C. Tassinari, S. Omer, L.A. Pennacchio, R.M. Myers, J. Dieguez-Lucena, R. Krahe, A. de la Chapelle, A. Lehesjoki. Unstable minisatellite expansion causing recessively inherited myoclonus epilepsy, EPM1. *Nature Genet.*, 15:393–396, 1997.

[46] Y. Wexler, Z. Yakhini, Y. Kashi, and D. Geiger. Web supplement: `http://bioinfo.cs.technion.ac.il/ATRHunter`.

[47] M. Waterman. *Introduction to computational biology.* Chapman & Hall, London, 1995.

[48] K. J. Woo, K. Sang-Ho, and C. Jae-Kwan. Association of the dopamine transporter gene with Parkinson's disease in Korean patients. *Journal of Korean Medical Science*, 15(4), 2000.