In Proc. of the International Conference on Intelligent Systems for Molecular Biology (ISMB2000), AAAI-Press, pp. 228-238, August 2000

Computation and Visualization of Degenerate Repeats in Complete Genomes

Stefan Kurtz[*] **Enno Ohlebusch**[*] **Chris Schleiermacher**[*]
Jens Stoye[†] **Robert Giegerich**[*]

Abstract

The repetitive structure of genomic DNA holds many secrets to be discovered. A systematic study of repetitive DNA on a genomic or inter-genomic scale requires extensive algorithmic support. The *REPuter* family of programs described herein was designed to serve as a fundamental tool in such studies. Efficient and complete detection of various types of repeats is provided together with an evaluation of significance, interactive visualization, and simple interfacing to other analysis programs.

Keywords: Genome, Degenerate Repeats, Efficient Algorithms, Software Tool, Visualization

Introduction

One of the most striking features of DNA is the extent to which it consists of repeated substrings. This is particularly true of eukaryotes. For example, most of the human Y chromosome consists of repeated substrings, and it is estimated that families of reiterated sequences account for about one third of the human genome (McConkey 1993). The presence of palindromic repeats hints to the formation of hairpin structures that may provide some structural or replicational mechanism (Huang *et al.* 1998). Furthermore some repeats have been shown to affect bacterial virulence by acting as the molecular basis of a mechanism used to successfully colonize and infect different human individuals (van Belkum *et al.* 1997). These properties make repeats an interesting research topic, and indeed, there is a vast literature on repetitive structures and their hypothesized functional and evolutionary role.

[*]Faculty of Technology, University of Bielefeld, P.O. Box 10 01 31, 33501 Bielefeld, Germany, corresponding author: Stefan Kurtz, Email: kurtz@techfak.uni-bielefeld.de, Phone: +49 521 106 2906, FAX: +49 521 106 6411

[†]German Cancer Research Center (DKFZ), Theoretical Bioinformatics (H0300), Im Neuenheimer Feld 280, 69120 Heidelberg, Germany

Repeat Analysis on a Genomic Scale

A tool for the systematic study of the repetitive structure of complete genomes must satisfy the following criteria:

Efficiency The size of the genomes to be studied ranges up to 3-4 billion base pairs. To do a complete analysis, algorithmic efficiency must be practically linear, both in terms of computer memory and execution time.

Flexibility and Significance While exact repeats often give a first hint at the overall repetitive structure, a biologically realistic model must recognize degenerate repeats, which allow a certain rate of error. Flexibility also requires to recognize not just direct repeats, but also palindromic repeats, and other sequence features closely related. In the presence of errors, the significance of a particular pattern is not easily judged, and a statistical assessment of significance is mandatory.

Interactive Visualization Since a large amount of data is generated, interactive visualization is required. Human investigators need to obtain an overview on a whole genome or chromosome basis, but also must be able to zoom in on the details of a particular repetitive region.

Compositionality In the long run, we expect that repeat finding is only a basic step in explaining genome structure. Further analysis will be built on top of the repeat finding. Hence, the repeat finding program must provide a simple interface to enable composition with such advanced analysis programs.

The *REPuter* program family described herein satisfies these requirements in the following way: *REPfind* uses an efficient and compact implementation of suffix trees in order to locate exact repeats in linear space and time. It has been estimated in (Kurtz 1999) that this time-critical task can be done in linear time for sequences up to the size of the human genome. These exact repeats are used as seeds from which significant degenerate repeats are constructed allowing for mismatches, insertions, and deletions. Note that our program is not heuristic: it guarantees to find all degenerate repeats as specified by the parameters. Output size

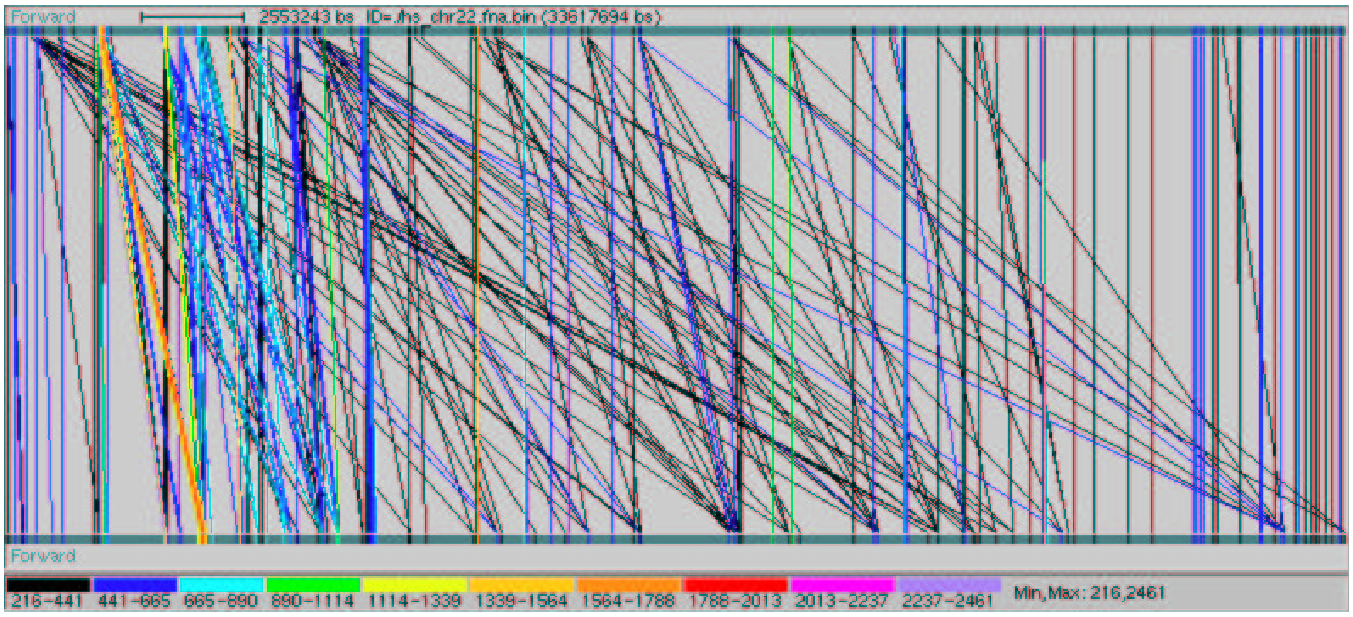


Figure 1: A view of the repeat structure of *Homo sapiens*, chromosome 22; see (Dunham *et al.* 1999). Current gaps in the chromosome sequence are being ignored. Degenerate direct repeats down to a length of 308 bases are shown. The most significant one is a repeat of length 2461. The fat shaded line near the left indicates a region rich of moderate-length repeats (1000-2000 bases), which calls for closer inspection via a zoom function, see Figure 6.

can be controlled via parameters for minimum length and maximum error. Output is sorted by significance scores (E-values) calculated according to the distance model used. *REPfind* produces voluminous output in a fixed format.

REPvis visualizes the output from *REPfind*; see Figures 1, 4, 5, and 6. A color-code indicates significance scores, and a scroll bar controls the amount of data displayed. A zooming function provides whole genome views as well as detailed presentations of selected regions.

REPselect allows to select interesting repeats from the output of *REPfind* as specified by user-defined criteria. It delivers a list of repeats of chosen length, degeneracy or significance into further analysis routines.

REPuter is available at our Bioinformatics web server under the following address: `http://BiBiServ.TechFak.Uni-Bielefeld.DE/reputer/`.

Related Work

Apart from many articles on finding exact repeats in a string, there exists a considerable number of papers that deal with the detection of degenerate repeats (which are called approximate repeats in the stringology literature). The methods generally divide into two groups: exact methods (Fitch, Smith, & Breslow 1986; Leung *et al.* 1991; Landau & Schmidt 1993; Benson 1994; Kannan & Myers 1996; Schmidt 1998; Sagot 1998) which (like ours) first formally define a model of a repeat, and then locate all regions in a given sequence which satisfy this definition, and heuristic methods (Benson & Waterman 1994; Agarwal & States 1994; Rivals *et al.* 1997; Benson 1999; Babenko *et al.* 1999; Vincens *et al.* 1998) which cannot guarantee to find all repeats under some specific model. We do not discuss the heuristic methods here.

The first paper that dealt with model-based recognition of degenerate repeats solved the problem of finding the highest-scoring pair of (possibly overlapping) substrings in a string (Fitch, Smith, & Breslow 1986) in $O(n^2)$ time and space, where n is the length of input string. (Kannan & Myers 1996; Benson 1994) restrict to pairs of non-overlapping substrings. Both algorithms run in $O(n^2 \log^2 n)$ time. The space usage in (Kannan & Myers 1996) is $O(n^2 \log n)$, which was improved in (Benson 1994) to $O(n^2)$.

A related question is to find degenerate tandem repeats, i.e., repeats where the two copies immediately follow each other in the string. (Landau & Schmidt 1993) study the problem of finding all tandem repeats whose Hamming distance is below a threshold k and present an algorithm that solves this problem in $O(nk \log(n/k))$ time. Another algorithm in that paper allows to find all tandem repeats whose edit distance is below k in $O(kn \log k \log(n/k))$ time. The algorithm by (Schmidt 1998) solves the more general problem of finding all "locally optimal" (non-extendable) repeats (both tandem and non-tandem) under a general alignment score in $O(n^2 \log n)$ time and $O(n^2)$ space. The algorithm is based on a general method to find all highest scoring paths in weighted grid graphs.

A different problem definition was used in (Waterman, Arratia, & Galas 1984; Sagot *et al.* 1995; Sagot 1998). They locate repeats which occur a minimum number q of times, where each occurrence has a maximum Hamming distance e to a repeat "model" (which may itself never exactly occur in the sequence). While the algorithms in (Waterman, Arratia, & Galas 1984; Sagot *et al.* 1995) are formulated such that the repeat must be common to several sequences, the algorithm by (Sagot 1998) also allows to find a repeat that multiply occurs in the same string. Sagot's algorithm uses the suffix tree for preprocessing the sequence and runs in time exponential in the number of errors.

(Sagot & Myers 1998) present an algorithm for finding tandem arrays (multiple occurrences of substrings similar to a common model in a row). Their approach is limited because the approximate pattern size (which is limited to at most 40 bases) and a range for the number of copies have to be specified in advance.

Another model for degenerate repeats is used by (Leung *et al.* 1991), who do not apply one of the standard distance measures normally used in biological sequence comparison. They define a repeat by an exactly matching "core block" of a certain length, which can be extended on both sides by short mismatching regions, so-called "error blocks", followed by matching "extension blocks". A repeat is reported if (in their terminology) the "printing criteria" are fulfilled, which are a number of parameters to the program: a minimal length for the core block, maximal lengths of the error blocks, and a minimal total length of the matching blocks. The model, while well defined, is only described in an operational way, and it is difficult to compare the output of their program to what the other approaches based on standard distance measures would find. That is why this approach has also been classified by other authors as a heuristic method.

To avoid confusion, we would like to point out that in the biological literature there is often a third kind of repeat finding programs, like RepeatMasker (unpublished, `http://ftp.genome.washington.edu/RM/RepeatMasker.html`). Here, a "repeat" is a substring that is known to occur very often in a genome. Such substrings tend to confuse sequence analysis programs, and hence they are masked to avoid spurious results. Such repeat masking programs use a dictionary of known repeat sequences and perform an exact or approximate string matching of the given sequence against all the dictionary entries. Additionally, some of the programs identify "low complexity regions", which more closely meet our notion of a repeat, but usually are limited to be very short or only find special patterns like the same character occurring several times in a row.

In all this work either the methods are restricted to small input or they do not implement the full model of degenerate repeats. *REPuter* provides the first solution to repeat analysis of complete genomes.

Basic Notions

Let S be a string of length $|S| = n$ over an alphabet Σ. $S[i]$ denotes the ith character of S, for $i \in [1, n]$. S^{-1} denotes the reverse of S. For $i \leq j$, $S[i, j]$ denotes the substring of S starting with the ith and ending with the jth character of S. Substring $S[i, j]$ is denoted by the *pair of positions* (i, j). The length of the substring (i, j) is $\ell(i, j) = j - i + 1$. To refer to the characters to the left and right of every character in S without worrying about the first and last character, we define $S[0]$ and $S[n + 1]$ to be two distinct characters not occurring anywhere else in S.

A pair of positions (i_1, j_1), $i_1 \leq j_1$ *contains* a pair (i_2, j_2), $i_2 \leq j_2$, if and only if $i_1 \leq i_2$ and $j_2 \leq j_1$. A pair (p_1, p_2) of substrings (i.e. a pair of pairs of positions) *contains* a pair (p_3, p_4) of substrings if and only if p_1 contains p_3 and p_2 contains p_4.

A pair of substrings $R = ((i_1, j_1), (i_2, j_2))$ is an *exact repeat* if and only if $(i_1, j_1) \neq (i_2, j_2)$ and $S[i_1, j_1] = S[i_2, j_2]$. The length of R is $\ell(R) = j_1 - i_1 + 1 = j_2 - i_2 + 1$. An exact repeat is *maximal* if it is not contained in any other exact repeat. Clearly, an exact repeat $R = ((i_1, j_1), (i_2, j_2))$ is maximal if and only if $S[i_1 - 1] \neq S[i_2 - 1]$ and $S[j_1 + 1] \neq S[j_2 + 1]$.

If S is a DNA-sequence, then we distinguish between two kinds of biologically interesting repeats. The repeats defined above are called *direct repeats* or *forward repeats*. A pair of substrings $P = ((i_1, j_1), (i_2, j_2))$ is a *palindromic repeat* or *reverse complemented repeat* if and only if $S[i_1, j_1] = \overline{S[i_2, j_2]}$, where $\overline{w}$ denotes the reverse complement of a DNA-sequence w. P is *maximal* if the complement of base $S[i_1 - 1]$ is different from $S[j_2 + 1]$ and the complement of base $S[j_1 + 1]$ is different from $S[i_2 - 1]$.

The *Hamming distance* of two equal-length strings S_1 and S_2, denoted by $d_H(S_1, S_2)$, is the number of positions where S_1 and S_2 differ.

There are three kinds of edit operations: *deletions*, *insertions*, and *mismatches* of single characters. The *edit distance* or *Levenshtein distance* of S_1 and S_2, denoted by $d_E(S_1, S_2)$, is the minimum number of edit operations needed to transform S_1 into S_2.

Models and Algorithms

It is well known (Gusfield 1997) that maximal exact repeats can be computed in linear time using the suffix tree of S. (Delcher *et al.* 1999) and (Kurtz 1999) independently showed how to practically construct suffix trees for genomic-size sequences. The space efficient implementation techniques developed in (Kurtz 1999) were the basis of the first *REPuter* program for finding exact repeats (Kurtz & Schleiermacher 1999). This subtask of our new algorithms is not discussed further.

We will present algorithms for finding degenerate repeats based on two different distance models: the Hamming distance model and the edit distance model. In the following, we assume that an error threshold $k \geq 0$ and a length threshold $l > 0$ is given.

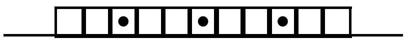


Figure 2: $k = 3$ mismatching characters (denoted by bullets) distributed equally over a repeat of length 11, yielding a minimal seed size of $\lfloor \frac{11}{4} \rfloor = 2$.

The Mismatches Repeat Problem

k-mismatch repeats are based on the notion of Hamming distance.

Definition 1 A pair of equal-length substrings $R = ((i_1, j_1), (i_2, j_2))$ is a *k-mismatch repeat* if and only if $(i_1, j_1) \neq (i_2, j_2)$ and $d_H(S[i_1, j_1], S[i_2, j_2]) = k$. The length of R is $\ell(R) = j_1 - i_1 + 1 = j_2 - i_2 + 1$. A k-mismatch repeat is *maximal* if it is not contained in any other k-mismatch repeat.

As with exact repeats, a k-mismatch repeat $R = ((i_1, j_1), (i_2, j_2))$ is maximal if and only if $S[i_1 - 1] \neq S[i_2 - 1]$ and $S[j_1 + 1] \neq S[j_2 + 1]$.

The *Mismatches Repeat Problem* is to enumerate all maximal k-mismatch repeats of length at least l that occur in S. Our algorithm MMR for solving this problem is based on the following lemma.

Lemma 1 Every maximal k-mismatch repeat R of length l contains a maximal exact repeat of length $\geq \left\lfloor \frac{l}{k+1} \right\rfloor$, called a *seed.*

Proof: In order to prove the lemma, let $R = ((i_1, j_1), (i_2, j_2))$ be a k-mismatch repeat. The k mismatches divide $S[i_1, j_1]$ and $S[i_2, j_2]$ into maximal exact repeats $w_0, w_1, w_2, \ldots, w_k$. The exact repeats w_0 and w_k occurring at the borders of the strings are maximal because R is maximal; the others are obviously maximal. Now $\max_{i \in [0,k]} |w_i|$ is minimal if the mismatching character pairs are equally distributed over R, yielding a pattern as shown in Figure 2. Obviously, for such an equal distribution the length of the longest w_i is $\geq \left\lceil \frac{l-k}{k+1} \right\rceil = \left\lfloor \frac{l}{k+1} \right\rfloor$.

Algorithm MMR Compute all seeds and test for each seed whether it can be extended to a k-mismatch repeat. More precisely, for each seed $((i_1, j_1), (i_2, j_2))$ tables T_{left} and T_{right} of size $k+1$ are computed such that for each $q \in [0, k]$:

$$\begin{aligned} T_{right}(q) &= \max\{p \mid d_H(S[j_1 + 1, j_1 + p], \\ &\qquad S[j_2 + 1, j_2 + p]) = q\} \\ T_{left}(q) &= \max\{p \mid d_H(S[i_1 - p, i_1 - 1], \\ &\qquad S[i_2 - p, i_2 - 1] = q\}. \end{aligned}$$

For each $q \in [0, k]$, if $j_1 - i_1 + 1 + T_{left}(q) + T_{right}(k-q) \geq l$, then output the maximal k-mismatch repeat $((i_1 - T_{left}(q), j_1 + T_{right}(k - q)), (i_2 - T_{left}(q), j_2 + T_{right}(k - q)))$.

Using Lemma 1, it is easy to prove that Algorithm MMR correctly solves the Mismatches Repeat Problem.

Table T_{right} can be computed in $O(k)$ time by using a suffix tree that allows to determine the length of the longest common prefix of two substrings of S in constant time. Since we construct the suffix tree of S anyway, this imposes virtually no overhead. Of course, the same approach can be applied to T_{left}. For details on this technique see (Harel & Tarjan 1984; Schieber & Vishkin 1988).

Algorithm MMR detects a maximal k-mismatch repeat more than once if it contains more than one seed. This can be avoided by stopping the computation of table T_{left} as soon as another seed is detected. This ensures that for a given seed the algorithm will output only those maximal k-mismatch repeats in which this particular seed is the leftmost.

The Differences Repeat Problem

We now extend our technique to allow for insertions and deletions.

Definition 2 A pair $R = ((i_1, j_1), (i_2, j_2))$ of substrings is a *k-differences repeat* if and only if $(i_1, j_1) \neq (i_2, j_2)$ and $d_E(S[i_1, j_1], S[i_2, j_2]) = k$. The *length* of R is $\ell(R) = \min\{j_1 - i_1 + 1, j_2 - i_2 + 1\}$. A k-differences repeat is *maximal* if it is not contained in any other k-differences repeat.

If $R = ((i_1, j_1), (i_2, j_2))$ is a k-differences repeat then $S[i_1 - 1] \neq S[i_2 - 1]$ and $S[j_1 + 1] \neq S[j_2 + 1]$ does *not* imply that R is maximal. This is in stark contrast to exact and k-mismatch repeats. Consider for instance the sequence $AC\underline{TTC}G\underline{CTTC}A$, where $l = 3$ and $k = 1$. Then $((3, 5), (7, 10))$ is a 1-difference repeat and $S[2] = C \neq G = S[6]$ as well as $S[6] = G \neq A = S[11]$. However, $((3, 5), (7, 10))$ is not maximal because it is e.g. contained in the 1-difference repeat $((1, 5), (6, 10))$.

The *Differences Repeat Problem* is to enumerate all maximal k-differences repeats of length at least l.

It can be shown that Lemma 1 also holds for k-differences repeats:

Lemma 2 Every maximal k-differences repeat R of length l contains a maximal exact repeat of length $\geq \left\lfloor \frac{l}{k+1} \right\rfloor$, called a *seed.*

Our algorithm for enumerating all k-differences repeats also crucially depends on Lemma 2.

Definition 3 Let U and V be strings of length m and n, respectively. For $q \in [0, k]$ define:

1. $lookright_E(U, V, q)$ is the set of all pairs $(x, y) \in [1, m] \times [1, n]$ which are maximal with respect to $d_E(U[1, x], V[1, y]) \leq q$.
2. $lookleft_E(U, V, q) = lookright_E(U^{-1}, V^{-1}, q)$

Here the pair (x, y) is called *maximal with respect to* $d_E(U[1, x], V[1, y]) \leq q$ if and only if:

- $d_E(U[1, x + 1], V[1, y]) > q$ if $x < n$,
- $d_E(U[1, x], V[1, y + 1]) > q$ if $y < m$, and
- $d_E(U[1, x + 1], V[1, y + 1]) > q$ if $x < n$ and $y < m$.

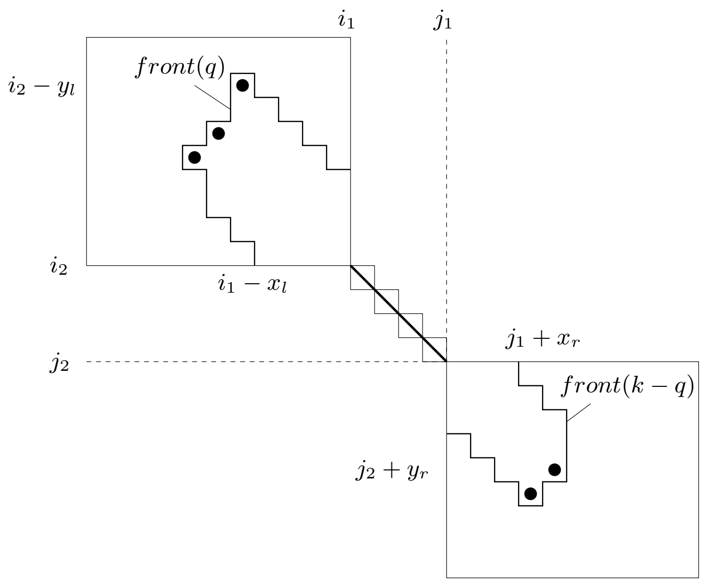


Figure 3: Extension of a seed in Algorithm MDR. The elements of $T_{left}(q)$ and $T_{right}(k-q)$ are marked by bullets.

Algorithm MDR Compute all seeds and try to extend these to k-differences repeats as shown in Figure 3. To be more precise, for every seed $((i_1, j_1), (i_2, j_2))$ compute tables T_{left} and T_{right} defined as follows:

$$\begin{aligned} T_{right}(q) &= lookright_E(S[j_1+1,n], S[j_2+1,n], q) \\ T_{left}(q) &= lookleft_E(S[1,i_1-1], S[1,i_2-1], q). \end{aligned}$$

For each $q \in [0,k]$, for each pair $(x_l, y_l) \in T_{left}(q)$, and each $(x_r, y_r) \in T_{right}(k-q)$: if $j_1 - i_1 + 1 + x_l + x_r \geq l$ and $j_2 - i_2 + 1 + y_l + y_r \geq l$, then output the maximal k-differences repeat $((i_1 - x_l, j_1 + x_r), (i_2 - y_l, j_2 + y_r))$.

Based on Lemma 2, one can show that Algorithm MDR correctly solves the Differences Repeat Problem.

One could of course use a standard dynamic programming algorithm (e.g. (Wagner & Fischer 1974)) to extend seeds in $O(n^2)$ time. However, there are faster methods: using the algorithm of (Ukkonen 1985), it is possible to compute tables T_{left} and T_{right} in $O(kn)$ time by computing only $front(k)$ of the DP-matrix. A combination of this algorithm with the longest common prefix technique yields an $O(k^2)$ time method to compute tables T_{left} and T_{right}.

By restricting to leftmost seeds, Algorithm MDR can be improved in a similar way as Algorithm MMR.

A different approach to search for degenerate repeats would be to initially search for inexact seeds and then to extend these with less errors. However, this approach suffers from the fact that there is no efficient algorithm for finding all inexact seeds, even if the number of errors is very small, see the section on related work.

Before we discuss the overall efficiency of the algorithms, we have to look at the significance of repeats.

Significance of Repeats

In order to assess the significance of a repeat found by our method, we compute its E-value, i.e., the number of repeats of the same length or longer and with the same number of errors or fewer, that one would expect to find in a random DNA of the same length.

As a model of random DNA the Bernoulli model is used, where a base $\alpha \in \{A, C, G, T\}$ has the same fixed probability p_α at each position of the sequence. We will start, however, with an even simpler model, the uniform Bernoulli model, where each base has the same probability of occurrence: $p_\alpha = p = 1/4$ for all α.

We first show how to compute E-values for maximal exact repeats. We use the fact that the number of maximal exact repeats of length $\geq l$ is the same as the number of (only) left-maximal repeats of length exactly l. Ignoring boundary effects, we get:

$$\begin{aligned} &\mathbb{E}[\#\text{ of maximal exact repeats of length} \geq l] \\ &\quad= \mathbb{E}[\#\text{ of left-maximal exact repeats of length } l] \\ &\quad= \sum_{1 \leq i_1 < i_2 \leq n} Pr[S[i_1, i_1+l-1] = S[i_2, i_2+l-1], \\ &\qquad\qquad\qquad S_{i_1-1} \neq S_{i_2-1}] \\ &\quad= \sum_{1 \leq i_1 < i_2 \leq n} p^l(1-p) \\ &\quad= \frac{1}{2}n(n-1)p^l(1-p). \end{aligned}$$

Considering effects at the sequence ends, one obtains in a similar way the following result:

$$\begin{aligned} &\mathbb{E}[\#\text{ of maximal exact repeats of length} \geq l] \\ &\quad= \frac{1}{2}(n-l+1)(n-l)p^l(1-p) + (n-l)p^{l+1}. \end{aligned}$$

Non-uniform Bernoulli Model. One can generalize this result for the non-uniform Bernoulli model by replacing

$$p \quad \text{by} \quad p^* = \sum_{\alpha \in \Sigma} p_\alpha^2.$$

This, however, is only an approximation to the exact solution because the different probabilities for self-overlapping repeats are ignored.

Hamming Distance. E-values for k-mismatch repeats can be computed in a similar way. First, assume fixed values for l and k. The probability of two independent sequences S_1 and S_2, both of length l, to have a Hamming distance of exactly k under the uniform Bernoulli model is

$$Pr[d_H(S_1, S_2) = k] = \binom{l}{k} p^{l-k}(1-p)^k.$$

To compute the expected number of maximal repeats of length l or longer and with k or fewer mismatches, one has to sum over all possible $k' \leq k$ and over all lengths $l' \geq l$. The latter is necessary, in contrast to the case of exact repeats, because for k-mismatch repeats it is no longer true that the number of maximal repeats of length $\geq l$ equals the number of left-maximal repeats of length l. Hence, we obtain:

$$\begin{aligned}
&\mathbb{E}[\#\text{ of maximal} \leq k\text{-mismatch repeats of length} \geq l] \\
&= \sum_{k'=0}^{k}\sum_{l'=l}^{n-1}\sum_{1\leq i_1<i_2\leq n} Pr[d_H(S[i_1, i_1+l'-1], S[i_2, i_2+l'-1]) = k', S_{i_1-1} \neq S_{i_2-1}, S_{i_1+l'} \neq S_{i_2+l'}] \\
&= \frac{1}{2}n(n-1)\sum_{k'=0}^{k}\sum_{l'=l}^{n-1}\binom{l'}{k'}p^{l'-k'}(1-p)^{k'+2}.
\end{aligned}$$

Because the sums are largely dominated by the terms for $k' = k$ and $l' = l$, this can be approximated by

$$\frac{1}{2}n(n-1)\binom{l}{k}p^{l-k}(1-p)^{k+2}.$$

Edit Distance. In the case of the edit distance there does not exist an analytic solution for $Pr[d_E(S_1, S_2) = k]$. For this reason we use the procedure of (Kurtz & Myers 1997) which estimates the probability of the event $A_k(P)$ that an arbitrary (not necessarily random) string P matches the prefix of a random string with edit distance k. This procedure is an unbiased estimator which gives good results in a matter of a thousand samples even for patterns of small probability. To obtain an estimation $Pr[d_E(S_1, S_2) = k]$, we precomputed a table E. Here $E(l, k)$ is the average of the estimation of the probability of the event $A_k(P)$. The estimation is delivered by running the above procedure with 1000 samples for 100 random patterns P, each of length l. The variance of the 100 estimations obtained for each l and k is very small and so we argue that $E(l, k)$ gives a good approximation for $Pr[d_E(S_1, S_2) = k]$ where $l = \max\{|S_1|, |S_2|\}$. Hence we estimate (ignoring boundary effects)

$$\begin{aligned}
&\mathbb{E}[\#\text{ of maximal } k\text{-differences repeats of length } l] \\
&= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(l,k) \\
&= \frac{1}{2}n(n-1)E(l,k).
\end{aligned}$$

Asymptotic Efficiency. The overall time efficiency of Algorithms MMR and MDR can be assessed as follows. The preprocessing phase of computing the suffix tree and locating the seeds takes $O(n)$ time. For a given seed, the extension phase of Algorithm MMR takes $O(k)$ time as shown above, yielding an overall time efficiency of $O(n + zk)$ where z is the number of seeds. The extension phase of Algorithm MDR takes time $O(k^3)$ per seed: As argued above, T_{right} and T_{left} can be computed in $O(k^2)$ time. For each $q \in [0, k]$ the algorithm tests $O(k^2)$ combinations of the values in $T_{left}(q)$ and $T_{right}(k - q)$, yielding an upper bound of $O(k^3)$ per seed. Hence the overall time efficiency of algorithm MDR is $O(n + zk^3)$.

The number of seeds z can be estimated by $\mathbb{E}[z] = O(n^2\frac{1}{|\Sigma|^s})$ where $s = \left\lfloor\frac{l}{k+1}\right\rfloor$ is the length of the seed as calculated above.

Implementation

We implemented Algorithm MMR and Algorithm MDR in the *REPuter* search engine *REPfind*. To detect seeds (i.e. exact repeats) we use the same program as in (Kurtz & Schleiermacher 1999). In MDR seeds are extended by the dynamic programming algorithm of (Ukkonen 1985). For both MMR and MDR, we compute the length of matches by pairwise character comparisons, which is very fast in practice.

Besides degenerate direct repeats, *REPfind* is capable to detect degenerate palindromic repeats. This is achieved by applying Algorithms MMR and MDR to the string $S\#\overline{S}$, where $\overline{S}$ is the reverse complement of S and $\#$ is a unique separator symbol.

To efficiently determine the significance of degenerate repeats we use precomputed tables H and E, where E are the estimations as specified above and $H(l, k) = \binom{l}{k}p^{l-k}(1-p)^{k+2}$ for any l and k. Note that the precomputed values are independent of n. Multiplying them by $\frac{1}{2}n(n-1)$ gives the E-value. Thus an E-value is computed in constant time.

Since Algorithm MMR and MDR are not heuristic, they find all maximal k-mismatch or k-differences repeats exceeding some given length l. However, usually the user only wants to see the most interesting repeats. For this reason, in the default mode, *REPfind* selects repeats according to the following rules:

(1) To be selected, the right instance of a direct repeat has to start at least $k+1$ positions to the right of the left instance of the repeat.

(2) For each seed only the most significant repeat containing that seed is selected. In this way, "clumps" of repeats are represented by only one repeat.

(3) Among all repeats selected according to (1) and (2), *REPfind* reports the b most significant repeats in order of significance. The parameter b can be defined by the user.

The output format of *REPfind* is either ASCII showing each repeat on a single line or a portable binary format. The latter is much more space-efficient and requires no parsing. The program can optionally report the two instances of a repeat in form of an alignment.

The program *REPselect* reads the binary format delivered by *REPfind*. It allows to select repeats according to user defined selection criteria. These are to be specified by the user in form of executable object code that is linked dynamically. Program code for several such selection functions is supplied to aid the user in developing his/her own selection functions.

Performance Results

Table 1 shows the running time and space consumption of *REPfind* when applied to several genomes or chromosomes. The construction of the suffix tree is dominating the running time. It requires more than 70% of the running time. The computation of exact repeats is only slightly faster than the computation of degenerate repeats. This surprising behavior can be explained as follows: To extend a seed, the only data that needs to be processed are two pairs of substrings of the input sequence. This is only a very small amount of data which is processed sequentially. As a consequence, the locality behavior of the extension phase is very good, and therefore it runs very fast. On the other hand, the locality behavior of the suffix tree is very poor, see (Kurtz 1999). That is, the suffix tree traversal leads to many cache misses, and it thus dominates the running time of the repeat searching phase.

The heuristic strategy determines the length parameter l such that we always find degenerate repeats. However, the number of repeats found differs very much, especially for the larger sequences. The number of exact repeats is always much smaller than the number of degenerate repeats. In most cases the number of mismatch repeats is about the same as the number of differences repeats. The remarkable exception is *Drosophila melanogaster* with 4200 mismatch repeats and 6731 differences repeats.

The space requirement for computing the differences repeats is on average about 13.7 bytes per input symbol including the space for the sequence. This is very similar to the space requirement for computing exact repeats, see (Kurtz & Schleiermacher 1999).

Visualization

REPvis, the visualization component of the *REPuter* program family, provides an easy to use interface for examining repeat structures computed by *REPfind*; see Figures 1, 4, 5, and 6. The program is designed to be used by the biologist, thus putting the data in the hands of those who can best interpret it.

A typical mode of use is as follows: The visualization comes up showing a single colored line, depicting either the longest or the most significant repeat. The first step is to obtain an impression of the overall number and distribution of repeats. By shifting a slider, we let further repeats rise on the screen, in the order of decreasing length or significance, which is coded in a ten-color scale (see Figure 4). Since black is used as the color for the shortest/least significant repeats, we may go down all the way: If we hit the noise level, the more significant repeats still shine up in colors before a black background of noise.

During the overview, we may catch interest in particular repeats or repeat-rich regions. A mouse click brings up the inspection window; see Figure 6. Here we can zoom in or out on a region by left or right clicking the mouse. Selecting a position on the strand symbol prints the information corresponding to this sequence position in a browser box below. There, a single repeat can be selected to view the alignment of the two instances of the repeat or to submit the corresponding nucleotide sequence for further investigation of biological significance to a FASTA or BLAST database search. This is achieved by invoking Netscape Navigator with the `-remote` argument, which allows to connect to and initiate the load of the database query data into an already-running Netscape process (Zawinski 1994).

Conclusion

The *REPuter* approach gives a complete account of degenerate direct and palindromic repeats, including significance scores, with an efficiency that allows the analysis of all genomes currently available. It allows inspecting repeats on a macroscopic scale as well as on the sequence level.

Aside from direct and palindromic repeats, *REPuter* also detects linguistic palindromes and forward, but complemented repeats. Although there is no biological mechanism known to produce such patterns, low complexity regions are typically exhibited as self-overlapping occurrences of the four kinds of repeats detected by *REPuter*.

At the moment, visual inspection of repeats found by *REPuter* will be the major mode of application. In the long run, models will need to be developed that explain the manifold aspects of repetitive genome structure. We expect that *REPuter* will serve as a basic vehicle for such research.

Acknowledgments. Eivind Coward, Sven Rahmann, and Rainer Spang helped on statistical issues of this work. Dirk Evers showed us how to dynamically load shared object code into *REPselect*. All their help is very much appreciated.

References

Agarwal, P., and States, D. J. 1994. The Repeat Pattern Toolkit (RPT): Analyzing the structure and evolution of the C. elegans genome. In *Proc. of the Second International Conference on Intelligent Systems for Molecular Biology, ISMB 94*, 1–9. Menlo Park, CA: AAAI Press.

Babenko, V. N.; Kosarev, P. S.; Vishnevsky, O. V.; Levitsky, V. G.; Basin, V. V.; and Frolov, A. S. 1999. Investigating extended regulatory regions of genomic DNA sequences. *Bioinformatics* 15(7/8):644–653.

Benson, G., and Waterman, M. 1994. A method for fast database search for all k-nucleotide repeats. *Nucl. Acids Res.* 22:4828–4836.

Benson, G. 1994. A space efficient algorithm for finding the best nonoverlapping alignment score. In Crochemore, M., and Gusfield, D., eds., *Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching, CPM 94. Asilomar, California, June 1994*, volume 807 of *LNCS*, 1–14. Berlin: Springer Verlag.

Benson, G. 1999. Tandem repeats finder: A program to analyze DNA sequences. *Nucl. Acids Res.* 27(2):573–580.

Genome	n	l	*Tree*	*Exact*		*hdist* ≤ 4		*edist* ≤ 4		*Space*
	(MB)		(sec)	#reps	(sec)	#reps	(sec)	#reps	(sec)	(MB)
Rhizobium sp. NGR234	0.51	120	1.10	9	1.71	11	1.71	13	1.71	7.14
Mycoplasma genitalium	0.55	130	1.19	9	1.84	59	1.89	62	1.90	7.71
Ureaplasma urealyticum	0.72	150	1.64	43	2.42	63	2.47	67	2.53	9.97
Mycoplasma pneumoniae	0.78	130	1.86	74	2.79	409	2.84	449	2.90	10.82
Borrelia burgdorferi	0.87	140	2.10	9	3.22	28	3.23	28	3.27	12.07
Chlamydia trachomatis	0.99	130	2.53	3	3.80	6	3.83	6	3.85	13.82
Chlamydia muridarum	1.02	130	2.64	4	3.91	8	3.94	8	3.98	14.16
Rickettsia prowazekii	1.06	140	2.65	9	4.02	10	4.08	10	4.08	14.71
Treponema pallidum	1.09	130	2.85	33	4.20	48	4.25	51	4.28	15.07
Chlamydo. pneum. AR39	1.17	130	3.16	6	4.63	7	4.66	8	4.67	16.27
Chlamydia pneumoniae	1.17	130	3.13	8	4.62	11	4.65	13	4.70	16.28
Aquifex aeolicus	1.48	140	4.15	12	6.06	22	6.08	23	6.13	20.50
Campylobacter jejuni	1.57	160	4.29	25	6.33	39	6.37	39	6.38	21.71
Methanococcus jannaschii	1.59	150	4.36	23	6.45	48	6.48	62	6.48	22.00
Helicobacter pylori	1.59	150	4.45	45	6.47	84	6.54	100	6.54	22.04
Pyrococcus horikoshii	1.66	140	4.76	3	6.85	3	7.00	3	7.09	22.97
M. thermoautotrophicum	1.67	140	4.79	29	6.98	51	7.00	57	7.16	23.14
Pyrococcus abyssi	1.68	140	4.82	0	5.00	4	7.00	4	7.09	23.32
Haemophilus influenzae	1.75	140	4.99	24	7.34	79	7.34	85	7.42	24.19
Plasmodium falciparum	1.91	240	4.94	46	7.43	107	7.53	126	7.81	26.51
Archaeoglobus fulgidus	2.08	140	6.11	29	8.93	58	8.98	59	8.99	28.77
Deinococcus radiodurans	2.92	170	8.85	35	12.79	41	12.87	47	12.89	40.40
Synechocystis PCC6803	3.41	160	11.27	347	15.64	655	15.68	686	15.82	47.15
Bacillus subtilis	4.02	150	13.61	286	18.80	411	18.86	496	18.88	55.60
M. tuberculosis	4.21	170	13.79	118	19.32	189	19.40	190	19.50	58.19
Escherichia coli	4.42	150	15.18	209	20.66	473	20.89	507	20.98	61.19
Saccharomyces cerevisiae	11.50	180	43.19	3379	58.08	9093	58.49	9571	58.96	158.95
Homo sapiens Chr. 22	32.06	670	136.56	58	185.88	482	186.71	548	187.33	443.04
A. thaliana Chr. 2 and 4	35.47	590	169.06	151	226.43	665	227.30	797	227.64	490.23
Caenorhabditis elegans	92.40	1905	584.76	74	762.44	191	767.31	227	769.86	1277.27
Drosophila melanogaster	114.44	700	737.73	1330	1047.90	4200	1052.52	6731	1053.92	1582.80

Table 1: The running time, the space consumption, and the number of repeats found when applying *REPfind* to several genomes and large chromosomes. The timings are in seconds. The program was run on a SUN-sparc computer under Solaris 2.5.1 with a 400 MHz-Processor and 2 Gigabytes of main memory. The second column shows the length of the genome in megabytes. The third column shows the length parameter l which was chosen according to the following strategy: We count, for each possible d, the number $b(d)$ of branching nodes exactly of depth d in the suffix tree. We then determine the largest d such that $b(d) \geq 10.000$ and set $l = 5 \cdot d \cdot log_{10}(d)$. This heuristic strategy proved to be good since it balances significance and speed. Column four of the table shows the construction time of the suffix tree. The last column shows the overall space requirement (in megabytes) for computing degenerate repeats with at most four differences. The remaining columns show the number of repeats found and the corresponding running time for *REPfind* when computing exact repeats or degenerate repeats with hamming and edit distance at most 4.

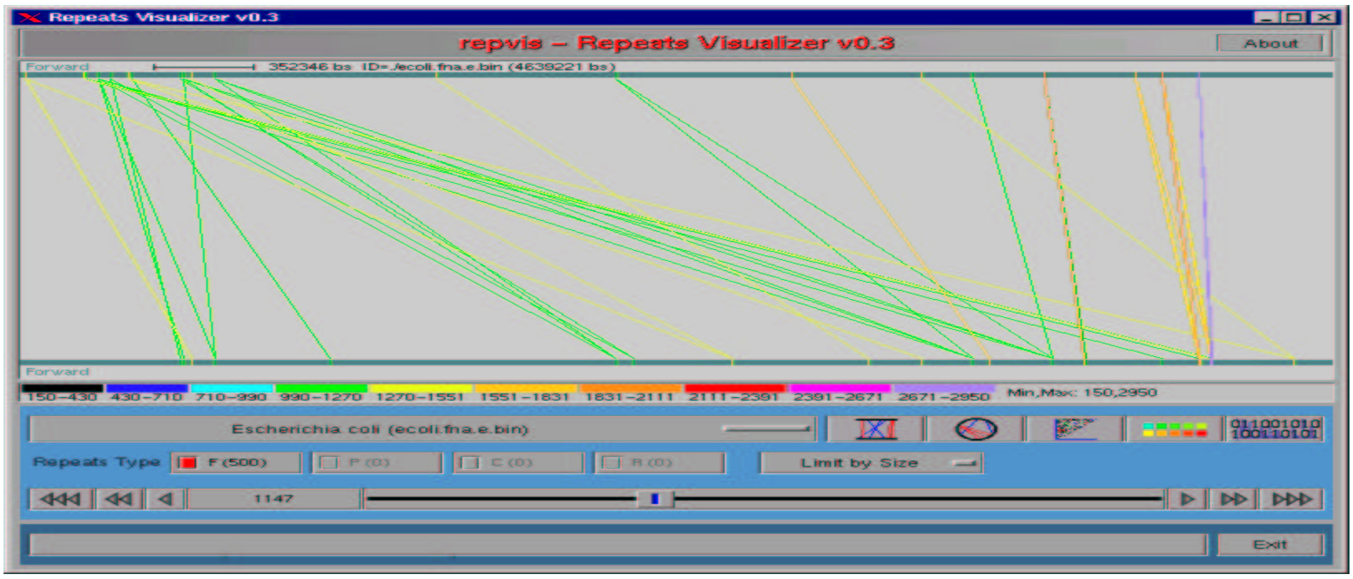


Figure 4: A typical application of *REPvis*, showing a view of the 50 most significant direct repeats in *E. coli* (4.6Mb), ranging from 1147 to 2950 bases in length. There are five repeats longer than the longest one found in *M. tuberculosis*; see Figure 5. In the main window graphics panel, two horizontal lines depict the input sequence and a copy of it. Diagonal lines stand for repeats by connecting their respective starting positions. Below the graphics panel, a choice box lists all calculated sequences in a user specified directory. Three further buttons switch the visualization mode to square graph, circle graph or dot plot. An additional button leads to the complete list of all repeats and their size distribution. Selector buttons specify which type of repeat to display. The symbols F, P, C, and R indicate direct (forward), palindromic (reverse complemented), complemented and reversed repeats; the number of repeats for each type is shown on the button.

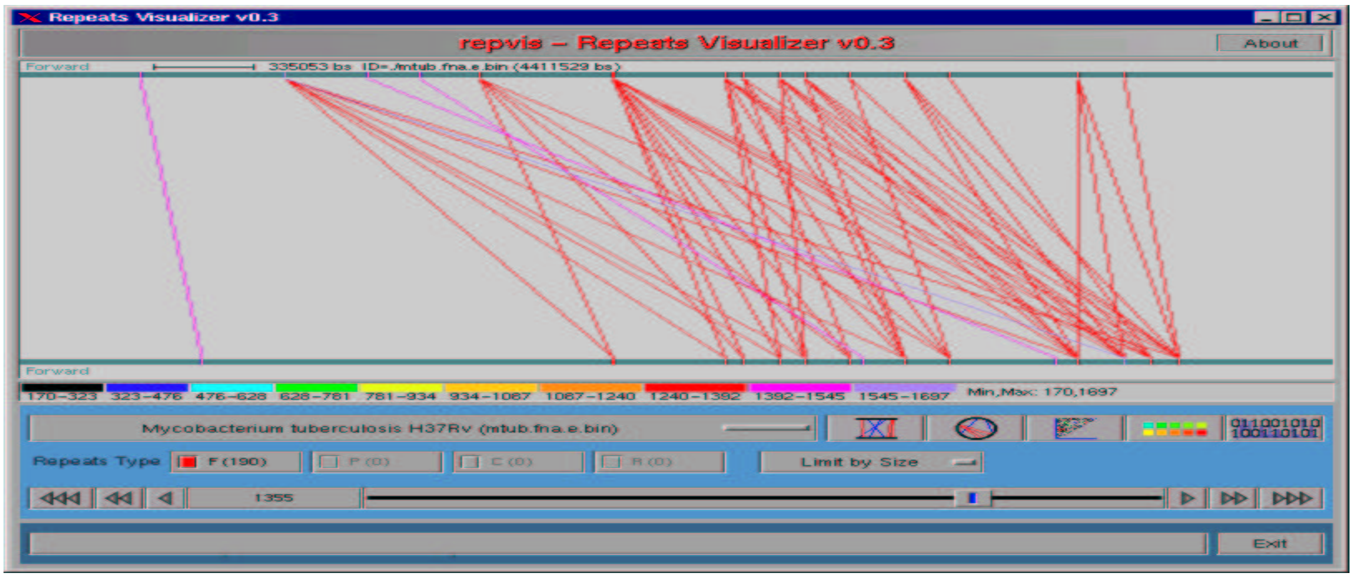
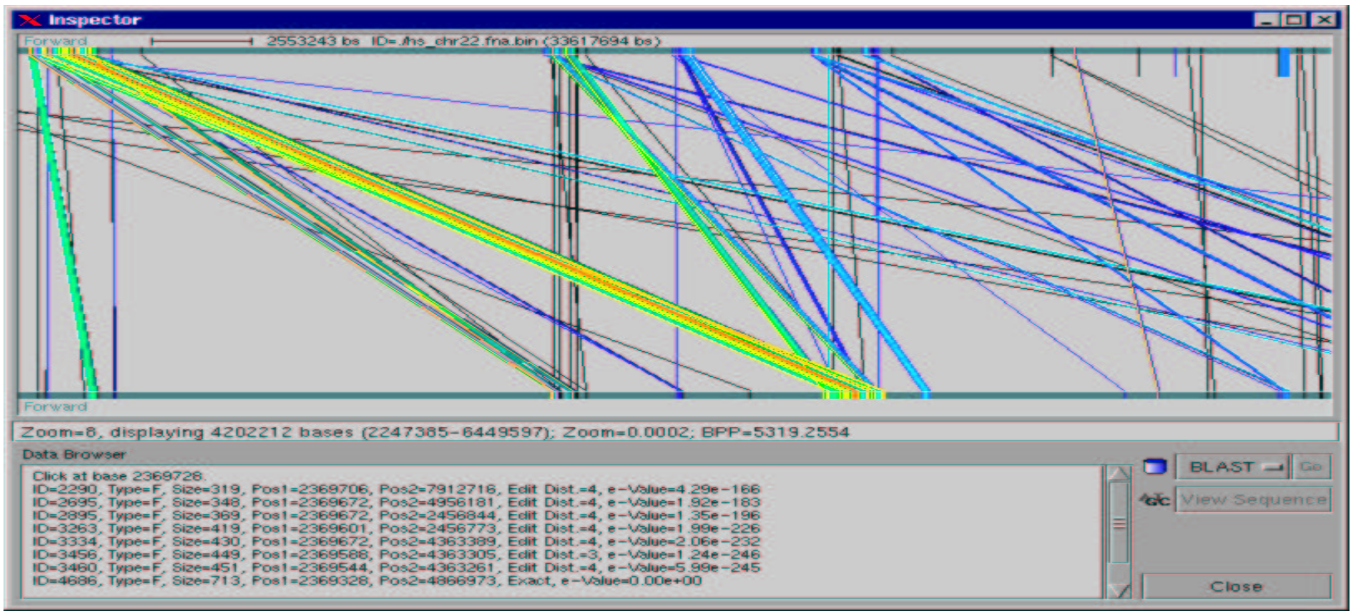


Figure 5: A view of the 50 most significant direct repeats in *M. tuberculosis* (4.4Mb), comparable in size to *E. coli*. Here the longest repeat has 1697 bases, and no others come close to this one. The mesh-like pattern, clearer than in *E. coli*, arises from multifold copies of the same repeat, around 1370 bases in length. Such patterns typically arise from insertion sequences, which is quickly confirmed: A database search indicates that this is an insertion sequence also common in other *mycobacteria*.



```
Zoom=8, displaying 4202212 bases (2247385-6449597); Zoom=0.0002; BPP=5319.2554
```

```
Click at base 2369728.
ID=2290, Type=F, Size=319, Pos1=2369706, Pos2=7912716, Edit Dist.=4, e-Value=4.29e-166
ID=2695, Type=F, Size=348, Pos1=2369672, Pos2=4956181, Edit Dist.=4, e-Value=1.92e-183
ID=2895, Type=F, Size=369, Pos1=2369672, Pos2=2456844, Edit Dist.=4, e-Value=1.35e-196
ID=3263, Type=F, Size=419, Pos1=2369601, Pos2=2456773, Edit Dist.=4, e-Value=1.99e-226
ID=3334, Type=F, Size=430, Pos1=2369672, Pos2=4363389, Edit Dist.=4, e-Value=2.06e-232
ID=3456, Type=F, Size=449, Pos1=2369588, Pos2=4363305, Edit Dist.=3, e-Value=1.24e-246
ID=3460, Type=F, Size=451, Pos1=2369544, Pos2=4363261, Edit Dist.=4, e-Value=5.99e-245
ID=4686, Type=F, Size=713, Pos1=2369328, Pos2=4866973, Exact, e-Value=0.00e+00
```

Figure 6: Zooming in on a repeat rich region on *Homo sapiens*, chromosome 22, here at zoom factor 2^8. (See Figure 1 for an overall view of the repeats.) Repeats are displayed with exact positions and E-values. An E-value smaller than $1.0 \cdot 10^{-300}$ is rounded to 0.00. The sequence information is available for database search via the FASTA/BLAST button.

Delcher, A.; Kasif, S.; Fleischmann, R.; Peterson, J.; White, O.; and Salzberg, S. 1999. Alignment of Whole Genomes. *Nucleic Acids Research* 27:2369–2376.

Dunham, I.; Shimizu, N.; Roe, B. A.; and Chissoe, S. 1999. The DNA sequence of human chromosome 22. *Nature* 402:489–495.

Fitch, W.; Smith, T.; and Breslow, J. 1986. Detecting internally repeated sequences and inferring the history of duplication. In Segrest, J. P., and Albers, J. J., eds., *Plasma Proteins. Part A: Preparation, Structure, and Molecular Biology*, volume 128 of *Methods in Enzymology*. San Diego, CA: Academic Press. chapter 45, 773–788.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences*. New York: Cambridge University Press.

Harel, D., and Tarjan, R. 1984. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Computing* 13:338–355.

Huang, C.; Lin, Y.; Yang, Y.; Huang, S.; and Chen, C. 1998. The telomeres of streptomyces chromosomes contain conserved palindromic sequences with potential to form complex secondary structures. *J. Mol. Biol.* 28(5):905–16.

Kannan, S. K., and Myers, E. W. 1996. An algorithm for locating nonoverlapping regions of maximum alignment score. *SIAM J. Computing* 25(3):648–662.

Kurtz, S., and Myers, G. 1997. Estimating the Probability of Approximate Matches. In *Proc. of the 8th Annual Symposium on Combinatorial Pattern Matching, Aarhus, Denmark, June/July 1997*, 52–64. Lecture Notes in Computer Science 1264, Springer Verlag.

Kurtz, S., and Schleiermacher, C. 1999. *REPuter*: Fast Computation of Maximal Repeats in Complete Genomes. *Bioinformatics* 15(5):426–427.

Kurtz, S. 1999. Reducing the Space Requirement of Suffix Trees. *Software—Practice and Experience* 29(13):1149–1171.

Landau, G. M., and Schmidt, J. P. 1993. An algorithm for approximate tandem repeats. In Apostolico, A.; Crochemore, M.; Galil, Z.; and Manber, U., eds., *Proc. of the 4th Annual Symposium on Combinatorial Pattern Matching, CPM 93. Padova, Italy, June 1993*, volume 684 of *LNCS*, 120–133. Berlin: Springer Verlag.

Leung, M.-Y.; Blaisdell, B. E.; Burge, C.; and Karlin, S. 1991. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Mol. Biol.* 221:1367–1378.

McConkey, M. 1993. *Human Genetics: The Molecular Revolution*. Boston, MA: Jones and Bartlett.

Rivals, É.; Delgrange, O.; Delahaye, J.-P.; Dauchet, M.; Delorme, M.-O.; Hènaut, A.; and Ollivier, E. 1997. Detection of significant patterns by compression algorithms: The case of approximate tandem repeats in DNA sequences. *CABIOS* 13:131–136.

Sagot, M.-F., and Myers, E. W. 1998. Identifying satellites and periodic repetitions in biological sequences. *J. Comp. Biol.* 5(3):539–553.

Sagot, M.-F.; Escalier, V.; Viari, A.; and Soldano, H. 1995. Searching for repeated words in a text allowing for mismatches and gaps. In Baeza-Yates, R., and Manber, U., eds., *The Second South American Workshop on String Processing. Viñas de Mar, Chili, 1995. Proceedings*, 87–100.

Sagot, M.-F. 1998. Spelling approximate repeated or common motifs using a suffix tree. In *Proc. of the Third Latin American Symposium on Theoretical Informatics, LATIN 98*, volume 1380 of *LNCS*, 111–127. Berlin: Springer Verlag.

Schieber, B., and Vishkin, U. 1988. On Finding Lowest Common Ancestors. *SIAM J. Computing* 17(6):1253–1263.

Schmidt, J. P. 1998. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM J. Computing* 27(4):972–992.

Ukkonen, E. 1985. Algorithms for Approximate String Matching. *Information and Control* 64:100–118.

van Belkum, A.; Scherer, S.; van Leeuwen, W.; Willemse, D.; van Alphen, L.; and Verbrugh, H. 1997. Variable number of tandem repeats in clinical strains of haemophilus influenzae. *Infect. Immun.* 65(12):5017–27.

Vincens, P.; Buffat, L.; André, C.; Chevrolat, J.-P.; Boisvieux, J.-F.; and Hazout, S. 1998. A strategy for finding regions of similarity in complete genome sequences. *Bioinformatics* 14(8):715–725.

Wagner, R., and Fischer, M. 1974. The String to String Correction Problem. *Journal of the ACM* 21(1):168–173.

Waterman, M. S.; Arratia, R.; and Galas, D. J. 1984. Pattern recognition in several sequences: Consensus and alignment. *Bull. Math. Biol.* 46(4):515–527.

Zawinski, J. 1994. Remote Control of UNIX Netscape. `http://home.netscape.com/newsref/std/x-remote.html`.