BIOINFORMATICS

Vol. 15 no. 12 1999
Pages 1012–1019

# Post-processing long pairwise alignments

Zheng Zhang[1], Piotr Berman[1], Thomas Wiehe[2,3] and Webb Miller[1,*]

[1]Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, USA and [2]IMB Jena, Department of Genome Analysis, P.O. Box 100 813, D-07708 Jena, Germany

Received on February 12, 1999; revised on May 20, 1999; accepted on June 4, 1999

## Abstract

***Motivation:*** *The local alignment problem for two sequences requires determining similar regions, one from each sequence, and aligning those regions. For alignments computed by dynamic programming, current approaches for selecting similar regions may have potential flaws. For instance, the criterion of Smith and Waterman can lead to inclusion of an arbitrarily poor internal segment. Other approaches can generate an alignment scoring less than some of its internal segments.*

***Results:*** *We develop an algorithm that decomposes a long alignment into sub-alignments that avoid these potential imperfections. Our algorithm runs in time proportional to the original alignment's length. Practical applications to alignments of genomic DNA sequences are described.*

***Availability:*** *Software is available at http://globin.cse.psu.edu/*

***Contact:*** *webb@cse.psu.edu*

## Introduction

Given an alignment scoring scheme, which assigns a score to each possible combination of aligned sequence entries and to each gap, a highest scoring alignment of two given sequences can be computed by a dynamic programming approach (Needleman and Wunsch, 1970). It is less obvious how one should determine a 'best' alignment that can ignore ends of either sequence (a so-called 'local' alignment), since in effect one must pick a region of each sequence and then align the chosen regions. The approach of Smith and Waterman (1981) is to find a highest scoring alignment of arbitrary regions of the two sequences, which can be done with a simple modification of the end-to-end alignment algorithm. When used with an alignment scoring scheme under which the average score of a random alignment is negative, the method provides an approach to solve the local alignment problem.

This approach to local alignment has been quite popular, in part because it is relatively straightforward to understand precisely what is being computed. Alternative formulations of the local alignment problem, e.g. by Sellers (1984), have sometimes been more cumbersome. Moreover, the Smith–Waterman results have a number of desirable properties, including that of being 'normal', in the sense that their score cannot be raised by shortening the alignment at either end. The approach has been adopted by many sequence alignment programs, including our own software for genomic DNA sequences (e.g. Huang *et al.*, 1990; Chao *et al.*, 1995).

An unfortunate property of the Smith–Waterman notion of local similarity is that arbitrarily poor regions can be included in a reported alignment, as illustrated in Figure 1. This is one manifestation of the sensitivity of the output to the choice of alignment scoring parameters (Vingron and Waterman, 1994).

For a practical example of the problem, consider the use of alignment scores that, in our experience, usually provide reasonable alignments of orthologous sequences from humans and mice. However, on occasion, long non-matching regions in both intergenic and intragenic parts of the genomic sequences can be included in the alignment, as shown in Figure 2. Similar problems arise if those same scores are used when comparing sequences from e.g. *C. elegans* and *C. briggsae*, which have a much higher density of coding DNA and shorter introns than do mammals, or when comparing sequences from two

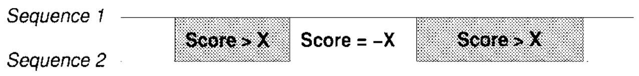

**Fig. 1.** Inclusion of an arbitrarily poor region in an alignment. If a region of score $-X$ is sandwiched between two regions scoring more than $X$, the Smith–Waterman approach will join the three regions into a single alignment, regardless of how large $X$ is.

[*]To whom correspondence should be addressed.
[3]Current address: Max Planck Institute for Chemical Ecology, Department of Molecular Genetics and Evolution, Tatzendpromenade 1a, 07745 Jena, Germany.

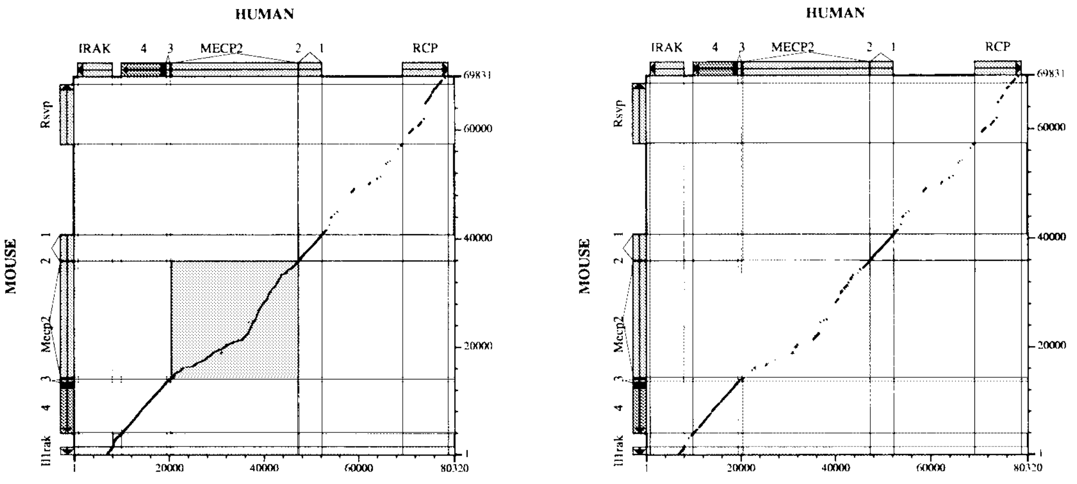

**Fig. 2.** Dotplot summaries of two sets of alignments of human and mouse genomic sequences (GenBank Acc. Nos AF030876, Z47046, Z47066 and Z68193 for human and AF121351 mouse). In both genomic sequences, repetitive elements were found (RepeatMasker with default settings; http://ftp.genome.washington.edu/RM/RepeatMasker.html) and removed from the sequences. The sequences contain three homologous genes. For clarity, only the gene structures for MECP2 and Mecp2, as determined by K. Reichwald *et al.* (submitted for publication), are indicated by black (CDS) and dark grey (UTR) boxes and exon numbers. The alignments on the left were computed by a program called SIM (http://globin.cse.psu.edu) that uses the Smith–Waterman notion of a local alignment. The longest local alignment (human positions 6817–53 081) joins highly conserved regions together using regions that are not conserved. Note especially the intergenic region between IRAK and MECP2 and intron 2 of MECP2, which are indicated by the interior grey boxes. The alignments on the right were determined by post- processing the alignments on the left, using the method described in this paper. Parameter settings were: $+1$ (match), $-1$ (mismatch), $-6 - 0.2k$ (penalty for a gap of length $k$) and $X = 100$.

mammalian species that diverged more recently than the mammalian radiation, e.g. human and monkey sequences.

A recent alternative formulation of the local alignment problem avoids alignments having arbitrarily poor regions. An *X-drop* within an alignment, where $X > 0$ is fixed in advance, is a region of consecutive columns scoring less than $-X$. The idea, then, is to consider only alignments that contain no $X$-drop, which we call *X-alignments*.

A number of theoretical results about $X$-alignments are proved by Zhang *et al.* (1998b). For the current discussion, the relevant observation is that optimal $X$-alignments are expensive to compute, so in practice some heuristic (i.e. approximation algorithm without a rigorous guarantee of solution quality) is employed. Altschul *et al.* (1997) and Zhang *et al.* (1998a) describe programs for searching protein sequence databases that use such methods.

These programs typically work by locating a statistically significant, gap-free local alignment (Altschul *et al.*, 1990), sometimes called a high scoring segment pair, abbreviated HSP. The alignment is extended in both directions, as described by Altschul *et al.* (1997). The portions of the alignment on either side of the extension point are guaranteed to be $X$-alignments, but the full alignment can only be guaranteed to be a $2X$-alignment (see Figure 3). Moreover, in cases where $X$ exceeds the score necessary for statistical significance of an HSP, a computed alignment might be non-normal, in the sense that removing columns from one end raises its score (see Figure 4 for an example of this phenomenon).

This paper develops techniques for decomposing a long alignment into sub-alignments that avoid both problems, i.e. the sub-alignments are free of very low scoring regions and their score cannot be improved by shortening them. In particular, we show how to scan an alignment to collect information from which a decomposition corresponding to any particular value of $X$ can be found almost instantaneously. Unlike the case with computing $X$-alignments *de novo*, the algorithm is both rigorous and efficient. Our approach is related to that of Huang *et al.* (1994). The final section of the paper applies the decomposition algorithm to develop a method for detecting variations in the rate of genome evolution.

## Outline of our approach

A *sub-alignment* of a given alignment is formed by deleting zero or more columns from each end of the alignment. Fix an alignment $\mathcal{A}$, and let $N$ denote the number of columns in $\mathcal{A}$. Let $\mathcal{A}$'s columns be given the respective scores $s_1, s_2, \ldots, s_N$. In what follows, the term

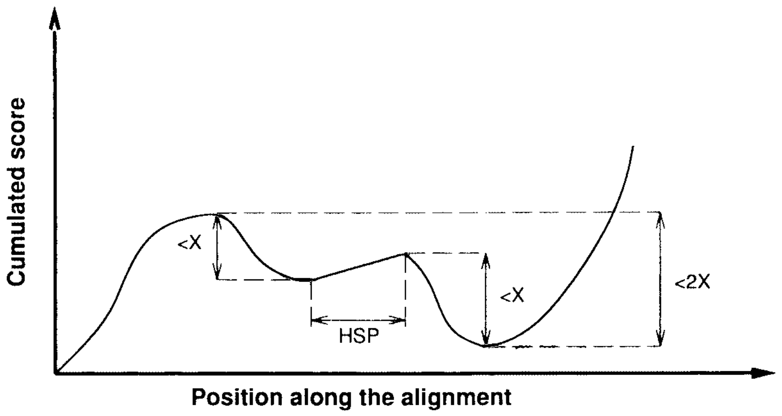

**Fig. 3.** Computation of an alignment violating the $X$-drop condition. Portions on either side of the HSP are $X$-alignments, and the entire alignment is a $2X$-alignment.

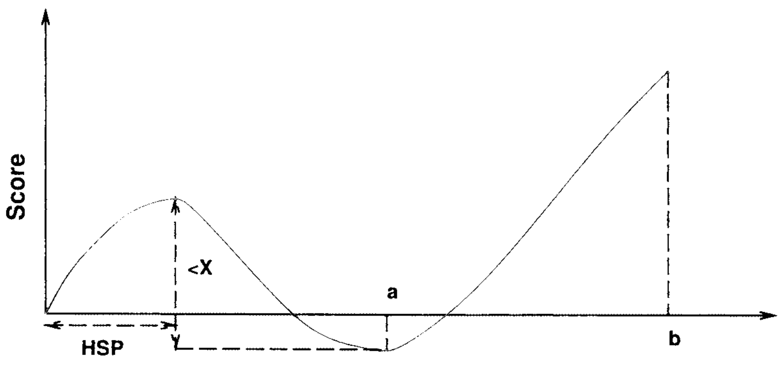

**Fig. 4.** Computation of a non-normal alignment. The HSP has been extended to the right in such a way that the entire alignment scores less than the section from $a$ to $b$.

alignment will refer only to sub-alignments of $\mathcal{A}$. The score of an alignment is taken to be the sum of its column scores. (Our approach also handles cases where a gap of length $k$ is charged the 'affine gap penalty' $x+ky$ for fixed $x$ and $y$.)

An alignment is *normal* if each of its prefixes or suffixes (i.e. initial or terminal runs of columns) has a non-negative score. This is equivalent to requiring that the alignment scores at least as high as any of its sub-alignments. A maximal normal alignment, i.e. one that is not contained in any longer normal alignment, is called *full*. An alignment is *X-normal* for $X \geq 0$ if it is normal and each of it sub-alignments scores at least $-X$. A maximal $X$-normal alignment is called *X-full*.

A 0-full alignment is simply a maximal run of columns of $\mathcal{A}$ with non-negative scores. For every $X$, $X$-full alignments are pairwise disjoint, and if $X < Y$, then each $X$-full alignment is contained in a $Y$-full alignment. A column that does not belong to any $X$-full alignment belongs to a maximal run of columns with negative scores, hence $X$-full alignments are separated by such runs. Finally, the $\infty$-full alignments are just the full alignments.

These properties, which are rigorously stated and proved below, permit assembly of a tree data structure that encodes the $X$-full alignments for all $X \geq 0$. Leaves of this tree are the 0-full alignments and the maximal runs of negative scoring columns that separate them; thus their scores alternate between negative and non-negative. We also add two special leaves at the beginning and the end of this sequence, both with score $-\infty$. Each internal node is a disjoint union of its three children. Besides the children of a node, we know the alignment's score and the minimum score of its sub-alignments. If the score is positive, and the minimum sub-alignment score is $-X$, then this node constitutes an $X$-normal alignment. Figure 5 gives an example.

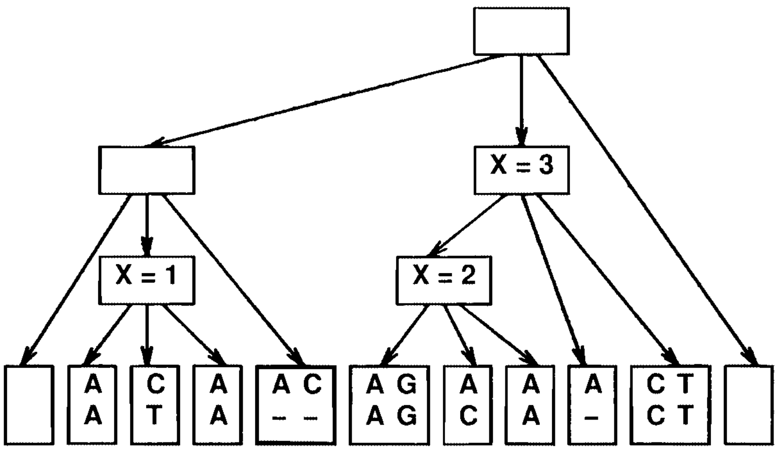

**Fig. 5.** Tree encoding all $X$-full alignments. Shaded and unshaded leaves are maximal runs of columns with negative and non-negative scores, respectively. Each node is marked with its score, internal nodes also have the largest drop value. Nodes that do not encode $X$-full alignments are shaded.

We will show how to construct such a tree in $O(N)$ time. Given the tree and a particular value of $X$, the $X$-full alignments can be found by inspection of the appropriate upper part of the tree; if there are $k$ such alignments, we will inspect at most $3k + 1$ nodes.

## Theoretical basis of our algorithm

As a preliminary step, we decompose the original alignment, $\mathcal{A}$, into a sequence of *atomic* sub-alignments $A_1, A_2, \ldots, A_{2n-1}$. If $i$ is odd, $A_i$ is a maximal run of columns with non-negative scores, and if $i$ is even, $A_i$ is a maximal run of columns with negative scores. Let $\sigma_i$ be the score of $A_i$. Any full alignment has the form $A_i A_{i+1} \ldots A_j$, so from now on we work exclusively with such runs of indices, and the result of our algorithm depends only on the vector $(\sigma_1, \ldots, \sigma_{2n-1})$. To make some of our definitions simpler, we also define $\sigma_0 = \sigma_{2n} = -\infty$.

We will be using the following definitions

- $[a, b) = \{a, a+1, \ldots, b-1\}$ is called a *segment*;
- $[i, j)$ and $[k, 1)$ are *consistent* if $[i, j) \cap [k, l) \in \{[i, j), [k, l), \emptyset\}$;
- a set of segments is consistent if every two elements are;
- a segment $t$ is consistent with a set $S$ if $\{t\} \cup S$ is consistent;
- $\sigma(i, j)$ is the score of $[i, j)$ and equals $\sum_{k=1}^{j-1} \sigma_k$,
- $\sigma_*(i, j) = \min\{\sigma(k, l) : i \le k \le l \le j\}$;
- $[i, j)$ is a normal rise if (1) $\sigma(i, j) \ge 0$ and (2) $i < k < j$ implies both $\sigma(i, k) \ge 0$ and $\sigma(k, j) \ge 0$;
- $[i, j)$ is a normal drop if (1) $\sigma(i, j) < 0$ and (2) $i < k < j$ implies both $\sigma(i, k) < 0$ and $\sigma(k, j) < 0$;
- $[i, j)$ is $X$-normal if it is a normal rise and $\sigma_*(i, j) \ge -X$;
- $[i, i)$ is $X$-full if it is $X$-normal and has no $X$-normal superset other than itself;
- $\chi$ is the set of segments that are $X$-full for any $X$.

LEMMA 1. *$\chi$ is consistent.*

PROOF. Note first that every inconsistent pair of segments has the form $[a, c)$ and $[b, d)$ where $a < b < c < d$. Suppose that $\chi$ contains such a pair, i.e. that $[a, c)$ is $X$-full and $[b, d)$ is $Y$-full. Assume that $X \ge Y$ (the other case is symmetric). We will show that $[a, d)$ is $X$-normal, which is a contradiction, because $[a, c)$ cannot be a subset of another $X$-normal segment.

First we will show that $[a, d)$ is a normal rise. To see (1), observe that $\sigma(a, d) = \sigma(a, c) + \sigma(c, d) \ge \sigma(a, c)$ (the inequality follows from the fact that $b < c < d$ and $[b, d)$ is a normal rise). To see (2), suppose that $a < k < d$. If $k \le c$, $\sigma(a, k) \ge 0$ because $[a, c)$ is a normal rise; otherwise $\sigma(a, k) = \sigma(a, b) + \sigma(b, k) \ge \sigma(b, k) \ge 0$. By a symmetric argument, $\sigma(k, d)$ is non-negative as well.

Now it suffices to show that $a \le k < l \le d$ implies $\sigma(k, l) \ge -X$. If $k \ge b$ or $l \le c$, this is true because both $[a, c)$ and $[b, d)$ are $X$-normal. Otherwise $\sigma(k, l) = \sigma(k, c) + \sigma(c, l) \ge \sigma(c, l) \ge -X$.□

LEMMA 2. *A normal drop is consistent with $\chi$.*

PROOF. Suppose that $[a, b)$ is a normal drop, $[c, d)$ is $X$-full and the segments $[a, b)$ and $[c, d)$ are inconsistent. We have two symmetric cases. If $a < c < b < d$, then $\sigma(c, b) < 0$ because $[a, b)$ is a normal drop, and $\sigma(c, b) \geq 0$ because $[c, d)$ is a normal rise, a contradiction.□

Our goal is to compute $\chi$. If we represent each $[i, j) \in \chi$ as a node, and let the minimal segment in $\chi$ that properly contains $[i, i)$ be its parent, we have a forest, which we call the $\chi$-forest. Below we describe a somewhat different tree structure that may be easier to build. Lemma 3 shows that the $\chi$-forest is easy to construct from any reasonable tree representation of $\chi$, including the ternary representation depicted in Figure 5.

LEMMA 3. *For the tree $T$, suppose that each node is a segment, that children of each internal node consist of two or more disjoint sub-segments of the node's segment, and that every segment of $\chi$ is a node of $T$. If for each node $[i, j)$, we can decide in $O(1)$ time if it is a normal rise and determine $\sigma_*(i, j)$, then in $O(n)$ time we can build the $\chi$-forest. Moreover, for a given $X$, if there are $m$ $X$-full segments, they can be listed in $O(m)$ time.*

PROOF. To compute the $\chi$-forest, we call purify $(R, \infty)$, where $R$ is the root of $T$ and *purify* is as follows. (The pseudo-code deliberately identifies a node with the corresponding segment, so that e.g. $\sigma_*(u)$ makes sense.)

```
boolean purify $(u, X)$
$b \leftarrow$ ($u$ is not a normal rise) **or** $\sigma_*(u) \leq -X$
**if** $b$ **then** $Y \leftarrow X$ **else** $Y \leftarrow -\sigma_*(u)$
**for** every $v$ on the list of children of $u$ **do**
    if purify $(v, Y)$ **then**
        replace $v$ on the list with the list of its
            children
return $b$
```

The arguments of purify are a node $u$ of $T$ and a positive real number that provides the lowest value of $X$ such that $u$ has an $X$-full proper ancestor. Given this information, we can easily decide if $u$ is $Y$-full for some $Y$ and compute the proper second argument for the children of $u$ (see the computation of $b$ and $Y$). The goal of purify is to remove from $T$ all descendants of the first argument that do not belong to $\chi$; this argument is not removed, but purify returns **true** if it should be. It is easy to see that if each recursive call made by purify $(u, X)$ achieves this goal, then so does their parent call. This completes the sketch of the proof of correctness of purify.

To see that this function terminates in time $O(|\chi|)$, we make three observations. First, purify works in time $O(|T|)$, where $|T|$ denotes the number of nodes in $T$. Second, under our assumptions, segment $[2k-1, 2k)$ is 0-full for $k = 1, \ldots, n$. Finally, because every node of $T$ has at least two children, and $T$ has at most $2n+1$ leaves, $|T| \leq 4n+1$.

To list the $X$-full segments, we can use a modified version of purify. The desired running time can be achieved because, with the exception of leaves with negative scores, every node of $T$ is an ancestor or descendant of an $X$-full segment. Because we can avoid calling the modified purify for the proper descendants, the overall number of calls is proportional to the size of the output.□

Our method is to compute $\chi$ by constructing a *useful tree*, which is defined as a tree $\mathcal{T}$ possessing the following properties.

- Each node of $\mathcal{T}$ is a segment consistent with $\chi$.
- Each leaf of $\mathcal{T}$ is of the form $[i, i+1)$.
- Each internal node $[a, d)$ has exactly three children, $[a, b)$, $[b, c)$ and $[c, d)$, and the signs of their scores alternate.

Figure 5 gives an example of a useful tree.

LEMMA 4. *If $\mathcal{T}$ is a useful tree with root $[0, 2n+1)$, then every segment of $\chi$ is a node of $\mathcal{T}$.*

PROOF. Let $[i, j)$ be a segment from $X$. Consider a node of $\mathcal{T}$, say $[a, d)$, that contains $[i, i)$ but none of its children does (observe that the root of $\mathcal{T}$ surely contains $[i, j)$). If $[i, j) = [a, d)$, we are done. If $[a, d)$ is a leaf, it had to be the case, because this segment is atomic. Therefore we can assume that $[i, j) \neq [a, d)$ and that $[a, d)$ has three children, $[a, b)$, $[b, c)$ and $[c, d)$. Because the children are consistent with $[i, i)$ and do not contain it, each is either disjoint with $[i, i)$ or a subset. In particular, it must be the case that $[i, j) = [a, c)$ or $[i, i) = [b, d)$. However, this leads to a contradiction. For example, if $[i, j) = [a, c)$, the contradiction comes from the fact that $[a, c)$ is allegedly a normal rise, but either $[a, b)$ or $[b, c)$ has a negative score (recall that the signs of scores of children alternate).□

Our method of constructing a useful tree with root $[0, 2n+1)$ is to start from the sequence (ordered forest) of $2n+1$ useful trees, each consisting of a single node: $[0, 1), [1, 2), \ldots, [2n, 2n+1)$. Then as long as there is more than one tree in the sequence, some three consecutive trees, say with roots $[a, b)$, $[b, c)$, $[c, d)$, will be merged by creating a common parent for these roots, namely $[a, d)$. Clearly, at every stage of this process, the sequence of the roots will partition the final root segment, $[0, 2n+1)$.

To assure that we can find the desired three consecutive trees, we will show that the sequence has the following invariant properties:

P1: the roots in the sequence alternate between normal drops and normal rises;

P2: if $[a, b)$, $[b, c)$, $[c, d)$ are roots of three consecutive trees, and $\sigma(b, c) \geq 0$, then $\sigma_*(b, c) \geq \sigma(a, b)$ and $\sigma_*(b, c) \geq \sigma(c, d)$.

The following three lemmas prove the correctness of our approach.

LEMMA 5. *Assume that three consecutive roots in our sequence,* $[a, b)$, $[b, c)$, *and* $[c, d)$, *satisfy* $0 \leq \sigma(b, c) < \min(-\sigma(a, b), -\sigma(c, d))$. *Then merging these trees into a single tree with root* $[a, d)$ *creates a useful tree and the resulting sequence still satisfies P1 and P2.*

PROOF. First we can show that $[a, d)$ is a normal drop. Assume $a < e < d$. If $e \leq b$, then $\sigma(a, e) < 0$, because $[a, e)$ is a prefix of $[a, b)$; if $b < e \leq c$, then $\sigma(a, e) = \sigma(a, b) + \sigma(b, c)\sigma(e, c) \leq \sigma(a, b) + \sigma(b, c) < 0$; finally, if $c < e$, then $\sigma(a, e) = \sigma(a, b) + (b, c) + \sigma(c, e) < \sigma(a, b) + \sigma(b, c) < 0$. A symmetric reasoning shows that $\sigma(e, d) < 0$. Because $[a, d)$ is a normal drop, it is consistent with $\chi$ by Lemma 2. By P1, the signs of the scores of its three children alternate, thus the new tree is useful. Property P1 is preserved, because the new root segment has a negative score. P2 is preserved, because $\sigma(a, d) < \sigma(a, b)$ and $\sigma(a, d) < \sigma(c, d)$, thus the low scores in the roots of the neighboring trees, which, by P2, cannot be smaller than $\sigma(a, b)$ and $\sigma(c, d)$ respectively, are larger than $\sigma(a, d)$.□

If $a$, $b$, $c$ and $d$ satisfy the premises of Lemma 5, we say that $(a, d)$ is a *possible negative merger*.

LEMMA 6. *Assume that five consecutive roots in our sequence,* $[a, b)$, $[b, c)$, $[c, d)$, $[d, e)$ *and* $[e, f)$ *satisfy*

- $0 > \sigma(c, d) \geq \max(\sigma(a, b), \sigma(e, f))$.
- *neither* $(a, d)$ *nor* $(c, f)$ *is a possible negative merger.*

*Then merging the trees with roots* $[b, c)$, $[c, d)$, $[d, e)$ *into a single tree with root* $[b, e)$ *creates a useful tree and the resulting sequence still satisfies P1 and P2.*

PROOF. Note that $0 \leq \sigma(b, c)$ and that $\min(-\sigma(a, b), -\sigma(c, d)) = -\sigma(c, d)$. Because $(a, d)$ is not a possible negative merger, this implies that $\sigma(b, c) \geq -\sigma(c, d)$. For similar reasons, we have $\sigma(d, e) \geq -\sigma(c, d)$. One can see that $[b, e)$ is a normal rise and $\sigma_*(b, e) = \sigma(c, d)$; thus the new sequence satisfies both P1 and P2. It remains to show that $[b, e)$ is consistent with $\chi$. Suppose not. Then there exists an $X$-full segment $[g, h)$ that is inconsistent with $[b, e)$. But it must be consistent with the existing root segments, and it cannot end or start with a normal drop, hence $[g, h)$ ends with $[b, c)$ or starts with $[d, e)$. In the first case it must also contain $[a, c)$, and thus $X \geq -\sigma(a, b) \geq -\sigma(c, d)$. It is easy to see that in this case $[g, h)$ is not $X$-full because it can be extended rightwards to $e$. The case when $[g, h)$ starts with $[d, e)$ is symmetric.□

If $a$, $b$, $c$, $d$, $e$ and $f$ satisfy the premises of Lemma 6, we say that $(a, f)$ is a *possible positive merger*.

## Implementation

Our algorithm is straightforward. We maintain a sequence of trees with roots $[0 = a_0, a_1)$, $[a_1, a_2), \ldots,$ $[a_{2m}, a_{2m+1} = 2_{n+1})$. While there is more than one tree in the sequence, we search for the least $k$ such that either $(a_{2k-2}, a_{2k+1})$ is a possible negative merger, or $(a_{2k-4}, a_{2k+1})$ is a possible positive merger, and execute the respective merger (negative or positive).

The previous three lemmas guarantee that if our algorithm properly terminates, then it will build a useful tree with root $[0, 2n + 1)$ which contains every segment of $\chi$. Thus to prove its correctness, it suffices to show that every search for a possible negative/positive merger is successful.

LEMMA 7. *Assume that we have a sequence of trees with roots* $[a_0, a_1)$, $[a_1, a_2)$, $[a_{2m}, a_{2m+1})$ *that satisfies conditions P1 and P2. Then there exists i such that either* $(a_{i-3}, a_{i-2})$ *is a possible negative merger, or* $(a_{i-5}, a_i)$ *is a possible positive merger.*

PROOF. Suppose there is no possible negative merger. Choose $i$ so that $|\sigma(a_{i-3}, a_{i-2})|$ is minimal. We can require that $i$ is odd, otherwise we will have $0 < \sigma(a_{i-3}, a_{i-2}) < \min(-\sigma(a_{i-4}, a_{i-3}), -\sigma(a_{i-2}, a_{i-1}))$ which means that $(a_{i-4}, a_{i-1})$ is a possible negative merger. Thus, $0 > \sigma(a_{i-3}, a_{i-2}) \geq \max(\sigma(a_{i-5}, a_{i-4}),$ $\sigma(a_{i-1}, a_i))$, and in the absence of any possible negative merger, this implies that $(a_{i-5}, a_i)$ is a possible positive merger.□

Now we need to show that we can search for possible mergers efficiently. Suppose that in our sequence the segment $(a_{i-1}, a_i)$ is the result of the last merger. Let $(a_k, a_l)$ be the next merger, i.e. a possible negative merger or possible positive merger with the least possible value of $a_l$. It is clear that $l \geq i$; otherwise we would perform the merger indicated by $(a_k, a_l)$ before the merger that created $(a_{i-1}, a_i)$, not after. We may conclude that in the sequence of possible negative merger and possible positive mergers executed by our algorithm the upper index is non-decreasing; therefore we can search for possible mergers by checking two possibilities that have the current upper index, and if neither of them applies, incrementing the

```
1. Push the first leaf on the stack.
2. while the stack size exceeds 1 or there is an unvisited leaf do
3.     if the top three stack items indicate a negative merger then
4.         pop three items, merge them and push the result onto the stack
5.     else if the top five segments indicate a positive merger then
6.         pop an item $[e, f)$, perform line 4, and push $[e, f)$ back
7.     else
8.         push the next two leaves onto the stack
```

**Fig. 6.** Algorithm to build a useful tree.

upper index and trying again. Because the upper index is always odd, we will consider only $n$ different values, from 3 to $2n + 1$.

The above ideas can be readily implemented using a stack to hold segments. We start with the sequence $[0, 1), \ldots, [2m, 2m + 1)$, and we treat it as the input of a PDA (pushdown automaton). Initially, $[0, 1)$ is on the stack, with the remaining segments waiting to be processed. If the top three segments of the stack are some $[a, b)$, $[b, c)$, $[c, d)$ such that $(a, d)$ is a possible negative merger, we merge their trees into one with root $[a, d)$; a negative merger that decreases the stack height by 2. If the top five segments are some $[a, b), \ldots, [e, f)$ such that $(a, f)$ is a possible positive merger, we perform a positive merger that produces segment $[b, e)$. If neither possibility holds, we push the next two (leaf) segments on the stack, thus incrementing the upper index for the next merger.

The algorithm is summarized in Figure 6. The pseudo-code omits details about propagation of the information required for application of Lemma 3, i.e. which nodes correspond to a normal rise, and the values $\sigma(u)$ and $\sigma_*(u)$. A leaf is a normal rise if its score is positive, and an internal node is a normal rise if it is created by a positive merger. The score of an internal node is the sum of its children's scores. Finally, if $u$ is created by a negative merger, then $\sigma_*(u) = \sigma(u)$, whereas if it is created by a positive merger of nodes $x$, $y$ and $z$, then $\sigma_*(u) = \sigma(y)$. It follows readily that the algorithm spends $O(1)$ time per node of the useful tree, and hence $O(n)$ time overall.

## An application

It has been known for some time that different regions of a mammalian genome evolve at different rates, though study of the phenomenon has been hampered by lack of data. Recently, with release of large amounts of human and mouse genomic DNA sequence data, systematic studies of relative mutation rates have become possible. However, there are pitfalls to be overcome to produce an objective and reliable method to measure the 'background' mutation rate in a given genomic region, particularly when the bulk of the detectable sequence similarity is confined to protein-coding regions. The methods developed in this paper can be profitably applied in such a study, as sketched in this section.

To compare the rates of evolution in different genomic regions where data is available from humans and mice, one approach is to align each pair of homologous regions and determine, say, the percentage of nucleotides that align according to some objective criterion. One subtlety is that interspersed repeats that inserted after the evolutionary divergence of humans and mice (such as Alu elements in humans and B1 elements in mice) should be removed from consideration, so as to distinguish the rate of nucleotide mutations from the rate of large-scale insertions. Another consideration is that functionally constrained intervals, in particular exons (protein-coding regions), should be excluded so as to expose the neutral rate of evolution.

Perhaps the simplest approach would be to align the regions in the usual way, but then tally statistics only at sequence positions not in exons. This is pitfall number 1; with most methods of computing local alignments, doing so would produce very biased results, since regions immediately adjacent to an exon will be aligned if their score is at least 0 (i.e. the alignment between the homologous exons will be extended to include the region), whereas an alignment that is far from any exon will be reported only if its score exceeds some positive threshold. A somewhat better approach is to remove the exons before producing the alignment. However, this is pitfall number 2; in regions that are very weakly conserved between the two species, this strategy may fail because the alignment program is unable to differentiate the biologically meaningful alignments from the ones that occur by chance; aligned exons are needed to pin down the remainder of the correct alignment. (Between humans and mice, homologous exons almost always align strongly. See Makalowski *et al.*, 1996.)

The results presented in this paper permit measurement of mutation rates in a manner that avoids both pitfalls. Simply put, the idea is to first align the sequences using the exons as guideposts, then re-score the alignment where positions within exons are 'masked' so that they cannot be aligned to another nucleotide. That is, the $X$-full sub-alignments are reported, for some appropriate value of $X$. In that manner, nucleotides aligned in the first step and adjacent to an exon will be treated as candidates for alignment in the second phase, but will not be reported as aligning unless they meet the same criterion that is applied far from the gene.

Figure 7 illustrates this approach using the genes ERCC2 (Lamerdin *et al.*, 1996) and XRCC1 (Lamerdin *et al.*, 1995). A portion of each human-mouse alignment containing several exons is shown, with weak matches in the exons' flanks. These initial alignments were computed using a Smith–Waterman criterion, where each alignment's score was required to exceed some threshold,

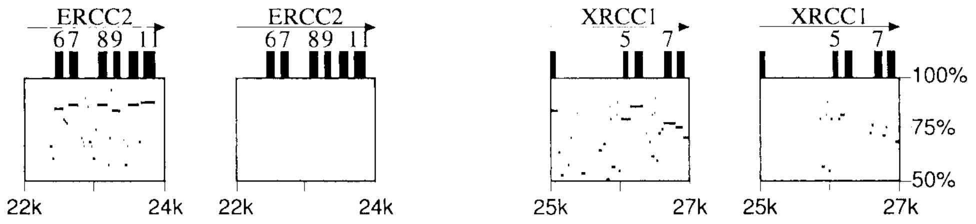

**Fig. 7.** Determination of regions that align, ignoring exons. For each of the ERCC2 and XRCC1 genes, a percent identity plot of a portion of the alignment is shown on the left, and the alignments that remain after filtering out the exons, as described in the paper, are shown on the right.

$\tau$. Then, the score of each column in a protein-coding region was set to $-\infty$, and $X$-full sub-alignments of score exceeding $\tau$ were computed as described in this paper, where $X$ was in essence set to $\infty$, corresponding to the fact that no $X$-drop condition was placed on the original alignments.

With XRCC1 but not ERCC2, some of the non-exon aligning regions remain, indicating that the rate of neutral mutation in the ERCC2 region may be substantially higher than around XRCC1. Note that this conclusion differs from what would be suggested by simply removing the coding-region matches from the first and third panel of Figure 7 (pitfall number 1), since that operation would leave similar-looking residues of tiny matches for both ERCC2 and XRCC1. Also, the approach of removing exons before aligning the sequences would fail (pitfall number 2), because the matches around XRCC1 that are pictured in the right-most panel of Figure 7 are too weak to be reliably determined within long surrounding sequences unless the matches between homologous exons are used to guide the process.

## Acknowledgements

Z.Z. and W.M. were supported by grant LM05110 from the National Library of Medicine. P.B was supported by NSF grant CCR 9700053. We thank Josep Abril for producing Figure 2. The referees provided an unusually insightful report.

## References

Altschul,S.F., Gish,W., Miller,W., Myers,B. and Lipman,D.J. (1990) A basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Altschul,S.F., Madden,T.L., Schäffer,A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped BLAST and PSI-BLAST — a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

Chao,K.-M., Zhang,J., Ostell,J. and Miller,W. (1995) A local alignment tool for very long DNA sequences. *Comput. Applic. Biosci.*, **11**, 147–153.

Huang,X., Hardison,R.C. and Miller,W. (1990) A space-efficient algorithm for local similarities. *Comput. Applic. Biosci.*, **6**, 373–381.

Huang,X., Pevzner,P. and Miller,W. (1994) Parametric recomputing in alignment graphs. *Combinatorial Pattern Matching.* Springer Lecture Notes in Computer Science, **807**, 87–101.

Lamerdin,J.E., Montgomery,M.A., Stilwagen,S.A., Scheidecker,L.K., Tebbs,R.S., Brookman,K.W., Thompson,L.H. and Carrano,A.V. (1995) Genomic sequence comparison of the human and mouse XRCC1 DNA repair gene regions. *Genomics*, **25**, 547–554.

Lamerdin,J.E., Stilwagen,S.A., Ramirez,M.H., Stubbs,L. and Carrano,A.V. (1996) Sequence analysis of the ERCC2 gene regions of human, mouse and hamster reveals three linked genes. *Genomics*, **34**, 399–409.

Makalowski,W., Zhang,J. and Boguski,M.S. (1996) Comparative analysis of 1196 orthologous mouse and human full-length mRNA and protein sequences. *Genome Res.*, **6**, 846–857.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Sellers,P.H. (1984) Pattern recognition in genetic sequences by mismatch density. *Bull. Math. Biol.*, **46**, 501–514.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular sequences. *J. Mol. Biol.*, **97**, 723–728.

Vingron,M. and Waterman,M.S. (1994) Sequence alignment and penalty choice. *J. Mol. Biol.*, **235**, 1–12.

Zhang,Z., Schäffer,A., Miller,W., Madden,T.L., Lipman,D.J., Koonin,E.V. and Altschul,S.F. (1998a) Protein sequence similarity searches using patterns as seeds. *Nucleic Acids Res.*, **26**, 3986–3990.

Zhang,Z., Berman,P. and Miller,W. (1998b) Alignments without low-scoring regions. *J. Computational Biol.*, **5**, 197–210.