ALIGNMENT OF MINISATELLITE MAPS: A MINIMUM SPANNING TREE BASED APPROACH

MOHAMED I. ABOUELHODA

Cairo University, Giza, Egypt
Email: mohamed.ibrahim@uni-ulm.de

ROBERT GIEGERICH

University of Bielefeld, 33501 Bielefeld, Germany

BEHSHAD BEHZADI

Google Switzerland GmbH, Switzerland and Ecole Polytechnique, France

JEAN-MARC STEYAERT

LIX, Ecole Polytechnique, Palaiseau cedex 91128, France

In addition to the well-known edit operations, the alignment of minisatellite maps includes duplication events. We model these duplications using a special kind of spanning trees and deduce an optimal duplication scenario by computing the respective minimum spanning tree. Based on best duplication scenarios for all substrings of the given sequences, we compute an optimal alignment of two minisatellite maps. Our algorithm improves upon the previously developed algorithms in the generality of the model, in alignment quality and in space-time efficiency. Using this algorithm, we derive evidence that there is a directional bias in the growth of minisatellites of the MSY1 dataset.

Keywords: Minisatellite maps; Sequence Analysis; Run length encoding.

1. Introduction

1.1. *Alignment of minisatellite maps*

A genomic region is classified as a minisatellite locus if it spans more than 500 bp and is composed of tandemly repeated DNA stretches. Each stretch, called *unit*, is a sequence of nucleotides whose length ranges between 7-100 bp. A potential mechanism responsible for the evolution of minisatellites is the *unequal cross-over*, where the paired homologous chromosomes exchange unequal segments during the cell division. This gives rise to a repeated segment in one chromosome and to a deletion in the other; see Figure 1 (b).

A minisatellite map represents a minisatellite region, where each unit is encoded by a character and handled as one entity; see Figure 1 (a). For one minisatellite locus, both the type and the number of units vary between individuals in a population. Therefore, minisatellite maps provide a means for studying the evolution of

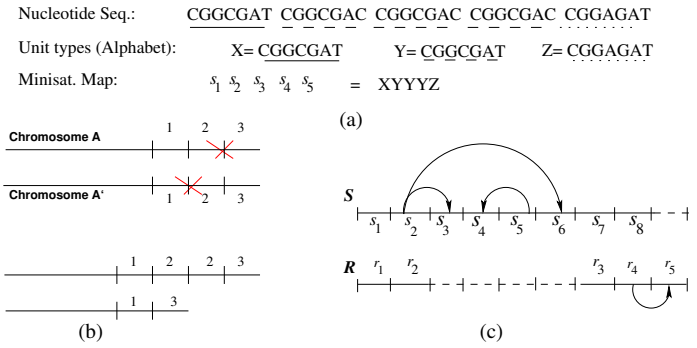


Fig. 1. (a): A minisatellite locus: five units, their nucleotide sequences, and the respective map are shown. (b): The unequal cross over producing duplication of the unit 2. (c): Alignment of two sequences. The matched copies are put above each other. The arcs represent duplication events.

populations. The key algorithm for this study is to align maps of individuals from different populations.

The traditional model of sequence alignment is based on the edit operations of replacement (including matches) and deletions/insertions (indels). In aligning minisatellite maps, one has to also consider that regions of the map have arisen as a result of duplication events from the neighboring units. The single copy duplication model, where only one unit can duplicate at a time, is the most popular, and its biological validation was asserted for the MSY1 minisatellites; see [1, 2]. The scoring of minisatellite map alignment accounts for common aligned units as well as for individual duplication histories. For example, the score of the alignment in Figure 1 (c) is composed of (1) replacement scores for the unit pairs (s_1, r_1), (s_2, r_2), (s_7, r_3) and (s_8, r_4), (2) costs of duplication of the units s_3, s_6 originated from the unit s_2, duplication of s_4 from s_5, and duplication of r_5 from r_4, and (3) insertion of the unit s_5. That is, the comparison delivers a three-stages scenario: The aligned units refer to common ancestors, the duplications refer to differences in the individual duplication histories, and the indels refer to units (possibly emerged by a transposition [3]) not homologous to the map units.

1.2. *Previous and new results*

The problem of comparing two minisatellite maps under the single copy duplication model was investigated for the first time by Bérard and Rivals [1] who presented an algorithm that takes $O(n^4)$ time and $O(n^3)$ space, where n is the average map length. Subsequently, Behzadi and Steyaert [4] followed a different approach and presented a transformation-distance based algorithm: one sequence is considered as a source and the other as a target. The optimal transformation distance is the minimum cost set of operations required to transform the source to the target. Their algorithm takes $O(n^3|\Sigma|)$ time and $O(n^2|\Sigma|)$ space, where $|\Sigma|$ is the alphabet size. Based on a run length encoding scheme, Behzadi and Steyaert [5] improved the

running time of their algorithm to $O(n^2 + nn'^2 + n'^3|\Sigma|)$, where n' is the length of the run-length compressed sequence. Very recently, Bérard et al. [6] argued that the alignment distance for two minisatellite sequences S and R is symmetric, while the transformation distance is not. They correspondingly refined the algorithm of [1] by incorporating ideas of [5] and presented an algorithm that takes $O(n^3 + n'^3|\Sigma|)$.

In this paper, we present an *alignment* algorithm that improves upon the previous ones in many respects: The alignment model is more general and our algorithm relaxes the constraint that the mutation distance $M(a, b)$ between two units is symmetric. The time complexity of our algorithm is alphabet-independent and we show that the run length encoding scheme can be incorporated, which yields an $O(n^2 + nn'^2 + n'^3)$ time and $O(n^2)$ space algorithm. This makes our algorithm the fastest map alignment algorithm in theory, and in practice as well, as we demonstrate by experiments.

From the biological point of view, our algorithm is so flexible that investigators can put constraints on the direction of duplications to study the structural variation and duplication dynamics. Based on this feature, we could quantitatively verify the assumption in [2] that the units duplicate in a biased fashion at the $5'$ end.

Our algorithm computes an optimal alignment by first computing and storing the cost of an optimal duplication history for each interval in each sequence separately. Then it computes the optimal alignment based on the precomputed costs.

In fact, reconstructing duplication histories of a minisatellite map which occurs here as a subproblem of map alignment is related, but in a somewhat modified form, to the problem of inferring the duplication history of tandemly repeated genes; see [7–10], and in particular [11], which is closely related to our work here. What distinguishes the general construction of duplication histories from their use in the map alignment problem is that here, it is required to compute, for each interval of units, a duplication history originated either from the leftmost or the rightmost unit. Furthermore, some units may be inserted (not duplicated from other units) and may then undergo further duplications.

2. The Duplication History

Let $S = s_1, s_2, \ldots, s_n$ denote a minisatellite map of n units. We write $S[i..j]$ to denote the $j - i + 1$ contiguous units $s_i, s_{i+1}, \ldots, s_j$. A sequence T is a subsequence of S, if there is a set of indices $i_1 < i_2 < \cdots < i_m$, $m \leq n$, such that $T = s_{i_1}, s_{i_2}, \ldots, s_{i_m}$. The cost of mutating a unit s' to s'' is denoted by $M(s', s'') \geq 0$.

We denote the cost of a duplication event affecting a unit s' by $DUP(s')$. We call the total cost of duplicating and mutating the unit s' to produce $s's''$ or $s''s'$, where $s', s'' \in S$, the *duplication cost* $d(s', s'') = DUP(s') + M(s', s'')$. Note that if $M(s', s'') = M(s'', s')$, and $DUP(s')$ is constant for every unit $s' \in S$, then $d(s', s'') = d(s'', s')$, i.e., we speak of a symmetric distance function. Our algorithms work for both symmetric and non-symmetric distance functions. We assume that $d(s', s'') \leq d(s', s''') + d(s''', s'')$, for all $\{s', s'', s'''\} \subset S$.

dup. (Edge)	subsequence
$s_3 \rightarrow s_1$	$s_1\ s_3$
$s_3 \rightarrow s_2$	$s_1\ s_2\ s_3$
$s_3 \rightarrow s_4$	$s_1\ s_2\ s_3\ s_4$
$s_4 \rightarrow s_6$	$s_1\ s_2\ s_3\ s_4\ s_6$
$s_6 \rightarrow s_7$	$s_1\ s_2\ s_3\ s_4\ s_6\ s_7$
$s_6 \rightarrow s_5$	$s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7$

(a)

dup. (Edge)	subsequence
$s_3 \rightarrow s_1$	$s_1\ s_3$
$s_3 \rightarrow s_2$	$s_1\ s_2\ s_3$
$s_3 \rightarrow s_4$	$s_1\ s_2\ s_3\ s_4$
$s_2 \rightarrow s_6$	$s_1\ s_2\ s_6\ s_3\ s_4$

(b)

Fig. 2. (a) The table on the left shows the duplication events producing the sequence $s_1, \ldots, s_7$, we show also the resulting intermediate subsequences. We show the corresponding ORDST and its arched-arrow representation (shown under the tree). (b) A spanning tree that cannot be an ORDST, along with the duplication events and the resulting subsequences.

The duplication history of a sequence of units describes a series of duplication events that created the observed sequence of units. Here, we represent these histories by *ordered directed spanning trees* (ORDSTs). The nodes of an ORDST are labeled with units of S, and an edge from s' to s'' corresponds to a duplication event where the unit s' is duplicated to $s's''$ or $s''s'$. (For brevity, we will write "node s" to mean the "node labeled with s".)

Figure 2 (a) shows an example of an ORDST, its arched-arrow representation, and the corresponding events for generating it. Note that the resulting sequences after each duplication event are subsequences of S; we call this property *the ordering constraint*. However, not every spanning tree on S is an ORDST describing the duplication history. For example, in the same figure part (b) we show a subtree that cannot lead to an ORDST. This is because the duplication of the unit s_2 produces s_6. Note that the resulting sequence s_1, s_2, s_6, s_3, s_4 is not a subsequence of S. In terms of the single copy duplication model, it is impossible to duplicate the unit s_2 to produce s_6 after the emergence of s_3 or s_4. That is, the ordering constraint preserves the properties of the evolutionary mechanism. The following lemma specifies properties of the ORDST used in our algorithm.

Lemma 2.1. *For an ORDST of n units, the following statements are equivalent:*

a. *For each node s_k in an ORDST, we can write the nodes of its full subtree as $S[i..j]$, $1 \leq i \leq k \leq j \leq n$. Moreover, this subtree can be divided into two subtrees sharing the root s_k and partitioning the interval $[i..j]$ into $S[i..k]$ and $S[k..j]$.*
b. *Let the interval $[i..j]$ include the nodes in the subtree of node s_k, and let the intervals $[l_1..r_1], .., [l_t..r_t]$ include the subtrees of the child nodes of s_k. Then $k \notin [l_i..r_i]$, and $[l_i..r_i] \cap [l_j..r_j] = \phi$ for all $1 \leq i, j \leq t, i \neq j$.*
c. *Let the interval $[i..j]$ include the nodes in the subtree of node s_k, and let s_l be a descendant node of s_k. If every element in $S[l+1..j]$ is neither an ancestor nor a right brother of s_l, we can divide the tree over $[i..j]$ at node s_l into two subtrees intersecting at s_l: The first includes the nodes $S[i..l]$ with s_k as a root, and the second includes $S[l..j]$ with s_l as a root.*

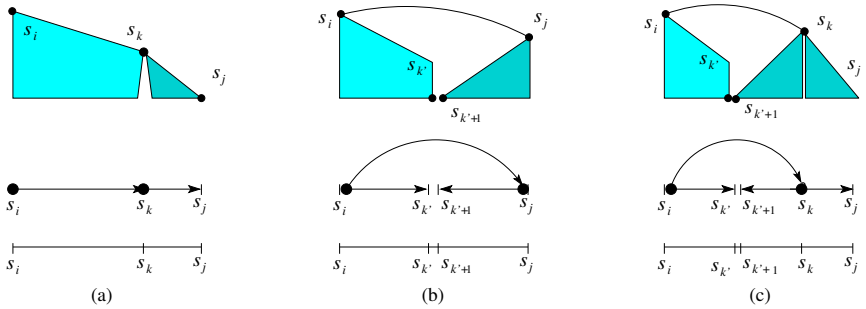


Fig. 3. The interval partitioning of the recurrence (3.1). The horizontal arrows correspond to *left*-ORDMST or *right*-ORDMST extending from the arrow beginning to its end. The arched arrows correspond to a duplication event between two units. The black circles correspond to roots of trees. (a) The recurrence for $C_l^A(i,j)$ partitioning the tree at node s_k. (b) The recurrence for $C_l^B(i,j)$. (c): A general topology for *left*-ORDMST.

Proof. This follows from the ordering constraint on the units of S. □

Definition 2.1. Each edge $s' \rightarrow s''$ in an ORDST for a sequence S is assigned a duplication cost $d(s', s'')$, where $s', s'' \in S$. The *ORDST cost* is the total sum of all duplication costs. An optimal ORDST for S is one of minimum cost.

3. Left- and Right Ordered Directed Spanning Trees

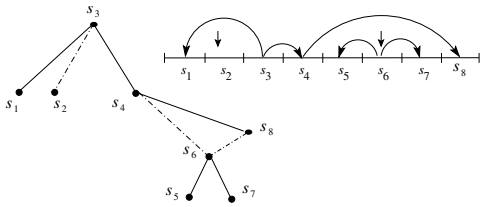
The alignment of minisatellites includes duplication events, where the duplicated units originate from the left- or rightmost unit of the interval containing the duplications. Therefore, we compute for each interval two optimal histories: one originated from the leftmost unit and the other from the rightmost unit. For ease of presentation, we will first present an algorithm that computes optimal histories from duplications only, without insertions. Then we will extend this algorithm to incorporate such insertions.

3.1. *Computation of optimal trees without insertion*

Based on Lemma 2.1, any ORDST can be expressed in terms of contiguous intervals. This suggests that a dynamic programming algorithm can be used to construct an optimal tree by recursively searching for an optimal partitioning of the intervals enclosing the units.

Given an interval $S[i..j] \subseteq S[1..n]$, we call an ORDST over this interval *left*-ORDST if the root of the respective tree is the leftmost unit s_i, and we call it *right*-ORDST if the root is the rightmost unit s_j. Let $C_l(i,j)$ denote the cost of a minimum *left*-ORDST defined over the units $S[i..j]$. Similarly, let $C_r(i,j)$ denote the cost of a minimum *right*-ORDST over $S[i..j]$. The values of $C_l(i,j)$ and $C_r(i,j)$ can be computed iteratively by examining subintervals of $[i..j]$ as follows.

Figure 3 (c) shows the general decomposition of a *left*-ORDST. s_k denotes the rightmost son of s_i, and the overall *left*-ORDST decomposes into two *left*-ORDSTs



Event	subsequence
dup. $s_3 \rightarrow s_1$	$s_1\ s_3$
Ins. s_2	$s_1\ s_2\ s_3$
dup. $s_3 \rightarrow s_4$	$s_1\ s_2\ s_3\ s_4$
dup. $s_4 \rightarrow s_8$	$s_1\ s_2\ s_3\ s_4\ s_8$
Ins. s_6	$s_1\ s_2\ s_3\ s_4\ s_6\ s_8$
dup. $s_6 \rightarrow s_7$	$s_1\ s_2\ s_3\ s_4\ s_6\ s_7\ s_8$
dup. $s_6 \rightarrow s_5$	$s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8$

Fig. 4. (a) An ORDST for $S = (s_1, .., s_8)$ with the inserted units: $\{s_2, s_5, s_6, s_7\}$. The unit s_2 was inserted but did not undergo further duplications. s_6 was inserted then duplicated to s_5 and s_7. An ORDST with insertions should be regarded as a forest; the dash-dotted lines are virtual links between the forest roots and their potential parent nodes. On the left, the respective events. The resulting sequences are subsequences of S, i.e., the ordering constraint property is conserved

over intervals $S[i..k']$ and $S[k..j]$, and a *right*-ORDST over the interval $S[k'+1..k]$. However, this most natural decomposition leads to a recurrence of time complexity $O(n^4)$, by iterating over k and k'. Therefore, we use different decompositions to reach $O(n^3)$ time complexity.

Consider the two cases $k < j$ and $k = j$ in the interval partitioning of Figure 3 (c). Figure 3 (a) deals with the case $k < j$. We see a partitioning at s_k into two *left*-ORDSTs over $S[i..k]$ and $S[k..j]$, where the first one accounts internally for the cost of generating s_k from s_i. Figure 3 (b) deals with the case $k = j$. We see a *left*-ORDST for $S[i..k']$ and a *right*-ORDST for $S[k'+1..j]$. Neither of the two accounts for the generation of s_j from s_i, so this must be added in the decomposition. (In the recurrence given below for this case, k' will be renamed to k.) In other words, the decomposition in Figure 3 (c) can be regarded as the concatenation of the two cases in Figure 3 (b) and (a). These two cases cover all tree topologies, and we reach the following recurrences.

If $i = j$, then $C_l(i,j) = C_r(i,j) = 0$. If $i = j-1$, then $C_l(i,j) = d(s_i, s_j)$ and $C_r(i,j) = d(s_j, s_i)$. (Note that in general $d(s_i, s_j)$ can be different from $d(s_j, s_i)$.)

For $j - i > 1$, we have, according to the above case analysis,

$$C_l(i,j) = \min\{C_l^A(i,j), C_l^B(i,j)\} \tag{3.1}$$

where

$$\begin{aligned} C_l^A(i,j) &= \min_k\{C_l(i,k) + C_l(k,j)\} && i < k < j \\ C_l^B(i,j) &= \min_k\{C_l(i,k) + C_r(k+1,j) + d(s_i,s_j)\} && i \le k < j \end{aligned}$$

By symmetry, $C_r(i,j)$, can be computed as follows:

$$C_r(i,j) = min \begin{cases} \min_k\{C_r(i,k) + C_r(k,j)\} & i < k < j \\ \min_k\{C_l(i,k) + C_r(k+1,j) + d(s_j,s_i)\} & i \le k < j \end{cases}$$

Computing this set of recurrences requires, for every i and j, to iterate over all k, which has a total time complexity of $O(n^3)$. We store for every interval the optimal $C_l(i,j)$ and $C_r(i,j)$, which takes $O(n^2)$ space.

3.2. *Incorporating insertions in optimal trees*

In the case of an insertion, a unit is not derived from a neighboring unit, but imported as an unrelated DNA fragment. This inserted unit may undergo duplication

events. Figure 4 shows an example of an ORDST with insertions over a sequence S. It is not difficult to see that Lemma 2.1 still holds for ORDST with insertions provided that the inserted units are taken into account. It is also clear that the tree topology itself is not affected by whether $s_x \in S$ is inserted or copied from some s_k. In other words, we may consider s_x as derived from some (arbitrary) s_k. The emergence of s_x has an insertion cost $I(s_x)$ instead of $d(s_k, s_x)$. To accommodate insertions in Recurrence 3.1, $d(s_i, s_j)$ must be replaced by $\min\{d(s_i, s_j), I(s_j)\}$. For $C_r(i, j)$, $\min\{d(s_j, s_i), I(s_i)\}$ replaces $d(s_j, s_i)$.

4. The Alignment Algorithm

4.1. *Formal model of map alignment*

Let two sequences $S = s_1, s_2, \ldots, s_n$ and $R = r_1, r_2, \ldots, r_m$ be given. In the alignment of two minisatellite maps, we have the following operations: (1) Match of two units s_i and r_j, (2) Indels in S/R, (3) left duplications in S/R, and (4) right duplications in S/R. Figure 4.1 (left) shows generation rules formalizing the alignment of minisatellite maps.

Relating models for map alignment is difficult. For comparison, we also show in Figure 4.1 (right) the model of Bérard et al. It is not difficult to see that it lacks a rule for simultaneous right duplications in S and R. In the example shown, there is no rule for generating $R[5..6]$ from r_7, and at the same time generating $S[7..9]$ from r_7.We clarify this point with an additional example. Consider $S = ab$ and $R = dc$, where $d(a, b) = d(b, c) = d(c, d) < d(a, c) = d(b, d) < d(a, d)$. The optimal alignment is to match b with c. Then b produces a by right duplication, and c produces d by right duplication. In Bérard et al. model there is no way for generating d from c, while b producing a. Our model overcomes this minor omission. Note also that although the last two rules of Bérard et al. seem to have no counterpart in our model, their effect is achieved by a combination of match, left and right duplication rules.

4.2. *An algorithm for computing an optimal alignment*

Let two sequences $S' = s_1, s_2, \ldots, s_n$ and $R' = r_1, r_2, \ldots, r_m$, $m \leq n$, be given. For ease of presentation, we prepend the character \$ to both S' and R', i.e. the alignment algorithm runs for $S = \$S' = \$, s_1, s_2, \ldots, s_n$ and $R = \$R' = \$, r_1, r_2, \ldots, r_m$. From left-to-right the units of S appear at positions $0,\ 1,\ 2, ..., n$ in S, and similarly for R. The mutation and duplication costs between the unit \$ and any other unit in S' and R' is much larger than the costs between the units of S' and R', i.e., $d(\$, v) \gg d(v, v')$, where $v, v' \in \{S' \cup R'\}$. (The rationale for introducing this unit is to allow that prefixes of S' and/or R' appear as insertions.) Let $\mathcal{C}_s(l, i, x)$ ($\mathcal{C}_r(l', j, y)$) denote the cost of an optimal duplication history of the units $S[l..i]$ ($R[l'..j]$) originating from the unit s_x (r_y). Note that $\mathcal{C}_s(l, i, l)$ ($\mathcal{C}_r(l', j, l')$) corresponds to an optimal *left*-ORDST over the interval $S[l..i]$ ($R[l'..j]$), because the duplications are originated from the leftmost unit s_l ($r_{l'}$). Note also that $\mathcal{C}_s(l, i, i)$ ($\mathcal{C}_r(l', j, j)$) corresponds to

Our Model

Operation	Generation rule
Match:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}s_i & A\\r_j & B\end{smallmatrix}$
Insertion in S:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}s_i & A\\ & B\end{smallmatrix}$
Insertion in R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix} & A\\r_j & B\end{smallmatrix}$
Left dup. in S and/or R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\overrightarrow{S[l..i]} & A\\\overrightarrow{R[l'..j]} & B\end{smallmatrix}$
Right dup. in S and/or R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\overleftarrow{S[l..i]} & A\\\overleftarrow{R[l'..j]} & B\end{smallmatrix}$
Termination:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\epsilon\\\epsilon\end{smallmatrix}$

Example :

$s_1\ \overrightarrow{s_2\ s_3\ s_4\ s_5}\ s_6\ \overleftarrow{s_7\ s_8\ s_9}$
$r_1\ \_\ \sim\ \sim\ \sim\ r_2\ \overleftarrow{r_3\ r_4\ r_5}$

Bérard et al. Model

Operation	Generation rule
Insertion in S:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}s_i & A\\ & B\end{smallmatrix}$
Insertion in R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix} & A\\r_j & B\end{smallmatrix}$
Left dup. in S and/or R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\overrightarrow{S[l..i]} & A\\\overrightarrow{R[l'..j]} & B\end{smallmatrix}$
Dup. in S from a unit in R:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\overleftarrow{S[l..i]} & A\\r_j & B\end{smallmatrix}$
Dup. in R from a unit in S:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}s_i & A\\\overrightarrow{R[l'..j]} & B\end{smallmatrix}$
Termination:	$\begin{smallmatrix}A\\B\end{smallmatrix} \rightarrow \begin{smallmatrix}\epsilon\\\epsilon\end{smallmatrix}$

Example :

$s_1\ \overrightarrow{s_2\ s_3\ s_4\ s_5}\ s_6\ \overleftarrow{s_7\ s_8\ s_9}$
$r_1\ \sim\ \sim\ \sim\ \sim\ \overrightarrow{r_2\ r_3\ r_4}\ r_5$

Fig. 5. The generation rules produces the alignment from left to right, where A and B are the alignment strings for S and R. (Think of them as a Turing machine writing on two tapes A and B.) The units of the interval $S[l..i]$ ($R[l'..j]$) are produced by duplication events, and the arrows $\rightarrow$, $\leftarrow$, and $\leftrightarrow$ above them specify if the duplications originate from the leftmost, rightmost, or any unit in the interval, respectively. In the example under our model, the alignment is generated through the following rules: Matching s_1 to r_1, insertion of s_2, left duplication of $S[2..4]$ originated from s_2, matching s_6 to r_2, right duplication of $S[7..9]$ from s_9 and right duplication of $R[3..5]$ from r_5, and finally matching s_9 to r_5. On the right, we show the Bérard et al. model. In this model there is neither explicit match nor right duplication. Instead, the duplication in S (R) from a unit in R (S) includes these events. In the respective example, r_5 produces $S[7..9]$. If r_5 produces, say s_9, then we can tell that r_5 matches s_9 and there is a right duplication of $S[7..9]$ from s_9.

an optimal *right*-ORDST over the interval $S[l..i]$ ($R[l'..j]$), because the duplications are originated from the rightmost unit s_i (r_j). We have $\mathcal{C}_s(t,t,t) = \mathcal{C}_r(t',t',t') = 0$, where $t \in [0..n], t' \in [0..m]$. Moreover, let $M'(s',s'') = \min\{M(s',s''), M(s'',s')\}$.

Let $\mathcal{A}(i,j)$ be the cost of aligning $S[0..i]$ to $R[0..j]$, where $0 \leq i \leq n$ and $0 \leq j \leq m$. We have the boundary values $\mathcal{A}(0,0) = M(\$,\$) = 0$. $\mathcal{A}(i,j)$ is computed by the following recurrence (noting the limits of i,j):

$$\mathcal{A}(i,j) = \min \begin{cases} M'(s_i,r_j) + \mathcal{A}(i-1,j-1) & \forall\ i>0 \textbf{ and } \forall\ j>0 \\ \mathcal{C}_s(l,i,l) \quad + \mathcal{A}(l,j) & \forall l \in [0,i-1],\ i>0,\ j\geq 0 \\ \mathcal{C}_r(k,j,k) + \mathcal{A}(i,k) & \forall k \in [0,j-1],\ i\geq 0,\ j>0 \\ \mathcal{C}_s(t_s,i,i) + \mathcal{C}_r(t_r,j,j) + & \\ M'(s_i,r_j) + \mathcal{A}(t_s-1,t_r-1) & \forall t_s \in [1..i],\ \forall t_r \in [1..j], \forall i>0, \forall j>0 \end{cases}$$

The optimal duplication costs for each interval originated from the unit either on the left or right boundary are computed in a pre-processing step, using the algorithm of Section 3. This takes totally $O(n^3)$ time and $O(n^2)$ space. Note that indels are not explicitly incorporated in this recurrence, because the duplication histories already take them into account. For computing the recurrences including $\mathcal{C}_s(l,i,l)$ and $\mathcal{C}_r(k,j,k)$, one iterates for all i and j over all l (concurrently k), which takes $O(n^3)$ time. For computing the recurrence involving $\mathcal{C}_s(t_s,i,i)$ and $\mathcal{C}_r(t_r,j,j)$, one iterates for all i and j over all t_s and t_r. This naively takes $O(n^4)$, but the time complexity can be reduced to $O(n^3)$, as follows. The righthand side $\mathcal{C}_s(t_s,i,i) + \mathcal{C}_r(t_r,j,j) + M(s_i,r_j) + \mathcal{A}(t_s-1,t_r-1)$ can be rewritten as $\mathcal{C}_s(t_s,i,i) +$

$M(s_i, r_j) + \mathcal{A}'(t_s - 1, j)$, where $\mathcal{A}'(t_s - 1, j) = \mathcal{A}(t_s - 1, t_r - 1) + \mathcal{C}_r(t_r, j, j)$. If the cost $\mathcal{C}_s(t_s, i, i) + \mathcal{C}_r(t_r, j, j) + M(s_i, r_j) + \mathcal{A}(t_s - 1, t_r - 1)$ is to be minimal, then $\mathcal{A}'(t_s - 1, j)$ must also be minimal. If the value $\mathcal{A}'(t_s - 1, j)$ is already precomputed and stored, then it takes only $O(n^3)$ time to compute the respective recurrence. To this end, an optimal $\mathcal{A}'(t_s - 1, j)$ is computed earlier when computing the alignment $\mathcal{A}(t_s - 1, j)$. For given t_s and j, this computation minimizes over t_r. Altogether, for computing $\mathcal{A}(i, j)$, one has to pre-compute and store an optimal $\mathcal{A}'(t_s - 1, j)$; which takes totally $O(n^3)$ and requires one extra table. Adding the complexity of the preprocessing step, the total complexity is $O(n^3)$ time and $O(n^2)$ space.

The run length encoding (RLE) scheme can be readily incorporated to further reduce the time complexity to $O(n^2 + nn'^2 + n'^3)$, where n' is the length of the run-length compressed sequence. We layout this improvement in Appendix I.

5. Experimental Results

5.1. *Performance evaluation*

For constructing the duplication history, we compare our algorithm to the alphabet-dependent algorithm using different alphabet sizes. We use random sequences between 80 and 60 units long, where each unit can duplicate at most 50 times. Figure 6 (left and middle) shows the results of the experiment over 1000 such random sequences. It is clear that our algorithm is invariant against the alphabet size, while the other is linearly dependent. We show also the results of applying the run-length encoding (RLE) scheme. The usage of RLE scheme has actually attenuated the running time of both algorithms, but our algorithm is still invariant against the alphabet size. Naturally, the effect of RLE decreases when increasing the alphabet *without* also increasing sequence length.

Regarding the alignment phase, we compared our algorithms `MSATcompare` (without RLE) and `MSATcompareRLE` (with RLE) to the only existing program for this task, `MS_ALIGN` [6], that runs without RLE. The table on the right of Figure 6 summarizes the comparison results for simulated and the largest real datasets. Our algorithms are faster than `MS_ALIGN`. Clearly, `MSATcompareRLE` is superior to other algorithms: It reduced the time of analyzing large datasets to a few minutes, including output of results. It is worth mentioning that for the MSY1 dataset our algorithms and `MS_ALIGN` yielded identical pairwise scores; that is, the limitation in the alignment model of Bérard et al. discussed in Section 4.1, did not matter. However, for other datasets, this may not be the case.

5.2. *Detecting directional duplication bias in minisatellites*

We analyzed the updated (and the largest available) dataset MSY1 [2, 12] that contains 465 maps of individuals from different populations. We excluded repeated maps (to have a non-redundant dataset) and also excluded those maps including ambiguous unit types. In the remaining 345 maps, the number of distinct unit types, including *null* repeats, is eight, i.e., the alphabet size is eight. The types are assigned

Without RLE		
$\|\Sigma\|$	Dep.	Indep.
5	147	65
10	262	65
20	472	61
30	703	65
50	1165	65
60	1428	67

With RLE		
$\|\Sigma\|$	Dep.	Indep.
5	0.46	0.46
10	0.59	0.55
20	0.95	0.59
30	1.11	0.56
50	1.5	0.48
60	1.7	0.6

Data	Algn. No.	MS_ALIGN	MSATcompare	MSATcompareRLE
rand 50	1225	5.58	2.3	0.23
rand 100	4950	24.2	10.2	0.98
rand 150	11175	49.8	21.4	2.1
rand 250	3112	161.5	70	5.9
rand 350	61075	317	140	12
MSY1 345	59340	87	25	4.8

Fig. 6. Left and middle tables: The running times, in *seconds*, of constructing duplication history for different alphabet sizes $|\Sigma|$ without and with RLE scheme, respectively; this is measured on a PC with 1.5GHz CPU and 512M RAM. The columns "Dep." and "Indep." are for Alphabet-dependent and independent algorithms, respectively. Right table: Running times, in *minutes*, of our algorithms compared to `MS_ALIGN`. The first four rows contain results for random data, and the last row is for the MSY1 real dataset. ("rand 50" means 50 random sequences). The second column contains the number of pairwise alignments for each set of sequences.

the codes $\{null, 1, 1a, 2, 3, 3a, 4, 4a\}$. The pairwise hamming distances d_H between the units (except for *null*) range between 1 and 3. The *null* repeats are unidentified types due to further base substitutions [2]; hence, we assumed it has $d_H = 4$ to other units. We consider three versions of this dataset. The first includes all 345 unique maps. The second (329 maps) excludes maps with more than 3 adjacent *null* repeats, as suggested in [6]. The third (249 maps) contains no *null* repeats.

The MSY1 dataset was previously analysed for studying population evolution [6]. Here, we ask whether the units duplicate at the $5'$ end (to the left) more than at the $3'$ end (to the right), or not; i.e., to verify the assumption given in [2]. In terms of our alignment model, Subsection 4.1, we want to examine if *left and right duplications* contribute equally to the duplication history. To answer this question, we measured the effect of ignoring left/right duplications in all pairwise alignments by comparing the respective costs to the optimal costs considering both kinds. The rationale is that if both kinds contribute equally, we obtain nearly the same number of alignments with increased cost. Otherwise, the numbers will be different. To this end, we used five cost schemes given in Figure 7 (left). These schemes sample the range from 1 to ∞ of the ratio M/DUP. The fourth scheme is the one recommended by [6], and the final scheme reduces the comparison of two maps to the comparison of their modular structures [2], e.g., the map "*aaabbc*" reduces to "*abc*".

We then ran three experiments under these cost schemes: In experiment E_{full}, we performed all-against-all comparisons over the above-mentioned three dataset versions. In Experiment E_{left}, we allowed only left-duplications (achieved by switching off the recurrence corresponding to right duplications). In experiment E_{right}, we allowed only right duplications (achieved by switching off the recurrences corresponding to left duplications). In the latter two experiments, we counted the number of alignments whose costs are higher than in E_{full}. All results of running E_{left} and E_{right} compared to E_{full} using various distance functions and versions of the dataset are shown in Figure 7 (right).

Interestingly, the number of alignments with optimal cost increased relative to E_{full} is clearly smaller in E_{right} than in E_{left}. Since our model is symmetric, this

	(M,DUP,I)
1	$(1 \times d_H, 1, 40)$
2	$(2 \times d_H, 1, 40)$
3	$(5 \times d_H, 1, 40)$
4	$(10 \times d_H, 1, 40)$
5	$(1 \times d_H, 0, 40)$

Dataset with	Total Algn.	$r=1 \times d_H$		$r=2 \times d_H$		$r=5 \times d_H$		$r=10 \times d_H$		$r = \infty$	
		L	R	L	R	L	R	L	R	L	R
nulls	59340	186	0	616	16	3005	57	1977	127	3219	107
max. 3 *nulls*	53956	148	0	398	0	2403	8	1487	10	2604	44
no *nulls*	30876	0	0	0	0	869	0	876	0	1040	0

Fig. 7. Left: The cost schemes used in our experiments. The triple (M,DUP,I) denotes the costs for mutation $M(x,y)$ as a function of $d_H(x,y)$, duplication $DUP(x)$, and insertion $I(x)$, $x,y \in \{null,1,..,4a\}$. For all schemes, the insertion cost $I = 40$. Right: The results for three versions of the dataset with the inclusion of all, at most three, and no *null* types. The 2nd column contains the total number of pairwise alignments for each dataset version. The other table entries are the number of alignments with costs higher than the optimal under different cost schemes, where $r = M/DUP$ is the ratio between the mutation and duplication costs. The columns titled with "L" and "R" correspond to E_{left} and E_{right}, respectively.

directly suggests a bias *in vivo* in the contribution of left- and right-duplications. To check against artefacts, we generated sequences representing minisatellites of random structural variations. The result was as expected: The number of optimal alignments with increased cost in E_{left} and E_{right}, compared to E_{full}, over this random dataset is nearly the same.

Back to the MSY1 dataset of [2], we can state that left and right duplications do not contribute equally to the duplication history, which (1) supports the idea that the types appear non-randomly at that locus and they are generated at the $5'$ ends with a limitation regarding type mutations [2], or (2) assumes the existence of further unkown dynamic constraints limiting the duplication of the MSY1 units. This observation calls for closer investigation.

References

[1] S. Bérard and E. Rivals, *Computational Biology* **10**, 357 (2003).
[2] M. Jobling, N. Bouzekri and P. Taylor, *Human Molecular Genetics* **7**, 643 (1998).
[3] C. Alkan, J. Bailey, E. Eichler and et al., *Genome Informatics* **13**, 93 (2002).
[4] B. Behzadi and J. M. Steyaert, An improved algorithm for generalized comparison of minisatellites, in *Proc. of the 14th CPM, LNCS 2676*, (Morelia, Mexico, 2003).
[5] B. Behzadi and J. M. Steyaert, The minisatellite transformation problem revisited: A run length encoded approach, in *Proc. of the 4th WABI, LNBI Vol. 3240*, (Bergen, Norway, 2004).
[6] S. Bérard, F. Nicolas, J. Buard and et al., *Evolutionary Bioinformatics* **2**, 327 (2006).
[7] D. Jaitly, P. Kearney, G. Lin and et al., *Computer and Systems Sciences* **65**, 494 (2002).
[8] O. Gascuel, M. D. Hendy, A. Jean-Marie and et al., *Systematic Biology* **52**, 110 (2003).
[9] M. Tang, M. Waterman and S. Yooseph, *Computational Biology* **9**, 429 (2002).
[10] L. Zhang, B. Ma, L. Wang and et al., *Bioinformatics* **19**, 1497 (2003).
[11] G. Benson and L. Dong, Reconstructing the duplication history of a tandem repeat, in *Proc. of the 7th ISMB*, (Heidelberg, Germany, 1999).
[12] N. Bouzekri, P. Taylor, M. Hammer and et al., *Human Molecular Genetics* **7**, 655 (1998).

Appendix I: The Inclusion of Run-length Encoding Scheme

In Run-Length Encoding (RLE) of sequences, i consecutive occurrences of symbol x is shown by x^i, which is called a *run*. For example, $S = aaaabbbbcccabbbcc$ is encoded as $a^4b^4c^3a^1b^3c^2$. For brevity, we call the sequence of symbols in the encoded sequence the *compressed sequence* and denote it with S', in the previous example $S' = abcabc$. Minisatellites are ideal patterns for using run-length encoding technique, because they consists of a large number of tandem repeats. As a result, the length of run-length encoded representation of minisatellite sequences is generally much shorter than the initial length.

The RLE technique helps us in both computing the duplication history and the alignment algorithm. If we compute the duplication histories for the RLE version of strings, then $C_l(i,j)$ and $C_r(i,j)$ for the non-compressed sequences, can be computed in constant time for any i and j. More precisely, let $e(i)$ denote the position of $S[i] = x_i$ in the compressed sequence S', and let $\hat{C}_l(i,j)$ ($\hat{C}_r(i,j)$) denote the cost for constructing the history in S'. If $DUP(x_i)$ is constant for all x_i, then $C_l(i,j) = \hat{C}_r(e(i),e(j)) + (j-i) - (e(j)-e(i))$. Otherwise, $C_l(i,j) = \hat{C}_r(e(i),e(j)) + F(j) - F(i) - \sum_{y=e(i)}^{y=e(j)} DUP(S'[y])$, where $F(i) = \sum_{r=1}^{r=i} DUP(s_r)$. Note that $e(i)$ and $F(i)$ can be pre-computed in linear time.

The score $\mathcal{A}(i,j)$ in the alignment algorithm can be correctly determined without iterating over all possible values of k, l, t_s and t_r. More precisely, it is enough to iterate over the block boundaries in S and R. Moreover, these iterations are launched only if i or j are block boundaries. We show how this works for the second clause that involves left duplications in S in the alignment recurrence of Subsection 4.2. Let $\hat{E}_l(i)$ and $\hat{E}_r(i)$ be the set of positions in S of all leftmost and rightmost boundaries of each block before $s_i \in S$. Let $\hat{E}(i) = \hat{E}_l(i) \cup \hat{E}_r(i)$ be an ordered list, w.r.t. positions in S, and let E_x denote the x^{th} entry in this list. During the iteration, if $s_i = s_{i-1}$, i.e., s_i is not the leftmost unit of a block. Then it is enough to consider only one value of l, namely $l = i-1$. The idea is that $s_{i-1} = s_i$, and the iteration over other values of $l \in [0..i]$ yields no better score. If $s_i \neq s_{i-1}$, it is enough that l iterates over all values of $\hat{E}(i)$. The idea is as follows: Let $l_x < i$ denote the position of an element within a block E_x (E_x is the position of the leftmost unit of block $x/2+1$), then $\mathcal{C}(l_x,i,l_x) = \mathcal{C}(E_{x+1},i,E_{x+1}) + Dup(l_x) \times (E_{x+1} - l_x)$, where E_{x+1} is the rightmost unit. The optimal value $A(l_x,j) + \mathcal{C}(l_x,i,l_x) = A(l_x,j) + \mathcal{C}(E_{x+1},i,E_{x+1}) + Dup(l_x) \times (E_{x+1} - l_x) = A(E_{x+1},j) + \mathcal{C}(E_{x+1},i,E_{x+1})$. The time complexity is derived as follows: for each s_j, we examine each s_i and s_{i-1}. We iterate over all the block boundaries in S only if i is a block boundary. This yields $O(n^2 + nn'^2)$ time. With similar arguments, we can prove how the RLE scheme works for the other clauses in the recurrence.

Computing the duplication histories for the RLE version of strings using our algorithm takes $O(n'^3)$. The aligment phase will take $O(n^2 + nn'^2)$. That is, our algorithm in this paper with the RLE techniques takes $O(n^2 + nn'^2 + n'^3)$ time and $O(n^2)$ space, which is also independent of the alphabet size.