Genomic Distances under Deletions and Insertions [1]

Mark Marron Krister M. Swenson Bernard M.E. Moret [2]

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, USA
ma_luen@eece.unm.edu,{kswenson,moret}@cs.unm.edu

Abstract

As more and more genomes are sequenced, evolutionary biologists are becoming increasingly interested in evolution at the level of whole genomes, in scenarios in which the genome evolves through insertions, deletions, and movements of genes along its chromosomes. In the mathematical model pioneered by Sankoff and others, a unichromosomal genome is represented by a signed permutation of a multiset of genes; Hannenhalli and Pevzner showed that the edit distance between two signed permutations of the same set can be computed in polynomial time when all operations are inversions. El-Mabrouk extended that result to allow deletions (or conversely, a limited form of insertions which forbids duplications). In this paper we extend El-Mabrouk's work to handle duplications as well as insertions and present an alternate framework for computing (near) minimal edit sequences involving insertions, deletions, and inversions. We derive an error bound for our polynomial-time distance computation under various assumptions and present preliminary experimental results that suggest that performance in practice may be excellent, within a few percent of the actual distance.

1 Introduction

Biologists can infer the ordering and strandedness of genes on a chromosome, and thus represent each chromosome by an ordering of signed genes (where the sign indicates the strand). These gene orders can be rearranged by evolutionary events such as inversions (also called reversals) and transpositions and, because they evolve slowly, give biologists an important new source of data for phylogeny reconstruction (see, e.g., (6; 13; 14; 16)). Appropriate tools for analyzing such data may help resolve some difficult phylogenetic reconstruction problems. Developing

[1] A preliminary version of this work appeared in (11).
[2] To whom all correspondence should be addressed.

such tools is thus an important area of research—indeed, the recent DCAF symposium (18) and IMA/RECOMB Workshop on Comparative Genomics (9) were devoted in good part to this topic.

A natural optimization problem for phylogeny reconstruction from gene-order data is to reconstruct an evolutionary scenario with a minimum number of the permitted evolutionary events on the tree. This problem is NP-hard for most criteria—even the very simple problem of computing a median [3] of *three* genomes with identical gene content under such models is NP-hard (4; 15). The problem of computing the edit distance between two genomes is itself difficult: for instance, even with equal gene content and with only inversions allowed, the problem is NP-hard for unsigned permutations (3).

Hannenhalli and Pevzner (8) made a fundamental breakthrough by developing an elegant theory for signed permutations and providing a polynomial-time algorithm to compute the edit distance (and the corresponding shortest edit sequence) between two signed permutations under inversions; Bader et al. (1) later showed that this edit distance is computable in linear time. El-Mabrouk (7) extended the results of Hannenhalli and Pevzner to the computation of edit distances for inversions and deletions and also for inversions and non-duplicating insertions; she also gave an approximation algorithm with bounded error for computing edit distances in the presence of all three operations (inversions, deletions, and non-duplicating insertion). Liu *et al.* showed that edit distances that allow only inversions and deletions can be computed in linear time (10).

In this paper, we extend El-Mabrouk's work by providing a polynomial-time approximation algorithm with bounded error to compute edit distances under inversions, deletions, and unrestricted insertions (including duplications) from the perfectly sorted sequence (the *identity* sequence) to any other. Our approach is based on a new canonical form for edit sequences: we show that shortest edit sequences can be transformed into equivalent sequences of equal length in which all insertions are performed first, followed by all inversions, and then by all deletions. This canonical form allows us to take advantage of El-Mabrouk's exact algorithm for inversions and deletions, which we then extend by finding the best possible prefix of insertions, producing an approximate solution with bounded error.

Section 2 introduces some notation and definitions. Section 3 gives two key theorems that enable us to reduce edit sequences to a canonical form. Section 4 outlines our method for handling unrestricted insertions. Section 5 presents the complete algorithm as well as an analysis of its error bounds. Section 6 gives some empirical results for the method presented here. Section 7 introduces a more general method that can be applied to arbitrary pairs of sequences.

---

[3] The median of k genomes is a genome that minimizes the sum of the pairwise distances between itself and each of the k given genomes.

2 Notation and Definitions

We denote a particular edit sequence with a Greek letter, π, its operations by subscripted letters, o_i, and its contents enclosed in angle brackets: $\pi = \langle o_1, o_2, \ldots, o_n \rangle$. We assume that the desired (optimal) edit sequence is that which uses the fewest operations, with all operations counted equally. As in the standard statement of the equal gene-content problem, we move from a subject sequence S to a perfectly sorted target sequence T with sequence elements in $\mathbb{Z}$.

We say that substring s_i is *adjacent* to substring s_j whenever they occupy sequential indices in a string. Let $\text{sign}_{\text{min}}(s_l)$ be the sign of the element of smallest index in s_l and $\text{sign}_{\text{max}}(s_l)$ be the sign of the element of largest index in s_l; we define the *parity*, ξ, of a pair of ordered strings (s_i, s_j) as $\text{sign}_{\text{min}}(s_i) \cdot \text{sign}_{\text{max}}(s_j)$.

Let σ be the *ordering* defined when s_i and s_j are single-element strings consisting of the elements e_i and e_j respectively; we just set $\sigma = e_i - e_j$. As an example, suppose we have $s_i = 5$ and $s_j = 3$; then we have $\sigma = e_i - e_j = 5 - 3 = 2$. Given a subject sequence and the target sequence, we say that two substrings, s_i and s_j, are *correctly oriented* relative to each other if and only if:

(1) s_i or s_j is ε.
(2) s_i and s_j are both of length 1 and adjacent with ordering σ in the target and source sequences.
(3) All substrings in s_i are correctly oriented relative to each other, all substrings in s_j are correctly oriented relative to each other and s_i is adjacent to s_j with parity ξ in the target and source sequences.

We say that an operation *splits* s_i and s_j if the two sequences are correctly oriented before the operation, but not after it.

3 Canonical Forms

In this section, we prove useful results about shortest transformation sequences, results that will enable us to obtain a *canonical form* into which any shortest sequence can always be transformed without losing optimality.

We make use throughout our derivation of operation reindexing; this reindexing provides a pliability to the indices that operations act upon so that their order can be manipulated. For example, take the string $(1,2,3,\text{-}5,\text{-}4,6,7,11,12)$ and suppose that the next operation to perform is an inversion starting at index 4 and going to index 7 (inclusive). This operation yields the new string $(1,2,3,\text{-}7,\text{-}6,4,5,11,12)$. Now, suppose that, in order to achieve a desired form, we needed an insertion of the element 10 at index 4 to precede the application of this inversion. The goal is to maintain the indices of the inversion so that it continues to act upon the substring (-5,-4,$6,7$). After the application of the insertion we are left with the string $(1,2,3,10,$-5,-4,$6,7,11,12)$. In order to maintain the integrity of the inversion, we adjust the start index of an inversion to be at 5 and the end index to be at 8. Application of the inversion from index 5 to index 8 will now yield the desired string $(1,2,3,10,$-7,-6,$4,5,11,12)$. The other types of reindexing that we use for inversions and deletions follow a similar pattern.

Our first theorem extends an earlier result of Hannenhalli and Pevzner (who proved that a sorted substring need not be split in an inversion-only edit sequence (8)) by showing that, whenever two substrings are correctly oriented, there is always a minimum edit sequence that does not split them. The idea behind this result is to show that the optimal edit sequence can be rewritten to keep the substrings together. First define $move(s_x, s_y, \xi)$ to move s_x to the immediate left of s_y resulting with parity ξ between s_x and s_y. Given an edit sequence $\langle o_1, o_2, \ldots, o_k, \ldots, o_m, \ldots \rangle$, where operation o_k is responsible for splitting the substrings s_i and s_j and operation o_m returns them to their correctly oriented state, we rewrite the operations to keep the substrings together. To accomplish this, each o_x, for $k \leq x \leq m$, is expanded into a tuple of operations $\langle f_x, \hat{o_x}, t_x \rangle$, in which f_x and t_x are *move* operations. This tuple is constructed so that the x^{th} tuple is functionally equivalent to o_x and t_x is the inverse of f_{x+1}; furthermore, the leading and trailing tuples are designed so that f_k and t_m are identity operations.

We illustrate this construction through a simple example. Suppose we have the sequence $(14,15,11,12,13,16)$ and the sorting sequence is $inv(2,5)$, $inv(1,4)$, $inv(4,4)$, $inv(5,5)$. The first operation in this sequence splits the substring $14,15$ and the fourth restores it. We can construct a new sorting sequence of the same length that preserves the adjacency $14,15$ by constructing the tuples as follows:

(1) $inv(2,5) \rightarrow \langle move(14,15,1), inv(1,5), move(\text{-}14,\text{-}13,\text{-}1) \rangle$
(2) $inv(1,4) \rightarrow \langle move(14,16,\text{-}1), inv(1,3), move(\text{-}14,\text{-}15,1) \rangle$
(3) $inv(4,4) \rightarrow \langle move(\text{-}14,16,\text{-}1), inv(\varepsilon), move(\text{-}14,\text{-}15,1) \rangle$
(4) $inv(5,5) \rightarrow \langle move(\text{-}14,16,\text{-}1), inv(4,5), move(14,15,1) \rangle$

The original sorting sequence produces:

$$
\begin{array}{lcl}
14,15,11,12,13,16 & \stackrel{inv(2,5)}{\Longrightarrow} & 14,\text{-}13,\text{-}12,\text{-}11,\text{-}15,16 \\
 & \stackrel{inv(1,4)}{\Longrightarrow} & 11,12,13,\text{-}14,\text{-}15,16 \\
 & \stackrel{inv(4,4)}{\Longrightarrow} & 11,12,13,14,\text{-}15,16 \\
 & \stackrel{inv(5,5)}{\Longrightarrow} & 11,12,13,14,15,16
\end{array}
$$

The new sequence of triples produces:

$$
\begin{array}{lclcl}
14,15,11,12,13,16 & \xrightarrow{move(14,15,1)} & 14,15,11,12,13,16 & \xrightarrow{inv(1,5)} & \\
 & & \text{-}13,\text{-}12,\text{-}11,\text{-}15,\text{-}14,16 & \xrightarrow{move(\text{-}14,\text{-}13,\text{-}1)} & 14,\text{-}13,\text{-}12,\text{-}11,\text{-}15,16 \\
14,\text{-}13,\text{-}12,\text{-}11,\text{-}15,16 & \xrightarrow{move(14,16,\text{-}1)} & \text{-}13,\text{-}12,\text{-}11,\text{-}15,\text{-}14,16 & \xrightarrow{inv(1,3)} & \\
 & & 11,12,13,\text{-}15,\text{-}14,16 & \xrightarrow{move(\text{-}14,\text{-}15,1)} & 11,12,13,\text{-}14,\text{-}15,16 \\
11,12,13,\text{-}14,\text{-}15,16 & \xrightarrow{move(\text{-}14,16,1)} & 11,12,13,\text{-}15,\text{-}14,16 & \xrightarrow{inv(\varepsilon)} & \\
 & & 11,12,13,\text{-}15,\text{-}14,16 & \xrightarrow{move(\text{-}14,\text{-}15,\text{-}1)} & 11,12,13,14,\text{-}15,16 \\
11,12,13,14,\text{-}15,16 & \xrightarrow{move(\text{-}14,16,\text{-}1)} & 11,12,13,\text{-}15,\text{-}14,16 & \xrightarrow{inv(4,5)} & \\
 & & 11,12,13,14,15,16 & \xrightarrow{move(14,15,1)} & 11,12,13,14,15,16
\end{array}
$$

It is evident that the end product of each triple (the rightmost column of permutations) is the same as that of the corresponding original operation; it can also easily be seen that the permutation before the third operation of each triple is identical to that produced by the first operation of the following triple (the middle column of permutations in each case). The new sorting sequence, $inv(1,5)$, $inv(1,3)$, $inv(\varepsilon)$, $inv(4.5)$, produces:

$$
\begin{array}{lcl}
14,15,11,12,13,16 & \xrightarrow{inv(1,5)} & \text{-}13,\text{-}12,\text{-}11,\text{-}15,\text{-}14,16 \\
 & \xrightarrow{inv(1,3)} & 11,12,13,\text{-}15,\text{-}14,16 \\
 & \xrightarrow{inv(\varepsilon)} & 11,12,13,\text{-}15,\text{-}14,16 \\
 & \xrightarrow{inv(4,5)} & 11,12,13,14,15,16
\end{array}
$$

Thus it preserves the $14,15$ adjacency throughout; moreover, we see directly that the new sorting sequence is in fact shorter, since one of its operations is an identity.

This example illustrates how the construction of the tuples can create an operation sequence where each tuple has the same effect as its corresponding operation in the original sequence and the opposing move operations cancel one another's effect (and can thus be discarded).

We formalize this insight with the following theorem—whose proof, easy but tedious, is omitted.

Theorem 1 *If subsequences s_i and s_j are correctly oriented relative to each other at some step during the execution of the minimum edit sequence π, say at the kth step, then there is another minimum edit sequence, call it π', that has the same first k steps as π, and never splits s_i and s_j. (For simplicity we assume that duplications do not arise.)*

Our next theorem shows that it is always possible to take any minimum edit sequence and transform it into a form where all of the insertions come first, followed by all of the inversions and then all of the deletions. The proof is again based on the idea of rewriting each operation preceding the first insert such that, at the beginning and end of each operation rewrite group, the sequence is the same as at each step in the original sequence, but when the terms are regrouped and cancellation occurs, the insert is pushed to the front of the operator sequence. (Once again, each rewrite group is a triple, but this time, the first member of each triple is an insertion, while the third member is a deletion.) Since each step produces the same sequence, we know that the resulting edit sequence is correct and the cancellation maintains the same number of operations in the new sequence as in the old one.

Theorem 2 *Given a minimal edit sequence* $\pi = \langle o_1, o_2 \ldots o_{k-1}, ins_1, o_{k+1} \ldots o_m \rangle$ *there is a* π' *such that (i)* $\pi' \equiv \pi$*; (ii)* $|\pi'| = |\pi|$*; and (iii)* $\pi' = \langle ins_1 \ldots ins_p, inv_1 \ldots inv_q, del_1 \ldots del_r \rangle$.

PROOF. For each $o_j \ni j \leq k-1$, set $\hat{o_j} = (ins'_j, o'_j, del'_j)$, with $del'{}_j = ins'{}^{-1}_{j+1}$ for $j \in [1, k-2]$ and ins_1^{-1} for $j = k-1$, where ins'_j is the inverse of del'_j and o'_j is o_j reindexed to compensate for the insertion. Thus del'_j deletes whatever was inserted by ins'_j when o'_j is applied and the construction of each tuple ensures $o_j \equiv \hat{o_j}$.

Write

$$\pi' = \langle \hat{o_1} \ldots, \hat{o_{k-1}}, ins_1, o_{k+1} \ldots, o_m \rangle$$

Expanding each term $\hat{o_j}$, we get

$$\pi' = \langle (ins'_1, o'_1, del'_1), (ins'_2, o'_2, del'_2), \ldots, (ins'_{k-1}, o'_{k-1}, del'_{k-1}), ins_1, \ldots, o_m \rangle$$

Since del'_j and ins'_{j+1} as well as del'_{k-1} and ins_1 cancel by construction, the expression reduces to

$$\langle ins'_1, o'_1, o'_2, \ldots, o'_{k-1}, o_{k+1}, \ldots, o_m \rangle$$

The construction for o'_j ensures that each $\hat{o_j}$ sequence is equivalent to o_j and the cancellation of the *ins* and *del* operators in $\hat{o_j}$, results in $|\pi| = |\pi'|$.

This reasoning shows how to move the first insertion to the front of the sequence; further insertion operations can be moved similarly. □

These two theorems allow us to define a canonical form for edit sequences. That canonical form includes only inversions and deletions in its second and third parts, which is one of the cases for which El-Mabrouk gave an exact polynomial-time algorithm. We can use her algorithm to find the minimal edit sequence of inversions and deletions, then reconstruct the preceding sequence of insertions. Because this approach fixes the sequence of inversions and deletions without taking insertions into account, and then only addresses insertions, it is an approximation, not an exact algorithm. We shall prove that the error is bounded and also give evidence that, in practice, the error is very small.

4 Unrestricted Insertions

4.1 The Problem

The presence of duplicates in the sequence makes the analysis much more difficult; in particular, it prevents a direct application of the method of Hannenhalli and Pevzner's and thus also of that of El-Mabrouk's. We can solve this problem by assigning distinct names to each copy, but this approach begs the question of how to assign such names. Sankoff proposed the exemplar strategy (17), which attempts to identify, for each family of gene copies, the "original" gene (known in biology as *ortholog*) as distinct from its copies (known is biology as *paralogs*), and then discards all copies, thereby reducing a multiset problem to the simpler set version. However, identifying exemplars is itself NP-hard (2)—and much potentially useful information is lost by discarding copies. Fortunately, we found a simple selection method, based on substring pairing, that retains a constant error bound.

4.2 Sequence Covers

Our job is to pick a group of substrings from the subject such that every element in the target appears in one of those substrings. To formalize and use this property, we need a few definitions. Call a substring $e_1e_2\ldots e_n$ *contiguous* if we have $\forall j,\, e_{j+1} = e_j + 1$. Given a contiguous substring s_i, define the *normalized* version of s_i to be s_i itself if the first element in s_i is positive and $inv(s_i)$ otherwise; thus the normalized version of s_i is a substring of the identity. Call a subsequence T_{nd} of the target string T, the *non-deleted* portion of T if T_{nd}, viewed as a set, is the largest subset of elements in T that is also contained in the subject string S (also viewed as a set). (Note that T_{nd} is not a substring, but a subsequence; that is, it may consist of several disjoint pieces of T; thus, in particular, it is unique.) Given a subset C of the set of normalized maximal contiguous substrings in S, we define $\uplus C$ to be the string produced by ordering the strings of C lexicographically and concatenating them in that order, removing any overlap. We say that a set C of contiguous substrings from S is a *cover* for T if T_{nd} is $\uplus C$. Note that a cover must contain only contiguous strings. We call the *size* of a cover the number of string from C used in $\uplus C$.

For example, pick $T = (1,2,3,4,5,6,7)$ and $S = (3,4,5,\text{-}4,\text{-}3,5,6,7)$. The set of normalized maximal contiguous substrings is $\{(3,4,5),(3,4),(5,6,7)\}$; T_{nd} is $(3,4,5,6,7)$; a possible cover for T is $\{(3,4,5),(5,6,7)\}$; and $\uplus C_p$ is $(3,4,5,6,7)$. The size of this cover is 2.

Let n be the size of (number of operations in) the minimal edit sequence.

Theorem 3 *There exists a cover for S of size* $2n+1$.

PROOF. By induction on n. For $n = 0$, S itself forms its own cover, since it is a contiguous sequence; hence the cover has size 1, obeying the bound. For the inductive step, note that deletions are irrelevant, since the cover only deals with the non-deleted portion; thus we need only verify that insertions and inversions obey the bound. An insertion between two contiguous sequences simply adds another piece, while one inside a contiguous sequence splits it and adds itself, for an increase of two pieces. Similarly, an inversion within a contiguous sequence cuts it into at most three pieces, for a net increase of two pieces, while an inversion across two or more contiguous sequences at worst cuts each of the two end sequences into two pieces, leaving the intervening sequences contiguous, also for a net increase of two pieces. Since we have $(2(n-1)+1)+2 = 2n+1$, the bound is obeyed in all cases. $\square$

4.3 Building the Minimal Cover

Let $\mathbf{C}(S)$ be the set of all (normalized versions of) maximal contiguous substrings of S. We will build our cover greedily from left to right with this simple idea: if, at some stage, we have a collection of strings in the current cover that, when run through the $\uplus$ operator, produces a string that is a prefix of length i of our target T, we consider all remaining strings in $\mathbf{C}(S)$ that begin at or to the left of position i (i.e., that can extend the current cover) and select that which extends farthest to the right of position i. Although this is a simple (and efficient) greedy construction, it actually returns a minimum cover, as we can easily show by contradiction.

Proposition 4 *The cover derived by our greedy algorithm is optimal.*

PROOF. We proceed by contradiction. Assume there exists a cover, say $C_{\min}$, that is smaller than the one provided by our construction, C_{const}. Order the sequences in $C_{\min}$ by increasing value of the smallest index in the sequence. Let α be the smallest element, say the kth element in this order such that α is not the same as the kth sequence of C_{const} under the same order. We have three cases:

(1) During the construction of C_{const}, α was not selected for C_{const} because the previous selection of a cover element in C_{const} did not cover all the way to the start index of α. Then α is not the first differing element in the order, a contradiction.
(2) During the construction of C_{const}, α was not selected for C_{const} because there was a sequence that had the same start index as α, but covered fewer elements than α. But this contradicts the selection criteria for our construction.
(3) During the construction of C_{const}, α was not selected for C_{const} because there was a sequence that had the same start index as α, but covered more elements than α. Then C_{const} has at most as many elements as $C_{\min}$, a contradiction. $\square$

5 Our Algorithm

Now that we have a method to construct a minimal cover, we can assign unique labels to all duplicates. which in turn enables the use of El-Mabrouk's approximation method. However, for greater control of the error and to cast the problem into a more easily analyzed form, we choose to use El-Mabrouk's exact method for deletions only, and then to extend the resulting solution to handle the needed insertions.

Theorem 5 *Let π be the minimal edit sequence from S to T, using l insertions and m inversions. Let T_{ir} denote T with all of the elements that do not appear in S removed. Let π' be the minimal edit sequence of just inversions and deletions from S to T_{ir}. The extension $\hat{\pi}$ of π' (with the needed insertions) has at most $l+m$ insertions.*

PROOF. Clearly, our method will do at least as well as looking at each inserted string in T and taking that as an insertion for $\hat{\pi}$. Now, looking at the possible effect of each type of operation on splitting a previous insertion, we have 3 cases:

(1) Inserting another substring cannot split an inserted substring—it just creates a longer string of inserted elements. (If x is inserted, $u\boxed{v_1v_2}w \rightarrow u\boxed{v_1xv_2}w$)
(2) Deletion of a substring cannot split an inserted substring—it just shortens it, even perhaps to the point of eliminating it and thus potentially merging two neighboring strings. (If v_2 is deleted, $u\boxed{v_1v_2v_3}w \rightarrow u\boxed{v_1v_3}w$)
(3) An inversion may split an inserted substring into two separate strings, thus increasing the number of inserted substrings by one. It cannot split a pair of inserted substrings because the inversion only rearranges the inserted substrings; it does not create new contiguous substrings. (If u_2v_1 is the substring inverted, $u_1u_2\boxed{v_1v_2}w \rightarrow u_1\boxed{\text{-}v_1}\text{-}u_2\boxed{v_2}w$)

Thus, if we have l insertions and m inversions in π, there can be at most $l+m \leq |\pi| = n$ inserted substrings in T. $\square$

If our selected cover C (of S) was the same as an optimal cover C_o (of S), we could sort our string with the optimal edit sequence π. Thus we can count how far off we are from $|\pi|$ for each wrong choice in our cover. The proof of this result is a constructive case analysis and is quite tedious. But to give an idea of how one would proceed, we provide a brief example. Let $(s_a, s_x, s_u, s_y, s_z)$ be substrings of S, let the $'$ denote that a particular substring was marked as a copy by a given cover, and let π_{tail} be the inversion and deletion portion of π. Set $S = (s_a \ldots s_x s_u s_y \ldots s_z)$ and $T = (s_a \ldots s_x s_u s_y \ldots s_u \ldots s_z)$; let T renamed according to the optimal covering be denoted $T_{\text{opt}} = (s_a \ldots s_x s'_u s_y \ldots s_u \ldots s_z)$; and let T renamed according to the chosen covering be denoted $T_{\text{chosen}} = (s_a \ldots s_x s_u s_y \ldots s'_u \ldots s_z)$. Further, suppose that s_u is at index I_1 in S and s'_u, according to T_{opt}, is inserted at index I_2. The construction proceeds by moving s_u to the location of the insertion of s'_u, then inserting s'_u in the location that s_u was in. Thus, for each wrong choice in the cover, we need three inversions to move and one to insert. In the given example, s_u should be moved to I_2 and s'_u should be inserted at index I_1. Now π_{tail} can be applied to this modified sequence to produce T_{chosen}.

If the minimal edit sequence is n operations long, then we have $|C| \leq 2n+1$. Assuming that each of the selections in C is in error and taking the results from above, the worst-case sequence that can be constructed from C is bounded by $4(2n+1) = 8n+4$ operations. Finally the extension of the edit sequence to include the insertions adds at most n insertions. Thus, the edit sequence produced by the proposed method has at most $9n+4$ operations.

Theorem 6 *The algorithm has an error bounded by* $9n+4$ *where n is the number of edit operations in the minimal edit sequence.*

While this error bound is large, it is a constant; it is also unrealistically large, as the assumptions used are not simultaneously realizable. Furthermore, the bounds can be easily computed on a case-by-case basis in order to provide information on the accuracy of the results for each run. Thus, we expect the error encountered in practice to be much lower and that further refinements in the algorithm and error analysis will bring the bound to a reasonable level.

6 Experimental Results

To test our algorithm and get an estimate of its performance in practice, we ran simulations. We generated pairs of sequences, one the sequence $(1,2,3,\ldots,n)$, for $n = 200, 400, 800$, and the other derived from the first through an edit sequence. Our edit sequences, of various lengths, include 80% of randomly generated inversions (the two boundaries of each inversions are uniformly distributed through the array), 10% of deletions (the left end of the deleted string is selected uniformly at random, the length of the deleted string is given by a Gaussian distribution of mean 20 and deviation 7), and 10% of insertions (the locus of insertion is uniformly distributed at random and the length of the inserted string is as for deletion), with half of the insertions consisting of new elements and the other half repeating a substring of the current sequence (with the initial position of the substring selected uniformly at random). Thus, in particular, the expected total number of duplicates in the subject sequence equals the generated number of edit operations—up to 400 in the case of 800-gene sequences. We ran 10 instances for each combination of parameters (in the figures below, we show the average, minimum, and maximum values over the 10 instances).

The results are gratifying: the error is consistently very low, with the computed edit distance staying below 3% of the length of the generated edit sequence in the linear part of the curve—that is, below saturation. (Of course, when the generated edit sequence gets long, we move into a regime of saturation where the minimum edit sequence becomes arbitrarily shorter than the generated one.) Figures 1, 2, and 3 show our results for sequences of 200, 400, and 800 genes, respectively.

7 Moving Beyond the Identity

To extend these results, we are studying the same problem with arbitrary subjects and targets; in other words, the target is no longer restricted to unique elements—a much more useful setting, since the genomes of most species contain many duplicated genes.

Building the cover with the identity permutation as the target was relatively easy, because all candidate cover elements from the subject were immediately apparent. With an arbitrary target, this correlation no longer exists. We do not yet know whether a minimal cover can be built efficiently between two arbitrary sequences. However, cases where it is hard to find a minimal cover are quite specialized and probably not seen in nature; moreover, when such a case does arise, it is unlikely to cause a large error.

We have implemented an algorithm to find such a cover and incorporated it into our code base; we have started to evaluate our approach and give here some preliminary results. We test this new method against tree distances rather than pairwise distances, since it is always tree distances that we need in phylogenetic reconstruction. Thus, in our current experiments, we generate a tree according to some model, evolve genomes down the tree under inversions, deletions, insertions, and duplications, and store the distance (based on the number of operations on the tree edges between the two leaves) between each pair of leaves in the tree. We then compare our distance estimates to these stored values.

Figure 4 shows results obtained on trees of 16 taxa, where the root genome had 800 genes, the edge lengths (number of evolutionary events along the edge) followed a Gaussian distribution around the indicated mean, and specific events were generated according to the same system as described in our earlier experiments. These results are representative of a variety of evolutionary parameters that we have used. Although the results are encouraging, it can be seen that they usually overestimate the tree distance and also that they suffer from significant variance when edges are long. Thus, our next step is to look at techniques that can be used to reduce the variance. Our initial work indicates that much of the error sensitivity is a result of the cover selection. Our goal is to improve selection heuristics and apply methods to reduce the chances that a single poor selection will bias the result too much. Fortunately, this problem is amenable to a large number of standard approximation and search techniques.

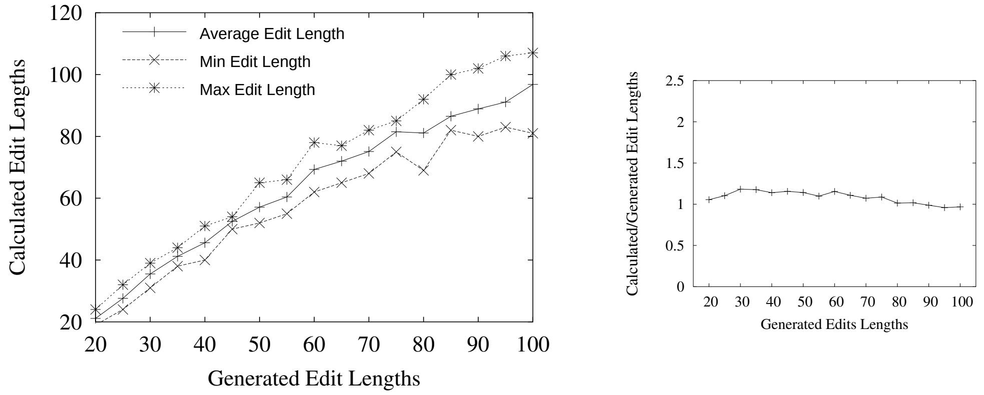


Fig. 1. Experimental results for 200 genes. Left: generated edit length vs. reconstructed length; right: the ratio of the two.

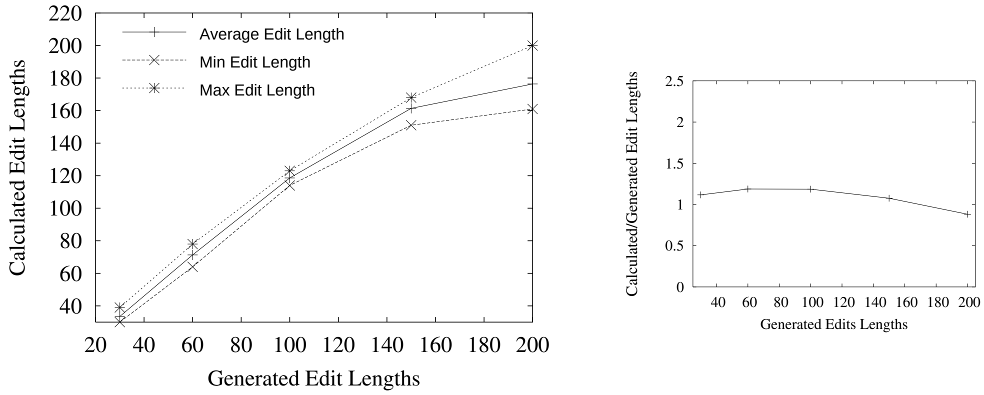


Fig. 2. Experimental results for 400 genes. Left: generated edit length vs. reconstructed length; right: the ratio of the two.

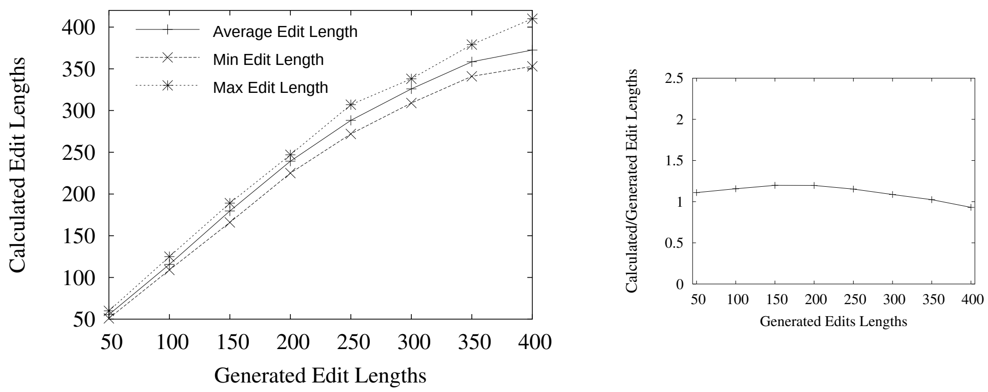
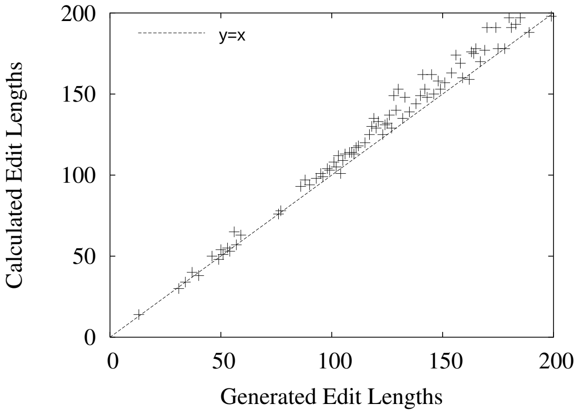
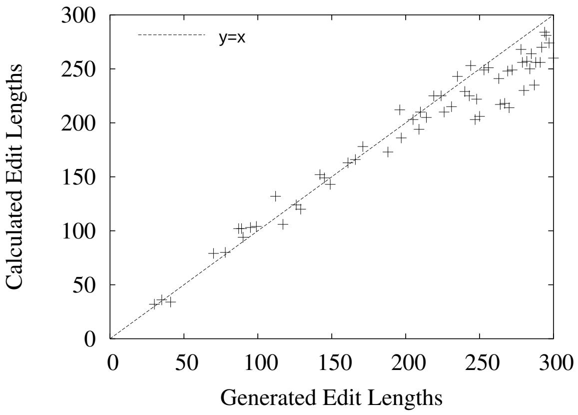


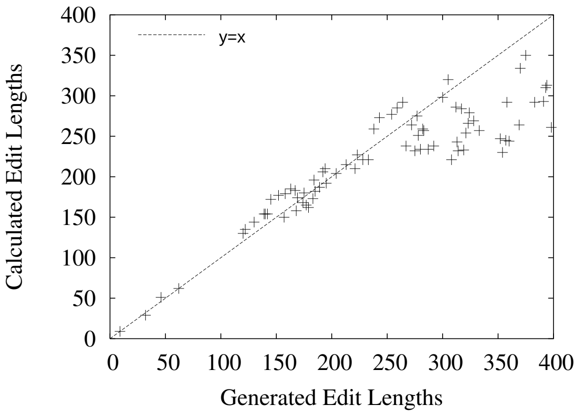
Fig. 3. Experimental results for 800 genes. Left: generated edit length vs. reconstructed length; right: the ratio of the two.



(a) expected edge length 20



(b) expected edge length 40



(c) expected edge length 60

Fig. 4. Tree distance estimates on trees of 16 taxa with an initial content of 800 genes and various expected tree edge lengths.

With these improvements and further empirical tests we expect to reduce the error of the method substantially and then to apply it to the problem of tree reconstruction and labeling of the internal nodes.

8 Conclusion and Future Directions

An exact polynomial-time algorithm for the computation of genomic distances under arbitrary insertions, deletions, and inversions remains to be found, but our work takes us a step closer in that direction. More thorough experimental testing will determine how well our algorithm does in practice under different regimes of insertion, deletion, and duplication, but our results to date are very encouraging. In order to be usable in many reconstruction algorithms, however, a further, and much more complex, computation is required: the median of three genomes. This computation is NP-hard even under inversions only (4; 15)—although the algorithms of Caprara (5) and of Siepel and Moret (19) have done well in practice (see, e.g., (12)). Good bounding is the key to such computations; our covering technique may be extendible to median computations.

9 Acknowledgments

This work is supported by the National Science Foundation under grants ACI 00-81404, DEB 01-20709, EIA 01-13095, EIA 01-21377, EIA 02-03584, and EF 03-31654. The first two authors would also like to thank Linda, Sarah, and Sage for their editorial assistance.

References

[1] D.A. Bader, B.M.E. Moret, and M. Yan. A fast linear-time algorithm for inversion distance with an experimental comparison. *J. Comput. Biol.*, 8(5):483–491, 2001.

[2] D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 207–212. Kluwer Acad. Pubs., Dordrecht, Netherlands, 2000.

[3] A. Caprara. Sorting by reversals is difficult. In *Proc. 1st Int'l Conf. on Comput. Mol. Biol. RECOMB97*, pages 75–83. ACM Press, 1997.

[4] A. Caprara. Formulations and hardness of multiple sorting by reversals. In *Proc. 3rd Int'l Conf. on Comput. Mol. Biol. RECOMB99*, pages 84–93. ACM Press, 1999.

[5] A. Caprara. On the practical solution of the reversal median problem. In *Proc. 1st Workshop on Algs. in Bioinformatics WABI 2001*, volume 2149 of *Lecture Notes in Computer Science*, pages 238–251. Springer-Verlag, 2001.

[6] S. Downie and J. Palmer. Use of chloroplast DNA rearrangements in recon-

structing plant phylogeny. In P. Soltis, D. Soltis, and J. Doyle, editors, *Plant Molecular Systematics*, pages 14–35. Chapman and Hall, 1992.

[7] N. El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *Proc. 11th Ann. Symp. Combin. Pattern Matching CPM 00*, volume 1848 of *Lecture Notes in Computer Science*, pages 222–234. Springer-Verlag, 2000.

[8] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th Ann. Symp. Theory of Computing STOC 95*, pages 178–189. ACM Press, 1995.

[9] J. Lagergren, B.M.E. Moret, and D. Sankoff, organizers, *1st IMA/RECOMB Satellite Workshop on Comparative Genomics*, Minneapolis (2003), at `http://www.ima.umn.edu/complex/fall/c2.html`

[10] T. Liu, B.M.E. Moret, and D.A. Bader. An exact linear-time algorithm for computing genomic distances under inversions and deletions *U. New Mexico*, TR-CS-2003-31.

[11] M. Marron, K.M. Swenson, and B.M.E. Moret. Genomic distances under deletions and inversions. In T, Warnow and B. Zhu, editors, *Proc. 9th Annual Int'l Conf. Comput. and Combinatorics (COCOON'03)*, volume 2697 of *Lecture Notes in Computer Science*, pages 537–547. Springer-Verlag, 2003.

[12] B.M.E. Moret, A.C. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In R. Guigo and D. Gusfield, editors, *Proc. 2nd Int'l Workshop Algorithms in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pages 521–536. Springer-Verlag, 2002.

[13] R. Olmstead and J. Palmer. Chloroplast DNA systematics: a review of methods and data analysis. *Amer. J. Bot.*, 81:1205–1224, 1994.

[14] J. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, editor, *Cell Organelles*, pages 99–133. Springer Verlag, 1992.

[15] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Elec. Colloq. on Comput. Complexity*, 71, 1998.

[16] L. Raubeson and R. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.

[17] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

[18] D. Sankoff and J. Nadeau, eds. *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families.* Kluwer Acad. Pubs., 2000.

[19] A.C. Siepel and B.M.E. Moret. Finding an optimal inversion median: Experimental results. In O. Gascuel and B.M.E. Moret, editors, *Proc. 1st Int'l Workshop Algorithms in Bioinformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, pages 189–203. Springer-Verlag, 2001.