Genomic Distances under Deletions and Insertions

Mark Marron, Krister M. Swenson, and Bernard M. E. Moret

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 multi-set 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 and 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.

Keywords: inversion distance, reversal distance, genomic distance, Hannenhalli-Pevzner

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., [7, 11, 12, 14]). Appropriate tools for analyzing such data may help resolve some difficult phylogenetic reconstruction problems. Developing such tools is thus an important area of research—indeed, the recent DCAF symposium [6] was devoted 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 the median[1] of *three* genomes with identical gene content under such models is NP-hard [4, 13]. The problem of computing the edit distance

[1] 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.

between two genomes is 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 [9] 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 [8] 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).

In this paper, we extend El-Mabrouk's work by providing a polynomial-time approximation algorithm with bounded error to compute edit distances under unrestricted inversions, deletions, and insertions (including duplications). Our basic approach is based on a new canonical form for edit sequences along with the notion of a *cover* to deal with duplicates. 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 inversions, 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 outlines our alternate method for analyzing the general case of insertion, deletion, and inversion along with an analysis of our algorithm's error bounds. Finally, Section 6 shows some preliminary empirical results.

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 perfectly sorted sequence S to the given target sequence T.

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

When two strings s_i and s_j each contain a single character, $s_i = e_i$ and $s_j = e_j$, define their ordering $\zeta = e_i - e_j$. Two substrings of a subject sequence, s_i and s_j, are *correctly oriented* relative to each other if and only if:

1. s_i or s_j is empty.

2. s_i and s_j are both of unit length and, whenever s_i is adjacent to s_j with ordering ζ in the target sequence, then s_i is also adjacent to s_j with ordering ζ in the subject sequence.
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, whenever s_i is adjacent to s_j with parity ξ in the target sequence, then also s_i is adjacent to s_j with parity ξ in the subject sequence.

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 several useful results about shortest edit sequences, results that will enable us to obtain a "canonical form" into which any shortest edit sequence can always be transformed without losing optimality.

We make use throughout our derivation of *reindexing*; this reindexing provides a pliability to the indices that operations act upon which enables us to manipulate the order in which these operations appear. For example, take the string $1, 2, 3, -5, -4, 6, 7, 11, 12$ and suppose that the next operation to perform is an inversion starting at index 4 and going to index 7 (inclusive). The result is the new string $1, 2, 3, -7, -6, 4, 5, 11, 12$. Now, suppose that, in order to achieve some desired form, we need to insert the element 10 at index 4 before the application of this inversion. The goal is to maintain the indices of the inversion so that it continues to act on the substring $-5, -4, 6, 7$. After the application of the insertion associated with index 4, we are left with $1, 2, 3, 10, -5, -4, 6, 7, 11, 12$. In order to maintain the integrity of the inversion, we now adjust the start index of the inversion to be at 5 and the end index to be at 8, upon which application of the inversion correctly yields $1, 2, 3, 10, -7, -6, 4, 5, 11, 12$. 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 [9]) by showing that, whenever two substrings are correctly oriented, there is always a minimum edit sequence that does not split them. The idea is to rewrite the optimal edit sequence to keep the substrings together. First define $move(s_x, s_y, \xi)$ to move s_x to the immediate left of s_y with parity ξ. 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 between o_k and o_m to keep the substrings together. To accomplish this each o_x is expanded into a tuple of operations $\langle f_x, \hat{o_x}, t_x \rangle_x$. This tuple is constructed so that the x^{th} tuple is functionally equivalent to o_x and that t_x is the inverse of f_{x+1}. Further, the leading and trailing tuples are designed such that f_k and t_m are identity operations.

For example, suppose we have the sequence $14, 15, 11, 12, 13, 16$ and the operation sequence is $inv(2, 5), inv(1, 4), inv(4, 4), inv(4, 5)$. The first operation in this sequence splits the substring $14, 15$ and the fourth restores it. In this case we relocate 14 and 15 together by constructing the tuples as follows:

1. $inv(2,5) \rightarrow \langle move(14,15,1), inv(1,5), move(-14,-13,-1)\rangle$
2. $inv(1,4) \rightarrow \langle move(-14,16,-1), inv(1,3), move(-14,-15,1)\rangle$
3. $inv(4,4) \rightarrow \langle move(-14,16,-1), inv(\epsilon), move(-14,-15,1)\rangle$
4. $inv(5,5) \rightarrow \langle move(-14,16,-1), inv(4,5), move(14,15,1)\rangle$

The original operation sequence produces:

– $14, 15, 11, 12, 13, 16 \Rightarrow 14, -13, -12, -11, -15, 16 \Rightarrow 11, 12, 13, -14, -15, 16 \Rightarrow 11, 12, 13, 14, -15, 16 \Rightarrow 11, 12, 13, 14, 15, 16$

The new operation sequence produces:
$14, 15, 11, 12, 13, 16$

1. $\Rightarrow 14, 15, 11, 12, 13, 16 \Rightarrow -13, -12, -11, -15, -14, 16 \Rightarrow 14, -13, -12, -11, -15, 16$
2. $\Rightarrow -13, -12, -11, -15, -14, 16 \Rightarrow 11, 12, 13, -15, -14, 16 \Rightarrow 11, 12, 13, -14, -15, 16$
3. $\Rightarrow 11, 12, 13, -15, -14, 16 \Rightarrow 11, 12, 13, -15, -14, 16 \Rightarrow 11, 12, 13, -14, -15, 16$
4. $\Rightarrow 11, 12, 13, -15, -14, 16 \Rightarrow 11, 12, 13, 14, 15, 16 \Rightarrow 11, 12, 13, 14, 15, 16$

We have demonstrated 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), thereby providing the intuition behind a proof of the following theorem.

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.*

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 based on the idea of rewriting each operation preceding the first insert such that, at the beginning and end of each operation rewrite group(tuple), 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. 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 $\pi' \equiv \pi$ and $\pi' = \langle ins_1 \ldots ins_p, inv_1 \ldots inv_q, del_1 \ldots del_r\rangle$.*

Proof. Reminiscent of our previous tuple expansion, for each o_j s.t. $j \leq k-1$ we create $\hat{o_j} = (ins'_j, o'_j, del'_j)$, where

$$del'_j = \begin{cases} ins'^{-1}_{j+1} & 1 \leq j \leq k-2 \\ ins_1^{-1} & j = k-1 \end{cases},$$

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} cancel, 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, 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 in fact 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 [15], which attempts to identify, for each family of gene copies, the "original" gene (as distinct from its copies) and then discards all copies, thereby reducing a multi-set 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 target such that every element in the source appears in one of those substrings. To formalize and use this property, we need a few definitions. Call a substring $e_1 e_2 \ldots e_n$ *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. A maximal subsequence S_{nd} of the source string S is the *non-deleted* portion of S if S_{nd}, viewed as a set, is the largest subset of elements in S that is also contained in the target string T, also viewed as a set. (Note that S_{nd} is not a substring, but a subsequence; that is, it may consist of several disjoint pieces of S.) Given a set C of normalized strings which are maximal in C under the substring relation, define $\uplus C$ to be the string produced as follows; order the strings of

C lexicographically and concatenate them in that order, removing any overlap. We will say that a set C of contiguous substrings from T is a *cover* for S if S_{nd} is $\uplus C$. Note that a cover must contain only contiguous strings.

Suppose we have $S = 1, 2, 3, 4, 5, 6, 7$ and $T = 3, 4, 5, -4, -3, 5, 6, 7, 8$; then the set of normalized contiguous strings is $\{(3, 4, 5), (3, 4), (5, 6, 7, 8)\}$, S_{nd} is $(3, 4, 5, 6, 7, 8)$, a possible cover for S is $C_p = \{(3, 4, 5), (5, 6, 7, 8)\}$, and we have $\uplus C_p = (3, 4, 5, 6, 7, 8)$.

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 by two pieces, and 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 by two pieces. Since we have $(2(n - 1) + 1) + 2 = 2n + 1$, the bound is obeyed in all cases.

4.3 Building a Minimum Cover

Let $\mathcal{C}(T)$ be the set of all (normalized versions of) contiguous substrings of T that are maximal (none is a substring of any other). 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 $\mathcal{C}(T)$ that begin at or to the left of position i—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. (The proof follows standard lines: use contradiction, assume a first point of disagreement between an optimal cover and the greedy cover, and verify that we can exchange cover elements to move the disagreement farther down the index chain.) However, it should be noted that, in our sorting algorithm, the best choice of cover need not be a minimum cover—a minimum cover simply allows us to bound the error.

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.

To do this we will need to look at the target sequence T with all the elements that do not appear in S removed, call this new sequence T_{ir} to denote that all the inserted elements have been removed.

Theorem 4. *Let π be the minimal edit sequence from S to T, using l insertions and m inversions. Let π' be the minimal edit sequence of just inversions and deletions from S to T_{ir}. If the extension $\hat{\pi}$ is obtained by adding an insertion operation to π' for each of the inserted strings in T then $\hat{\pi}$ 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 (in all cases v is an inserted substring and x another):

1. Inserting another substring cannot split an inserted substring—it just creates a longer string of inserted elements: if x is inserted within v, then $uvw = uv_1v_2w$ becomes $uv_1xv_2w = uv'w$.
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 part of v is deleted, then $uvw = uv_1v_2v_3w$ becomes $uv_1v_3w = uv'w$.
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. For instance, given $uvw = u_1u_2v_1v_2w$, an inversion that acts on u_2v_1 yields the string $u_1\overline{v_1}\,\overline{u_2}v_2w = u_1v'\overline{u_2}v''w$.

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 and $l+m \leq n$ insertions in $\hat{\pi}$.

Now, the optimal edit sequence defines a corresponding optimal cover C_0 of S; if the cover we obtain, C, is in fact the optimal cover, then our algorithm produces the optimal edit sequence. In order to bound the error of our algorithm, then, we look at the differences between the unknown optimal cover C_0 and our constructed cover C.

The proof is constructive and rather laborious due to the multiple cases, so we restrict ourselves to an illustration of one of the cases. Let s_a, s_x, s_u, s_y, s_z be substrings of S; we use a prime accent ($'$) to denote that a particular substring was marked as a copy by a given cover. Let π_{tail} be the inversion and deletion portion of π and set $S = s_a \ldots s_x s_u s_y \ldots s_z$, $T = s_a \ldots s_x s_u s_y \ldots s_u \ldots s_z$, $T_{\text{opt}} = s_a \ldots s_x s'_u s_y \ldots s_u \ldots s_z$ (this T_{opt} is T renamed according to the optimal cover C_0), and $T_{\text{chosen}} = s_a \ldots s_x s_u s_y \ldots s'_u \ldots s_z$ (this T_{chosen} is T renamed according to the chosen covering), Further, suppose that s_u is at index I_1 in S and that 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 previously occupied. Thus, for each wrong choice in the cover, we need 3 inversions to move and 1 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 contains n operations, 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 $4(2n+1)+n = 9n+4$ 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 $10n+4$ operations. While this error bound is large, it is a constant and it is also unrealistically large—the assumptions used are not truly 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 should bring the bound to a more 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 sequences. Our edit sequences, of various length, 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% 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 target 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 (the figures show the average, minimum, and maximum values over the 10 instances).

The results are very 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; our estimated length shows this phenomenon very clearly.) Figures 1, 2, and 3 show our results for sequences of 200, 400, and 800 genes, respectively.

7 Conclusion and Future Directions

An exact polynomial-time algorithm for the computation of genomic distances under 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 preliminary results are extremely 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, 13]—although the algorithms of Caprara [5] and of Siepel and

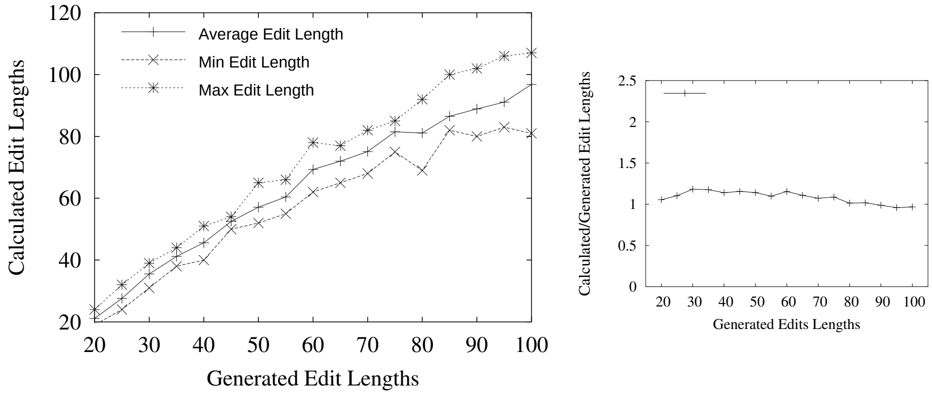

Fig. 1. Experimental results for 200 genes: (a) generated edit length vs. reconstructed length; (b) the ratio of the two.

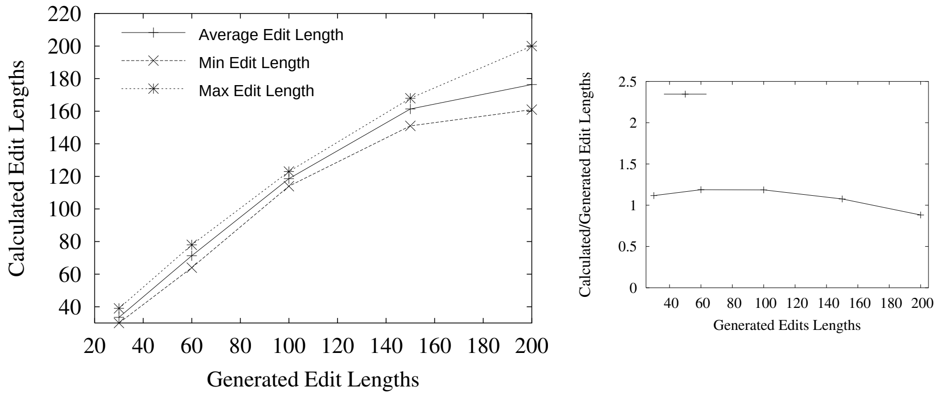

Fig. 2. Experimental results for 400 genes: (a) generated edit length vs. reconstructed length; (b) the ratio of the two.

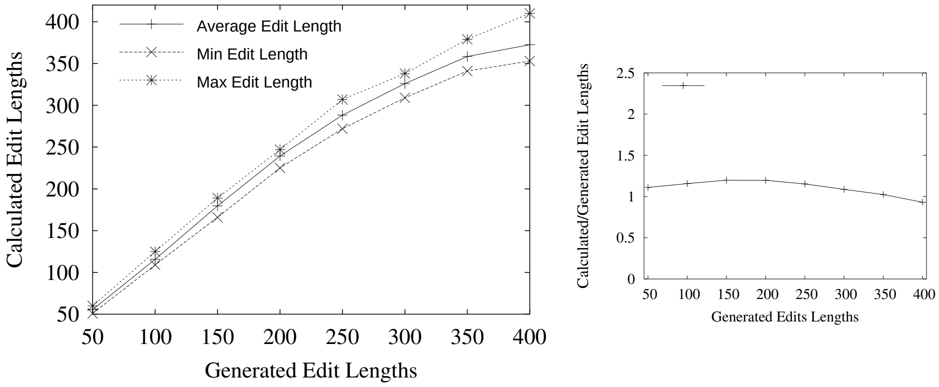

Fig. 3. Experimental results for 800 genes: (a) generated edit length vs. reconstructed length; (b) the ratio of the two.

Moret [16] have done well in practice (see, e.g., [10]). Good bounding is the key to such computations; our covering technique may be extendible to median computations.

8 Acknowledgments

This work is supported by the National Science Foundation under grants ACI 00-81404, DEB 01-20709, EIA 01-13095, EIA 01-21377, and EIA 02-03584.

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 Academic 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. M. Cosner, R. Jansen, B.M.E. Moret, L. Raubeson, L. Wang, T. Warnow, and S. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics*, pages 99–122. Kluwer Acad. Pubs., 2000.
7. S. Downie and J. Palmer. Use of chloroplast DNA rearrangements in reconstructing plant phylogeny. In P. Soltis, D. Soltis, and J. Doyle, editors, *Plant Molecular Systematics*, pages 14–35. Chapman and Hall, 1992.

8. 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.
9. 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.
10. 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.
11. R. Olmstead and J. Palmer. Chloroplast DNA systematics: a review of methods and data analysis. *Amer. J. Bot.*, 81:1205–1224, 1994.
12. J. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, editor, *Cell Organelles*, pages 99–133. Springer Verlag, 1992.
13. I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Elec. Colloq. on Comput. Complexity*, 71, 1998.
14. L. Raubeson and R. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.
15. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.
16. 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.