Of mice and men: Algorithms for evolutionary distances between genomes with translocation*

John D. Kececioglu† R. Ravi‡

Abstract

A new and largely unexplored area of computational biology is combinatorial algorithms for genome rearrangement. In the course of its evolution, the genome of an organism mutates by processes that can rearrange whole segments of a chromosome in a single event. These rearrangement mechanisms include inversion, transposition, duplication, and translocation, and a basic problem is to determine the minimum number of such events that transform one genome to another. This number is called the rearrangement distance between the two genomes, and gives a lower bound on the number of events that must have occurred since their divergence, assuming evolution proceeds according to the processes of the study.

In this paper, we begin the algorithmic study of genome rearrangement by *translocation*. A translocation exchanges material at the end of two chromosomes within a genome. We model this as a process that exchanges prefixes and suffixes of strings, where each string represents a sequence of distinct markers along a chromosome in the genome. For the general problem of determining the translocation distance between two such sets of strings, we present a 2-approximation algorithm. For a theoretical model in which the exchanged substrings are of equal length, we derive an optimal algorithm for translocation distance.

We also examine for the first time two types of rearrangements in concert. An *inversion* reverses the order of markers within a substring, and flips the orientation of the markers. For genomes that have evolved by translocation and inversion, we show there is a simple 2-approximation algorithm for data in which the orientation of markers is unknown, and a $\frac{3}{2}$-approximation algorithm when orientation is known.

These results take a step towards extending the area from the analysis of simple organisms, whose genomes consist of a single chromosome, and whose evolution has largely involved a single type of rearrangement event, to the analysis of organisms such as man and mouse, whose genomes contain many chromosomes, and whose history since divergence has largely consisted of inversion and translocation events.

1 Introduction and motivation

Molecular biology is entering a new era in which data will soon become available on the entire complement of DNA, known as the *genome*, of man and model organisms such the mouse [4]. Genomes of these and other complex organisms are organized into *chromosomes*, which contain the genes of the organism arranged in a linear order. Much of the current data for genomes is in the form of *maps*, which identify the location of genes and other markers of interest along the chromosomes. Maps are now becoming available that show the location of markers common to both man and mouse [10], and such data permits for the first time the study of the evolution of such organisms at the scale of the genome [9].

In contrast to evolution at the level of individual genes, which proceeds by local operations, such as *point mutation*, which substitutes, inserts, and deletes individual letters of the DNA sequence, evolution at the level of the genome proceeds by non-local, large-

*This paper appeared in Proceedings of the 6th ACM-SIAM *Symposium on Discrete Algorithms*, 604–613, 1995. Work carried out while the authors were at the Department of Computer Science of the University of California, Davis.

†Department of Computer Science, The University of Georgia, Athens, GA 30602. Electronic mail: kece@cs.uga.edu.
This research was supported by a DOE Human Genome Distinguished Postdoctoral Fellowship.

‡Department of Computer Science, Princeton University, Princeton, NJ 08544. Electronic mail: ravi@cs.princeton.edu.
Research supported by DOE Grant DE-FG03-90ER60999 and a DIMACS Postdoctoral Fellowship.

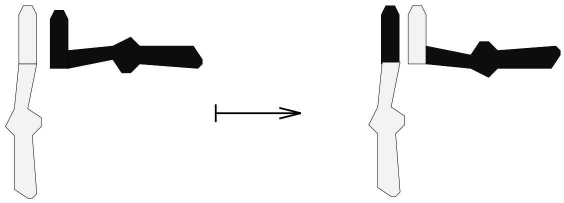

Figure 1 An example of a translocation between two chromosomes. A segment from the end of each chromosome is exchanged.

scale operations, which can rearrange a whole segment of the chromosome in one evolutionary event. These non-local operations include: *inversion*, which replaces a segment of a chromosome with its reverse DNA sequence, and has the effect of reversing the order and orientation of markers within the segment; *transposition*, which moves a segment to a new location in the genome; *duplication*, which copies a segment to a new location; and *translocation*, which exchanges segments between the ends of two chromosomes (see Figure 1). Such large-scale rearrangements occur at a much slower rate than point mutations, and are now a subject of intensive study, as they may permit the reconstruction of evolutionary history for long-diverged organisms whose divergence-times cannot be recovered from study of point mutations alone [11, 12].

In this paper, we take a first step towards the algorithmic study of genome rearrangement by translocation. Generally, *genome rearrangement* is concerned with determining the minimum number of rearrangement events that transform one genome into another. This number is often called the *rearrangement distance* between the genomes. We present approximation algorithms for rearrangement distances. An α-*approximation algorithm* for a minimization problem is a an algorithm that runs in polynomial time and delivers a solution whose value is at most α times the minimum.

Prior algorithmic work on genome rearrangement has largely focused on analysis of a single chromosome, in terms of inversions, and more recently in terms of transpositions. Markers correspond to unique segments of DNA, which have a left-to-right orientation. With inversion, which flips the orientation of markers as well as reversing their order, there are two forms of the inversion distance problem, according to whether or not the orientation of markers is known. Kececioglu and Sankoff [7] undertook the first formal study of algorithms for inversion distance; they developed a 2-approximation algorithm and an exact algorithm using branch-and-bound and linear programming for data in which the orientation of markers is unknown; this was simplified in [8] for the case of oriented markers. Bafna and Pevzner [1] improved the approximation guarantees significantly, to $\frac{7}{4}$ for unoriented markers, and $\frac{3}{2}$ for oriented markers; they also determined the exact inversion-distance diameter. Most recently, Bafna and Pevzner discovered a $\frac{3}{2}$-approximation algorithm for transposition distance [2], and application of these results to biological data is now underway [3, 5].

In the next section, we formulate our problems, and summarize the results. In Section 3, we consider the case of translocations alone, and in Section 4, translocations and inversions together. We conclude with some directions for further research.

2 Problem formulations and results

We consider the basic problem of comparing two genomes. Formally a genome is a collection of chromosomes, each of which we represent as a string. Each character in the string corresponds to a genetic marker, such as an identified gene. We assume that all markers in a genome are distinct; in other words, no character is repeated in the strings. We also assume that the alphabet of the two genomes is identical, as the rearrangements that we consider do not create or destroy markers.

A translocation exchanges prefixes or suffixes of two strings in our model. Suppose that a translocation operates on the two strings vw and xy, where v, w, x, and y are non-empty substrings. A *prefix-prefix translocation* that exchanges v and x has the following effect:

$$vw,\, xy \;\mapsto\; xw,\, vy.$$

A *prefix-suffix translocation* that exchanges v and y has the effect

$$vw,\, xy \;\mapsto\; y^R w,\, xv^R,$$

where s^R denotes the reverse of string s.[1] We allow a prefix-suffix translocation to operate within one string, as long as the prefix and the suffix are non-overlapping.[2]

All chromosomes contain a *centromere*, which is important for the proper function of the genome during cell division. Depending on the position of the

[1] Note that a suffix-suffix translocation is equivalent to a prefix-prefix translocation. Similarly, a suffix-prefix translocation is equivalent to a prefix-suffix translocation.

[2] A translocation of this type is known as a *pericentric inversion* in the literature.

centromere along the length of the chromosome, chromosomes may be classified into two types. In *acrocentric chromosomes*, the centromere occurs at one end of the chromosome. In *metacentric chromosomes*, the centromere occurs towards the middle. Within a genome, all chromosomes are either acrocentric or metacentric.

In acrocentric chromosomes, the location of the centromere distinguishes the two ends of a chromosome, and establishes a reading direction. When forming the string representing such a chromosome, we assume the markers are read from the end containing the centromere. After a viable translocation, both chromosomes must be left with a centromere. Thus, for organisms with acrocentric chromosomes, we can restrict ourselves to prefix-prefix translocations. We define the *directed model* to be one in which all translocations are prefix-prefix.

When comparing genomes with metacentric chromosomes, we are concerned only with the adjacency of markers. In other words, the direction that a string corresponding to a chromosome is read is irrelevant. In this case, we allow both prefix-prefix and prefix-suffix translocations, since a prefix-prefix translocation is equivalent to a prefix-suffix translocation when the second string is read in the reverse direction. We define the *undirected model* to be one in which translocations may be both prefix-prefix and prefix-suffix.

A special case that we study is *equal-length translocation*, in which the substrings exchanged are of equal length. In this case, we further assume that all strings in both genomes have the same length. The majority of the paper is devoted to the general case, with no implicit assumptions on the lengths of the substrings exchanged, and the lengths of the strings in the genome.

Physically, our markers correspond to short, fixed stretches of the DNA sequence. When a rearrangement reverses the order of markers, it changes the direction of the sequence for each marker. When forming the string for a chromosome, we can associate a *sign* with the character representing a marker as follows. If when reading the chromosome, the fixed sequence for a marker appears, we associate a positive sign with the character corresponding to the marker; if the reverse sequence appears, we associate a negative sign. We treat signed data only in the case of chromosomes with no absolute reading direction. In this case, the string $+a+b+c$ is identical to $-c-b-a$.

We now summarize our results for these models. *Sorting by translocations* refers to transforming one genome into another using the fewest number of translocations. We denote the total number of markers in a genome by m.

Theorem 2.1 *In both the directed and undirected models, there is an exact algorithm for sorting by equal-length translocations that runs in $O(m)$ time.*

The case of equal-length translocations may arise when the markers are obtained by sampling the same position across several chromosomes of uniform size. If no marker is present at a sampled position, we can denote this by recording a *blank*. Theorem 2.1 extends to such data.

Theorem 2.2 *For data containing blanks, in both the directed and undirected models, there is an exact algorithm for sorting by equal-length translocations that runs in $O(m)$ time.*

For the general case of arbitrary-length translocations we have the following results.

Theorem 2.3 *In both the directed and undirected models, there is a 2-approximation algorithm for sorting by translocations that runs in $O(m^2)$ time.*

Our remaining results allow both translocations and inversions, and treat both signed and unsigned data. With signed data, a translocation that exchanges a prefix with a suffix flips the signs of markers in the exchanged segments. Similarly, an inversion on signed data flips signs as well as reverses the order of markers.

Theorem 2.4 *For unsigned data, in both the directed and undirected models, there is a 2-approximation algorithm for sorting by translocations and inversions that runs in $O(m^2)$ time.*

We use the 2-approximation algorithm of Kececioglu and Sankoff [7] in this result. We note that for sorting unsigned data by inversions alone, Bafna and Pevzner have shown that a performance guarantee of $\frac{7}{4}$ can be achieved [1].

Theorem 2.5 *For signed data, in the undirected model, there is a $\frac{3}{2}$-approximation algorithm for sorting by translocations and inversions that runs in $O(m^2)$ time.*

This makes use of the algorithm and analysis of Bafna and Pevzner [1] for sorting signed data by inversions alone, and has the same performance guarantee.

3 Translocations alone

For rearrangement distance with translocations alone, we first examine the equal-length case.

3.1 Equal-length translocations

When all translocations in a series exchange segments of equal length, and the strings in both genomes are all of equal length, the problem of determining the translocation distance between two such sets of strings can be reduced a problem of sorting a matrix.

For a genome containing n strings, each of length k, we form a $k \times n$ matrix of integers from the strings as follows. Each column corresponds to a string, and each row corresponds to a position in a string. Let us first consider the directed model in which translocations only exchange prefixes with prefixes. With equal-length translocation, a character at position i in a string remains at position i after moving to another string. Thus we can assign the characters in each row i integer names in the range 1 to n. Given a genome γ_1 to transform into a genome γ_2, we assign numbers to characters as follows. Arbitrarily number the strings in γ_2 from 1 to n, and assign the characters in γ_2 the number of their string. This converts γ_1 into strings of numbers as well. If we then arrange the strings of γ_1 in increasing order of the number at their kth position, the numbers at position i in the strings of γ_1 form a permutation λ_i of 1 to n.

Treating the permutations λ_1 through λ_k as the rows of a $k \times n$ matrix, our problem is equivalent to sorting the rows into $1\,2\,\cdots\,n$ by repeatedly selecting two columns, and a length l, and exchanging the elements in the first l rows of the two columns. The minimum number of such column-exchanges to sort the matrix gives the translocation distance between γ_1 and γ_2.

Denote a column exchange by a triple (i, j, l), where i and j specify the columns and l specifies the length of the prefix of the columns that is exchanged. We say a series of exchanges is *bottom-up* if the third coordinate of the triples is nonincreasing, in other words, if the series progresses from the bottom to the top of the matrix.

Lemma 3.1 *Every matrix has an optimal sorting series that is bottom-up.*

Proof A series that sorts a matrix and is not bottom-up contains a pair of consecutive exchanges

$$(i_t, j_t, l_t),\ (i_{t+1}, j_{t+1}, l_{t+1}),$$

with $l_t < l_{t+1}$. Without loss of generality, suppose $i_t \neq i_{t+1}$ and $j_t = j_{t+1}$. In this case, performing exchange $t+1$ first, followed by a slight modification of exchange t:

$$(i_{t+1}, j_{t+1}, l_{t+1}),\ (i_t, i_{t+1}, l_t),$$

produces the same effect. Repeatedly applying this transformation to a shortest series yields one that is both optimal and bottom-up. □

We say a bottom-up series is *locally optimal* if, when sorting each row, it uses the minimum number of translocations to sort the row from its current state.

Lemma 3.2 *A bottom-up series that is locally optimal is globally optimal.*

Proof Consider a bottom-up series just before it sorts row $i+1$. Let π be the current state of row $i+1$, and λ_i be the state of row i before any rows have been sorted. After the series sorts row $i+1$, the state of row i is

$$\lambda_i \circ \pi^{-1},$$

which is independent of the translocations used to sort row $i+1$. Applying this inductively, the state of a row just before it is sorted is independent of the translocations that were used to sort the rows below it. In other words, the particular translocations that are used to sort higher-numbered rows do not affect the number of the translocations needed for lower-numbered rows. Hence using the fewest translocations to sort a row is optimal. □

Lemmas 3.1 and 3.2 reduce the problem of sorting a matrix by translocations to the problem of sorting a row by translocations. This is precisely the problem of sorting a permutation by the mininum number of transpositions, where a *transposition* on a permutation exchanges the values at two arbitrary positions.[3]

This classical problem was solved by Cayley in 1849 (see [6]). The solution is to associate a graph $G(\pi)$ with the given permutation π. For each element i of π there is a vertex v_i, and an edge is directed from v_i to v_j if $\pi(i) = j$. As every vertex of $G(\pi)$ has exactly one in-edge and one out-edge, $G(\pi)$ consists of vertex-disjoint cycles. Denote the *number of cycles* of $G(\pi)$ by $\Psi(\pi)$.

Lemma 3.3 (Cayley) *The minimum number of transpositions to sort a permutation π on n elements is*

$$n - \Psi(\pi).$$

Combining Lemmas 3.1, 3.2, and 3.3 yields the algorithm of Figure 2. The algorithm proceeds bottom-up. It accumulates the effect of sorting prior rows in permutation π, and accumulates the number of

[3] Note that this differs from the use of the term in molecular biology.

algorithm EqualLengthTranslocationDistance(γ) **begin**
Denote the rows of the $k \times n$ matrix γ by $\lambda_1, \ldots, \lambda_k$.
$\pi := 1\,2 \cdots n$
$d := 0$
for $i := k$ **downto** 1 **do begin**
$\pi := \lambda_i \circ \pi^{-1}$
$d := d + n - \Psi(\pi)$
end
return d
end

Figure 2 Exact algorithm for sorting by equal-length translocations.

translocations to sort the rows in d. The work for a given row, which involves computing an inverse, composing two permutations, and computing the number of cycles, can be performed in $O(n)$ time, which gives an $O(kn) = O(m)$ time algorithm, which is optimal. This proves the first part of Theorem 2.1.

Extension to the undirected model

To handle the undirected model, which adds prefix-suffix translocations, we reduce it to the directed model. With equal-length translocations, every translocation can be considered to exchange substrings of length at most $\lfloor \frac{k}{2} \rfloor$, where k is the length of the strings in the genome. Hence we can double-up the ends of each string, and assign each of the k positions of a string a depth between 1 and $\lfloor \frac{k}{2} \rfloor$, by their distance from the nearest end.[4] It is then straightforward to show there is an optimal series that progresses from greater to smaller depths, and is locally optimal at each depth, in analogy to Lemmas 3.1 and 3.2.

The only question is how to get started at depth $\lfloor \frac{k}{2} \rfloor$. Once the characters at this depth are correctly paired, we can separate each string into halves at depth $\lfloor \frac{k}{2} \rfloor$, and treat the halves independently in a directed problem over a $\lfloor k/2 \rfloor \times 2n$ matrix. The problem of pairing the characters at depth $\lfloor \frac{k}{2} \rfloor$ by the fewest translocations has a solution similar to Lemma 3.3. This leads to the second part of Theorem 2.1.

Extension to data with blanks

We can extend our results to genomes that contain a blank character as follows. When a row contains blanks, we define *impartial sorting* to mean getting the nonblank elements into position, while treating all columns with blanks alike. In other words, when sorting the nonblank elements, we do not care which blank elements are displaced. As before, we say a bottom-up series is *locally optimal* if, when impartially sorting a given row, it uses the minimum number of translocations for that row.

Lemma 3.4 *Every matrix with blanks has a bottom-up series that is optimal. Furthermore, every impartial bottom-up series that is locally optimal is globally optimal.*

The problem of finding an impartial series that is locally optimal can be solved by defining the graph of Lemma 3.3 so that the number of cycles is maximum. Using this graph to identify an impartial series results in Theorem 2.2.

3.2 Arbitrary-length translocations

In this section, we consider sorting by arbitrary-length translocations. Our undirected model implicitly allows inversions, since an inversion can be realized as a prefix-suffix translocation on the same string. [5] Theorem 2.4 implies the result in this case. The reminder of this section addresses the directed model.

In the directed model, all strings in a genome have an implicit ordering of their markers. Define a marker that is at the left end of a string to be a *top marker*, and one that is at the right end of a string to be a *bottom marker*. Let γ_1 denote a genome that is to be sorted by translocations into genome γ_2. Since translocations exchange non-empty prefixes, the set of top markers in γ_2 is identical to the set of top markers in γ_1, and similarly for the bottom markers. We denote the top and bottom marker of a string A in γ_2 by $\top_A$ and $\perp_A$ respectively.

A lower bound construction

We now describe the construction of an auxiliary digraph that will be used to derive a lower bound on translocation distance. We define a directed graph $G(\gamma_1)$ whose node set is the set of all markers. The edges in $G(\gamma_1)$ are of two types: *solid* and *dashed* edges. Solid edges represent adjacencies in the genome γ_2, while dashed edges represent adjacencies

[4] When k is odd, the positions at depth $\lceil \frac{k}{2} \rceil$ can be ignored.

[5] Actually, an inversion performed on a prefix or suffix of a string cannot be realized by a translocation. Notice, however, that in the present context, arbitrary inversions are not allowed. This implies that the characters at the ends of the strings in the sorted genome are at the ends of strings in the unsorted genome, so such inversions will never be required by our algorithm.

in γ_1. In particular, for every string $a_1a_2\ldots a_p$ in γ_2, the graph $G(\gamma_1)$ contains $p-1$ solid edges (a_i, a_{i+1}) for $1 \le i < p$. For every string $b_1b_2,\ldots b_q$ in γ_1, the graph $G(\gamma_1)$ contains $q-1$ dashed edges (b_i, b_{i-1}) for $1 < i \le q$.

Define an *alternating cycle* to be a cycle in which the edges are alternately solid and dashed. The definition of the edges of $G(\gamma_1)$ implies that every edge of a certain type that is directed into a node must be uniquely followed by the single edge of the other type directed out of this node in any alternating cycle. This pairing at every node defines a unique partition of the edges of $G(\gamma_1)$ into disjoint alternating cycles. Let $\Psi(\gamma_1)$ denote the number of alternating cycles in the partition of $G(\gamma_1)$. Let m denote the number of markers, and n denote the number of strings.

Lemma 3.5 *Any series of translocations that sorts a genome γ_1 has length at least $m - n - \Psi(\gamma_1)$.*

Proof A translocation performed on γ_1 affects only two dashed edges in $G(\gamma_1)$, and exchanges the tails of these two edges. Depending on whether these two dashed edges are in different cycles or in the same cycle in $G(\gamma_1)$, this translocation either decreases or increases the cycle count by one. Notice that $\Psi(\gamma_2) = m-n$. Thus any series that sorts γ_1 must use at least $m - n - \Psi(\gamma_1)$ translocations. □

An approximation algorithm

The greedy approximation algorithm motivated by this lower bound attempts to increase the cycle count by one at every step. Let γ denote the current genome. Define a *good* translocation to be one that increases the number of cycles in the decomposition of $G(\gamma)$. The algorithm attempts to perform a good translocation at every step, preferring those that leave behind good translocations. There is a particularly simple implementation of a good translocation whenever two markers a and b that are adjacent in γ_2 with b following a, are in two distinct strings in γ. In this case, the translocation that brings b immediately below a increases the alternating cycle count by one.

Define a pair of markers a, b to be *out of order* in γ if b immediately follows a in γ_2 while a follows b (not necessarily immediately) in γ. If no good translocations are available in γ and $\gamma \neq \gamma_2$, then γ contains a pair of markers that are out of order. In this case, our algorithm first performs a preparatory translocation that separates these two markers onto different strings. Then it continues to find and perform good translocations. We view the algorithm as working in *rounds* where each round begins with a preparatory translocation, followed by a series of good translocations. The algorithm is presented in detail in Figure 3.

```
algorithm TranslocationSort(γ) begin
    i := 0;
    while γ is not sorted do begin
        i := i + 1
        if γ has a good translocation then
            Choose a good translocation τ_i, favoring
            translocations that leave behind good
            translocations.
        else begin
            (Start a new round.)
            Find a pair of markers a and b that are
            out of order in γ.
            Choose a translocation τ_i that separates
            a and b onto different strings.
        end
        γ := γ ∘ τ_i
    end
    return i, τ_1 τ_2 ··· τ_i
end
```

Figure 3 Greedy algorithm for sorting by translocations in the directed model.

Lemma 3.6 *Every round of the greedy algorithm contains at least three good translocations.*

Proof Consider a round that begins with a preparatory translocation involving a pair of markers a and b that are out of order. The key observation is that any good translocation following the preparatory translocation leaves a genome that permits two subsequent good translocations. This proves the lemma.

We identify good translocations by finding pairs of markers that are adjacent in γ_2 but in distinct strings in γ. Figure 4 (a) illustrates a genome with two markers a and b that are out of order in the current genome; Figure 4 (b) depicts the result of a translocation separating a and b. The good translocation that makes a and b adjacent in γ results in Figure 4 (c). Tracing along the path of solid edges in $G(\gamma)$ from $\top_A$ to $\perp_A$, we identify a pair of markers p and q connected by a solid edge but in separate strings. Marker q is depicted above a in Figure 4 (c). The good translocation that makes p and q adjacent in γ leaves $\top_A$ and $\perp_A$ in different strings (Figure 4 (d)). Again, tracing the path of solid edges from $\top_A$ to $\perp_A$, we identify yet another pair of separated markers r and s, leading to a third good translocation. □

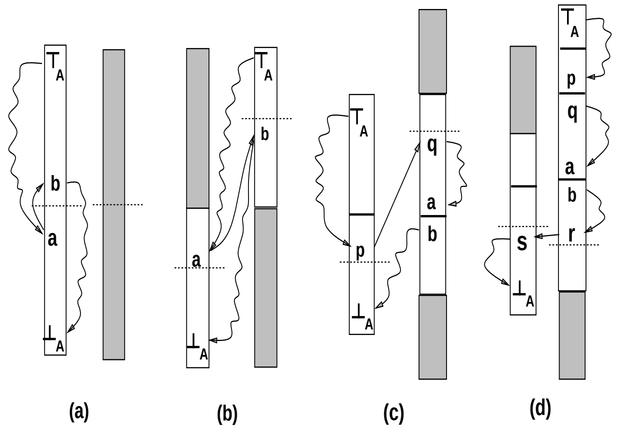

Figure 4 The first four translocations in a typical round of the greedy algorithm. There is always a path of solid edges from $\top_A$ to $\perp_A$ containing all the markers in A; portions of this path are depicted by wavy edges. The positions at which translocations cut are indicated by dashed lines.

Lemma 3.7 *The greedy algorithm sorts any genome γ in at most $2(m-n-\Psi(\gamma))$ translocations.*

Proof The first translocation in any round of the greedy algorithm decreases $\Psi(\gamma)$ by one. Combining this observation with Lemma 3.6, during every round of the approximation algorithm, $\Psi(\gamma)$ increases at the rate of at least two per four translocations. Furthermore, before the first round, every translocation the algorithm performs increases the cycle count by one. Since the unsorted genome γ has $\Psi(\gamma)$ cycles and the sorted genome has $m-n$ cycles, the number of translocations performed by the algorithm is at most $2(m-n-\Psi(\gamma))$. □

Combining the above lemma with Lemma 3.5 proves the performance ratio. It is straightforward to show an $O(m^2)$ running time, which completes the proof of Theorem 2.3.

4 Translocations together with inversions

We now consider the problem of transforming one genome into another by a shortest series of translocations and inversions. We discuss our result for unsigned data very briefly, and devote most of the section to signed data in the undirected model.

Given an unsigned genome γ_1 to transform into γ_2, define a *breakpoint* in γ_1 to be a pair of characters a and b that are consecutive in γ_1, but not in γ_2. Notice that any unsorted genome γ_1 contains a breakpoint, while the sorted genome γ_2 contains none. Whenever γ_1 contains a pair of characters that are on the same string in γ_2, but are separated onto different strings in γ_1, there is a translocation that removes a breakpoint. Thus, until characters are properly segregated into strings, we can remove at least one breakpoint per translocation. Once characters are properly segregated, we can use the inversion algorithm of Kececioglu and Sankoff [7] to finish the sorting; this algorithm removes at least one breakpoint per inversion. Since any translocation or inversion can remove at most two breakpoints, a performance guarantee of 2 follows. This leads to Theorem 2.4.

We now turn to signed data in the undirected model. As before, we associate a graph G with genomes γ_1 and γ_2. Vertices in G correspond to characters, and edges record the adjacency of characters in γ_1 and γ_2. With signed data and inversions, the construction becomes a little more complicated; it uses the idea of a bidirected edge from Kececioglu and Sankoff [7], and an alternating path from Bafna and Pevzner [1].

A lower bound construction

For the construction we choose an arbitrary reading direction for each string, both in the unsorted genome γ_1 and the sorted genome γ_2. This establishes a successor for each character in the initial genome γ_1 and the final genome γ_2.

We establish a sign for each character as follows. All characters in γ_2 are considered to be positive. A character a in a string of γ_1 is positive if, when reading the corresponding chromosome of γ_1 in the chosen direction, the sequence for the marker corresponding to a appears in the same direction as in γ_2. Otherwise, character a is negative in γ_1. Notice that reading a string $+a\,+b\,+c$ of γ_1 in the reverse direction gives string $-c\,-b\,-a$. Graph G will be invariant under this transformation.

To simplify the remainder of the presentation, we will consider the final sorted genome γ_2 to be fixed and implicit, and define graph G in terms of what we call the *current genome* γ. At the start, γ is the initial unsorted genome γ_1, but γ changes as translocations and inversions are performed.

With these conventions, graph $G(\gamma)$ is formally defined as follows. For every character there is a vertex in G, and for every string in γ there are four *dummy vertices*. For a character a at the left end of a string in γ, one dummy vertex represents the null predecessor of a; for a character b at the right end of a string in γ, one dummy vertex represents the null successor of b. Similarly, two more sets of dummy vertices represent null predecessors and successors in the final sorted configuration.

Edges of G are bidirected,[6] and of two colors: solid and dashed. *Solid edges* represent the adjacency of characters in the final sorted genome, while *dashed edges* represent adjacencies in the current unsorted genome. We assign directions to these edges as follows. A dashed edge between a character a and its successor in the current unsorted genome, a'_{current}, is always directed from the successor to the predecessor:

$$\pm a \quad \leftarrow\leftarrow \quad \pm a'_{\text{current}}$$

A solid edge between a character a and its successor in the final sorted genome, a'_{final}, has a direction that depends on the signs of the two characters in the current genome, and is given as follows:

$$\begin{array}{lcl} +a & \rightarrow\rightarrow & +a'_{\text{final}} \\ +a & \rightarrow\leftarrow & -a'_{\text{final}} \\ -a & \leftarrow\rightarrow & +a'_{\text{final}} \\ -a & \leftarrow\leftarrow & -a'_{\text{final}} \end{array}$$

The key property of this construction is that the edge set of G can be uniquely decomposed into disjoint alternating paths and cycles. By a *path* in a bidirected graph we mean a sequence of distinct edges, where a vertex that is encountered via an in-edge in the sequence is left via an out-edge, and vice versa. A vertex may be visited repeatedly in such a path. A *cycle* is a path where the first and last vertices coincide. A path or cycle is *alternating* if the edges alternate in color between solid and dashed.

The decomposition is specified by pairing each solid edge incident to non-dummy vertex v with a dashed edge incident to v. Notice that every non-dummy vertex has exactly two incident solid edges, one an in-edge and one an out-edge. Similarly, every non-dummy vertex has exactly two incident dashed edges, again one an in-edge and one an out-edge. This implies there is a unique pairing that correctly mates edge directions, hence a unique decomposition into maximal alternating paths and cycles.

Using this decomposition we can derive the following lower bound. For a genome γ, we denote the *number of cycles* in the decomposition of $G(\gamma)$ by $\Psi(\gamma)$. The number of paths in the decomposition is always $2n$.

[6] A *bidirected* edge has arrowheads at both ends that can be independently directed into or out of its endpoints. A normal directed edge is an out-edge for one endpoint, and an in-edge for the other endpoint. A bidirected edge can in addition be an in-edge for both endpoints, or an out-edge for both endpoints.

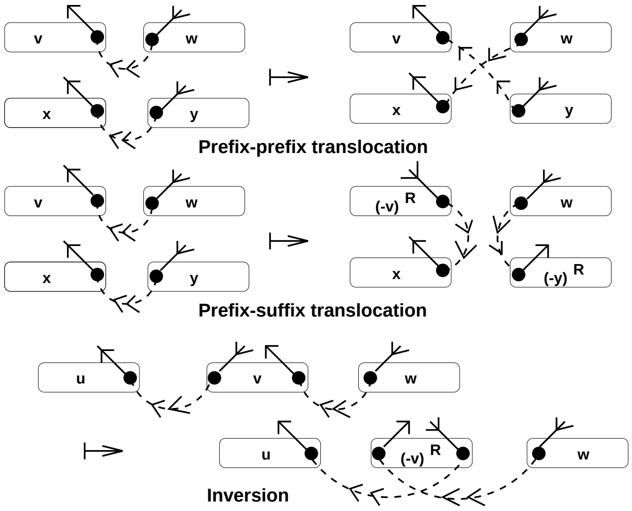

Figure 5 The effect of a translocation or inversion on $G(\gamma)$. The structure of the paths that connect the four dangling solid edges does not change.

Lemma 4.1 *The minimum number of translocations and inversions to sort a genome γ of n chromosomes and m markers, with signed data in the undirected model, is at least*

$$(m - n) - \Psi(\gamma).$$

Proof sketch Consider the effect on $G(\gamma)$ of a translocation or inversion in γ (see Figure 5). The construction of G is carefully designed to ensure that the edge decomposition changes only where the chromosome is cut, by simply reconnecting the two dashed edges corresponding to the cut sites. With a case analysis, one can then show that any translocation or inversion changes the number of cycles by $\Delta\Psi \in \{-1, 0, +1\}$.

When the current genome coincides with the final genome, the number of cycles in the decomposition is $m - n$. As any series that sorts γ must raise the number of cycles from $\Psi(\gamma)$ to this number, any such a series must use as least $m - n - \Psi(\gamma)$ translocations and inversions. □

An approximation algorithm

We obtain an approximation algorithm that comes within $\frac{3}{2}$ of the lower bound using the idea of a *breakpoint*. A string in the current genome γ has

```
algorithm SignedTranslocationInversionSort(γ) begin
    i := 0
    while γ contains a breakpoint do begin
        i := i + 1
(1)     if γ has a 2-translocation τ then
            μ_i := τ
(2)     else if γ has a 2-inversion then
            Choose a 2-inversion ρ, favoring inversions
                that leave behind 2- or 1-inversions.
            μ_i := ρ
(3)     else if γ has a 1-translocation τ then
            μ_i := τ
(4)     else if γ has a 1-inversion then
            Choose a 1-inversion ρ, favoring inversions
                that leave behind 2- or 1-inversions.
            μ_i := ρ
(5)     else begin
            Choose a 0-inversion ρ for which ΔΨ ≥ 0,
                favoring inversions that leave behind
                2-inversions.
            μ_i := ρ
        end
        γ := γ ∘ μ_i
    end
    return i, μ_1 μ_2 ··· μ_i
end
```

Figure 6 Greedy algorithm for sorting signed data by translocations and inversions in the undirected model.

a breakpoint between two consecutive characters if they are not of the form $+a\,+a'_{\text{final}}$, or $-a'_{\text{final}}\,-a$. We denote the *number of breakpoints* in a genome γ by $\Phi(\gamma)$.

Notice that any translocation or inversion changes the number of breakpoints by $\Delta\Phi \in \{-2,-1,0,+1,+2\}$, since the operations cut at exactly two positions. We say a translocation τ is a *k-translocation* if $\Delta\Phi = -k$ on performing τ. Similarly, an inversion ρ is a *k-inversion* if $\Delta\Phi = -k$ on performing ρ. Thus 2-translocations and 2-inversions remove two breakpoints.

A natural strategy for sorting γ is to repeatedly pick a translocation or inversion that removes the most breakpoints, since for every γ that is not sorted, $\Phi(\gamma) > 0$, while the sorted configuration has zero breakpoints.

This results in the greedy algorithm of Figure 6. Cases (2) and (4) are from the inversion algorithm of Kececioglu and Sankoff [8], while case (5) is the refinement of Bafna and Pevzner [1]. Our proof of the performance guarantee is based on their analysis.

In the following, $\Psi_k(\gamma)$ denotes the number of cycles of length k in $G(\gamma)$.

Lemma 4.2 *The greedy algorithm sorts any genome γ with signed data in the undirected model in at most*

$$\Phi(\gamma) \;-\; \tfrac{1}{2}\Psi_4(\gamma)$$

translocations and inversions.

Proof sketch Notice that a configuration has a 1- or 2-translocation iff there is a pair of characters that should be on the same string in the sorted configuration, but are separated onto different strings in the current genome. Hence all translocations are performed before any 0-inverson. This gives a natural division of the algorithm into two phases: the first phase consists of all rearrangments preceding the first 0-inversion; the second phase contains all remaining rearrangements. At the conclusion of the first phase, all characters are correctly segregated onto strings; the second phase consists of sorting each string by inversions.

Bafna and Pevzner have shown that cases (2), (4), and (5) sort by inversions within the bound of the lemma [1]. Thus it suffices to show that for every rearrangement operation performed by the algorithm in the first phase,

$$\Delta\Phi \;-\; \tfrac{1}{2}\Delta\Psi_4 \;\geq\; 1,$$

where $\Delta\Phi$ measures the decrease in the number of breakpoints, and $\Delta\Psi_4$ measures the decrease in the number of cycles of length 4, due to performing the operation. This inequality holds both for 2- and 1-translocations, and 2- and 1-inversions. □

This implies the following performance guarantee.

Theorem 4.1 *The greedy algorithm is a $\frac{3}{2}$-approximation algorithm for sorting signed data by translocations and inversions in the undirected model.*

Proof Notice that the number of breakpoints can be rewritten as $\Phi(\gamma) = m - n - \Psi_2(\gamma)$. Then by Lemma 4.1,

$$\begin{aligned}
\mathrm{Opt}(\gamma) &\geq (m-n) \;-\; \Psi(\gamma) \\
&= (\Phi(\gamma) + \Psi_2(\gamma)) \;-\; \Psi(\gamma) \\
&= \Phi(\gamma) \;-\; (\Psi_4(\gamma) + \Psi_{\geq 6}(\gamma)) \\
&\geq \Phi(\gamma) \;-\; \big(\Psi_4(\gamma) \;+\; \tfrac{1}{3}(\Phi(\gamma) - 2\Psi_4(\gamma))\big) \\
&= \tfrac{2}{3}\big(\Phi(\gamma) - \tfrac{1}{2}\Psi_4(\gamma)\big),
\end{aligned}$$

where the last inequality follows from the observation that every cycle of length 6 or more contains at least three breakpoints. Applying Lemma 4.2 yields the theorem. □

5 Further research

It appears that a significant obstacle to obtaining improved approximations for translocations alone is deriving a lower bound that reflects the constraint that a translocation must exchange segments from *distinct* chromosomes. In particular, is there a lower bound sensitive to the following: when markers are properly segregated onto chromosomes but not in sorted order, translocations that separate markers but do not increase the cycle count are required.

One important issue we do not address is the comparison of genomes with differing numbers of chromosomes. For such organisms, the evolutionary history must consider chromosome *fissions* and *fusions*, which split and join chromosomes, in addition to the other rearrangement operations. While we do not know of any performance guarantees for sorting with fissions and fusions, we do know that any history involving translocations, fissions, and fusions has a normal form in which all fissions precede all translocations, which precede all fusions. Thus we can restrict our attention to series of this form in the search for a shortest history.

The position of the centromere influences the model of translocation, as no viable exchange can produce a chromosome lacking a centromere. This motivated us to define the directed and undirected models. A more faithful formulation, however, would be desirable, that explicitly models the location of the centromere, and takes this into account when defining the set of allowable exchanges.

Acknowledgments

The first author wishes to thank David Sankoff for helpful discussions, and for suggesting the problem with equal-length translocations.

References

[1] Bafna, Vineet and Pavel A. Pevzner. Genome rearrangements and sorting by reversals. Proceedings of the 34th *Symposium on Foundations of Computer Science*, 148–157, November 1993.

[2] Bafna, Vineet and Pavel A. Pevzner. Sorting by transpositions. Proceedings of the 6th ACM-SIAM *Symposium on Discrete Algorithms*, January 1995.

[3] Bafna, Vineet and Pavel A. Pevzner. Sorting by reversals: genome rearrangements in plant organelles and evolutionary history of X chromosome. Technical Report CSE-94-032, Department of Computer Science, Pennsylvania State University, April 1994.

[4] Copeland, N.G., N.A. Jenkins, D.J. Gilbert, J.T. Eppig, L.J. Maltais, J.C. Miller, W.F. Dietrich, A. Weaver, S.E. Lincoln, R.G. Steen, L.D. Stein, J.H. Nadeau, E.S. Lander. A genetic linkage map of the mouse: current applications and future prospects. *Science* 262, 57–66, 1993.

[5] Hannenhalli, Sridhar, Colombe Chappey, Eugene V. Koonin, and Pavel A. Pevzner. Algorithms for genome rearrangements: herpesvirus evolution as a test case. To appear in Proceedings of the 3rd *International Conference on Bioinformatics and Complex Genome Analysis*, 1994.

[6] Jerrum, Mark R. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science* 36, 265–289, 1985.

[7] Kececioglu, John and David Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. To appear in *Algorithmica*, 1994. (An earlier version appeared as Exact and approximation algorithms for the inversion distance between two chromosomes, Proceedings of the 4th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 684, 87–105, June 1993.)

[8] Kececioglu, John and David Sankoff. Efficient bounds for oriented chromosome-inversion distance. Proceedings of the 5th *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 807, 307–325, June 1994.

[9] Nadeau, Joseph H. and Benjamin A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences USA* 81, 814–818, 1984.

[10] O'Brien, S.J., J.E. Womack, L.A. Lyons, K.J. Moore, N.A. Jenkins, and N.G. Copeland. Anchored reference loci for comparative genome mapping in mammals. *Nature Genetics* 3, 103–112, 1993.

[11] Sankoff, David. Edit distance for genome comparison based on non-local operations. Proceedings of the 3rd *Symposium on Combinatorial Pattern Matching*, Springer-Verlag Lecture Notes in Computer Science 644, 121–135, April-May 1992.

[12] Sankoff, David, Guillaume Leduc, Natalie Antoine, Bruno Paquin, B. Franz Lang, and Robert Cedergren. Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences USA* 89, 6575–6579, 1992.