

Simple Rearrangement using Length-Weighted Inversions

Lina Dong¹, Saurabh Sethia², and Pavel Sumazin¹

¹ Portland State University, P.O. Box 751, Portland, Oregon 97207, USA

² Oregon State University, 102 Dearborn Hall, Corvallis, OR 97331-3202, USA

Abstract. Recent studies of genome rearrangement indicate a high proportion of short inversion events in genome evolution. This data motivates genome rearrangement models that reward short inversions and penalize long inversions. We study the optimization problem of sorting by inversions, where the cost of an inversion is proportional to its length. For the problem of sorting binary sequences by inversions, we give a polynomial time algorithm and a fast greedy approximation algorithm with close to optimal performance in our experiments. Unfortunately, we find that a previously proposed approximation scheme for sorting n -alphabet sequences by repeated binary-sequence-sorting operations fails to produce favorable results in practice.

Keywords: genome rearrangement; sorting by inversions; sorting by reversals

1 Introduction

The problem of inferring evolutionary distance from gene-rearrangement data has been extensively studied in the last decade. Gene-order rearrangement data is recognized to be an important source for phylogeny reconstruction, and computational tools to analyze gene-order dynamics are improving rapidly. Much attention has been given to the inversion-distance (sometimes referred to as reversal-distance) problem, which was a focus of the 2000 *Gene Order Dynamics, Comparative Maps and Multi-gene Families* symposium.

The inversion-distance problem is described as an optimization problem, where the object is to find the highest likelihood transformations between gene sequence pairs. The prevalent mathematical model for inversion-based genomic distance assumes that (1) inversions are randomly distributed throughout the chromosome, (2) the likelihood of an inversion event in the evolutionary history of a chromosome is independent of the inversion length, and (3) gene content can be described in terms of an ordered set without repetition. Computational approaches based on this model attempt to find the shortest transformations, in terms of the number of inversions, between two permutations.

In this paper we study the sorting-by-inversions problem using the additive cost model of Pinter and Skiena [16]. The likelihood of inversion events is a function of the number of inverted elements (as opposed to the number of inversions) and gene repetition is allowed. We study inversion-based sorting of binary sequences, where the cost of each inversion is proportional to its length. As proposed by Pinter and Skiena [16], sequences of alphabet of size k can be sorted using $\lceil \lg k \rceil$ applications of the alphabet-size-two algorithms, giving a $\lceil \lg k \rceil$ approximation ratio for this general problem.

In Section 2 we provide insight into the special structure of inversion transformations of binary sequences, and show that there is always a *simple* transformation schedule into sorted order. In Section 3 we present a polynomial algorithm for binary-sequence sorting by inversions. In Section 4 we prove a lower bound for the *MedianEject* approximation algorithm for binary-sequence sorting by inversions proposed by Pinter and Skiena [16], and propose a greedy approximation algorithm that is experimentally dominant. Our work corrects and extends the work of Bender et al. [2] for additive cost inversions.

We compare the solution quality and time performance of the *MedianEject* approximation algorithm, greedy approximation algorithm and our polynomial-time algorithm on (1) random binary sequences of increasing length, (2) binary sequences generated according to the random-breakage model of Ohno [13], (3) binary sequences generated according to the length-weight-cost inversion model of Pinter and Skiena [16], and (4) binary sequences generated according to a geometric distribution of inversion lengths. The solutions found by greedy are near optimal and are better than the solutions found by *MedianEject* on all sequences tested.

We compare the quality of approximation on representations of human, sea urchin and fruit fly mitochondrial gene-order data given by Sankoff, Sundaram and Kececioğlu [21], and on gene-order representations of genome-segments

of twelve members of the *Campanulaceae* family used by Cosner, Jansen, Moret, Raubeson, Wang, Warnow and Wyman [5]. Our results on binary-sequence sorting agree with the results from simulated data; gene-order binary-representation sorting schedules found by greedy are near optimal and better than those found by MedianEject on all of our gene-order data. Our results on sorting n -alphabet representations using the proposed $\lceil \lg n \rceil$ approximation scheme suggest that the approximation scheme is far too inaccurate on real data. We see a poor translation from good sorting schedules for binary-representations to good overall n -alphabet sorting schedules, implying that the recursive sorting technique often includes unnecessary steps. We conclude that using $\lceil \lg n \rceil$ applications of near-optimal or even optimal binary-sequence sorting algorithms to approximate n -alphabet sorting fails to produce accurate results on real data.

1.1 Notation

We study inversion-based transformations of sequences of the normal form³ $(a^+b^+)^+$ to sequences of the normal form b^+a^+ , where the cost of an inversion is proportional to its length. We use 'a' and 'b' as labels so that the first element of the initial sequence is labeled 'a' and the last element is labeled 'b'. To simplify presentation we will assume that all transformations are of this form. The optimization problem where the cost of an inversion v is $\alpha|v| + \beta$, and α is a positive constant, is equivalent to the optimization problem with inversion cost $|v|$. We will assume inversion cost $|v|$ in this paper.

An inversion in a sequence s_0 over a range $[i, j]$ transforms s_0 to a new sequence s_1 by inverting the range $[i, j]$ so that $s_1[i+k] = s_0[j-k]$ for $k \in \{0 \dots j-i\}$. The sequence s_0 is transformed to sequence s_t via an inversion schedule $V = v_1, v_2, \dots, v_t$, where v_i transforms s_{i-1} to s_i . The cost of an inversion is equal to its length, and the total cost of the inversion schedule V is the sum of the lengths of the inversions in V . We use n to represent the length of the sequence.

1.2 Related Work

The analysis of genome rearrangement was pioneered by Dobzhansky and Sturtevant [6], who studied disruptions of gene order to explain the gene arrangement difference between *Drosophila pseudoobscura* and its sibling specie *Drosophila miranda*. Watterson, Ewens, Hall, and Morgan [23] were the first to systematically study pair-wise genomic distance in terms of the number of breakpoints between two genomes. Nadeau and Taylor [12] were the first to partition genomes into corresponding pairs of conserved sequences in order to estimate chromosomal rearrangements.

The most studied algorithmic approach for inversion-based genomic distance inference is focused on minimizing the inversion distance between two permutations. This model was motivated by the study of small and highly related organisms, most notably the plant mitochondrial DNA rearrangement studies of Makaroff and Palmer [11] and Palmer and Herbon [14]. The computational approach for genome rearrangement was pioneered by Sankoff, Nadea and colleagues [9, 18–20] with later contributions from Pevzner, Hannenhalli, Bafna and colleagues [1, 7, 8]. The inversion distance problem for unoriented permutations was shown to be hard by Caprara [3], but Hannenhalli and Pevzner [7] gave a polynomial time algorithm for the oriented permutation version.

There is compelling evidence that inversion likelihood is dependent on the length of the inverted sequence. Most recently Lefebvre, El-Mabrouk, Tillier and Sankoff [10] presented an experimental study to support the claim that short inversions are more abundant. Computational work on the sorting-by-inversions problem that considers inversion length includes [16, 2, 4]. Pevzner and Tesler [15] present evidence against the assumption that inversions are normally distributed across the chromosome. Gene repetition is common in higher order eukaryotes and even bacteria such as *Chlamydia trachomatis* [17].

2 An Optimal Simple-Inversion Divide-and-Conquer Schedule

In this section we show that there is a recursive optimal schedule, where two elements $s_0[i]$ and $s_0[j]$ are inverted only by the last inversion v_t of the schedule v_1, \dots, v_t . The schedule-prefix v_1, \dots, v_{t-1} can be partitioned into three

³ The normal form notation a^+ stands for a sequence of 'a's of length greater than 0.

independent sub-schedules that transform $s_0[1 \dots i - 1]$, $s_0[i + 1 \dots j - 1]$ and $s_0[j + 1 \dots n]$ to the target form and each can be again repartitioned until reaching a homogeneous sequence. We call such a schedule a *divide-and-conquer schedule*. The existence of a divide-and-conquer schedule implies that any sequence can be transformed to the target form in $O(n \log n)$ cost, and that the optimal transformation cost for some sequences is in $\Omega(n \log n)$. Our polynomial-time algorithm proceeds by constructing a divide-and-conquer schedule.

We refer to same element runs as *strips*, and use m to represent the number of strips in the sequence. Position j in sequence s is called a *strip break* if $s[j] \neq s[j + 1]$. An inversion in s over range $[i, j]$ where $i - 1$ and j are strip breaks does not break any strips and is called a *strip inversion*. An inversion schedule that includes only strip inversions is called a *strip schedule*. *Even inversions* include an equal number of 'a'-element strips and 'b'-element strips. Strip inversions of a sequence containing only one strip break are called *simple inversions*, and an inversion schedule that includes only simple inversions is called a *simple schedule*.

First we show that the existence of an optimal simple schedule implies the existence of an optimal divide-and-conquer schedule. Then we prove the existence of an optimal simple schedule by providing a replacement policy to convert any schedule to a simple schedule of same or lower cost.

2.1 Optimal Simple Schedule Implies Optimal Divide-and-Conquer Schedule

We begin by proving the existence of a strip schedule, and give a constructive algorithm to convert any schedule to a strip schedule of equal or lower cost. To ease our discussion, before applying an inversion schedule to a sequence we extend the sequence with a 'b' on the left and an 'a' on the right. The added elements have no impact on the total cost since they are never included in an inversion. This extension simplifies the exposition by eliminating the need for special treatment of inversions that include the left or right most element of the sequence.

Lemma 1. *The deletion of an element that participates in x inversions in a given schedule reduces the cost of the schedule by at least x .*

Proof. Follow the schedule as if the element e was not deleted. Each inversion that included e is at least one element shorter, and some inversions may be eliminated. The total cost of the resulting schedule is at least x cheaper. \square

Lemma 2. *An inversion that breaks a strip can be replaced by a strip inversion to produce a schedule of same or lower cost.*

Proof. Consider a schedule that includes an inversion v_i that breaks a strip of 'b's between element u at position j and element w at position $j + 1$. Without loss of generality assume that v_i includes w . Element u participates in x_u inversions and w participates in x_w inversions in the schedule v_i, \dots, v_t .

Two cases are possible:

$x_u \leq x_w$. We show that replacing v_i with a shorter inversion that starts at position $j + 2$ instead of position $j + 1$ does not increase the cost of the schedule. Consider inserting a 'b' in position $j + 1$ after the completion of v_i . This inserted element will participate in each of the inversions that include u . The total cost of the schedule will increase by x_u . Deleting w before v_i will decrease the inversion's cost by at least x_w . The resulting schedule is no worse than the original schedule, and it is equivalent to the schedule that results from replacing v_i with a shorter inversion that starts at position $j + 2$.

$x_u > x_w$. By a similar argument, inserting a 'b' at position $j + 1$ and including this 'b' in every inversion made to w will add x_w to the cost of the original schedule. Removing u will reduce the total cost by at least x_u resulting in a schedule that is at least as good as the original.

Continue by induction. At each step we consider breaking the strip at a position that was never considered before until the entire strip is either contained in the replacement inversion or is external to it. The resulting schedule is at least as good as the original schedule. \square

Lemma 2 implies that non-strip schedules can be converted to strip schedules of same or lower cost. Represent the sequence in terms of its alternating strip composition $A_1 B_1 \dots A_{m/2} B_{m/2}$, where m is the number of strips, B_i stands for the i_{th} strip of 'b's, the size of a strip B_i is $|B_i|$ and the number of inversions B_i participates in is b_i . The total cost of a strip schedule can be represented in terms of the strip cost, as given in Equation 1. The total cost is the

sum, over all elements of s_0 , of the number of times each element is inverted by V , which is equal to the sum, over all strips in s_0 , of the number of times each strip is inverted by V multiplied by its length.

$$\text{cost of } V = \sum_{v_i \in V} |v_i| = \sum_{i=1}^n \text{number of times } s_0[i] \text{ is inverted} = \sum_{i=1}^{m/2} b_i |B_i| + a_i |A_i| \quad (1)$$

Lemma 3. *Strip schedules consist of even inversions.*

Proof. Assume v is a strip inversion over the range $[i, j]$ that has same elements at positions i and j . Reduce the length of v by $2k$ so that its new range $[i+k, j-k]$ is the longest to have different elements in positions $i+k$ and $j-k$. The shorter replacement-inversion is cheaper. If the replacement inversion is not a strip inversion, convert it to a strip inversion using the algorithm described in Lemma 2. \square

Corollary 1. *Any schedule can be converted to an even schedule of same or lower cost.*

Observation 1 *Strip inversions decrease the number of strip breaks by 2.*

We define *anti-schedules* and *anti-inversions* as follows. If V is an inversion schedule then its anti-schedule $\tilde{V} = \tilde{v}_t, \tilde{v}_{t-1}, \dots, \tilde{v}_1$ transforms s_t to s_0 , so that if $v_i \in V$ inverts the range $[j, k]$ and transforms s_{i-1} to s_i then its anti-inversion \tilde{v}_i inverts the same range and transforms s_i to s_{i-1} . If V is a simple schedule then \tilde{V} is a simple anti-schedule, and if v is a simple inversion then \tilde{v} is a simple anti-inversion.

Observation 2 *Anti inversions in a strip anti-schedule increase the number of strip breaks by 2.*

Observation 3 *Anti-inversions in a simple anti-schedule include one strip break and split 2 strips.*

Lemma 4. *There are two elements in anti-inversion \tilde{v}_i that are not inverted by the remainder of the simple anti-schedule $\tilde{v}_{i-1}, \dots, \tilde{v}_1$.*

Proof. We consider the effect of \tilde{v}_i , which inverts $s_i[j \dots k]$. The subsequence $s_{i-1}[j-1 \dots k+1]$ includes elements from exactly 4 strips. Label these strips $B_1 A_1 B_2 A_2$. There is an element in A_1 that does not participate in any of the remaining anti-inversions, or there is an anti-inversion u that constructs a sequence where an element from B_1 and an element from B_2 are adjacent. The anti-inversion u increases the number of strip breaks by at most 1, which contradicts Observation 2. Following a symmetric argument there must be element in B_2 that does not participate in any of the remaining anti-inversions. \square

Theorem 1. *The existence of an optimal simple schedule implies the existence of an optimal divide and conquer schedule.*

Proof. The anti-inversion \tilde{v}_i transforms a sequence of the target form BA to a sequence of the form $B_1 A_1 B_2 A_2$. By Lemma 4 there are elements in strip A_1 and in strip B_2 that do not participate in the remaining anti-inversions in \tilde{V} . These elements partition the sequence to 3 subsequences that will be inverted independently by the remaining anti-inversions. Inductively, each anti-inversion contains two such elements that further partition each subsequence to three subsequences. \square

2.2 There is an Optimal Simple Schedule

In order to prove that there is an optimal simple schedule we show how to convert a non-simple even schedule to a simple schedule of equal or lower cost. Consider a schedule that includes a non-simple even-inversion v_i that operates on range $[l, r]$, followed by a simple schedule r_1, \dots, r_z that consists of the inversions that transform the subsequence $s_i[l \dots r]$ to a sequence with a single strip break and include no strips outside of $s_i[l \dots r]$. The schedule r_1, \dots, r_z is a sub-schedule of V , and need not be contiguous. Any inversion not in r_1, \dots, r_z that is scheduled after v_i and before r_z can be rescheduled before v_i . We call v_i, r_1, \dots, r_z a *contained schedule*.

Lemma 5. *A contained schedule can be replaced by a simple schedule.*

Proof. The inversion schedule v_i, r_1, \dots, r_z operates on range $[l, r]$ of the original sequence. If the inversion r_i operates on range $[j, k]$, then define r'_i to be an inversion that operates on range $[l + r - k, l + r - j]$ of the same sequence. The cost-equivalent simple-inversion-schedule r'_1, \dots, r'_z, v_i can replace the contained-inversion schedule with no effect on the rest of the schedule. The inversion v_i in the new schedule inverts the same range as it did before, but now it is a simple inversion. \square

Next we show how to replace non-contained non-simple inversions with simple inversions. We proceed by repeatedly replacing the last non-contained non-simple inversion v_i over range $[l, r]$ and the simple-inversion sub-schedule $v_{i+1} \dots v_t$ with a schedule of simple inversions.

First reschedule all inversions where either both inverted strips are not in $s_i[l \dots r]$ or both inverted strips are in $s_i[l \dots r]$ before v_i . The range of the enclosed inversions needs to be adjusted as in the proof of Lemma 5. The schedule $U = u_1, \dots, u_t$ contains the remaining inversions.

Follow U to the inversion $u_f = v_q$ that generates the first sequence s_q containing a range $[l', r']$, where each strip in $s_{i-1}[l, r]$ is also in $s_q[l', r']$ and $s_q[l' \dots r']$ includes exactly one strip-break. We call the sequence $s_{i-1}[l' \dots r']$ a *containing sequence*, and the schedule v_i, u_1, \dots, u_f a *containing schedule*. We say that a strip is internal to $s[l \dots r]$ if it neither includes $s[l]$ nor $s[r]$.

Lemma 6. *A containing schedule includes a strip internal to $s_i[l \dots r]$ that is inverted by exactly the first and last inversions of the schedule.*

Proof. A containing schedule $v_i, u_1 \dots, u_f$ over range $[l', r']$ transforms $s_{i-1}[l' \dots r']$ to the target form, and includes a non-simple even inversion that is followed by simple inversions. The last inversion in the schedule u_f includes two elements, e_x and e_y , that are not inverted by any of u_1, \dots, u_{f-1} (Lemma 4). We first show that at least one of e_x, e_y is inverted by v_i .

Assume that neither e_x nor e_y is inverted by v_i . There are two possible cases:

Both e_x and e_y are on the same side of $s_i[l \dots r]$. The elements e_x and e_y partition $s_i[l' \dots r']$ into three subranges – one that includes $[l, r]$ and two that do not. This contradicts the construction of the containing schedule since u_1, \dots, u_{f-1} generates a sequence that contains each strip in $s_i[l \dots r]$ and is in target form.

The range defined by the elements spans $s_i[l \dots r]$. The spanning range contains a single strip break, implying that u_1, \dots, u_{f-1} is a containing schedule.

Now we show that at least one of e_x, e_y is internal to $s_i[l \dots r]$. We consider two cases. In the first case both e_x, e_y are in $s_i[l \dots r]$, and in the second case only one of e_x and e_y is in $s_i[l \dots r]$. In the first case e_x and e_y must be adjacent in order for u_f to sort $[l', r']$, which implies that one of them is internal to $s_i[l \dots r]$. In the second case if the element in $s_i[l \dots r]$ is not internal then u_1, \dots, u_{f-1} is a containing schedule. \square

Finally, we show how to replace v_i with an inversion of fewer strips or a longer non-contained inversion. Repeated applications of the replacement rules will result in either reducing v_i to a simple inversion, or reducing the containing schedule into a contained schedule.

Lemma 7. *The last non-simple even inversion can be replaced by an even inversion that includes fewer strips or by a longer even inversion with the same number of strips.*

Proof. Label the strip composition of $s_{i-1}[l \dots r]$ as $A_k B_k \dots A_p B_p$. We present replacement rules for v_i and either the first inversion (w_k) to include A_k or the first inversion (w_p) to include B_p based on the relationship between a_k and a_p and b_k and b_p , where a_k, a_p, b_k, b_p are the number of times A_k, A_p, B_k, B_p are inverted by U respectively. The replacement rules will never call for the use of non-existent strips. A corollary to lemma 6 is that B_p and A_k can be inverted independently.

The replacement rules assume that B_p and A_k can be inverted independently. Next we show that the rules will hold even if one of them is not allowed to be inverted without the other. Lemma 4 implies that there are two strips in s_i (C left of D) that are inverted by u_f and are not inverted by u_1, \dots, u_{f-1} . Lemma 6 implies that at least one of these strips lies in the range $s_i[l \dots r]$. The inversions w_k and w_p are dependent only if both C and D lie in the

$a_k < a_p$	$b_k > b_p$	$a_k \geq a_p$ and $b_k \leq b_p$
Original Schedule: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}\mathbf{B}_{k-1}\mathbf{B}_p\mathbf{A}_p\cdots B_kA_kA_{p+1}B_{p+1}$ $A_{k-1}A_pB_pB_{k-1}\cdots B_kA_kA_{p+1}B_{p+1}$ Replacement: $A_{k-1}B_{k-1}A_kB_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{B}_pA_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_p\cdots B_kA_kA_{p+1}B_{p+1}$	Original Schedule: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_pA_p\cdots B_k\mathbf{A}_k\mathbf{A}_{p+1}\mathbf{B}_{p+1}$ $A_{k-1}B_{k-1}B_pA_p\cdots B_kB_{p+1}A_{p+1}A_k$ Replacement: $A_{k-1}B_{k-1}A_kB_k\cdots A_pB_p\mathbf{A}_{p+1}\mathbf{B}_{p+1}$ $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_p\mathbf{B}_{p+1}A_{p+1}$ $A_{k-1}B_{k-1}\underline{B_{p+1}}B_pA_p\cdots B_kA_kA_{p+1}$	Original Schedule: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_pA_p\cdots B_kA_kA_{p+1}B_{p+1}$ Replacement: $A_{k-1}B_{k-1}A_k\mathbf{B}_k\cdots\mathbf{A}_pB_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}\underline{A_k}A_p\cdots B_k\underline{B_p}A_{p+1}B_{p+1}$
Original Schedule: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $\mathbf{A}_{k-1}\mathbf{B}_{k-1}\mathbf{B}_pA_p\cdots B_kA_kA_{p+1}B_{p+1}$ $B_pB_{k-1}A_{k-1}A_p\cdots B_kA_kA_{p+1}B_{p+1}$ Replacement: $\mathbf{A}_{k-1}\mathbf{B}_{k-1}A_kB_k\cdots A_pB_pA_{p+1}B_{p+1}$ $B_{k-1}\mathbf{A}_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $B_{k-1}B_pA_p\cdots B_kA_k\underline{A_{k-1}}A_{p+1}B_{p+1}$	Original Schedule: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_pA_p\cdots\mathbf{B}_k\mathbf{A}_k\mathbf{A}_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_pA_p\cdots A_{p+1}A_kB_kB_{p+1}$ Replacement: $A_{k-1}B_{k-1}\mathbf{A}_k\mathbf{B}_k\cdots A_pB_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}B_k\mathbf{A}_k\cdots\mathbf{A}_p\mathbf{B}_pA_{p+1}B_{p+1}$ $A_{k-1}B_{k-1}\underline{B_k}B_pA_p\cdots A_kA_{p+1}B_{p+1}$	

Fig. 1. Replacement policy for the last non-simple even inversion v_i that inverts the sequence $A_kB_k\cdots A_pB_p$ depends on (1) the number of times a_k, b_k, a_p, b_p that the strips A_k, B_k, A_p, B_p are inverted by the remaining simple schedule, (2) the first simple inversion that includes A_k , and (3) the first simple inversion that includes B_p . We consider three cases: $a_k < a_p$: B_p is inverted with its left or its right neighbor. In the first case, excluding A_p from v_i and uniting it with A_k reduces the number of strips in v_i and results in a schedule that is at least as good; in the second case including A_{k-1} in v_i increases the length of v . The case where $b_k > b_p$ is symmetric. In the case where $a_k \geq a_p$ and $b_k \leq b_p$, excluding both A_k and B_p from v_i reduces the number of strips inverted by v_i . In all cases the resulting schedule is at least as good as the original schedule.

range $s_i[l \dots r]$ and either C is B_p or D is A_k . The replacement rules fail when calling for the replacement of a simple inversion that includes C or D . The replacement rules call for the replacement of a simple inversion that includes C only if $a_k < a_p$, but if C is B_p then D must be A_p and $a_k \geq a_p$. The case where replacement rules call for the replacement of a simple inversion that includes follows by symmetry.

We next describe the replacement rules.

To choose a replacement we distinguish between 3 cases, (1) $a_k < a_p$, (2) $b_k > b_p$, and (3) $a_k \geq a_p$ and $b_k \leq b_p$. Consider the case where $a_k < a_p$. The replacement choice will depend on w_p . If w_p is $B_{k-1}B_pA_p$ then v_i and w_p can be replaced by the inversion of A_pB_p followed by the inversion of $A_kB_k \dots A_{p-1}B_{p-1}B_p$. The strip A_p will participate in all inversions that include A_k . The replacement is cheaper, and the new schedule is cheaper since the new a_p is smaller. If w_p is $A_{k-1}B_{k-1}B_p$, then v_i and w_p can be replaced by the inversion of $A_{k-1}B_{k-1}$ followed by the inversion of $A_{k-1}A_kB_k \dots A_pB_p$. The strip A_{k-1} will participate in all inversions that include A_k . The replacement inverts A_{k-1} one time more, but the new a_{k-1} is smaller and so the cost for the new schedule is no greater.

Consider the case where $b_k > b_p$. The replacement choice depends on w_k . If w_k is $A_kA_{p+1}B_{p+1}$ then v_i and w_k can be replaced by the inversion of $A_{p+1}B_{p+1}$ followed by the inversion of $A_k \dots B_pB_{p+1}$. If w_k is $B_kA_kA_{p+1}$, then v_i and w_k can be replaced by the inversion of A_kB_k followed by the inversion of $A_kA_{k+1} \dots A_pB_p$.

In the case that $a_k \geq a_p$ and $b_k \leq b_p$, replace v_i by the inversion $B_kA_{k+1} \dots B_{p-1}A_p$. The inversion v_i includes fewer strips, the replacement is cheaper, and the rest of the schedule is at most as expensive. The resulting sequence can be described as a deletion of A_k and B_p from s_{i-1} , and their re-insertion so that A_k is adjacent to A_p and B_p is adjacent to B_k . A_k will be included in each inversion that includes A_p , and B_p will be included in each inversion that includes B_k . The cost of the new schedule did not increase since the new a_k and b_p are no larger. □

Theorem 2. *There is an optimal simple-inversion schedule.*

Proof. Convert all inversions to even inversions. Repeatedly replace the last non-simple inversion in the schedule with simple inversions, until no non-simple inversions remain, as follows.

If this inversion is a part of a contained schedule, then replace the contained schedule with a schedule of simple inversions. Otherwise, reschedule all independent and enclosed inversions in the remainder of the schedule before this inversion. Follow the procedure outlined in Lemma 7 to replace this inversion with an inversion that includes fewer strips or with a longer inversion that includes as many strips. Repeat the rescheduling and replacement procedures until arriving at a simple inversion schedule or a contained inversion schedule. □

2.3 Lower Bound on Worst Case Sorting Cost

Theorem 3 is a direct consequence of the existence of an optimal divide-and-conquer schedule.

Theorem 3. *Sorting by length-weight inversions cost $\Omega(n \log n)$ in the worst case.*

Proof. Consider the anti-inversion complement of the optimal divide-and-conquer schedule that transforms the sequence $(ab)^+$ to the target form. The cost of the anti-inversion schedule is equal to the cost of the inversion schedule. The first anti-inversion transforms $B_1B_2A_1A_2$ to $B_1A_3aA_4B_3bB_4A_2$, where $|A_1| = |A_3| + |A_4| + 1$ and $|B_2| = |B_3| + |B_4| + 1$. The strips A_3 and A_4 are separated by an 'a' that will not be anti-inverted again, and the strips B_3 and B_4 are separated by a 'b' that will not be anti-inverted again. Consequently, the remaining anti-inversions will not include elements that are both from A_3 and A_4 or elements that are both from B_3 and B_4 . Since $\text{abs}(|B_1| - |A_3|) \leq 1$ and $\text{abs}(|B_4| - |A_2|) \leq 1$, the length of the first anti inversion is greater than $n/2$. The sum of node costs in the implied recursion tree is at least $n \log_3 n$, and the total cost for the divide-and-conquer recursion is greater than $\frac{n \log_3 n}{2}$. □

3 Polynomial Algorithm

We describe a dynamic-programming polynomial algorithm for binary-sequence sorting by inversions. We call the algorithm *StripSort*. StripSort relies on the existence of an optimal divide-and-conquer schedule of simple inversions

shown in Section 2. StripSort proceeds as follows. Compile the m -length list of the strip breaks in the initial sequence $s_0 = A_1B_1 \cdots A_{m/2}B_{m/2}$. We describe the sorting procedure on this list. By Lemma 2 and 4 there are two strips C and D , where C is left of D , that are inverted by the last inversion in the optimal schedule and by no other inversion. C must be an 'a' strip and D must be a 'b' strip. Sort all strips left of C to BA form. The strips on the right of D will also be sorted to BA form. The strips between C and D will be sorted to AB form. After sorting the subproblems we get a sequence of form $BABA$. Perform the middle simple inversion to bring the sequence into BA form. The recurrence on the strip list is given in Equation 2; the sizes of the left and right strips in each sorted or reverse sorted sequence are recorded, and the cost of the last inversion in each subproblem is equal to the sum of the lengths of the right strip of the sequence left of C ($\text{RightStripLength}(i, l - 1)$), left strip of the sequence right of D ($\text{LeftStripLength}(k + 1, j)$) and the sequence from C to D ($\text{Length}(l, k)$).

$$\begin{aligned} \text{StripSort}(i, j) = & \text{Min}_{i \leq l < k \leq j} \{ \text{RightStripLength}(i, l - 1) + \text{LeftStripLength}(k + 1, j) \\ & + \text{StripSort}(i, l - 1) + \text{StripSort}(l + 1, k - 1) + \text{StripSort}(k + 1, j) \\ & + \text{Length}(l, k) \} \end{aligned} \quad (2)$$

There are $\Theta(m^2)$ different subproblems, each requiring $O(m^2)$ lookups in the dynamic programming table. Thus, we get the following theorem.

Theorem 4. *StripSort finds an optimal schedule in $O(m^4)$ time and $O(m^2)$ space.*

4 Approximation Algorithms

In this section we give an $\Omega(\log \log n)$ lower bound on the approximation-ratio for the *MedianEject* algorithm proposed by Pinter and Skiena [16], and we propose a greedy approximation algorithm with $O(\log m)$ approximation-ratio. Tighter bounds on the approximation-ratio are left as an open problem. In our experiments the greedy approximation algorithm has comparable running time to the *MedianEject* approximation algorithm, and it finds near-optimal solutions for all inputs. The worst-case running time for the greedy approximation algorithm is $O(m \log m)$, and the worst-case running time for *MedianEject* is $O(m)$. Both algorithms use $O(m)$ space.

The greedy approximation algorithm proceeds as follows. At every step make the shortest simple inversion possible, but with one exception. Give preference for inversions at the extreme left and right: if an inversion inverts a sequence of the form AB and there are no 'a' strips on its left, consider the length of this inversion to be the length of its 'a' strip; if there are no 'b' strips on its right, consider the length of this inversion to be the length of its 'b' strip.

Lemma 8. *The approximation ratio of MedianEject is in $\Omega(\log \log n)$.*

Proof. Consider the n -length sequence of the normal form below, where $i = \log_3^{n/2} - 2 = \Theta(\lg n)$. The total cost for the schedule found by *MedianEject* is $\Theta(n \lg \lg n)$, while the total cost for the schedule found by the greedy algorithm is $2 + \sum_{k=0}^{i+1} 6 \cdot 3^k = 3^{i+3} - 1 = \Theta(n)$.

$$a^{3^i \cdot 6} b^{3^{i-1} \cdot 6} \dots a^{3^2 \cdot 6} b^{3^1 \cdot 6} a^6 b^2 a b a^2 b^6 a^{3^1 \cdot 6} b^{3^2 \cdot 6} \dots a^{3^{i-1} \cdot 6} b^{3^i \cdot 6}$$

□

Lemma 9. *The approximation ratio of the greedy approximation algorithm is $O(\log m)$.*

Proof. The length of the shortest inversion is less than $\frac{2n}{m-1}$ long, and the schedule cost is

$$\sum |\text{shortest inversion at each iteration}| < \sum_{k=1}^{m/2} \frac{2n}{2k-1} < 2n \sum_{k=1}^{m-1} \frac{1}{k} = 2n(\ln(m-1) + O(1))$$

□

Lemma 10. *The greedy approximation algorithm is not optimal.*

Proof. Consider the sequence of the following normal form, where $i = \lg(n+2) - 2$.

$$a^{\frac{n}{2}-i} b^{2^0} a b^{2^1} a b^{2^2} \dots a b^{2^{i-1}} a b^{2^i}$$

The greedy algorithm will first transform the sequence to AB form and then perform an inversion of all elements. The cost for moving all b^{2^j} strips to the right is $\sum_{k=0}^{i-1} \sum_{j=0}^k 2^j = 2^{i+1} - i - 2$, and the total transformation cost is $i + 2^{i+1} - i - 2 + n = 2^{\lg(n+2)-1} - 2 + n = \frac{3n}{2} - 1$. The following method outperforms the greedy approximation algorithm on this sequence. Invert the second and third right strips repeatedly until reaching a sequence of form AB , then transform AB to BA using one n -length inversion. Before the final inversion each 'b' strip is inverted exactly once. The cost for moving all 'b' strips to the right is $\frac{n}{2} - 2^i$. The cost for moving all 'a' strips to the left is $\sum_{k=1}^i k$. The total transformation cost is $\frac{n}{2} - 2^i + \sum_{k=1}^i k + n = \frac{5n}{4} + \frac{i(i+1)-1}{2} < \frac{3n}{2} - 1$. \square

Table 1. The worst approximation ratios observed on all sequences by generation method.

Algorithm	Normal Large	Normal Small	Linear	Geometric	Random
Greedy	1.0685	1.0715	1.0617	1.1007	1.0714
MedianEject	1.3586	1.3772	1.3996	2.3370	1.3103

4.1 Experimental Evaluation

Our experiments were done on an AMD Athlon personal computer with a 2.8GhZ processor, 1GB RAM and 514K L2 cache. Source code is available for download at <http://www.cs.pdx.edu/~ps/code/invert/index.html>. The indexing web page is entitled *Simple Rearrangement using Length-Weighted Inversions*.

Experimental Evaluation on Constructed Sequences. We constructed random binary sequences of increasing length, and binary sequences of length 1000 with an increasing number of inversions. Inversion sizes were taken from (1) a normal distribution with mean 200 and standard deviations of 20 and 200, (2) a geometric distribution where the probability of an inversion-length is proportional to $1/e$ raised to that length, and (3) a linear distribution where the probability of an inversion-length x is $(2/1000)[1 - x/1000]$. Distributions were chosen according to different proposed models for inversion lengths. The random-breakage [13] model implies that inversion lengths are taken from a normal distribution. Our computational study is based on the model proposed by Pinter and Skiena [16], where the probability of an inversion is inversely proportional to its length (Linear distribution). A possible model to account for the abundance of short inversions advocates a geometric distribution of inversion lengths; sequences generated via inversions with length taken from a geometric distribution are expected to be harder to approximate.

We compared the cost of the sorting schedules found by the MedianEject approximation algorithm, the greedy approximation algorithm and the polynomial-time algorithm StripSort of Section 3 for increasing sequence length and increasing number of inversions in a sequence with fixed-length. We also compared the time-performance gap between the 2 approximation algorithms and StripSort for increasing number of strips.

In Figure 1 we report the solution cost found by the two approximation algorithms and compare this cost to the optimal sorting cost. Each data point in the sorting-cost plots corresponds to the median sorting-cost of 30 generated sequences, and each data point in the time-performance plot corresponds to the cumulative sorting time of 30 generated sequences. Each of the plots presents data for 1800 sequences. In Table 1 we report the worst approximation ratio of each algorithm over all sequences of a given inversion-generation method. In our experiments, there was no sequence for which MedianEject produced a better solution than greedy. We conclude that the greedy approximation algorithm has running time near that of MedianEject, and provides a better approximation to the optimal sorting algorithm.

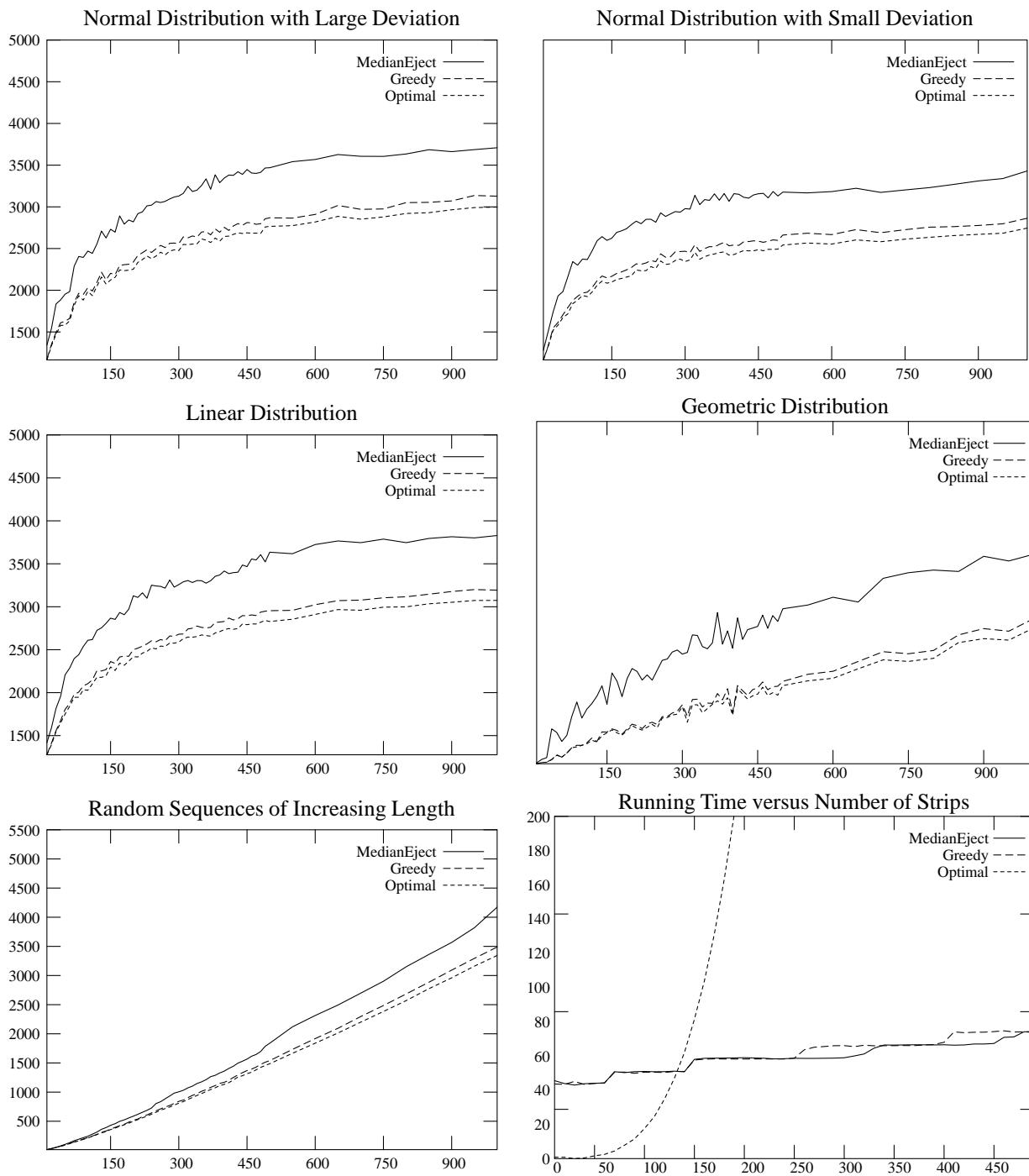


Fig. 2. TopFour. Sorting cost for sequences of length 1000 as a function of number of inversions. **TopLeft.** Inversion lengths are taken from a normal distribution with a large standard deviation. **TopRight.** Inversion lengths taken from a normal distribution with a small standard deviation. **MiddleLeft.** Inversion lengths taken from a linear distribution. **MiddleRight.** Inversion lengths taken from a geometric distribution. **BottomLeft.** Sorting cost for random sequences as a function of sequence length. **BottomRight.** Sorting time in milliseconds as a function of the number of strips.

Table 2. Total sorting cost of gene-order representations using StripSort, Greedy, MedianEject, and GRIMM. The first three representations (Human, Sea Urchin and Fruit Fly) are permutations of gene-order lists of length 33, and the remaining twelve representations are permutations of gene-order lists of length 105. The first three columns corresponding to StripSort, Greedy and MedianEject record the sorting cost of the of the gene-order-list binary representations. The next three columns corresponding to StripSort, Greedy and MedianEject record the total cost of the gene-order sorting schedules. The right most column records the cost of the sorting schedule given by GRIMM.

Organism	First Recurrence Step Cost			Total Sorting Cost			
	StripSort	Greedy	MedEject	StripSort	Greedy	MedEject	GRIMM
Human	54	55	55	171	160	181	213
Sea Urchin	58	61	66	165	161	184	199
Fruit Fly	53	53	64	185	171	180	194
Adenophora	59	59	59	251	251	267	240
Asyneuma	104	104	104	321	321	359	268
Campanula	59	59	59	246	247	262	222
Codonopsis	25	25	25	137	137	137	259
Cyananthus	44	44	44	202	178	202	190
Legousia	127	127	129	399	402	334	333
Merciera	78	78	78	307	307	319	392
Platycodon	47	47	47	191	191	192	234
Symphyandra	67	67	80	250	250	266	227
Trachelium	59	59	59	242	242	258	219
Triodanus	104	104	110	410	410	339	299
Wahlenbergia	80	80	80	340	340	340	298

Experimental Evaluation on Gene-Order Representations. We evaluated the algorithms on the representations of human, sea urchin and fruit fly mitochondrial gene-order data given by Sankoff, Sundaram and Kececioğlu [21]. These representations are signed permutations of the integer list from 1 to 33. We also evaluated the algorithms on representations of genome-segments of the twelve plants from the Campanulaceae family used by Cosner, Jansen, Moret, Raubeson, Wang, Warnow and Wyman [5]. These representations are signed permutations of the integer list from 1 to 105, corresponding to the gene order in Tobacco. Orientation was disregarded when using StripSort, Greedy and MedianEject.

We compared the total sorting cost when using recursive applications of StripSort, Greedy and MedianEject to the cost of the sorting schedule found by GRIMM [22]. The total sorting cost is computed as the sum of the sorting costs over all levels of the recurrence. GRIMM uses orientation information and sorts the representations to minimize the number of inversions. The algorithm to sort gene-order representations partitions the unoriented gene-order list, encoding elements greater than the list’s median with a ‘1’ and the remaining elements with a ‘0’. It then uses StripSort, Greedy or MedianEject to sort the resulting binary sequence, arranging all elements encoded with a ‘0’ in sorted order with respect to elements encoded with a ‘1’. The two lists are then decoded and sorted independently. Confirming the simulation results, StripSort optimally sorts the binary representations of the gene-order lists and the Greedy approximation algorithm always outperforms MedianEject. However, we are not able to predict the relative total-sorting-cost performance of recursive applications of these algorithms. Although our results indicate that uniform-cost-model solutions are not the best linear-cost-model solutions, GRIMM’s sorting schedule was best for many of the longer gene-order representations. Our results, summarized in Table 2, suggest that there is a large error associated with recursive applications of the binary-sequence sorting algorithms and that it is not effective on real data.

Acknowledgment. We thank Steven Skiena for bringing the problem to our attention, and Bernard Moret for pointing us to related publications. This work is supported by NSF grant DBI-0306152.

References

1. V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.
2. M. A. Bender, D. Ge, S. He, R. Y. Pinter, S. S. Skiena, and F. Swidan. Improved bounds on sorting with length-weighted reversals. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 919–928, 2004.
3. A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, pages 75–83, 1997.
4. T. T. Chen and S. S. Skiena. Sorting with fixed-length reversals. *Discrete Applied Mathematics*, 71(1-3):269–295, Dec. 1996.
5. M. E. Cosner, R. K. Jansen, B. M. E. Moret, L. A. Raubeson, L. S. Wang, T. Warnow, and S. Wyman. A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data. In *Proceedings of the 8th international conference on Intelligent Systems for Molecular Biology*, pages 104–115, 2000.
6. T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *Drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.
7. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 178–189, 1995.
8. S. Hannenhalli and P. A. Pevzner. Transforming men into mice: polynomial algorithm for genomic distance problem. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 581–592, 1995.
9. J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, 1995.
10. J. F. Lefebvre, N. El-Mabrouk, E. Tillier, and D. Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, 19(1):190–196, 2003.
11. C. Makaroff and J. D. Palmer. Mitochondrial dna rearrangements and transcriptional alterations in the male sterile cytoplasm of ogura radish. *Molecular Cellular Biology*, 8:1474–1480, 1988.
12. J. H. Nadeau and B. A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. In *Proceedings of the National Academy of Sciences*, volume 81(3), pages 814–818, 1984.
13. S. Ohno. Ancient linkage groups and frozen accidents. *Nature*, 244(5414):259–262, 1973.
14. J. D. Palmer and L. A. Herbon. Plant mitochondrial dna evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28:87–97, 1988.
15. P. A. Pevzner and G. Tesler. Transforming men into mice: the Nadeau-Taylor chromosomal breakage model revisited. In *Proceedings of the Seventh Annual International Conference on Computational Molecular Biology*, pages 247–256, 2003.
16. R. Y. Pinter and S. S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:103–111, 2002.
17. T. D. Read et al. Genome sequences of *Chlamydia trachomatis* MoPn and *Chlamydia pneumoniae* AR39. *Nucleic Acids Res.*, 28(6):1397–1406, 2000.
18. D. Sankoff. Edit distances for genome comparisons based on non-local operations. In *Proceedings of the Symposium on Combinatorial Pattern Matching*, volume 644, pages 121–135, 1992.
19. D. Sankoff, R. Cedergren, and Y. Abel. Genomic divergence through gene rearrangement. *Methods in Enzymology*, 183:428–438, 1990.
20. D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. F. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. In *Proceedings of the National Academy of Sciences*, volume 89, pages 6575–6579, 1992.
21. D. Sankoff, G. Sundaram, and J. Kececioglu. Steiner points in the space of genome rearrangements. *International Journal of Foundations of Computer Science*, 7:1–9, 1996.
22. G. Tesler. Grimm: genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.
23. G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.