An Algorithm to Find All Sorting Reversals*

Adam C. Siepel†
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, USA

National Center for Genome Resources
2935 Rodeo Park Drive East
Santa Fe, NM 87505, USA

acs@cse.ucsc.edu

ABSTRACT

The problem of estimating evolutionary distance from differences in gene order has been distilled to the problem of finding the reversal distance between two signed permutations. During the last decade, much progress was made both in computing reversal distance and in finding a minimum sequence of sorting reversals. For most problem instances, however, many minimum sequences of sorting reversals exist, and obtaining the complete set can be useful in exploring the space of genome rearrangements (e.g., in pursuit of solutions to higher-level problems). The problem of finding *all* minimum sequences of sorting reversals reduces easily to the problem of finding all sorting reversals of one permutation with respect to another. We derive an efficient algorithm to solve this latter problem, and present experimental results indicating that our algorithm offers a dramatic improvement over the best known alternative. It should be noted that in asymptotic terms the new algorithm does not represent a significant improvement: it requires $O(n^3)$ time (where n is the permutation size), while the problem can now be solved trivially in $\Theta(n^3)$ time.

Keywords

Genome rearrangements, sorting by reversals

1. INTRODUCTION

It is generally believed that chromosomal rearrangements occur by a few well-defined processes, including inversion, transposition, and reciprocal translocation [16]. Of these, inversion is considered the most important for many genomes [10, 5]. Computing evolutionary distance with gene order data, assuming an inversions-only model of genome rearrangement (with exactly one copy of each of n genes present in each genome, and all inversions of equal cost), is equivalent to solving the "reversal distance" problem: that is, to find the minimum number of reversals required to convert one permutation into another. A related but distinct problem is to find an actual sequence of reversals that will "sort" one permutation with respect to another. Both of these problems are NP-Hard in the general (unsigned) case [6]. In the case where elements of a permutation are "signed," however, they have polynomial-time solutions [8] (the genome rearrangement problem can be modeled with signed permutations if the direction of transcription is known of each gene in each genome). The mathematical characterization of genome rearrangements, with particular attention to inversions (reversals), has been a subject of intense interest in computational molecular biology during the last decade (see summaries in [19, 13, 16]). Many results have been purely theoretical, but authors have also found applications in several areas, including phylogeny reconstruction [18, 14, 15, 11], comparative mapping [7], and the estimation of the lengths or boundaries of homologous chromosomal segments [12, 17, 7, 21].

Bafna and Pevzner discovered early on [2] that the number of sorting reversals separating signed permutations was closely correlated to the number of cycles in a particular diagram—the "breakpoint graph," or (more colorfully) "Diagram of Reality and Desire" [19]. Later, Hannenhalli and Pevzner characterized certain peculiar structures in the breakpoint graph—which they called "hurdles" and "fortresses"—that caused the correlation between cycles and distance not to be exact [8]. Hannenhalli and Pevzner arrived at a duality theorem that exactly expresses reversal distance as a function of the numbers of cycles and hurdles; whether a fortress is present determines which case of the theorem to apply (if a fortress is present, an additional sorting reversal is required). Using these results, they produced a $O(n^4)$ algorithm to sort by reversals (where n is the permutation size). Not long afterwards, Berman and Hannenhalli arrived at an improved $O(n^2\alpha(n))$ solution. Since then, Kaplan, Shamir, and Tarjan have shown that sorting can be achieved in $O(n^2)$ time [9], and Bader, Moret, and Yan have shown how to compute

*An expanded version of this material appears as Chapter 3 of the author's Master's thesis [20].

†Current address: Department of Computer Science, Baskin Engineering Building, University of California, Santa Cruz, CA 95064

reversal distance (without actually sorting) in $O(n)$ time. Most recently, Bergeron [3, 4] published an alternative sorting approach that also takes $O(n^2)$ time, but that sidesteps much of the complexity of earlier approaches.

All reversal-sorting algorithms published so far find a single minimum-length sequence of sorting reversals. While they generally can be adapted to find multiple sequences of sorting reversals, none will find *all* minimum sequences. We have found in other work that it is sometimes desirable to know all minimum sequences of sorting reversals; such knowledge can enable one efficiently to explore the space of genome rearrangements, in pursuit of solutions to higher-level problems [20]. Knowing all minimum sequences of sorting reversals also can help to complete the biological picture in real scientific applications of reversal sorting algorithms. For example, a biologist might want to evaluate the scientific merits of various parsimonious rearrangement scenarios that seek to explain a particular data set.

The problem of finding all minimum sequences of sorting reversals between a permutation π and a permutation ϕ reduces easily to the problem of finding all *individual* sorting reversals of an intermediate permutation π' with respect to ϕ: a search to address the former need only use an algorithm for the latter as its branching step. We will call the problem of finding all individual sorting reversals the "all sorting reversals problem" and will abbreviate it as *ASR*.

In this paper, we derive an efficient $O(n^3)$ solution to *ASR*. The algorithm is developed as follows: we begin by outlining a straighforward classification of all possible reversals; then we introduce a simplified version of the problem, which we call the "Fortress-Free Model" (FFM), and using the FFM, we prove exactly which classes of reversals can be sorting reversals, and under what conditions they sort; finally, we adapt our results for the general case by re-introducing fortresses. Using the principles we have developed in this way, we describe an algorithm that solves *ASR*. After presenting the algorithm, we report on experimental results that demonstrate its efficiency and affirm its correctness.

2. NOTATION AND DEFINITIONS

Let π and ϕ be signed permutations of size n, such that $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ and $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$. Let the unsigned permutation $\pi' = (\pi'_0, \pi'_1, \ldots, \pi'_{2n+1})$ be defined such that $\pi'_0 = 0$, $\pi'_{2n+1} = 2n + 1$, and for all i, $1 \leq i \leq n$, $\pi'_{2i} = 2\pi_i$, $\pi'_{2i-1} = 2\pi_i - 1$ (if $\pi_i > 0$) or $\pi'_{2i} = 2|\pi_i| - 1$, $\pi'_{2i-1} = 2|\pi_i|$ (if $\pi_i < 0$); let the unsigned permutation $\phi' = (\phi'_0, \phi'_1, \ldots, \phi'_{2n+1})$ be defined exactly the same way with respect to ϕ. We say two elements π_i and π_{i+1} are *adjacent* in π, and we say the corresponding elements π'_{2i} and π'_{2i+1} are adjacent in π'; similarly for ϕ and ϕ'. Let the *breakpoint graph* $\mathcal{B}$ of π with respect to ϕ be defined as follows (see Figure 1). $\mathcal{B}$ contains a sequence of $2n + 2$ vertices labeled with the elements of π'. Every two of these vertices that reflect an adjacency in π' are connected with a *black* edge (a "reality" edge), and every two that reflect an adjacency in ϕ' are connected with a *gray* edge (a "desire" edge, often depicted as a dashed line). Let the *overlap graph* $\mathcal{O} = (V, E)$ for $\mathcal{B}$ be defined such that there exists a distinct $v_e \in V$ for every gray edge e in $\mathcal{B}$, and two vertices v_e and $v_{e'}$ are connected by an edge ($\{v_e, v_{e'}\} \in E$) iff gray edges e and e' *overlap* in $\mathcal{B}$ (see Figure 1). A *cycle* in $\mathcal{B}$ is a sequence of connected vertices $(v_0, v_1, \ldots, v_{2i}, v_{2i+1}, \ldots, v_{2m}, v_{2m+1}, v_0)$ ($m \geq 0$), such that, for all i, $0 \leq i \leq m$, v_{2i} and v_{2i+1} are connected with a black edge, and v_{2i+1} and v_{2i+2} (or v_{2i+1} and v_0, if $i = m$) are connected with a gray edge. A *connected component* in $\mathcal{O}$ has the usual meaning, and is sometimes called simply a "component."

Every gray edge is said to be *oriented* if it spans an odd number of vertices in $\mathcal{B}$, and *unoriented* otherwise. A cycle in $\mathcal{B}$ and a connected component in $\mathcal{O}$ are each said to be *oriented* if they contain at least one oriented gray edge. We call cycles and components *unoriented* if they are not oriented, except when they are *trivial*. A trivial cycle consists of a single gray edge and a single black edge, and corresponds to an adjacency shared in permutations π and ϕ. A component is trivial if it consists of a single, isolated vertex in $\mathcal{O}$. Note that the gray edges of a cycle always belong to the same connected component, so we can say that the cycle belongs to that component. For convenience, we will refer in this paper to a component that is either oriented or trivial as a *benign* component[1].

Every unoriented component can be classified as either a *hurdle* or a *protected nonhurdle*. A hurdle is an unoriented component that does not *separate* other unoriented components, and a protected nonhurdle is one that does. A component u is said to *separate* two other components v and w if, in a traversal of the vertices of $\mathcal{B}$, it is impossible to pass (in either circular direction) from a vertex belonging to v to a vertex belonging to w without encountering a vertex belonging to u. Note that, while separation is primarily used with respect to unoriented components, the definition applies as well to oriented and trivial ones[2]. A hurdle is called a *superhurdle* if, were it eliminated, a protected nonhurdle would emerge as a hurdle; otherwise it is called a *simple hurdle*.

By Hannenhalli and Pevzner's duality theorem, the distance $d(\pi, \phi)$ between π and ϕ is given by $d(\pi, \phi) = n+1-c+h+f$, where c is the number of cycles and h is the number of hurdles in $\mathcal{B}$. The parameter f is equal to one if there is a *fortress* in $\mathcal{B}$ and zero otherwise. A fortress exists iff there are an odd number of hurdles and all are superhurdles [19].

A reversal $\rho(i, j)$ ($1 \leq i < j \leq n$) applied to $\pi = (\pi_1, \ldots, \pi_n)$ transforms π into $\rho(\pi) = (\pi_1, \ldots, -\pi_j, \ldots, -\pi_i, \ldots, \pi_n)$. A reversal ρ is a *sorting reversal* iff $d(\rho(\pi), \phi) = d(\pi, \phi) - 1$. Note that $d(\rho(\pi), \phi) < d(\pi, \phi)$ iff $d(\rho(\pi), \phi) = d(\pi, \phi) - 1$. We will use the term Δd to indicate the quantity $d(\rho(\pi), \phi) - d(\pi, \phi)$. The end-points i and j of a reversal $\rho(i, j)$ correspond to the ith and $(j+1)$st black edges of $\mathcal{B}$. We say that ρ *acts on* these edges. Note that $\Delta d \in \{-1, 0, 1\}$ for every reversal.

Setubal and Meidanis [19] have presented an alternative distinction to the one between oriented and unoriented gray edges, based on black edges. They define two black edges of

[1]We differ from the literature also in the way we have distinguished and named trivial cycles and components.

[2]The notion of separation was introduced in [19]; our definition is stated slightly more generally.

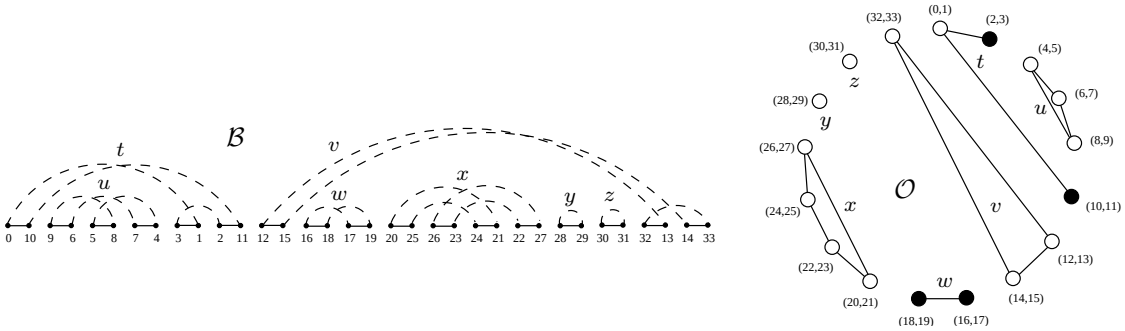

Figure 1: Breakpoint graph $\mathcal{B}$ and overlap graph $\mathcal{O}$ for the permutation $\pi = \{-5, -3, -4, -2, +1, +6, +8, -9, +10, +13, +12, +11, +14, +15, +16, +7\}$ with respect to the identity permutation of size $n = 16$. Connected components t and w are oriented, y and z are trivial, and u, v, and x are unoriented. Unoriented components u and x are hurdles, but unoriented component v is a protected nonhurdle because it separates u and x. Oriented gray edges are represented in $\mathcal{O}$ by solid circles.

the same cycle as *convergent* if in a traversal of the cycle, these edges induce the same (circular) ordering of the vertices of $\mathcal{B}$; otherwise the edges are *divergent*. It can be shown easily that an oriented gray edge always connects divergent black edges and an unoriented gray edge always connects convergent black edges (if the unoriented gray edge is part of a trivial cycle, it connects a black edge to itself, and the correspondence holds trivially). Setubal and Meidanis have shown that any reversal that acts on divergent black edges will split the cycle to which the edges belong, and any reversal that acts on convergent black edges will not split the cycle to which they belong.

When a gray edge has one vertex within and one vertex outside the range of a reversal, we say that the edge is *affected by* the reversal. We say that a component is affected by a reversal when it has at least one gray edge that is affected by the reversal. A reversal causes an edge to change orientation iff it affects it (only when a reversal affects a gray edge can the adjoining black edges change from convergent to divergent or vice versa). As a result, a reversal that affects an unoriented component will cause it to become oriented. This observation gives us a simple explanation for the phenomena Hannenhalli and Pevzner have called "hurdle cutting" and "hurdles merging" [8].

Let M be the set of connected components in $\mathcal{O}$, and let C_i be the set of cycles that belong to $m_i \in M$. Every $c_{i,j} \in C_i$ has a set of black edges $B_{i,j}$. If there exist $b_{i,j,k}, b_{i,j,l} \in B_{i,j}$ such that $b_{i,j,k}$ and $b_{i,j,l}$ are divergent, then $c_{i,j}$ is oriented; otherwise $c_{i,j}$ is unoriented, unless $|B_{i,j}| = 1$, in which case $c_{i,j}$ is trivial. If there exists $c_{i,j} \in C_i$ such that $c_{i,j}$ is oriented, then m_i is oriented; otherwise, m_i is unoriented, unless $m_i = \{c_{i,j}\}$ and $c_{i,j}$ is trivial, in which case m_i also is trivial. If m_i is unoriented, then m_i is either a hurdle or a protected nonhurdle, depending on whether it separates other unoriented components.

3. SORTING REVERSALS IN THE ABSENCE OF FORTRESSES

Based on these definitions, we can describe an exhaustive classification scheme for reversals.

LEMMA 1. *Suppose ρ is an arbitrary reversal, such that ρ acts on black edges $b_{i,j,k}$ and $b_{i',j',k'}$, belonging respectively to cycles $c_{i,j}$ and $c_{i'j'}$ and to connected components m_i and $m_{i'}$. Then one of the following must be true:*

1. *$i = i'$ and $j = j'$ ($b_{i,j,k}$ and $b_{i',j',k'}$ belong to the same cycle, $c_{i,j}$)*
   (a) *$c_{i,j}$ is oriented and m_i is oriented. Here, $b_{i,j,k}$ and $b_{i',j',k'}$ may be convergent or divergent.*
   (b) *$c_{i,j}$ is unoriented and m_i is either oriented or unoriented. Here, $b_{i,j,k}$ and $b_{i',j',k'}$ must be convergent.*
2. *$i = i'$ and $j \neq j'$ ($b_{i,j,k}$ and $b_{i',j',k'}$ belong to different cycles of the same component, m_i). Here, m_i may be oriented or unoriented.*
3. *$i \neq i'$ ($b_{i,j,k}$ and $b_{i',j',k'}$ belong to different components, m_i and $m_{i'}$). Here, each of m_i and $m_{i'}$ may be benign or unoriented.*

PROOF. Either $i = i'$ or $i \neq i'$ (the edges are part of the same or different components); if $i = i'$ then either $j = j'$ or $j \neq j'$ (the edges are part of the same or different cycles); if $i = i'$ and $j = j'$ then $b_{i,j,k}$ and $b_{i',j',k'}$ are either convergent or divergent (the latter only if $c_{i,j}$ is oriented). Each of $c_{i,j}$, $c_{i',j'}$, m_i, and $m_{i'}$ by definition is oriented, unoriented, or trivial. Trivial cycles or components are not possible in cases 1 and 2, and case 1 need not consider the possibility of an oriented cycle belonging to an unoriented component (which is prohibited by definition). □

Let $d'(\pi, \phi) = n + 1 - c + h$ (i.e., $d' = d - f$), and $\Delta d' = d'(\rho(\pi), \phi) - d'(\pi, \phi)$. We denote as the "Fortress-Free Model" (FFM) of ASR a version of the problem in which a *sorting reversal* is redefined to be a reversal that causes $\Delta d' = -1$.

LEMMA 2 (CONSERVATION OF DISTANCE). *Under the FFM, a reversal is a sorting reversal iff: $\Delta c = -1$ and $\Delta h = -2$; or $\Delta c = 0$ and $\Delta h = -1$; or $\Delta c = 1$ and $\Delta h = 0$.*

PROOF. We know that $\Delta c \in \{-1, 0, 1\}$ (because a reversal can only merge cycles, be neutral with respect to cycle number, or split a cycle). By definition, $\Delta d' = -1$ in the case of a sorting reversal. Because $d' = n+1-c+h$, it must be true that $\Delta h - \Delta c = -1$. □

The lemmata below address cases 1a, 1b, 2, and 3 of Lemma 1. They rely heavily on the idea of conservation of distance.

LEMMA 3 (CASE 1A). *Under the FFM, a reversal ρ that acts on two black edges belonging to the same oriented cycle is a sorting reversal iff the edges are divergent and ρ does not increase the number of hurdles.*

PROOF. Either the black edges are divergent and ρ splits the cycle, or the black edges are convergent and ρ is neutral with respect to cycle number [19]. If ρ splits the cycle ($\Delta c = 1$), then it is a sorting reversal iff $\Delta h = 0$ (conservation of distance). If ρ is neutral ($\Delta c = 0$), then it is a sorting reversal iff $\Delta h = -1$ (conservation of distance). But if ρ is neutral, we know that $\Delta h = 0$, because the reversal acts on black edges of an oriented cycle, and therefore of an oriented component. Therefore, to be a sorting reversal, ρ must split the cycle in question and avoid increasing the number of hurdles. □

LEMMA 4 (CASE 1B). *Under the FFM, a reversal ρ that acts on two black edges belonging to the same unoriented cycle c is a sorting reversal iff c belongs to a simple hurdle.*

PROOF. Because c is unoriented, all of its black edges are convergent; therefore ρ must result in $\Delta c = 0$, and to be a sorting reversal, it must cause $\Delta h = -1$. The component m to which c belongs is either oriented, a hurdle, or a protected nonhurdle. If m is oriented, then $\Delta h = 0$, as described in the proof of Lemma 3. If m is a protected nonhurdle, ρ cannot orient a hurdle, and $\Delta h \geq 0$ (orienting a protected nonhurdle cannot decrease the number of hurdles). If m is a hurdle, then ρ cuts m. If m is a superhurdle, however, then when it is cut another hurdle will emerge, and $\Delta h = 0$; thus, ρ will not be a sorting reversal. Only if m is a simple hurdle will $\Delta h = -1$, and only in this case will ρ be a sorting reversal. □

LEMMA 5 (CASE 2). *Under the FFM, a reversal ρ cannot be a sorting reversal if ρ acts on two black edges belonging to different cycles of the same component.*

PROOF. Because ρ acts on different cycles, $\Delta c = -1$ [19]. Therefore, by conservation of distance, ρ is a sorting reversal iff $\Delta h = -2$. It is impossible, however, for ρ to eliminate two hurdles, because it affects only a single component. □

To address Case 3 of Lemma 1, we will introduce two new terms.

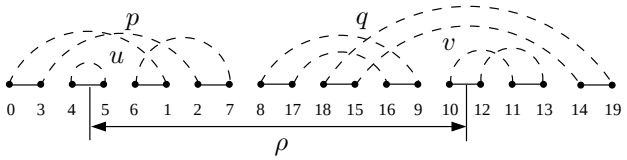

Figure 2: A reversal ρ that acts on black edges belonging to two benign components (u and v) can still cause the destruction of two hurdles (p and q), if the hurdles are distinct separating hurdles of the benign components. The effect is similar to what would be observed if the hurdles were merged.

Definition 1. Suppose in a permutation π there exist four unoriented components u, v, w, and p, such that p is a protected nonhurdle that separates hurdles u and v from w but does not separate u and v from each other (Figure 3). Further suppose that every other unoriented component in π either separates u and v or is separated by p from both u and v, and that p does not separate any two of the components that it separates from u and v. Then we say that hurdles u and v form a ***double superhurdle***.

Definition 2. A hurdle that separates a benign component from all other unoriented components is said to be the **separating hurdle** of the benign component.

It follows from this definition that a benign component may have at most one separating hurdle, but a hurdle may be the separator of multiple benign components.

LEMMA 6 (CASE 3). *Under the FFM, a reversal ρ that acts on black edges belonging to different components u and v is a sorting reversal iff all of the following are true:*

1. *Each of u and v is a hurdle or a benign component that has a separating hurdle.*
2. *u and v are not benign components sharing the same separating hurdle.*
3. *u and v or their separating hurdles do not form a double superhurdle.*

PROOF. As in Lemma 5, $\Delta c = -1$ and ρ is a sorting reversal iff $\Delta h = -2$. If $\Delta h = -2$, then ρ must orient at least two hurdles; and it is impossible for a single reversal to orient more than two hurdles, so ρ must orient exactly two hurdles. To orient two hurdles, ρ must act on two black edges, each of which belongs to a hurdle or to a benign component separated from the other by a hurdle (a protected nonhurdle cannot be separated from a hurdle by a hurdle; otherwise a hurdle would separate unoriented components). Furthermore, if each black edge belongs to a benign component, the two benign components cannot be separated by a single hurdle (if they were, ρ would orient only one hurdle). If a benign component is separated by a hurdle from another hurdle, the former hurdle must be its separating hurdle (another separating hurdle could not exist; if it did a

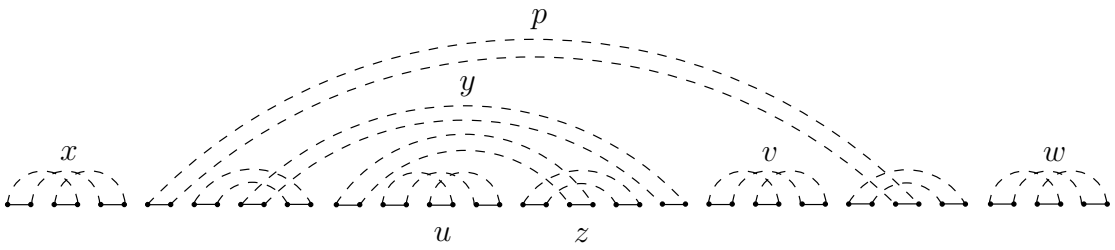

Figure 3: Breakpoint graph in which hurdles u and v form a double superhurdle with respect to protected nonhurdle p. Note that all unoriented components besides u, v, and p either separate u and v (y and z) or are separated from both u and v by p (x and w). We call this construct a "double superhurdle" because a reversal that destroys u and v will cause p to become a hurdle. In this example, x and w also form a double superhurdle with respect to p.

hurdle would separate unoriented components). Thus, each of u and v must be a hurdle or a benign component that has a separating hurdle; and if both are benign components, they cannot share a separating hurdle. For it to be true that $\Delta h = -2$, however, ρ must avoid introducing any new hurdles. We claim that a reversal that destroys two hurdles causes a new hurdle to emerge iff those hurdles form a double superhurdle. It is clear that, if each of u and v is a hurdle or a benign component separated by a hurdle, and they do not share a separating hurdle, then ρ will orient exactly two hurdles; thus, if we prove our claim about double superhurdles, we will have completed both directions of the proof of Lemma 3.

We prove as follows that a reversal ρ that destroys two hurdles u and v will cause a new hurdle to emerge iff u and v form a double superhurdle. It follows directly from the definition of a double superhurdle that, if u and v form a double superhurdle, then a reversal that destroys them must cause a new hurdle to emerge; therefore, we will focus on the converse claim. Suppose ρ causes a protected nonhurdle p to emerge as a hurdle. Then p must separate each member of a nonempty set U of unoriented components from each member of a nonempty set V of unoriented components, and ρ must orient all members of U or all members of V (it cannot orient members of both, because if ρ spanned U and V it would orient p). Assume that ρ orients all members of U. Then $u, v \in U$, and u and v are separated from all members of V. Furthermore, u and v must be separated from one another by all other members of U; otherwise either they could not be hurdles, or a reversal that destroyed them could not also destroy the other members of U. Thus, u and v meet the definition of a double superhurdle. □

4. ACCOMMODATING FORTRESSES

In this section, we adapt what we have established under the FFM to allow for the possibility of a fortress. For economy of space, we leave complete proofs for a supplement to this paper (available at `http://www.cse.ucsc.edu/~acs/pubs.html`). The FFM differs from the general case exactly when $f \neq 0$ (either before or after a candidate reversal). Thus, we have two major cases to consider with respect to a given reversal ρ. The first occurs when a fortress does not exist before the reversal; here it is possible that ρ is a sorting reversal under the FFM, but it introduces a fortress, and thus is not a sorting reversal in the general model. The second case occurs when there exists a fortress; here it is possible that ρ does not sort under the FFM, but it eliminates the fortress, and thus is a sorting reversal in the general model. The first case is addressed by the following lemma.

LEMMA 7. *A reversal ρ that meets the criteria for a sorting reversal under the FFM will introduce a fortress iff one of the following is true:*

1. *ρ acts on divergent black edges of the same oriented cycle, and ρ introduces at least one unoriented component so that there are an odd number of hurdles all of which are superhurdles.*
2. *ρ cuts the only simple hurdle and there are an odd number of superhurdles.*
3. *ρ acts on two black edges belonging to different components such that two hurdles are destroyed, and the set of hurdles is reduced so that there remain an odd number and all are superhurdles.*

The second case is more difficult and requires the introduction of several new concepts. See Figure 4 for an illustration of the following definitions.

Definition 3. We say two unoriented components u and v are **adjacent** in a breakpoint graph $\mathcal{B}$ iff at least one pair of vertices from u and v have between them, in a circular traversal of the vertices of the breakpoint graph, no vertex belonging to another unoriented component.

Definition 4. Let U be the set of unoriented components in a breakpoint graph $\mathcal{B}$. We define the **hurdle graph**[3] for $\mathcal{B}$ to be a graph $\mathcal{H} = (V, E)$ such that $V = U$ and $E = \{\{v_i, v_j\} \mid v_i, v_j \in V$ and v_i, v_j are adjacent in $\mathcal{B}\}$.

[3]The hurdle graph might more precisely be called the "unoriented component graph." We will retain the name originally used in [20], however, because it has a better ring, and because a primary purpose of the graph is to identify various species of hurdles and their relationships.

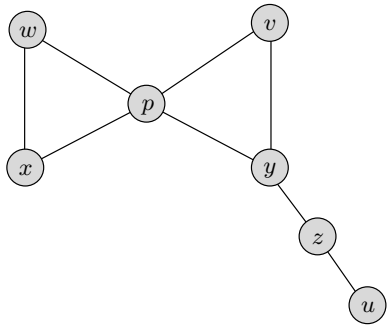

Figure 4: The hurdle graph for the breakpoint graph of Figure 3. Every unoriented component is represented by a vertex, and adjacencies between unoriented components are represented by edges. Notice here that unoriented components y, z, and u form a superhurdle chain for superhurdle u, with y as the anchor.

Definition 5. A **hurdle chain** is a chain of vertices in the hurdle graph that consists of a hurdle and zero or more other vertices, such that every vertex v in the chain is a hurdle, has degree 2 and does not belong to a cycle, or is the last vertex in the chain and either belongs to a cycle or has degree greater than 2. A **superhurdle chain** is a hurdle chain that contains a superhurdle.

If a hurdle chain has one end that is not a hurdle, we call the vertex at that end the *anchor* of the chain.

Definition 6. We say a superhurdle is a **single protector** if it belongs to an anchored hurdle chain of length 2. We call the neighbor of a single protector a **pseudohurdle**.

Simple hurdles, single protectors and other superhurdles, pseudohurdles and other protected nonhurdles, and double superhurdles are all readily identifiable from local properties of vertices in the hurdle graph. Now we can address the second case, above. Note that we return here to the original, general-purpose definition of a *sorting reversal*.

Lemma 8. *If there exists a fortress, then a reversal ρ will eliminate the fortress and be a sorting reversal iff one of the following is true:*

1. *ρ splits a cycle and increases the number of hurdles.*
2. *ρ cuts a single protector or a pseudohurdle.*
3. *ρ acts on edges of two components u and v such that one of u and v is a superhurdle or a benign component that has a separating superhurdle. Let u be this component, and let the hurdle w be either u or u's separating hurdle. Then one of the following is true:*
   (a) *v is the anchor of w's chain.*
   (b) *v is a protected nonhurdle not belonging to w's chain.*
   (c) *v is a benign component that is not separated by a hurdle, and v is not separated by one component in w's chain from another.*
   (d) *v is a superhurdle or a benign component with a separating superhurdle, and v or its separating hurdle forms a double superhurdle with w.*

Lemmata 7 and 8 allow us to generalize Lemmata 3, 4, 5, and 6 to accommodate fortresses.

Theorem 1 (Generalization of Lemma 3). *A reversal ρ that acts on two black edges belonging to the same oriented cycle is a sorting reversal iff the edges are divergent and one of the following is true:*

1. *ρ does not introduce an unoriented component.*
2. *There exists no fortress ($f = 0$) and ρ introduces an unoriented component, but that oriented component does not cause there to be an odd number of hurdles all of which are superhurdles.*
3. *There exists a fortress ($f = 1$) and ρ increases the number of hurdles.*

Theorem 2 (Generalization of Lemma 4). *A reversal ρ that acts on two black edges belonging to the same unoriented cycle c is a sorting reversal iff one of the following is true:*

1. *There exists no fortress ($f = 0$), c belongs to a simple hurdle u, and either u is not the only simple hurdle or the number of superhurdles is even.*
2. *There exists a fortress ($f = 1$) and c belongs to a single protector or a pseudohurdle.*

Theorem 3 (Generalization of Lemma 5). *A reversal ρ cannot be a sorting reversal if ρ acts on two black edges belonging to different cycles of the same component.*

Theorem 4 (Generalization of Lemma 6). *A reversal ρ that acts on black edges belonging to different components u and v is a sorting reversal iff one of the following is true:*

1. *There exists no fortress ($f = 0$) and all of the following are true:*
   (a) *Each of u and v is a hurdle or a benign component that has a separating hurdle.*
   (b) *u and v are not benign components sharing the same separating hurdle.*
   (c) *u and v or their separating hurdles do not form a double superhurdle.*

Input: Two signed permutations of size n, π and ϕ; assume functions `get_revs_split_cycles`, `get_revs_cut_hurdles`, `get_revs_merge_nofort`, and `get_revs_merge_fort` that find all sorting reversals which respectively split cycles, cut hurdles (or pseudohurdles), merge components (when there is no fortress), and merge components (when there is a fortress).

Output: A list L of all sorting reversals of π with respect to ϕ.

begin

Construct the breakpoint graph $\mathcal{B}$ of π with respect to ϕ;

Identify all black edges, cycles, and connected components in $\mathcal{B}$. Let $e_{i,j,k}$ represent the ith black edge belonging to the jth cycle of the kth component;

Define an $o_{i,j,k}$ corresponding to each $e_{i,j,k}$ such that $o_{i,j,k} \in \{-1, +1\}$ and $o_{i_1,j,k} \cdot o_{i_2,j,k} = -1$ iff $e_{i_1,j,k}$ and $e_{i_2,j,k}$ are *divergent*;

Label each component as oriented, trivial, or unoriented;

Build hurdle graph $\mathcal{H}$; use $\mathcal{H}$ to label each unoriented component as a simple hurdle, a single protector, a pseudohurdle, a (non-single-protector) superhurdle, or a (non-pseudohurdle) protected nonhurdle. Also label each member of a double superhurdle with its partner (a single hurdle can have at most two);

Let c be the number of cycles in $\mathcal{B}$, h be the number of hurdles, and s be the number of superhurdles; let $f = 1$ if $h = s$ and s is odd; otherwise let $f = 0$.

Initialize list L;

`append_all(`L, *get_revs_split_cycles()*`)`;

`append_all(`L, *get_revs_cut_hurdles()*`)`;

if $f = 0$ **then** `append_all(`L, *get_revs_merge_nofort()*`)`;
else `append_all(`L, *get_revs_merge_fort()*`)`;

return L;

end

Algorithm 1: find_all_sorting_reversals

(d) The elimination of the hurdles associated with u and v will not leave an odd number of hurdles all of which are superhurdles.

2. There exists a fortress ($f = 1$) and either:

(a) Each of u and v is a superhurdle or a benign component that has a separating superhurdle, and u and v are not benign components sharing the same separating hurdle; or

(b) Case 3 of Lemma 8 applies

5. AN ALGORITHM TO ENUMERATE ALL SORTING REVERSALS

Theorems 1, 2, 3, and 4 lead directly to an algorithm to address *ASR* (Algorithm 1). For clarity of presentation and economy of space, we have described as separate subroutines the steps of the algorithm enumerating sorting reversals that split cycles (`get_revs_split_cycles`; Theorem 1), cut

Input: All $e_{i,j,k}, o_{i,j,k}$, and the values of h, s, and f from `find_all_sorting_reversals`; assume the existence of a function `detect_new_unoriented_components` that returns a list of sets of edges, such that each set represents a new unoriented component introduced by a given reversal ($\emptyset$ is returned if no new unoriented component is introduced).

Output: A list M of all sorting reversals that split cycles

begin

Initialize list M;
foreach $e_{i_1,j,k}$ *and* $e_{i_2,j,k}$ *such that* $o_{i_1,j,k} \cdot o_{i_2,j,k} = -1$ **do**
$P \leftarrow$ `detect_new_unoriented_components(`$\rho(e_{i_1,j,k}, e_{i_2,j,k})$`)`;
if $P = \emptyset$ **then**
/ no new unoriented component */*
`append(`$M, \rho(e_{i_1,j,k}, e_{i_2,j,k})$`)`;
end
else
/ at least one new unoriented component */*
Add components represented by the elements of P to hurdle graph;
Label types of new components, and relabel neighbors in hurdle graph as needed;
Compute new number of hurdles (h') and whether a fortress exists in new permutation (f');
if $h' + f' \leq h + f$ **then**
`append(`$M, \rho(e_{i_1,j,k}, e_{i_2,j,k})$`)`;
end
end
end
return M;

end

Algorithm 2: get_revs_split_cycles

Input: All $e_{i,j,k}, o_{i,j,k}$, and the values of h, s, and f from `find_all_sorting_reversals`

Output: A list M of all sorting reversals that cut hurdles (or pseudohurdles)

begin

Initialize list M;
$H \leftarrow \{k \mid$ component k is a simple hurdle$\}$;
if $f = 1$ **then** $H \leftarrow H \cup \{k \mid$ component k is a pseudohurdle or a single protector$\}$;
if $f = 0$ *and* $s = 2a + 1, a \in \mathcal{Z}^+$ *and* $h = 2a + 2$ **then**
do not cut hurdles; */* avoid a fortress! */*
end
else foreach $e_{i_1,j,k}, e_{i_2,j,k}$ *such that* $k \in H$ *and* $i_1 \neq i_2$ **do**
`append(`$M, \rho(e_{i_1,j,k}, e_{i_2,j,k})$`)`;
end
return M;

end

Algorithm 3: get_revs_cut_hurdles

Input: All $e_{i,j,k}, o_{i,j,k}$, and the values of h and s from `find_all_sorting_reversals`.

Output: A list M of sorting reversals that merge separate components, assuming there is not a fortress

begin
  Initialize list M;
  Find all separating hurdles: let $S_k \in S$ be defined such that $S_k = \{j \mid$ component j is a benign component whose separating hurdle is component $k\}$;
  $H \leftarrow \{k \mid$ component k is a hurdle$\}$;
  foreach $i \in H$ **do**
    foreach $k \in H$ *such that* $k \neq i$ **do**
      /* *avoid a fortress* */
      if ($s = 2a+1, a \in \mathcal{Z}^+$ *and* $h = 2a+3$ *and hurdles* i *and* k *are both simple hurdles*) ***or*** ($s = 2a+2, a \in \mathcal{Z}^+$ *and* $h = 2a+3$ *and one of hurdles* i *and* k *is a superhurdle*) **then**
        Do not merge i and k;
      end
      /* *avoid a double superhurdle* */
      else if *hurdles* i *and* k *form a double superhurdle* **then**
        Do not merge i and k;
      end
      else
        foreach $j \in \{i\} \cup S_i$ **do**
          foreach $e_{x_1,y_1,z_1}, e_{x_2,y_2,z_2}$ *such that* $z_1 = j$ *and* $z_2 \in \{k\} \cup S_k$ **do**
            `append`$(M, \rho(e_{x_1,y_1,z_1}, e_{x_2,y_2,z_2}))$;
          end
        end
      end
    end
  end
  return M;
end

Algorithm 4: get_revs_merge_nofort

Input: All $e_{i,j,k}, o_{i,j,k}$, and the values of h and s from `find_all_sorting_reversals`.

Output: A list M of sorting reversals that merge separate components, assuming there is a fortress

begin
  Initialize list M;
  Find all separating hurdles: let $S_k \in S$ be defined such that $S_k = \{j \mid$ component j is a benign component whose separating hurdle is component $k\}$, and let $S_\neg \in S$ be defined such that $S_\neg = \{j \mid$ component j is a benign component that has no separating hurdle$\}$;
  $H \leftarrow \{k \mid$ component k is a hurdle$\}$;
  $U \leftarrow \{k \mid$ component k is an unoriented component$\}$;
  foreach $k \in H$ **do**
    foreach $j \in U$ *such that* j *belongs to hurdle chain* k **do** $\gamma_j \leftarrow k$;
    if j *is the anchor of the chain* **then** $\alpha_k \leftarrow j$;
  end
  foreach $j \in U$ *such that* j *belongs to no hurdle chain* **do** $\gamma_j \leftarrow -1$;
  foreach $i \in H$ **do**
    $V \leftarrow \{k \mid k \in U$ and $k \notin H$ and $\gamma_k \neq i\}$;
    $P \leftarrow \{k \mid k$ is a double superhurdle partner of $i\}$;
    $W \leftarrow \{k \mid (k \in H$ and $k \neq i)$ or $(k \in S_j$ such that $j \in H$ and $j \neq i)\}$;
    $S'_\neg \leftarrow \{k \mid k \in S_\neg$ and k is not separated by one component of chain γ_i from another$\}$;
    foreach $j \in \{i\} \cup S_i$ **do**
      foreach $e_{x_1,y_1,z_1}, e_{x_2,y_2,z_2}$ *such that* $z_1 = j$ *and* $z_2 \in \{\alpha_i\} \cup V \cup P \cup W \cup S'_\neg$ **do**
        `append`$(M, \rho(e_{x_1,y_1,z_1}, e_{x_2,y_2,z_2}))$;
      end
    end
  end
  return M;
end

Algorithm 5: get_revs_merge_fort

hurdles or pseudohurdles (`get_revs_cut_hurdles`; Theorem 2), and merge components (`get_revs_merge_nofort` and `get_revs_merge_fort`; Theorem 4). Note that Theorem 3 is addressed implicitly, by the absence of a corresponding step. The correctness of Algorithm 1 follows from Lemma 1 and Theorems 1, 2, 3, and 4.

One step in the routine to enumerate sorting reversals that split cycles (`get_revs_split_cycles`) is particularly important: the step that detects whether a reversal that splits a cycle introduces new unoriented components. This step is relatively expensive, and turns out to be a bottleneck for the algorithm, because most sorting reversals are of the class that splits cycles (the other classes are only relevant when hurdles are present, and we know hurdles to be rare), and it must be performed for every such reversal. We refer to a routine to execute this step as `detect_new_unoriented_components`, and abbreviate it as `detect`. Our efforts to improve the efficiency of `detect` are outside the scope of this paper; we will report, however, that we implemented two versions of the routine and compared their performance. The first version simply re-runs the `connected_components` algorithm of [1] (on a portion of the breakpoint graph) and tests whether more unoriented components are present after a reversal than before; the second is a novel algorithm that extends ideas presented in [3] and [4] for modeling changes to the overlap graph induced by a reversal with bit-vector operations. This latter "bitwise" version of `detect` is fully described in [20].

Algorithm 1 takes $O(n^3)$ or $O(n^4)$ time, depending on whether the connected-components ($O(n)$) or bitwise ($O(n^2)$) version of `detect` is used. While all sorting reversals can be enumerated trivially in $\Theta(n^3)$ time, we will see in the next section that, in practice, Algorithm 1 offers a considerable improvement in performance.

6. EXPERIMENTAL METHODS AND RESULTS

We implemented Algorithm 1 in C and tested it for correctness and speed. Our implementation, program `find-all-sr`, allows an implementation of either `detect` algorithm to be selected at compile time. The program comprises about 1600 lines of code.

Test data fell into three classes. The first class consisted of pairs of random signed permutations, such that one member of each pair had been "scrambled" with respect to the other by a specified number of random reversals. The second

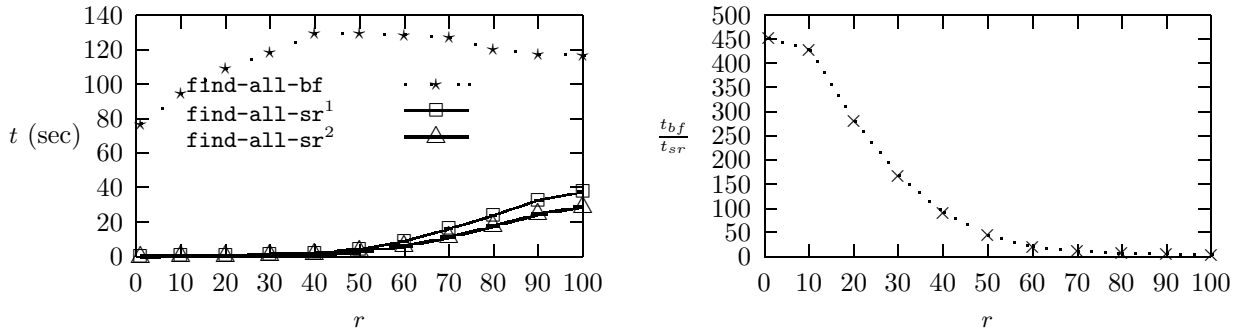

Figure 5: (a) Running times of programs `find-all-bf` and `find-all-sr` for $n = 100$ and various values of r. Plotted are total times required to process 500 pairs of permutations. Times are plotted for `find-all-sr` using both the connected-components (`find-all-sr`^{1}) and bitwise (`find-all-sr`^{2}) versions of `detect`. (b) Speed-up of `find-all-sr`^{2} with respect to `find-all-bf` for the same experiment.

class was similar to the first except that permutations were scrambled not with random reversals but with a procedure designed to introduce unoriented components (to stress the parts of the algorithm that are exercised only when multiple unoriented components are present). The third class consisted of hand-picked pairs of permutations representing special cases unlikely to appear in the other two classes (configurations involving fortresses, long hurdle chains, double superhurdles, and the like). A total of several thousand pairs of permutations were produced. Correctness testing was performed by comparing the output of program `find-all-sr` with that of a control called program `find-all-bf`. The latter program finds all sorting reversals of one permutation with respect to another by brute force—that is, by considering all $\binom{n+1}{2}$ "neighbors" of one permutation and computing the reversal distance of each neighbor from the other permutation. Program `find-all-bf` directly uses the well-tested code for reversal distance by Bader *et al.* [1], and because it is also very simple, is believed to be highly reliable. Program `find-all-sr` was not confirmed to be correct until on all test cases it produced identical results to program `find-all-bf`. Note that the method of program `find-all-bf` is the only reasonable alternative known to us for solving *ASR*. Performance testing focused on two types of comparisons: the performance of `find-all-sr` versus that of `find-all-bf`, and the performance of `find-all-sr` when using the connected-components version of the `detect` algorithm versus that when using the bitwise version. All testing was performed on a SONY laptop with a 700 MHz Pentium III processor and 128 MB of RAM, running the Linux operating system (RedHat 7.0). The most extensive testing was performed using test data of the first class, as the presence of multiple unoriented components did not appear to change performance significantly. We ran tests for values of n between 25 and 100 and numbers of random reversals (a parameter we call r) between 0% and 100% of n.

Figure 5 shows results for $n = 100$ and $0 < r \leq 100$, which are typical of what we observed. Plots are shown for `find-all-bf` and both versions of `find-all-sr`. As would be expected the brute force algorithm shows approximately constant performance[4] for all values of r, and the implementations of Algorithm 1 perform significantly better[5] at small r than at large r. Note that Algorithm 1 still performs approximately three times as fast as the brute force approach even when $r = n$, when it should be closest to its worst-case performance. Note also that the bitwise implementation of `detect` outperforms the connected-components implementation consistently, for all values of r. Figure 5 plots the speed-up of `find-all-sr` with respect to `find-all-bf`. In this experiment, a speed-up of over 400 times is achieved for $r \leq 10$. These results are particularly encouraging because small values of r are often of greatest interest when solving higher-level genome rearrangement problems with real biological data [20]. Figure 6 shows the number of sorting reversals for $0 < r \leq 100$. When r is close to n, as one might expect, a significant fraction of all possible reversals are sorting reversals; but even when $r = 0.5n$, hundreds of sorting reversals are possible.

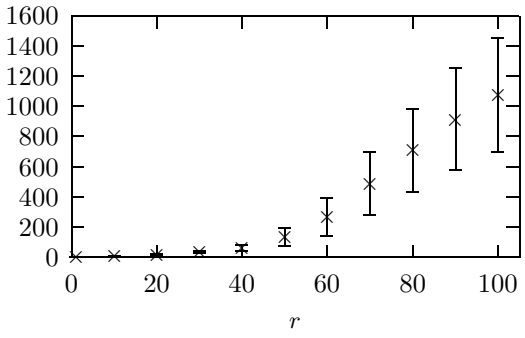

Figure 6: Average number of sorting reversals for the experiment shown in Figure 5. Error bars indicate one standard deviation.

[4]The slightly better performance seen at small r is probably the result of improvements in the speed of distance calculations due to the presence of numerous trivial components.

[5]As r increases, so does the average number of pairs of divergent black edges of the same cycle, the dominant factor in the running time of the algorithm

7. SUMMARY

We have shown that it is possible, although not straightforward, to enumerate all sorting reversals of one signed permutation with respect to another, using the principles of the Hannenhalli-Pevzner cycle decomposition theory. Our algorithm can be used to enumerate all minimum sequences of sorting reversals, although in practice, that number is likely to be very large. The algorithm can be of great use, however, for other types of exploration of the space of genome rearrangements, as we show in [20].

8. ACKNOWLEDGEMENTS

Thanks to Bernard Moret, the author's adviser at the University of New Mexico, for encouragement and support, and to the anonymous reviewers of this paper for corrections and helpful suggestions.

9. REFERENCES

[1] D. Bader, B. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. In F. Dehne, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures: Seventh International Workshop, WADS 2001, Brown University, Providence, RI, August 8-10, 2001, Proceedings*, volume 2125 of *Lecture Notes in Computer Science*, pages 365–376. Springer, 2001.

[2] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS93)*, pages 148–157. IEEE Press, 1993.

[3] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In A. Amir and G. M. Landau, editors, *Combinatorial Pattern Matching, 12th Annual Symposium, CPM 2001 Jerusalem, Israel, July 1-4, 2001 Proceedings*, volume 2089 of *Lecture Notes in Computer Science*, pages 106–117. Springer, 2001.

[4] A. Bergeron and F. Strasbourg. Experiments in computing sequences of reversals. In O. Gascuel and B. M. E. Moret, editors, *Algorithms in Bioinformatics, First International Workshop, WABI 2001, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2149 of *Lecture Notes in Computer Science*, pages 164–174. Springer, 2001.

[5] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:GC11–GC17, 1996.

[6] A. Caprara. Sorting by reversals is difficult. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*, pages 75–83, 1997.

[7] D. Goldberg, S. McCouch, and J. Kleinberg. Algorithms for constructing comparative maps. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics*. Kluwer Academic Press, 2000.

[8] S. Hannenhalli and P. 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.

[9] H. Kaplan, R. Shamir, and R. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal of Computing*, 29(3):880–892, 1999.

[10] A. McLysaght, C. Seoighe, and K. Wolfe. High frequency of inversions during eukaryote gene order evolution. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics*, pages 47–58. Kluwer Academic Press, 2000.

[11] B. Moret, L.-S. Wang, T. Warnow, and S. Wyman. New approaches for reconstructing phylogenies from gene-order data. *Bioinformatics*, 17:S165–S173, 2001. Presented at the Ninth International Conference on Intelligent Systems for Molecular Biology, ISMB-2001.

[12] J. Nadeau and B. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences*, 81:814–818, 1984.

[13] P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000.

[14] D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570, 1998.

[15] D. Sankoff, D. Bryant, M. Deneault, B. Lang, and G. Burger. Early eukaryote evolution based on mitochondrial gene order breakpoints. *Journal of Computational Biology*, 7(3/4):521–535, 2000.

[16] D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, Y. Xu, and M. Zhang, editors, *Topics in Computational Biology*. MIT Press, 2001.

[17] D. Sankoff, V. Ferretti, and J. Nadeau. Conserved segment identification. *Journal of Computational Biology*, 4(4):559–565, 1997.

[18] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, and B. Lang. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences*, 89:6575–6579, 1992.

[19] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, 1997.

[20] A. Siepel. Exact algorithms for the reversal median problem. Master's thesis, University of New Mexico, Albuquerque, New Mexico, December 2001. Available at `http://www.cse.ucsc.edu/~acs/masters-thesis.html`.

[21] D. Waddington. Estimating the number of conserved segments between species using a chromosome-based model. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics*. Kluwer Academic Press, 2000.