Algorithms for Molecular Biology **Fall Semester, 2000**

Lecture 10: December 25, 2000

Lecturer: Ron Shamir and Itsik Pe'er *Scribe: Yaniv Nahum and Sonny Ben-Shimon*

10.1 Genome Rearrangements

10.1.1 Preface

The genome undergoes rearrangements, or what seem to be a general scrambling of the order of the genome, during the evolution. This phenomenon has been a well known fact for over 80 years.

In the salivary glands of *Drosophila*, a phenomenon of chromosomes doubling in thickness during mitosis has been noticed. This appears to be two *homologs* (identical copies of a chromosome segment created during cell division) that have glued together somehow.

The chromosomes have an observable pattern of bands perpendicular to their length, which was studied since the 1920's. This pattern is a characteristic of a species. However, at times one can find two individuals of the species who show different patterns of these bands; usually the differences appear to be segment reversals along the pattern of bands.

10.1.2 Operations on Chromosomes

What kind of genome rearrangement events (also called *operations*) take place?

1. Operations on a single chromosome:

   - Deletions (a certain part is lost, for example $abc \rightarrow ac$)
   - Insertions (a part is added, for example $ac \rightarrow abc$)
   - Duplications (can be tandem, for example $abc \rightarrow abbc$, or not, for example $abc \rightarrow abcb$)
   - Reversals, or inversions (a part is turned around, head to tail, for example $abc_1c_2c_3c_4de \rightarrow abc_4c_3c_2c_1de$) - see figure 10.1.
   - Transpositions (two parts change places, for example $abcd \rightarrow acbd$) - this operation is believed to be the most rare since it requires 3 points of leverage along the choromosome.

How do these operations take place?

If two regions along a chromosome are very similar, they might hybridize just like two different strands of the double helix. Once they are attached, a loop forms. This loop might be discarded (deletion), or its direction might switch (inversion).

2. Operations on two chromosomes:

   - Translocation: two chromosomes swap their "tails". It is important to note that not all translocations are possible. A chromosome contains a part called a *centromere* which is crucial to cell division; the centromere usually lies somewhere in the middle of the chromosome, and if upon translocation it will be lost from one of the chromosomes, the cell will surely die.
   - Fusion: two chromosomes merge.
   - Fission: one chromosome splits up into two chromosomes.
     It is not known what exactly happens to the centromere in these cases.

10.1.3 Why Study Genome Rearrangements?

- Genome rearrangements are useful in studying evolution. Since the operations described above are by far more rare than point mutations, one can track the genome rearrangements through the evolutionary history of the species much further back than regular mutations allow.
- There is a very small chance that reverse mutations will affect the exact same location on the genome, so we have less ambiguity in interpreting the mutations.
- Since the rearrangements affect whole chromosomes, this is a larger scale of data which is more appropriate for studying evolution of species. For example, one can take man and mouse. There are about 80 million years of evolutionary distance between them, but only about 140-150 operations of rearrangments.

10.2 Unsigned Permutations

We will assume that we are able to identify genes along the chromosome, and we will discuss a single chromosome. We will also assume that all the genes are different. The order of the genes, which might be different in different taxa, is a permutation of these genes. Thus we will be discussing sequences of unsigned, different integers, where each permutation $\pi = (\pi_1 \dots \pi_n)$ represents a different order of genes. We write this sequence horizontally, using the terms $left$ and $right$ to denote directions along it.

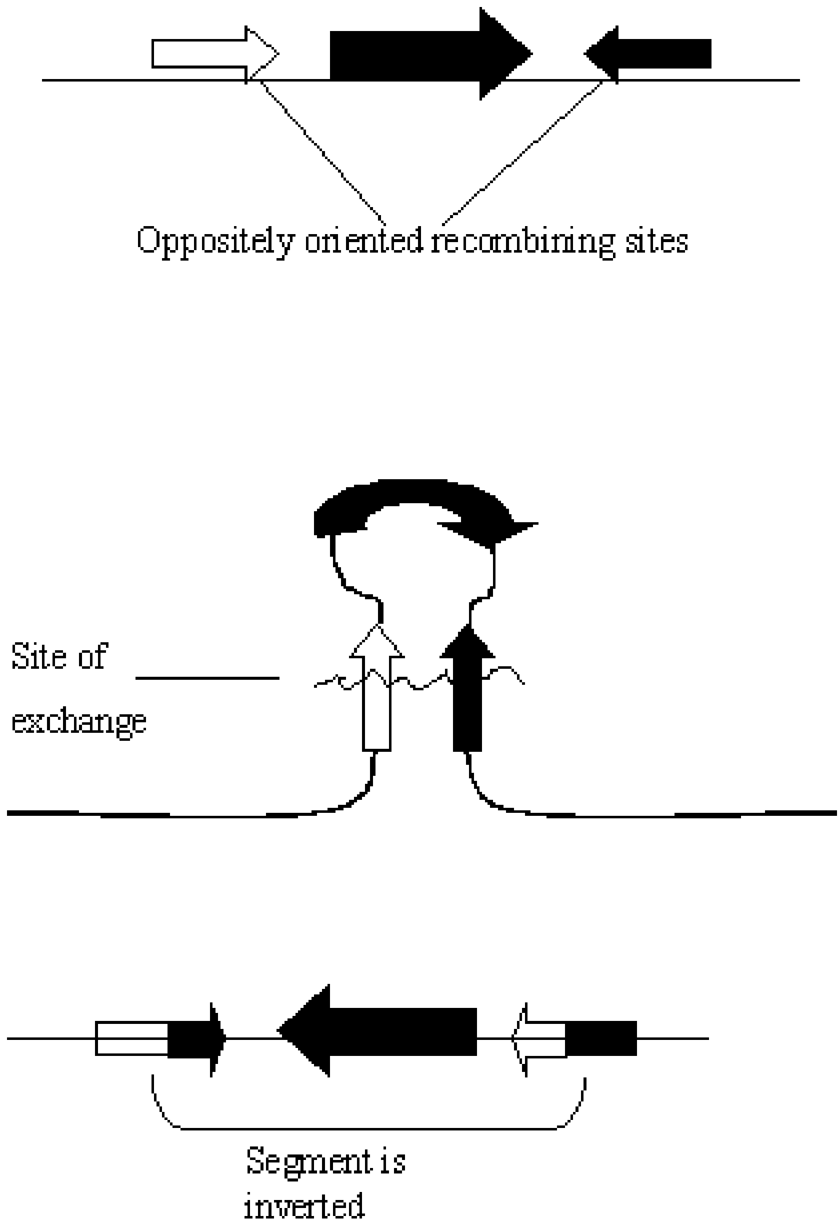

Figure 10.1: Illustration of a reversal operation on a DNA strand

Definition A *reversal* transformation on a sequence is the operation of taking a subsequence and reversing it, for example 12345 $\rightarrow$ 14325.

Definition The *reversal distance* between two sequences is the minimum number of reversals needed to transform one of them into the other (see figure 10.2).

$$\boxed{\begin{aligned} \pi_1 &= (1,\underline{2,3,4},5,6) \\ \pi_2 &= (1,4,\underline{3,2,5,6}) \\ \pi_3 &= (\underline{1,4,6},5,2,3) \\ \pi_4 &= (6,4,1,5,2,3) \end{aligned}}$$

Figure 10.2: Example of reversals; the underlined segments show where the reversals took place. The reversal distance between π_1 and π_3 is 3.

Problem 10.1 Sorting by reversals.
INPUT: A permutation π.
QUESTION: Find $d(\pi)$, the reversal distance between π and the identity permutation (id).

This problem has been investigated in the last few years with the following results:

1. 2-approximation algorithm [16]
2. 1.75-approximation algorithm [2]
3. NP Completeness proof [6]
4. 1.5-approximation algorithm [7]
5. 1.375-approximation algorithm [11]

Definition A *breakpoint* is any place in the sequence where two adjacent numbers are not consecutive ($|\pi_i - \pi_{i+1}| \neq 1$) (For example, in the sequence 123654 there is a breakpoint between the 3 and the 6).

We denote the number of breakpoints in π by $b(\pi)$. When performing a reversal, transforming π into π', we denote $b(\pi') - b(\pi)$ by Δb.

Theorem 10.1 *[16]*

$$\tfrac{b(\pi)}{2} \leq \lceil \tfrac{b(\pi)}{2} \rceil \leq d(\pi) \leq n$$

Proof The lower bound holds since a reversal can fix up at most two breakpoints, and $d(\pi)$ is an integer. On the other hand, the upper bound follows, since it will take us at most n reversals to create any sequence. ∎

Definition A *strip* is a maximal subsequence without breakpoints. For example, in the sequence 0 7 6 4 1 9 8 2 3 5 10 , "7 6" is a strip. A strip can be either *increasing* or *decreasing*; in the above example the strip "2 3" is increasing, whereas the strip "7 6" is decreasing.

Lemma 10.2 *If $\pi \neq id$ contains a decreasing strip, there is a reversal that decreases $b(\pi)$ by k, $k \geq 1$. Such a reversal is called a good reversal.*

Proof

1. Find the decreasing strip with the minimal number, let K be this number. K will be at the right end of the strip.

2. Find $(K-1)$ in π; it will have to be in an increasing strip of length 1 or more, and therefore will also be at its right end.

3. Reverse the entire sequence between these two numbers, so that K and $(K-1)$ will be adjacent. Having joined these two numbers, a breakpoint is reduced (see figure 10.3).

■

$\leftarrow 7\ 6\ 5\ 4 \ldots\ldots 2\ 3 \rightarrow \Longrightarrow \leftarrow 7\ 6\ 5\ 4\ 3\ 2 \ldots$
OR:
$2\ 3 \rightarrow \ldots\ldots \leftarrow 7\ 6\ 5\ 4 \Longrightarrow 2\ 3\ 4\ 5\ 6\ 7 \rightarrow \ldots$

Figure 10.3: Two possible cases to reduce a breakpoint using a decreasing strip ($K = 4$).

Lemma 10.2 gives rise to the following algorithm:
If there exists a decreasing strip, find and perform a good reversal ($\Delta b = -1$). Else reverse an increasing strip, thus creating a decreasing strip ($\Delta b = 0$).

This algorithm leads to performance of at most 4 times the optimum, since there are at most $2b(\pi)$ reversals.

Lemma 10.3 *[16] If there exists a decreasing strip and every reversal that removes a breakpoint results in a permutation without any decreasing strip, then there exists a reversal that removes 2 breakpoints.*

Proof Let $\pi = \pi_1 \ldots \pi_n$ be a permutation such that every reversal which removes a breakpoint results in a permutation without a decreasing strip. We use the following notation:
π_i - the smallest element in a decreasing strip
π_j - the greatest element in a decreasing strip

$(\pi_i - 1)$ has to be to the left of π_i, otherwise we can reverse the strip that includes $(\pi_i - 1)$, thus removing a breakpoint and still maintaining a decreasing strip - the one that includes π_i (see figure 10.4, top). Similarly, $(\pi_j + 1)$ has to be to the right of π_j (see figure 10.4, bottom).

$$\boxed{\begin{array}{l} \leftarrow \pi_i \ldots\ldots (\pi_i - 1) \rightarrow \\ (\pi_j + 1) \rightarrow \ldots\ldots \leftarrow \pi_j \end{array}}$$

Figure 10.4: Two impossible scenarios.

Consider the interval between π_j and $(\pi_j + 1)$ along π, calling it ρ_j (including π_j but not including $(\pi_j + 1)$) ; and the interval between $(\pi_i - 1)$ and π_i, calling it ρ_i (including π_i but not including $(\pi_i - 1)$) (see figure 10.5).

ρ_j and ρ_i must overlap, otherwise we can reverse just one of them, leaving a decreasing strip in the other. Similarly, neither ρ_j, nor ρ_i contains the other, nor can π_j be to the left of $(\pi_i - 1)$.

The only remaining case is (see figure 10.6):

$$(\pi_j + 1) \notin \rho_i \qquad \pi_j \in \rho_i \tag{10.1}$$
$$(\pi_i - 1) \notin \rho_j \qquad \pi_i \in \rho_j \tag{10.2}$$

If $\rho_i \setminus \rho_j$ contains a decreasing strip, then reversing the entire ρ_j interval leaves us a decreasing strip (and removes a breakpoint). Furthermore, if $\rho_i \setminus \rho_j$ contains an increasing strip, then reversing the entire ρ_i interval leaves us a decreasing strip (and one breakpoint less). Hence, $\rho_i \setminus \rho_j = \emptyset$.

Similarly, $\rho_j \setminus \rho_i = \emptyset$, implying that $\rho_j = \rho_i$. Reversing $\rho_j = \rho_i$ is therefore a reversal that removes two breakpoints.

■

Lemma 10.3 gives rise to the following algorithm:

For as long as possible, either:

1. Perform a good reversal using a decreasing strip, resulting in a permutation with a decreasing strip ($\Delta b = -1$).

   Or, if no such reversal exists:

2. Perform a reversal with $\Delta b = -2$, and then reverse any strip.

   This algorithm leads to performance of at most twice the optimum, since $\Delta b = -1$ on the average.

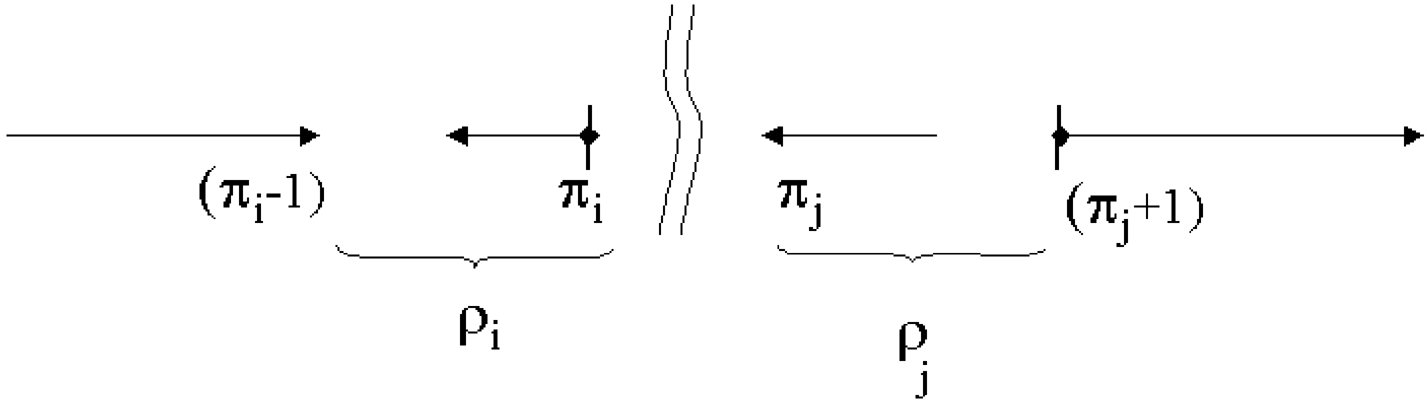

Figure 10.5: A situation where the two strips do not overlap.

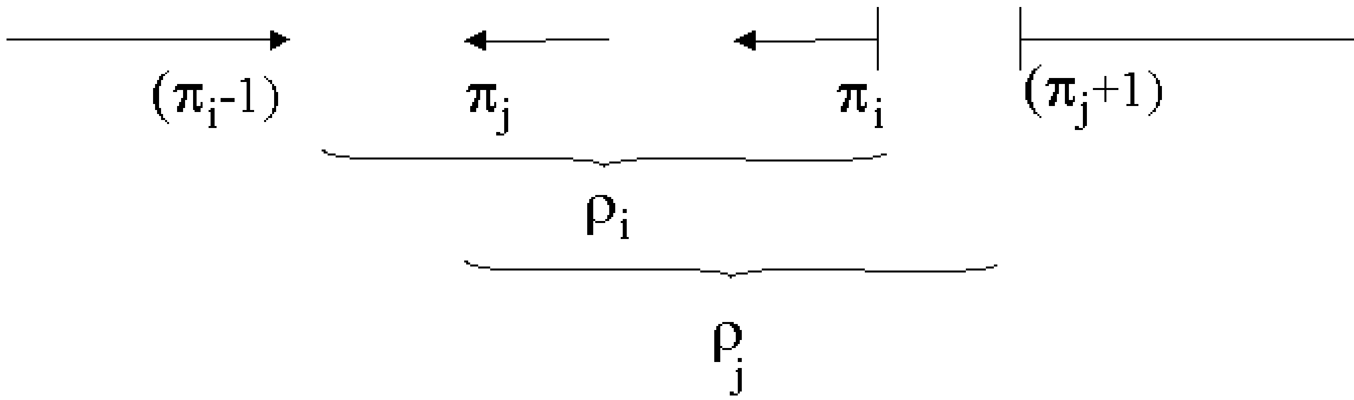

Figure 10.6: The remaining case where the two strips overlap.

10.3 Examples of Genome Rearrangements

A few years ago, the genome of yeast has been fully mapped and sequenced. An interesting fact that was discovered is that almost every DNA subsequence happens to have a twin subsequence almost identical to it in the genome. This appears to be due to a doubling of the entire genome at one point during the course of evolution, and since that doubling, various genome rearrangements took place, mixing the genome into the shape we know today.

A comparison of the DNA of mice and men shows that any specific mouse chromosome contains various parts that can be found in different human chromosomes. The explanation for this is also genome rearrangements, that took place both in the mouse genome and in human genome, ever since the two split apart in the evolutionary tree, some 80 million years ago (see figure 10.7).

A comparison of human X-chromosome to cow and mouse X-chromosomes is also shown. Sites which are conserved between the species are shown (see figure 10.8). Note that since the X chromosome contains only genes that appear either once or twice in a cell, its overall content is rather conserved among mammals.

10.4 An Algorithm for Sorting Signed Permutations

10.4.1 Introduction

We shall introduce the problem of sorting signed permutations by reversals. A *signed permutation* is a permutation $\pi = (\pi_1, \ldots, \pi_n)$ on the integers $\{1, \ldots, n\}$, where each number is also assigned a sign of plus or minus. A *reversal,* $\rho(i,j)$ on π transforms π into

$$\pi' = \pi \cdot \rho(i,j) = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n).$$

This conforms with the usual definition of the product (i.e., composition) between permutations, defining $\rho(i,j) = (1, 2, \ldots, i-1, -j, -(j-1), \ldots, -i, j+1. \ldots, n)$. As in the case of unsigned permutations, the minimum number of reversals needed to transform one permutation to another is called the *reversal distance* between them. The problem of *sorting signed permutations by reversals* is defined as follows:

Problem 10.2
INPUT: A signed permutation π.
QUESTION: What is the reversal distance between π and the signed identity permutation $(+1, +2, \ldots, +n)$?

Our motivation for studying this problem comes from genome comparison problems. Due to the fast progress in the Human Genome Project, genetic and DNA data is accumulating

Total Homologies: 4963
Total mapped in both species: 3383

mouse, laboratory

human	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	X	Y	XY	UN	MT
1	103	1	91	104	2	1		6			2	1	7	1				1					115	
2	78	46			7	29					9	17	1				10	1					63	
3	1		19	1		29		1	62		1			12		32	1						59	
4			28		62	4		21					2				1		1				33	
5	1										44		55		13		4	35					41	
6	5			6	1		1		8	30			24				114						46	
7		1			72	70	1				14	14	6				1						41	
8	6		8	10	1			26						21	33	3							30	
9		45		54	1								11						12				36	
10		12				3	12	1		13			1	17		1		3	49				25	
11		33		1			83		51		1		1						48				73	
12		1	1		23	68				41					47								58	
13	3		2		6			14						24									19	
14			1				1		1	1		52		29									41	
15		30					33		37		1		1										26	
16		1					23	58			8					17	20						41	
17		1	1			1		1			202	2		1									66	
18	3				2			1								1	4	34					13	
19						1	103	31	13	29		1					17	1					67	
20		80																					22	
21										24						32	6						18	
22	1				6	3		3		13	9				39	34							24	
X																				194		1	37	
Y																					6			
XY																								
UN	11	25	11	15	7	5	14	5	9	17	13	5	9	1	6	4	10	5	3	4			407	
MT																								37

Figure 10.7: Man-mouse pairs of homologous genes were examined for their chromosomal location. The (i, j) entry in the table registers the number of such pairs mapped to human chromosome i and mouse chromosome j [21].

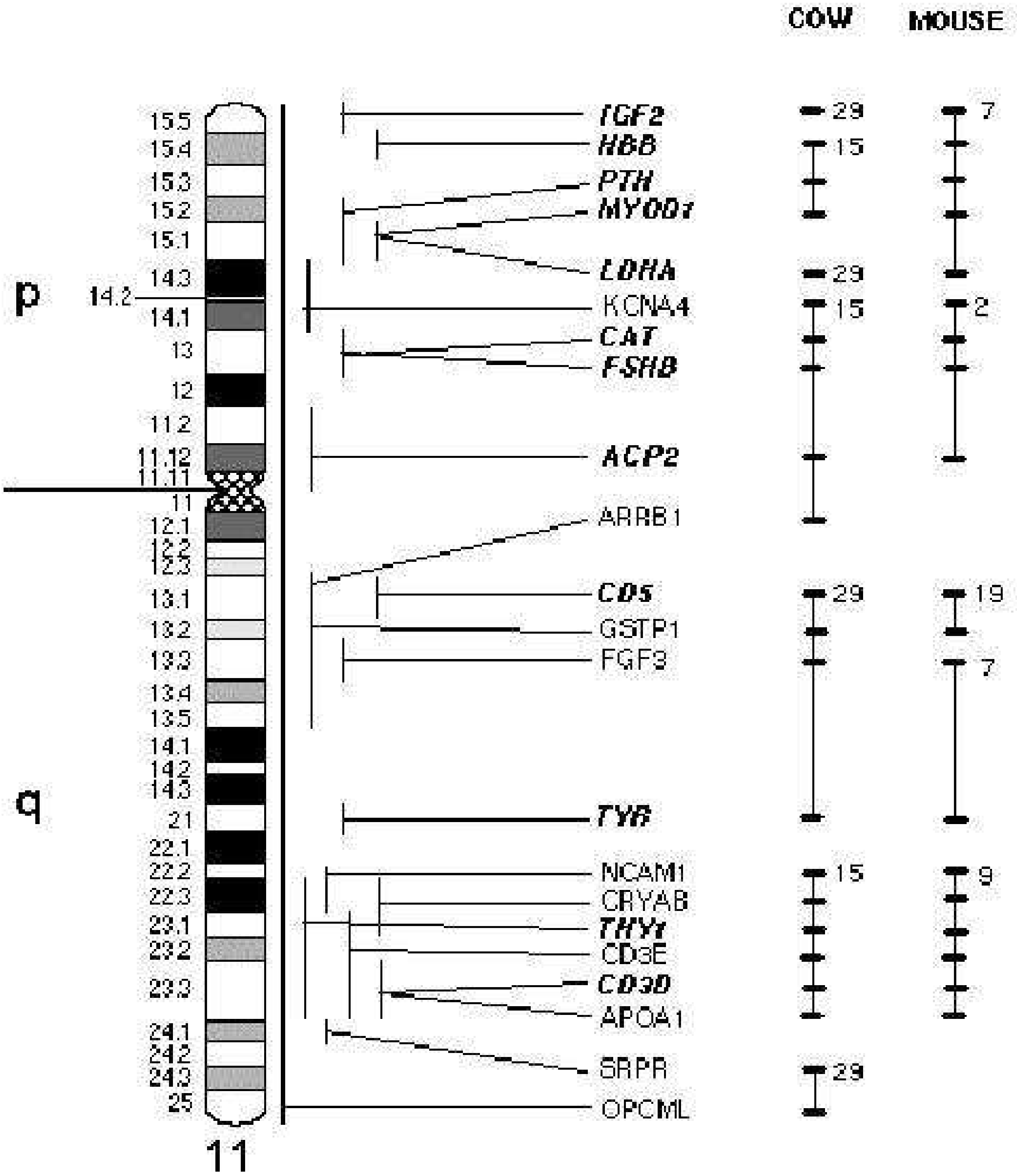

Figure 10.8: Comparison of cow and mouse to human X chromosome [4].

rapidly, and consequently the ability to compare genomes of different species has grown dramatically. One of the most promising ways of checking large scale similarity between genomes is to compare the order of appearance of identical genes in the two species. Dobzhansky and Sturtevant have shown in 1938 [8] evidence of inversions in chromosomes of *Drosophila*. In the 1980's, Palmer [17, 18, 19, 20, 13] has demonstrated that different species may have essentially the same genes, but the gene order may differ between species.

A mathematical description of this problem suggests that genes along a chromosome can be thought of as points along a line. Numbers identify the particular genes, and as genes have directionality, denoted by signs corresponding to their orientation. The difference in order between genomes can be explained by some reversals between them. These reversals correspond to evolutionary changes along the history between the two genomes, so the number of reversals represents the evolutionary distance between the species. Hence, given two such permutations, their reversal distance measures their evolutionary distance.

Studies of problem 10.2 resulted in a 1.5 polynomial approximation algorithm [16]. Unlike problem 10.1, in 1995, Hannenhalli and Pevzner [12] have shown that the problem of sorting a *signed* permutation by reversals is polynomial, and can be done in $O(n^4)$ time. More recently, Berman and Hannenhalli [3] described a faster implementation that finds a minimum sequence of reversals in $O(n^2\alpha(n))$ time, where α is the inverse Ackerman's function [1].

In this lecture we present a $O(n^2)$ algorithm for sorting a signed permutation of n elements, thereby improving upon the previous bound. In fact, if the reversal distance is r, our algorithm requires $O(n \cdot r + n\alpha(n))$ time [15].

10.4.2 Group Theory Viewpoint

From a group theory point of view, the sorting of signed permutations can be viewed as follows: Consider S_n, the symmetric group (group of all permutations) on n elements. The set $\{\rho(i, j)\}$ of all possible reversals is a set of generators of S_n, Therefore, from the group theory point of view, problem 10.2 is a special case of the following general problem:

Problem 10.3
INPUT: Two permutations $\pi_1, \pi_2 \in S_n$, and a set $\{g_1, \ldots, g_k\}$ of generators.
QUESTION: What is their *distance*, i.e.,what is the shortest product of generators that transforms π_1 into π_2?

Even and Goldreich showed that this problem is NP-Hard [9]. Jerrum generalized this result by proving it PSPACE-complete [14].

Problem 10.4
INPUT: A set $\{g_1, \ldots, g_k\}$ of generators.
QUESTION: What is the *diameter* of S_n, where the *diameter* is the longest distance between two permutations?

Gates and Papadimitriou have shown [10] that by using only *prefix reversals* as generators, the diameter can be bounded by $\frac{17}{16}n \leq \textit{diameter} \leq \frac{5}{3}n + \frac{5}{3}$.

10.4.3 Definitions

Let $\pi = (\pi_1, \ldots, \pi_n)$ denote a permutation of $\{1, \ldots, n\}$. Augment π to a permutation on $n+2$ vertices by adding $\pi_0 = 0$ and $\pi_{n+1} = n+1$ to it. A pair (π_i, π_{i+1}), $0 \leq i \leq n$ is called a *gap*. Gaps are classified into two types: A gap (π_i, π_{i+1}) is called a *breakpoint* of π if $|\pi_i - \pi_{i+1}| > 1$; otherwise, it is called an *adjacency* of π. We denote by $b(\pi)$ the number of breakpoints in π.

Recall from section 10.4.1 that a *reversal*, $\rho(i,j)$, on a permutation π transforms π to

$$\pi' = \pi \cdot \rho(i,j) = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$$

We say that a reversal $\rho(i,j)$ *acts on* the gaps (π_{i-1}, π_i) and (π_j, π_{j+1}).

10.4.4 The Breakpoint Graph

Definition The *breakpoint graph* $B(\pi)$ of a permutation $\pi = (\pi_1, \ldots, \pi_n)$ is an edge colored graph on $n+2$ vertices $\{\pi_0, \pi_1, \ldots, \pi_n + 1\} = \{0, 1, \ldots, n+1\}$. We join vertices π_i and π_j by a *black edge* if (π_i, π_j) is a breakpoint in π and by a *gray edge* if (i,j) is a breakpoint in π^{-1}.

Let us now define a one-to-one mapping u from the set of signed permutations of order n into the set of unsigned permutations of order $2n$. Let π be a signed permutation. To obtain $u(\pi)$ replace each positive element x in π by $2x-1$ and $2x$, and each negative element $-x$ by $2x$ and $2x-1$. For any signed permutation π, let $B(\pi) = B(u(\pi))$. This description of $B(\pi)$ is equivalent to the following description: given a permutation $\pi = (\pi_1, \ldots, \pi_n)$, obtain a graph of $2n+2$ vertices by replacing each positive element x in π by $2x-1$ and $2x$, each negative element $-x$ by $2x$ and $2x-1$, and augment with begin and end vertices, 0 and $2n+1$. Black edges connect vertices π_{2i}, π_{2i+1} and gray edges connect vertices $2i$ and $2i+1$.

We may now limit our discussion to signed permutations given that the reversals we perform on our unsigned permutation have a one-to-one correspondance to reversals in signed permutations.

Definition A reversal $\rho(i,j)$ such that i is odd and j even is called an *even reversal*. An even reversal $\rho(2i+1, 2j)$ on $u(\pi)$ mimics the reversal $\rho(i+1, j)$ on π.

Thus, sorting π by reversals is equivalent to sorting the unsigned permutation $u(\pi)$ by even reversals. From now on we will consider the latter problem and by reversals we will always mean an even reversal.

Note that in $B(\pi)$ every vertex has exactly one black edge and one gray edge incident on it. Therefore, there is a unique decomposition of $B(\pi)$ into cycles. The edges of each cycle are alternating gray and black. Our goal is to sort our given graph $B(\pi)$ into $n+1$ trivial cycles.
Notice that in this formulation all reversals are one of three types:

- A reversal can act on two cycles, joining them. We call this move a *bad* move.
- It can act on one cycle, changing it. We call this move a *profitless* move.
- It can act on one cycle, splitting it. We call this a *good* move.

Let $b(\pi) = b(u(\pi))$ and let $c(\pi)$ be the number of cycles in $B(\pi)$.

Figure 10.10(a) shows the breakpoint graph of the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. It has eight breakpoints and decomposes into two alternating cycles, i.e. $b(\pi) = 8$, and $c(\pi) = 2$. The two cycles are shown in figure 10.10(b).

An alternative construction of the breakpoint graph constructs $B'(\pi)$, with vertices $0, \ldots, 2n+1$, black edges (π_{2i}, π_{2i+1}), and grey edges $(2i, 2i+1)$ for all $i = 0, \ldots, n$. All vertices of $B'(\pi)$ are in disjoint cycles, but we ignore trivial cycles, i.e. cycles of length 2. Figure 10.11(a) shows the breakpoint graph of $\bar{\pi} = (4, -3, 1, 2, 5, 7, 6)$, $B'(\pi)$ that has seven breakpoints and decomposes into two cycles (the trivial cycle that is composed of the vertices {2,3} is ignored). Let $c'(\pi)$ denote the number of cycles in $B'(\pi)$.

For an arbitrary reversal ρ on a permutation π, define $\Delta b(\pi, \rho) = b(\pi, \rho) - b(\pi)$ and $\Delta c'(\pi, \rho) = c'(\pi, \rho) - c'(\pi)$. When the reversal ρ and the permutation π will be clear from the context we will abbreviate $\Delta b(\pi, \rho)$ to Δb and $\Delta c'(\pi, \rho)$ to $\Delta c'$. As Bafna and Pevzner [2] observed, the following values are taken by Δb and $\Delta c'$ depending on the types of the gaps $\rho(i, j)$ acts on (see figure 10.9):

1. Two adjacencies: $\Delta c' = 1$ and $\Delta b = 2$.
2. A breakpoint and an adjacency: $\Delta c' = 0$ and $\Delta b = 1$.
3. Two breakpoints each belonging to a different cycle: $\Delta b = 0$, $\Delta c' = -1$.
4. Two breakpoints of the same cycle C:
   a. (π_i, π_{j+1}) and (π_{i-1}, π_j) are gray edges: $\Delta c' = -1$, $\Delta b = -2$.
   b. Exactly one of (π_i, π_{j+1}) and (π_{i-1}, π_j) is a gray edge: $\Delta c' = 0$, $\Delta b = -1$.
   c. Neither (π_i, π_{j+1}) nor (π_{i-1}, π_j) is a gray edge, and when breaking C at i and j vertices $i-1$ and $j+1$ end up in the same path: $\Delta b = 0$, $\Delta c' = 0$.
   d. Neither (π_i, π_{j+1}) nor (π_{i-1}, π_j) is a gray edge, and when breaking C at i and j vertices $i-1$ and $j+1$ end up in different paths: $\Delta b = 0$, $\Delta c' = 1$.

Theorem 10.4 *[2] $d(\pi) \geq (n+1) - c(\pi)$. Alternatively, using the breakpoint graph $B'(\pi)$ it may be phrased as $d(\pi) \geq b(\pi) - c'(\pi)$.*

Proof The number of cycles in $B(\pi)$ increases by at most 1 for every reversal. Alternatively, the identity permutation has no breakpoints and no non-trivial cycles, thus $b(id)-c'(id)=0$. We have seen that every reversal changes $\Delta b - \Delta c'$ by at most 1, therefore we need at least $b(\pi)-c'(\pi)$ reversals to sort π.

■

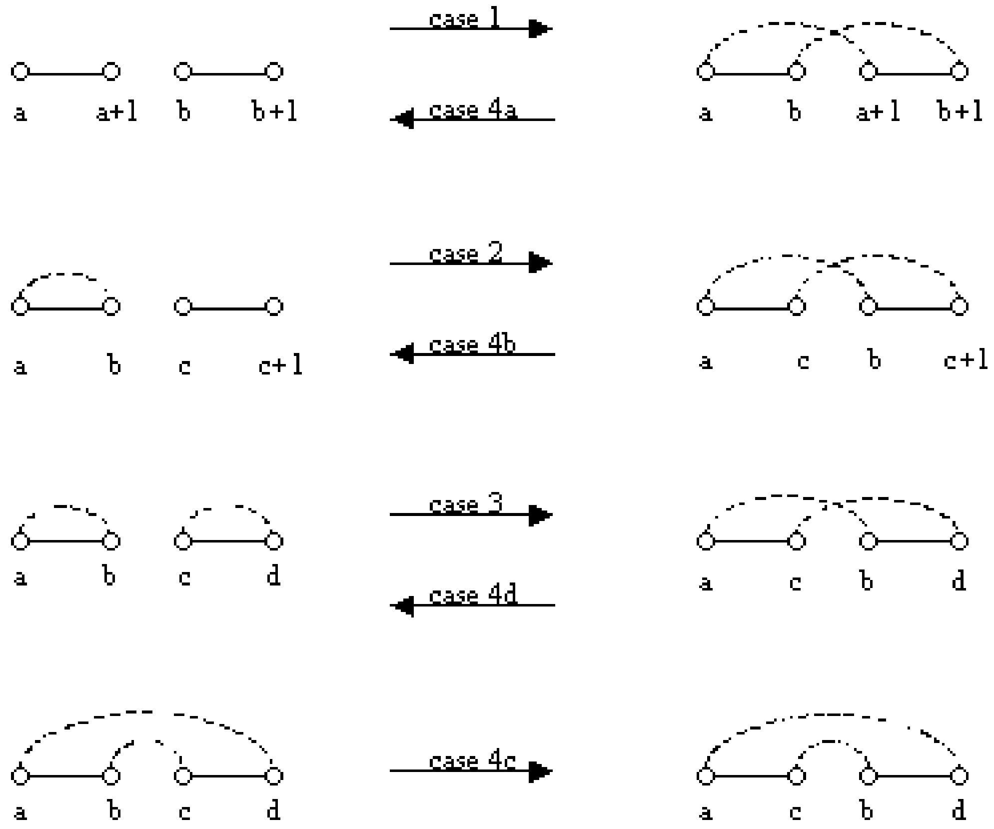

Figure 10.9: All possible cases of changes to Δb and $\Delta c'$ by applying a reversal (see section 10.4.4).

We call a reversal *proper* if $\Delta b - \Delta c' = -1$, i.e. it is either of type 4a, 4b, or 4d. We say that a reversal ρ *acts on* a gray edge e if it acts on the breakpoints which correspond to the black edges incident on e. A gray edge is *oriented* if a reversal acting on it is proper, otherwise it is *unoriented*. Notice that a gray edge (π_k, π_l) is oriented if and only if $k+l$ is even. For example, the gray edge $(0,1)$ in the graph of figure 10.10(a) is unoriented, while the gray edge $(7,6)$ is oriented.

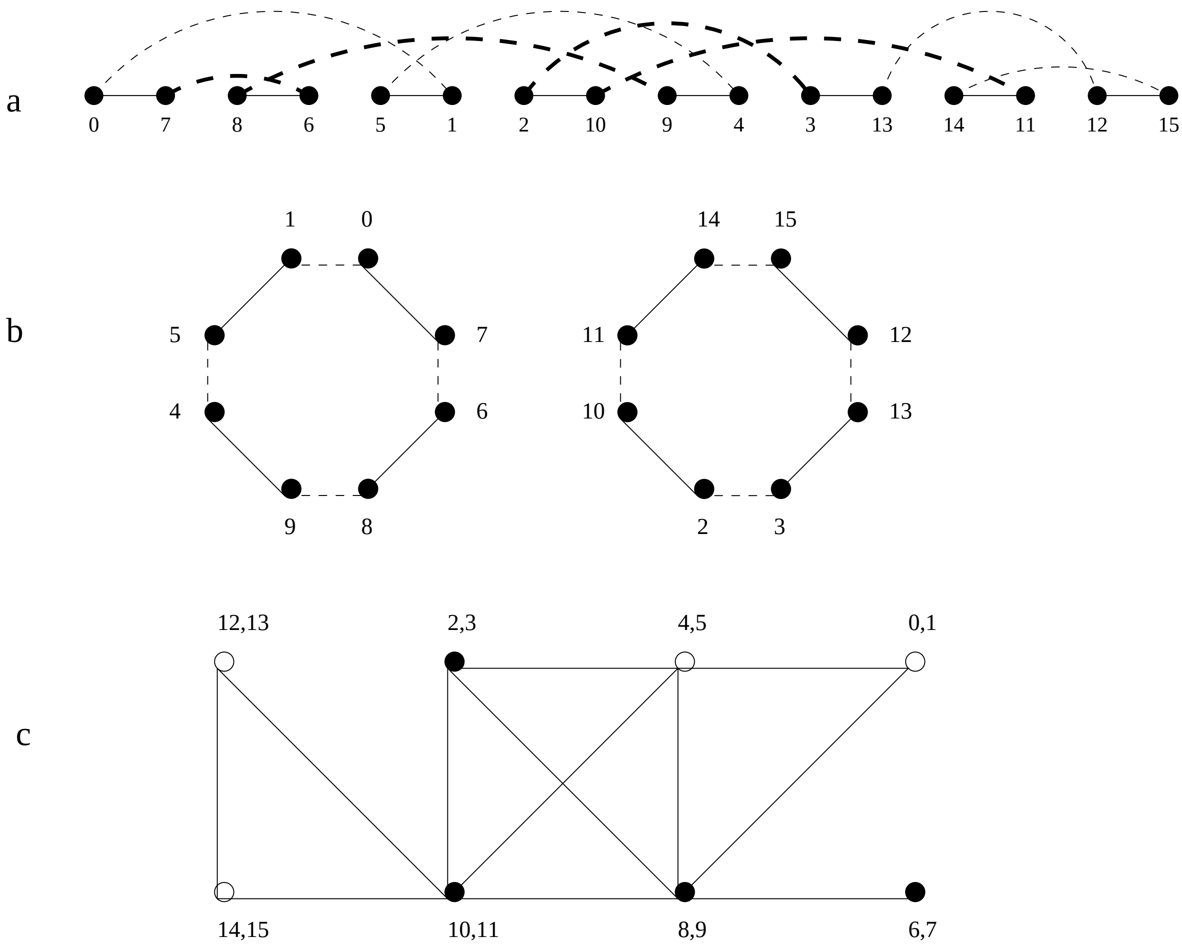

Figure 10.10: (a) The breakpoint graph, $B(\pi)$, of the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. Black edges are solid; gray edges are dashed; oriented edges are bold. (b) $B(\pi)$ decomposes into two disjoint alternating cycles. We disregard whether grey edges are oriented or not. (c) The overlap graph, $OV(\pi)$. Black vertices correspond to oriented edges.

10.4.5 The Overlap Graph

Definition Two intervals on the real line *overlap* if their intersection is nonempty but neither one of them properly contains the other.

Definition An *interval overlap graph* is a graph $G(N, A)$, for which there is an assignment of an interval to each node in N, with an arc in A between two nodes iff their corresponding intervals overlap.

Definition The *overlap graph* of a permutation π, denoted by $OV(\pi)$, is the interval overlap graph of the gray edges of $B'(\pi)$, where we associate with each gray edge (π_i, π_j) the interval $[i, j]$. (We use the terms *vertex* and *edge* for $B'(\pi)$, reserving the names *node* and *arc* for $OV(\pi)$).

In other words, the node set of $OV(\pi)$ is the set of gray edges in $B(\pi)$, and two nodes are connected by an arc if the intervals associated with their gray edges overlap. We shall identify a node in $OV(\pi)$ with the edge it represents and with its interval in the representation. Thus, the endpoints of a gray edge are actually the endpoints of the interval representing the corresponding node in $OV(\pi)$. A connected component of $OV(\pi)$ that contains an oriented edge is called an *oriented component*, otherwise, it is called an *unoriented component*. Figure 10.10(c) shows the interval overlap graph for $\pi = (4, -3, 1, -5, -2, 7, 6)$. It has only one oriented component. Figure 10.11(b) shows the overlap graph of the permutation $\pi' = (4, -3, 1, 2, 5, 7, 6)$, which has two connected components, one oriented and the other unoriented.

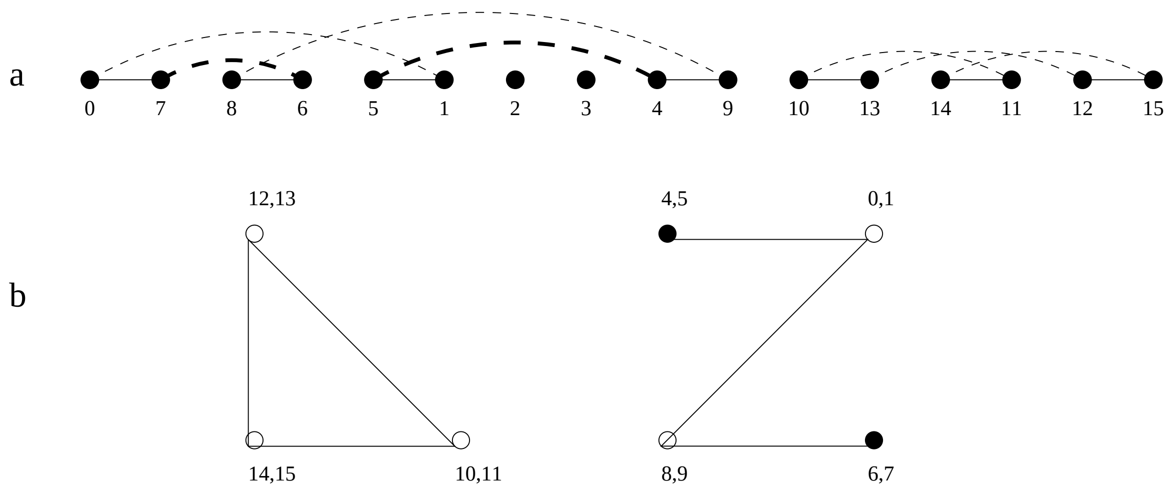

Figure 10.11: (a) The breakpoint graph, $B'(\pi')$ of $\pi' = (4, -3, 1, 2, 5, 7, 6)$. π' was obtained from π of figure 10.10 by the reversal $\rho(7, 10)$; or, equivalently, by the reversal defined by the gray edge $(2, 3)$. (b) The overlap graph of π'.

Lemma 10.5 *The reversal acting on a gray edge flips the orientation of all edges overlapping it, leaving all other edges unchanged.*

Lemma 10.6 *A reversal acting on a gray edge is good iff the edge is oriented.*

We shall see that any connected component which is oriented can be transformed by a series of reversals to a set of trivial connected components that correspond to the identity permutation. The unoriented connected components pose us a problem since we cannot split any of their cycles, nor delete any of their breakpoints by applying a single reversal.

In some cases we can eliminate unoriented components. This is done either by applying a reversal that does not increase the number of cycles, but rather transforms some of the edges to oriented edges, or by applying a reversal that merges two or more unoriented connected components into one oriented component.

The above idea for eliminating unoriented components allows a characterization of the unoriented components, on which we have to spend an extra reversal operation. We denote these components as *hurdles*.

10.4.6 Hurdles

Let $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ be the subsequence of $0, \pi_1, \ldots, \pi_n, n+1$ consisting of those elements incident to gray edges that occur in unoriented components of $OV(\pi)$. Order $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ on a circle CR such that π_{i_j} follows $\pi_{i_{j-1}}$ for $2 \leq j \leq k$ and π_{i_1} follows π_{i_k}. Let M be an unoriented connected component in $OV(\pi)$. Let $E(M) \subset \{\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}\}$ be the set of endpoints of the edges in M. An unoriented component M is a *hurdle* if the elements of $E(M)$ occur consecutively along CR. Let $h(\pi)$ denote the number of hurdles in a permutation π.

Lemma 10.7 *Unoriented connected components cannot be* resolved *(transformed into the identity permutation) only by good moves.*

Therefore, from lemma 10.7 we see that hurdles are unoriented connected components that cannot be solved by good moves. We can still make either a profitless move on a hurdle, that can possibly change some unoriented edges into oriented ones, or make a bad move, joining cycles from different hurdles, thus merging them and flipping the orientation of many edges and components on the way.

For some hurdles, called *superhurdles*, there exist another unoriented component, which upon deletion of the superhurdle by a profitless move, becomes a hurdle itself. Furthermore, note that when merging two hurdles which are not consecutive along CR, no new hurdles are formed. Therefore, if we denote the number of hurdles in $B(\pi)$ by $h(\pi)$, and denote the change in $h(\pi)$ by Δh when a reversal is applied, we conclude that:

Theorem 10.8 *Unless π has exactly three hurdles, all of which are superhurdles, there exist a reversal for which $\Delta b - \Delta c + \Delta h = -1$. Thus the minimum number of reversals required to sort π is $b(\pi) - c(\pi) + h(\pi)$.*

Proof A hurdle is destroyed by a profitless move, or at most two are destroyed (merged) by a bad move. In either cases at least one reversal is required to eliminate each hurdle. The argument above shows that it is possible to do so without generating any new hurdles along the way. ■

The situation described in theorem 10.8 is bound to occur sometime during the sorting process if π has an odd number of hurdles, all of which are superhurdles. We call π a *fortress* in such a case, and write $f(\pi) = 1$; otherwise we write $f(\pi) = 0$. Note that in this case, one extra reversal is required to sort π.

Theorem 10.9 *[12]. If π is a signed permutation, then $d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi)$.*

10.4.7 A New Proof

We will show that for the case of $h = 0$ and $d = b - c$, there is an $O(n^2)$ algorithm for sorting signed permutations by reversals. Our key idea is to prove (constructively) that the following condition is fulfilled for every step of the algorithm:

Condition 10.10
There exists a reversal r, such that $b(\pi r) - c(\pi r) = b(\pi) - c(\pi) - 1$, and the overlap graph of πr does not contain unoriented components.

A node in the overlap graph, i.e., a gray edge e in the breakpoint graph, *defines* the reversal acting on the two black edges adjacent to e. Thus the effect of a reversal r on the overlap graph is as follows:

- Delete the node n that corresponds to the edge defining r.
- Complement the subgraph induced by n's neighbors, switching oriented edges by unoriented ones, and vice versa.

The choice of reversals needs to be a good one, e.g., one that maintains condition 10.10. We must therefore make sure no unoriented components are generated when applying the reversals. Such reversals are called *safe*.

10.4.8 Happy Cliques

Definition Let $G(N, A)$ be an interval overlap graph. A *happy clique* $C \subseteq N$ is a clique of oriented nodes s.t $\forall$ *orineted* $y \notin C$, if $(x, y) \in A$ and $x \in C$, then $\exists$ *oriented* $z \notin C$ where $(z, y) \in A$ and $(z, x) \notin A$

For example, in the overlap graph shown in figure 10.10(c) $\{(2,3),(10,11)\}$ and $\{(6,7)\}$ are happy cliques, but $\{(2,3),(10,11),(8,9)\}$ is not.

Claim 10.11 *The reversal defined by a node x with maximum unoriented degree (maximum number of unoriented neighbors) in a happy clique C creates no new unoriented components.*

Proof Suppose that such a reversal created an unoriented component M. Consider $OV(\pi)$ before applying the reversal.

- M contains a neighbor y of x:
  Suppose otherwise. We know that M is unoriented, therefore neighborhood relationships and orientation in M are unchanged, thus M must have been unoriented **before** the reversal, and this contradicts to the happy clique definition.

- M contains no neighbor of x outside C. Therefore $y \in C$:
  Suppose to the contrary that there exist $e \in M \setminus C$ such that (e, x) is an arc in $OV(\pi)$). There are two cases to examine: Either e was unoriented before applying the reversal r, hence e is oriented and so is M - a contradiction. Otherwise, e was oriented, and by the definition of the happy clique C, e has an oriented neighbor g, unadjacent to x. Therefore $g \in M$, and its orientation remains unchanged by applying r, thus M is oriented - a contradiction.

- Every unoriented neighbor of x is adjacent to y:
  Suppose to the contrary that z is an unoriented neighbor of x, unadjacent to y. Then after applying r, z is oriented, and adjacent to y, hence $z \in M$, contradicting M being unoriented.

- $|M| > 1$:
  Every unoriented edge π_{2i}, π_{2j-1} has a neighbor. Otherwise, suppose $i < j$ and π_{2i} is odd (the other cases are analogous). Then $\pi_{2i} + 2$ appears between π_{2i} and π_{2j-1}, and so is $\pi_{2i} + 2k$ for all k, by induction - a contradiction. Therefore, y has, after applying r, an unoriented neighbor z. Then $z \notin C$ and z is not adjacent to x. Then y has more unoriented neighbors then x, a contradiction to the choice of x.

■

Claim 10.11 implies, for example, that the reversal defined by the gray edge $(10, 11)$ is a safe proper reversal for the permutation of figure 10.10 (a), since it corresponds to the node with maximum unoriented degree in the happy clique $\{(2, 3), (10, 11)\}$. On the other hand, the reversal defined by $(2, 3)$ creates a new unoriented component, as it yields the permutation shown in figure 10.11.

10.4.9 Implicit Representation of the Overlap Graph

Note that an explicit representation of the overlap graph uses $\Theta(n^2)$ space, and since the neighborhood of a node may be of size $\Omega(n)$ nodes (and $\Omega(n^2)$ arcs), it seems we need to perform $\Omega(n^2)$ steps per reversal, finally reaching a time bound of $\Omega(n^3)$.

We shall therefore use an *implicit* representation of the overlap graph, constructed as follows: We assume that the input is given as a sequence of n signed integers representing π^0. Initially the permutation $\pi = u(\pi^0)$ is constructed as described in Section 10.4.4 and stored in an array. The array holds n intervals and $2n$ endpoints, thus it is linear in size. We also construct an array representing π^{-1}. It is straightforward to verify that with these two arrays we can determine, in constant time, for each element in π whether it is a left or a right endpoint of a gray edge. In case the element is an endpoint of a gray edge we can also find the other endpoint and check whether the edge is oriented in constant time. Finding whether two edges overlap is also trivial in constant time.

Thus the arrays π and π^{-1} comprise a representation of $OV(\pi)$. Our algorithm will maintain these two arrays while carrying out the reversals that it finds. The time to update the arrays is proportional to the length of the interval being reversed, which is $O(n)$.

It is easy to produce a list of the intervals in the representation of $OV(\pi)$ sorted by either left or right endpoint from the arrays π and π^{-1}. It is also possible to maintain them without increasing the asymptotic time bound of the algorithm. In practice it may be faster to maintain such lists instead of, or in addition to π and π^{-1}.

10.4.10 Finding a Happy Clique

Theorem 10.12 *The oriented neighborhood of every oriented node contains a happy clique.*

Let $n_1, \ldots, n_k$ be the oriented nodes in $OV(\pi)$ in increasing left endpoint order (we ignore unoriented nodes at this stage). To locate a happy clique in $OV(\pi)$, the algorithm traverses the oriented nodes in $OV(\pi)$ according to this order. Let $L(e)$ and $R(e)$ be the left and right endpoints, respectively, of the interval corresponding to a node e in the realization of $OV(\pi)$. After traversing $n_1, \ldots, n_i$ for $1 \le i \le k$, the algorithm maintains a happy clique C_i in the subgraph of $OV(\pi)$ induced by these vertices. Assume $|C_i| = j$, $j \le i$ and let $n_{i_1}, \ldots, n_{i_j}$ be the vertices in C_i where $i_1 < i_2 < \ldots < i_j$. The vertices of C_i are maintained in a linked list ordered in increasing left endpoint order. If there exists an interval that contains all the intervals in C_i then the algorithm maintains a minimal such interval t_i. The clique C_i and the node t_i (if exists) satisfy the following invariant:

Invariant 10.13
1) Every node $n_l \notin C_i$, $l \leq i$, such that $L(n_{i_1}) < L(n_l)$ must be adjacent to t_i.
2) Every node $n_l \notin C_i$, $L(n_l) < L(n_{i_1})$ that is adjacent to a node in C_i is either adjacent to an interval v_p such that $R(n_p) < L(n_{i_1})$ or adjacent to t_i.

We prove the correctness of this invariant by induction: Initially $C_1 = \{n_1\}$ and t_1 is undefined. If $R(n_{i_j}) < L(n_{i+1})$ then C_i is guaranteed to be happy in $OV(\pi)$ (see figure 10.12(a)), therefore we need to focus only on cases with $L(n_{i+1}) \leq R(n_{i_j})$. The induction step: We assume correctness up until i and show how to obtain C_{i+1} and t_{i+1} if $L(n_{i+1}) \leq R(n_{i_j})$. We have to consider the following cases:
Case 1. The interval t_i is defined and $R(t_i) < R(n_{i+1})$. Continue with $C_{i+1} = C_i$ and $t_{i+1} = t_i$. See figure 10.12(b).
Case 2. The interval t_i is not defined or $R(n_{i+1}) \leq R(t_i)$.
a) $R(n_{i_j}) < R(n_{i+1})$ and $L(n_{i+1}) \leq R(n_{i_1})$. C_{i+1} is obtained by adding n_{i+1} to C_i and $t_{i+1} = t_i$. See figure 10.12(c).
b) $R(n_{i_j}) < R(n_{i+1})$ and $L(n_{i+1}) > R(n_{i_1})$. The clique C_{i+1} consists of v_{i+1} alone and $t_{i+1} = t_i$. See figure 10.12(d).
c) $R(n_{i+1}) < R(n_{i_j})$. As in the previous case $C_{i+1} = \{n_{i+1}\}$. In this case t_{i+1} is set to n_{i_j}, the last interval in C_i. See figure 10.12(e).

The fact that C_i is happy in the subgraph induced by $n_1, \ldots, n_i$ follows from this invariant. It is straightforward to see that the clique C_l that the algorithm stops with, is happy.

The running time of the algorithm is proportional to the number of oriented nodes traversed since a constant amount of work is performed for each oriented node it encounters.

10.4.11 Computing the Unoriented Degrees

After locating a happy clique C in $OV(\pi)$ we need to search it for a node with a maximum number of unoriented neighbors. In this section we give an algorithm that performs this task.

Let $e_1, \ldots, e_j$ be the intervals in C ordered in increasing left endpoint order. Clearly, $L(1) < L(2) < \ldots < L(j) < R(1) < R(2) < \ldots < R(j)$. Thus the endpoints of the j vertices in C partition the line into $2j + 1$ disjoint intervals $I_0, \ldots, I_{2j}$, where $I_0 = (-\infty, L(1)]$, $I_l = (L(l), L(l+1)]$ for $1 \leq l < j$, $I_j = (L(j), R(1)]$, $I_l = (R(l-j), R(l-j+1)]$ for $j < l < 2j$ and $I_{2j} = (R(j), \infty)$. The algorithm consists of the following three stages.
Stage 1: Let e be an unoriented node that has a non-empty intersection with the interval $[L(1), R(j)]$. Mark each of e's endpoints with the index of the interval that contains it.
Stage 2: Let o be an array of j counters, each corresponding to a node in C. The intention

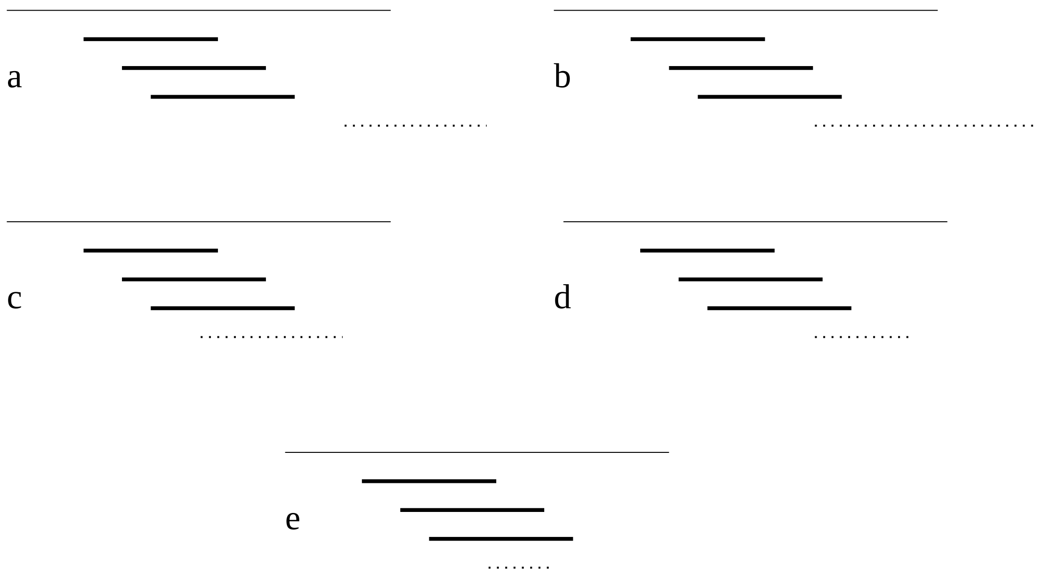

Figure 10.12: The various cases of the algorithm for finding a happy clique. The topmost interval is always t_i. The three thick intervals comprise C_i. The dotted interval corresponds to n_{i+1}.

is to assign values to o such that the sum $\sum_{i=1}^{l} o[i]$ is the unoriented degree of the node $e_l \in C$. The counters are initialized to zero. For each unoriented node e that overlaps with the interval $[L(1), R(j)]$ we change at most four of the counters as follows. Let I_l and I_r be the intervals in which $L(e)$ and $R(e)$ occur, respectively. We may assume $l < r$ as otherwise e is not adjacent to any node in C and we can ignore it. We continue according to one of the following cases.
Case 1: $r \leq j$. All the vertices from e_{l+1} to e_r are adjacent to e: we increment $o[l+1]$ and decrement $o[r+1]$ (if $r < j$).
Case 2: $j \leq l$. All the vertices from e_{l-j+1} to e_{r-j} are adjacent to e: we increment $o[l-j+1]$ and decrement $o[r-j+1]$ (if $r < 2j$).
Case 3: $l < j$ and $j < r$. Let $m = \min\{l, r-j\}$. If $m > 0$ then all the vertices from e_1 to e_m are adjacent to e: we increment $o[1]$ and decrement $o[m+1]$. Similarly let $M = \max\{l, r-j\}$. If $M < j$ then the vertices from e_{l+1} to e_j are adjacent to e: we increment the counter $o[l+1]$.
Stage 3: Compute $f = \max_l \{\sum_{i=1}^{l} o[i] | 1 \leq l \leq j\}$. Return e_f.

The following theorem summarizes the result of this section.

Theorem 10.14 *Given a clique C, the node $e_f \in C$ computed by the algorithm above has maximum unoriented degree among the nodes in C.*

The complexity of the algorithm is proportional to the size of C plus the number of unoriented nodes in $OV(\pi)$, and hence, it is $O(n)$.

10.4.12 Algorithm Summary

Figure 10.13 gives a schematic description of the algorithm.

algorithm SIGNED REVERSALS(π);
/* π is a signed permutation */
1. Compute the connected components of $OV(\pi)$.
2. Perform h reversals ($h+1$ in the fortress case)
leading to π' with $d' = d - h(+1)$
and no unoriented components.
3. while π is not sorted **do** :
  /* iteration */
  begin
    a. find a happy clique C in $OV(\pi)$.
    b. find a node $e_f \in C$ with maximum unoriented
    degree, and perform a reversal on e_f;
    c. update π and the representation of $OV(\pi)$.
  end
4. output the sequence of reversals.
end

Figure 10.13: Sorting signed permutations

Theorem 10.15 *Algorithm* SIGNED REVERSALS *finds the reversal distance r in $O(n\alpha(n) + r \cdot n)$ time, and in particular in $O(n^2)$ time.*

Proof Step 1 takes $O(n\alpha(n))$ time by the algorithm of Berman and Hannenhalli [3]. Step 2 takes $O(n)$ time. Step 3 takes $O(n)$ time per reversal, by the previous discussion. ∎

10.4.13 Open Problems

- Is there a faster algorithm for sorting signed permutations using reversals ?
- Given 3 signed permutations π_1, π_2, π_3, find an efficient algorithm that minimizes $\sum_{i=1}^{3} d(\pi, \pi_i)$ (finding an exact solution is NP-hard [5]).
- Find the reversal distance between two signed digit sequences with equal number of occurrences of each digit.
- Find how many sequences of reversals realize d.

- Find among the minimum sequences one that has some additional properties.

Bibliography

[1] W. Ackermann. Zum hilbertshen aufbau der reelen zahlen. *Math. Ann.*, 99:118–133, 1928.

[2] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[3] P. Berman and S. Hannenhalli. Fast sorting by reversal. In *Proc. Combinatorial Pattern Matching (CPM)*, pages 168–, 1996. LNCS 1075.

[4] Bovine and Mouse on Human Comparative Maps. http://bos.cvm.tamu.edu.

[5] A. Caprara. Formulations and complexity of multiple sorting by reversals. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 84–93, Lyon, France, 1999. ACM Press.

[6] Alberto Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, February 1999.

[7] D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proc. ninth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 98)*, pages 244–252. ACM Press, 1998.

[8] T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.

[9] S. Even and O. Goldreich. The minimum-length generator sequence is np-hard. *J. of Algorithms*, 2:311–313, 1981.

[10] W. H. Gates and C. H. Papadimitriou. Bound for sorting by prefix reversals. *Discrete Mathematics 27*, pages 47–57, 1979.

[11] S. Hannenhalli. Private communication. unpublished, 1998.

[12] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, January 1999.

[13] S. B. Hoot and J. D. Palmer. Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. *J. Molecular Evooution*, 38:274–281, 1994.

[14] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theor. Comput. Sci.*, 36:265–289, 1985.

[15] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892, June 2000.

[16] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180–210, January 1995.

[17] J. D. Palmer and L. A. Herbon. Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion. *Nucleic Acids Research*, 14:9755–9764, 1986.

[18] J. D. Palmer and L. A. Herbon. Unicircular structure of the Brassica hirta mitochondrial genome. *Current Genetics*, 11:565–570, 1987.

[19] J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *J. Molecular Evolution*, 28:87–97, 1988.

[20] J. D. Palmer, B. Osorio, and W.R. Thompson. Evolutionalry significance fo inversions in legume chorloplast DNAs. *Current Genetics*, 14:65–74, 1988.

[21] Oxford Grid: Human vs. Mouse. http://www.informatix.jax.org.