Algorithms for Molecular Biology **Fall Semester, 2001**

Lecture 10: January 10, 2002

Lecturer: Ron Shamir *Scribe: Elad Verbin and Inon Axel*[1]

10.1 Genome Rearrangements

10.1.1 Preface

It has been long known that during mitosis, a phenomenon of chromosomes repeated doubling in thickness occurs in the salivary glands of *Drosophila*. These appear to be the result of two *homologs* (identical copies of a chromosome segment created during cell division) that have been glued together somehow.

In general, chromosomes have an observable pattern of bands perpendicular to their length. This was studied since the 1920's, and is 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.

This phenomenon suggests that the genome undergoes rearrangements, or what seems to be a general scrambling of its order, during evolution.

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, $abc \rightarrow ac$ (a, b, c are subsequences))
   - Insertions (a part is added, $ac \rightarrow abc$)
   - Duplications (can be tandem, $abc \rightarrow abbc$, or not, $abcd \rightarrow abcbd$ *or* $abcd \rightarrow acbcd$)
   - Reversals, or inversions (a part is turned around, head to tail, for example $abc_1c_2c_3c_4de \rightarrow abc_4c_3c_2c_1de$).
   - Transpositions (two adjacent parts switch places, for example $abcd \rightarrow acbd$) - this operation is believed to be the most rare since it requires 3 points of breakage along the chromosome.
   - Transversals (two parts change places and one is reversed).

[1] Based partially on notes taken by Yaniv Nahum and Sonny Ben-Shimon, fall 2000.

How do these operations take place?

It is not exactly known when and why the above operations occur though there are several reasonable hypotheses for some of them. 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) - see Figure 10.1.

An Insertion might happen when a foreign DNA segment enters the cell and combines itself with the cell's DNA, for example, during a virus infection or by *horizontal* $transfer$.

Duplication sometimes happen when some error occurs during cell mitosis.

Transpositions are rare, and it is not exactly clear how they take place.

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 may die.
   - Fusion: two chromosomes merge.
   - Fission: one chromosome splits up into two chromosomes.
     It is not known what exactly happens to the centromere in the latter two cases.

10.1.3 Genome Rearrangements and Evolution

There are several factors that make the study of genome rearrangements useful for studying evolution:

- Since the operations described above are much 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 rearrangements are easily noticeable mutations, we can locate them simultaneously in large portions of the genome, and therefore can inspect them in a larger scale of data. This allows us to look at a more comprehensive picture of the evolutionary process.

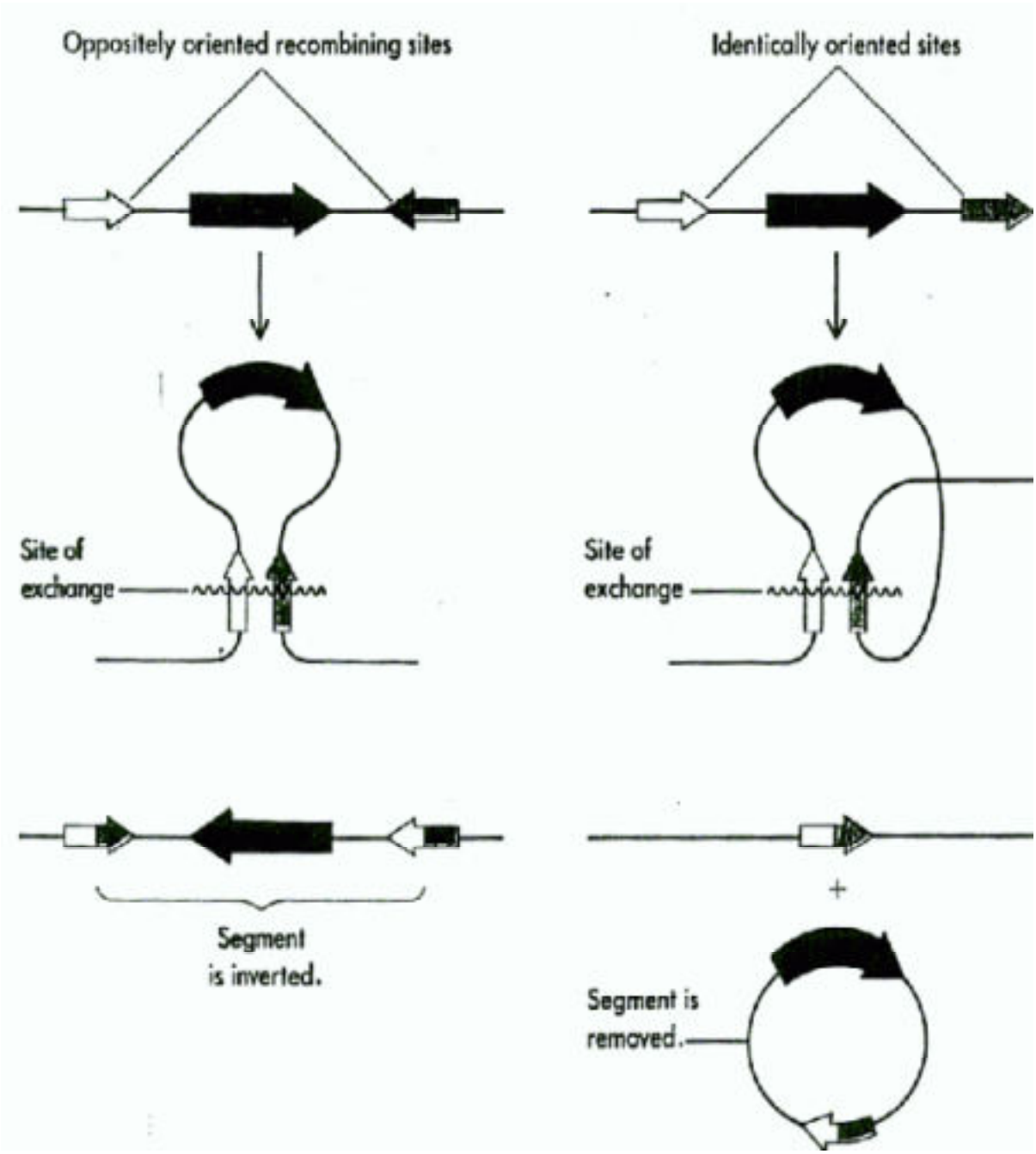

Figure 10.1: Illustration of a hypothetical model for reversal and deletion operations on a DNA strand.

For example, one can compare human and mouse. There are about 80 million years of evolutionary distance between them, but only about 140-150 operations of rearrangements. See Figure 10.2.

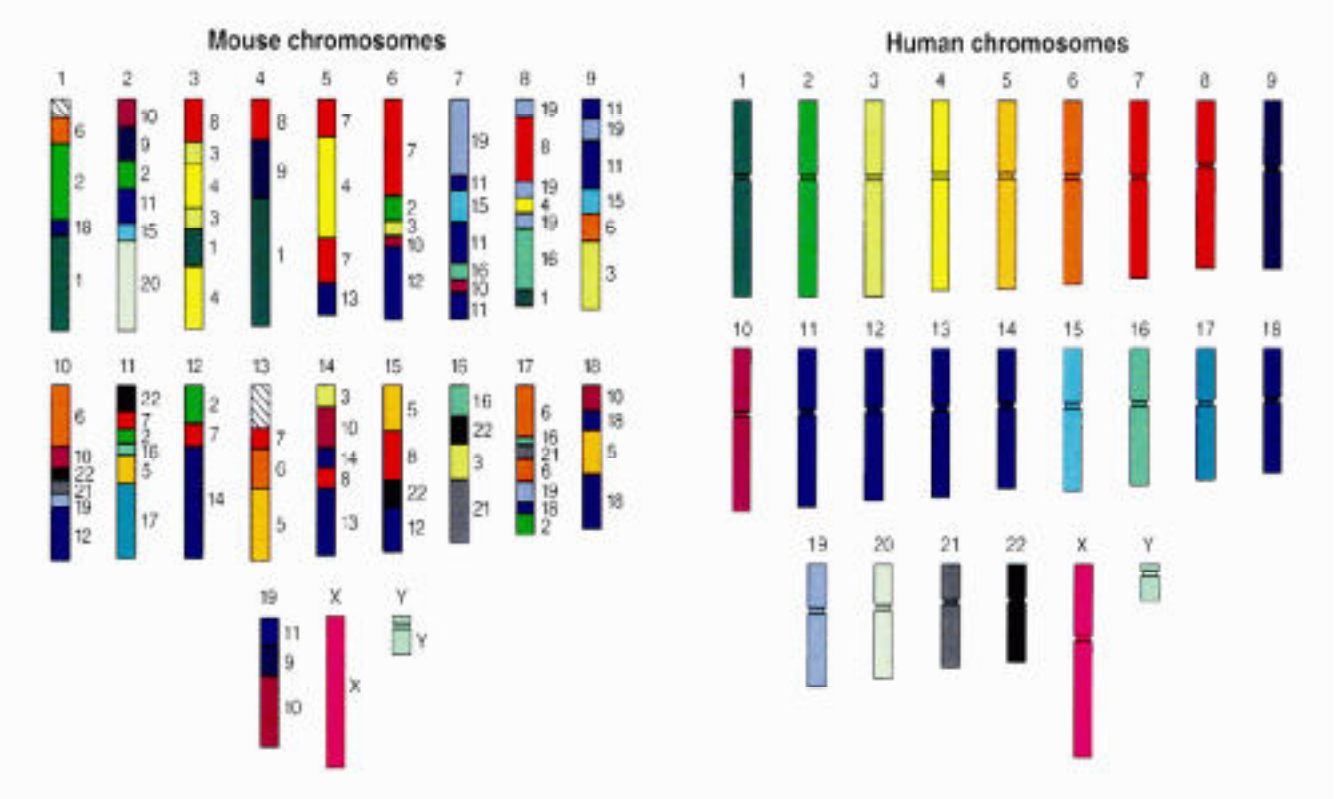

Figure 10.2: Source:[19]. Mouse and human genetic comparison.

10.2 Unsigned Permutations

In the following section we discuss the study of reversals on unsigned permutations. The sequence segments will be represented by the elements of the permutation. This formalism ignores the directionality of each segment. We discuss a single chromosome and work under two basic (not biologically exact) assumptions:

- It is possible to identify genes uniquely along the chromosome and their homologues in other species.
- All of the genes are different.

The order of the genes (or gene homologues), which might be different in different species, is a permutation of these, unique, genes (or homologues thereof). 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.

Definition A *reversal transformation on a sequence is the operation of taking a subsequence and reversing the order of elements within it. For example 12345 → 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 10.3).*

$\pi_1 = (1,\underline{2,3,4},5,6)$
$\pi_2 = (1,4,\underline{3,2,5,6})$
$\pi_3 = (\underline{1,4,6},5,2,3)$
$\pi_4 = (6,4,1,5,2,3)$

Figure 10.3: Examples of reversals; the underlined segments show where the reversals took place. The reversal distance between π_1 and π_4 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 [18].
2. 1.75-approximation algorithm [2].
3. NP Completeness proof [7].
4. 1.5-approximation algorithm [8].
5. 1.375-approximation algorithm [12].

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 3 and 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.

The following theorem gives lower and upper bounds for $d(\pi)$

Theorem 10.1 ([18]) $\frac{b(\pi)}{2} \leq \lceil \frac{b(\pi)}{2} \rceil \leq d(\pi) \leq n-1$

Proof: The lower bound holds since a reversal can cancel 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 $n-1$ reversals to create any sequence. For example, a sequence that does that is one that at each step operates on the positions $(i \dots \pi^{-1}(i))$. ■

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*; A strip of size 1 is defined as 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 eliminated (see Figure 10.4).

■

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

Figure 10.4: Two possible cases to eliminate a breakpoint using a decreasing strip ($K = 4$).

Lemma 10.2 gives rise to the following approximation algorithm: If there exists a decreasing strip, find and perform a good reversal ($\Delta b = -1$). Otherwise, reverse an increasing strip, thus creating a decreasing strip ($\Delta b = 0$). This algorithm 4-approximates the algorithm, since there are at most $2b(\pi)$ reversals.

Lemma 10.3 ([18]) *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 $\pi = \pi_1 \dots \pi_n$ 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 10.5, top). Similarly, $(\pi_j + 1)$ has to be to the right of π_j (see Figure 10.5, bottom).

$$\leftarrow \pi_i \ldots\ldots (\pi_i - 2)\ (\pi_i - 1) \rightarrow$$
$$(\pi_j + 1)\ (\pi_j + 2) \rightarrow \ldots\ldots \leftarrow \pi_j$$

Figure 10.5: Two impossible scenarios.

Consider the interval ρ_j between π_j and $(\pi_j + 1)$ along π (including π_j but not including $(\pi_j + 1)$) ; and the interval ρ_i between $(\pi_i - 1)$ and π_i (including π_i but not including $(\pi_i - 1)$) (see Figure 10.6).

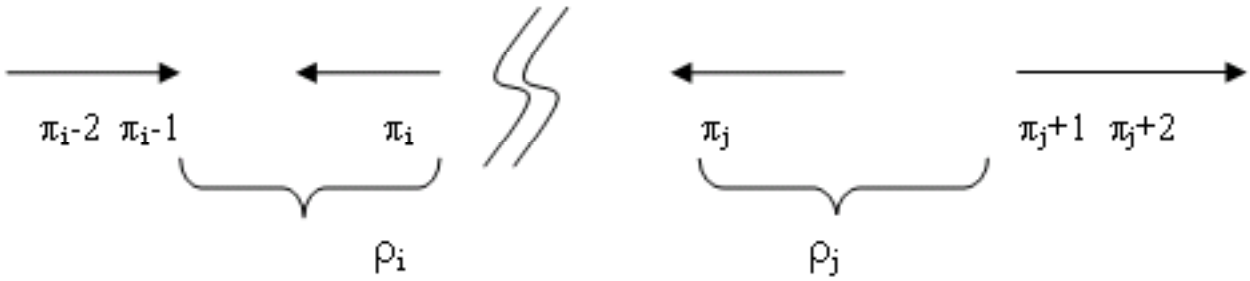

Figure 10.6: A situation where the two strips do not overlap.

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, and π_j must be to the right of $(\pi_i - 1)$. The only remaining case is (see Figure 10.7):

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

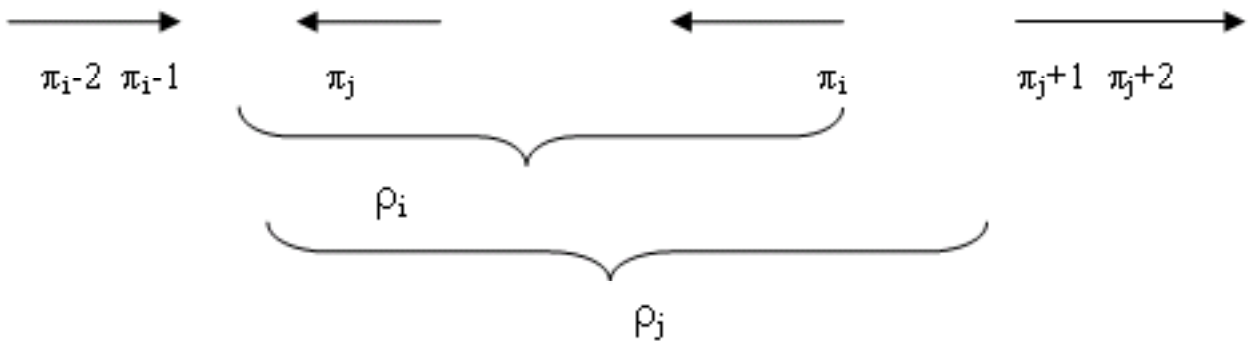

Figure 10.7: The remaining case where the two strips overlap.

If $\rho_i \setminus \rho_j$ contains a decreasing strip, then reversing ρ_j leaves us a decreasing strip (and removes a breakpoint). Furthermore, if $\rho_i \setminus \rho_j$ contains an increasing strip, then reversing ρ_i 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 $\rho_j = \rho_i$. Therefore, the reversal on $\rho_j = \rho_i$ 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 2-approximates sorting by reversals, since $\Delta b = -1$ on the average.

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 some point during the course of evolution, and since that doubling, various genome rearrangements took place, mixing the genome into the shape we know today. This specific case poses a different computational problem in which each gene (number) appears twice and can be either in one direction (positive) or reversed (negative). In this chapter we will not relate to this problem.

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 10.8).

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 10.9). Note that since most of the X chromosome does not undergo recombination, its overall content is rather conserved among mammals.

Palmer et al. [23] have shown that the evolution of the chloroplast genome of the pea can be modeled as a series of rearrangements of a soybean-like ancestor (see Figure 10.10).

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. 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 (Sorting Signed Permutation by Reversals)
INPUT: A signed permutation π.
QUESTION: What is the reversal distance between π and the signed identity permutation $(+1, +2, \ldots, +n)$?

A simple upper bound for the reversal distance would be $2n$, since we can create the correct sequence (disregarding signs) by n reversals (see Theorem 10.1) and then another n (at most) for sign flipping (a reversal of length one).

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 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

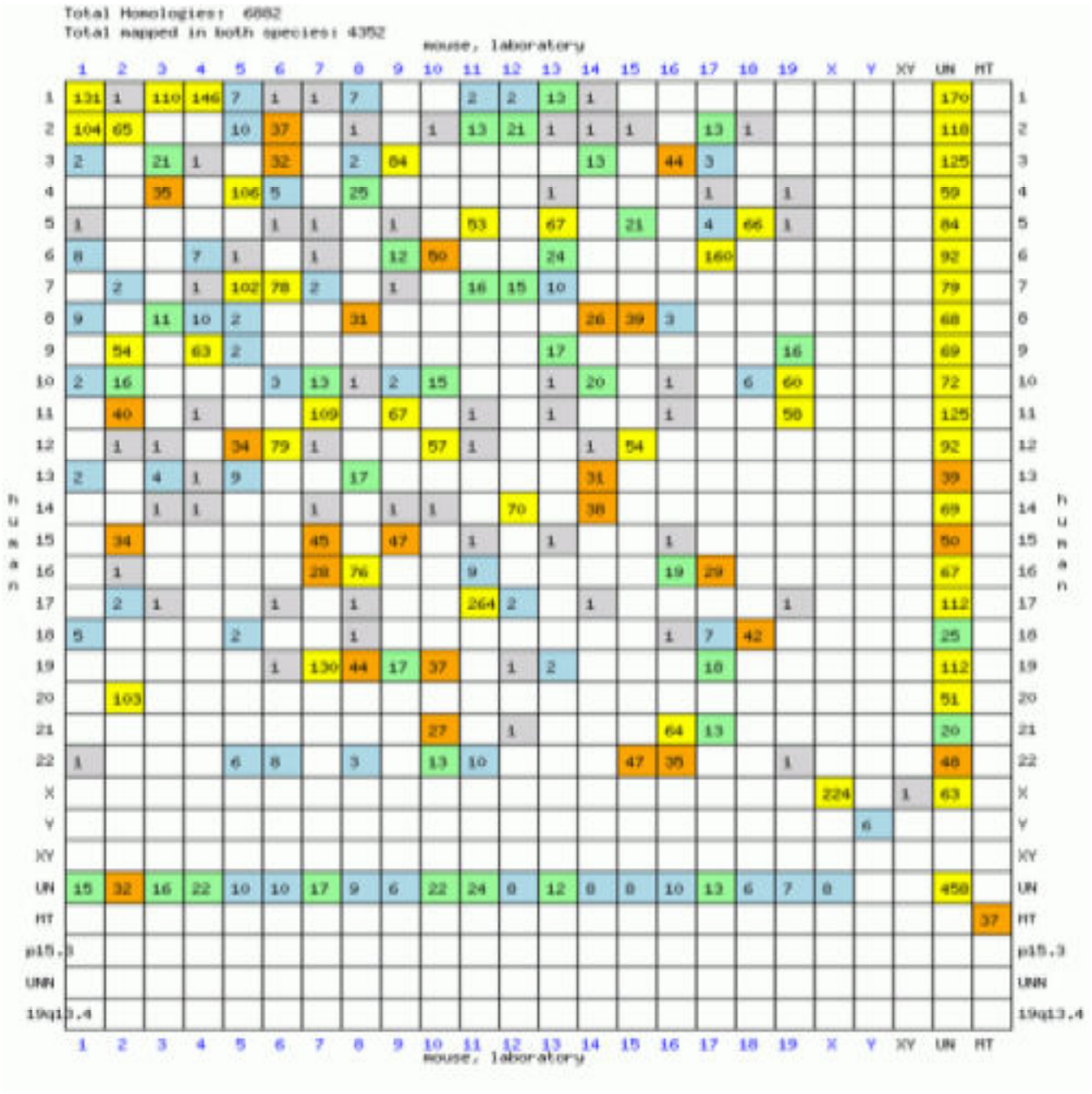

Figure 10.8: Man-mouse pairs of homologous genes were examined for their chromosomal location. The (i, j) entry in the table registers the number of such pairs mapped to human chromosome i and mouse chromosome j. UN stand for "'unknown"' and MT for mitochondrial DNA.

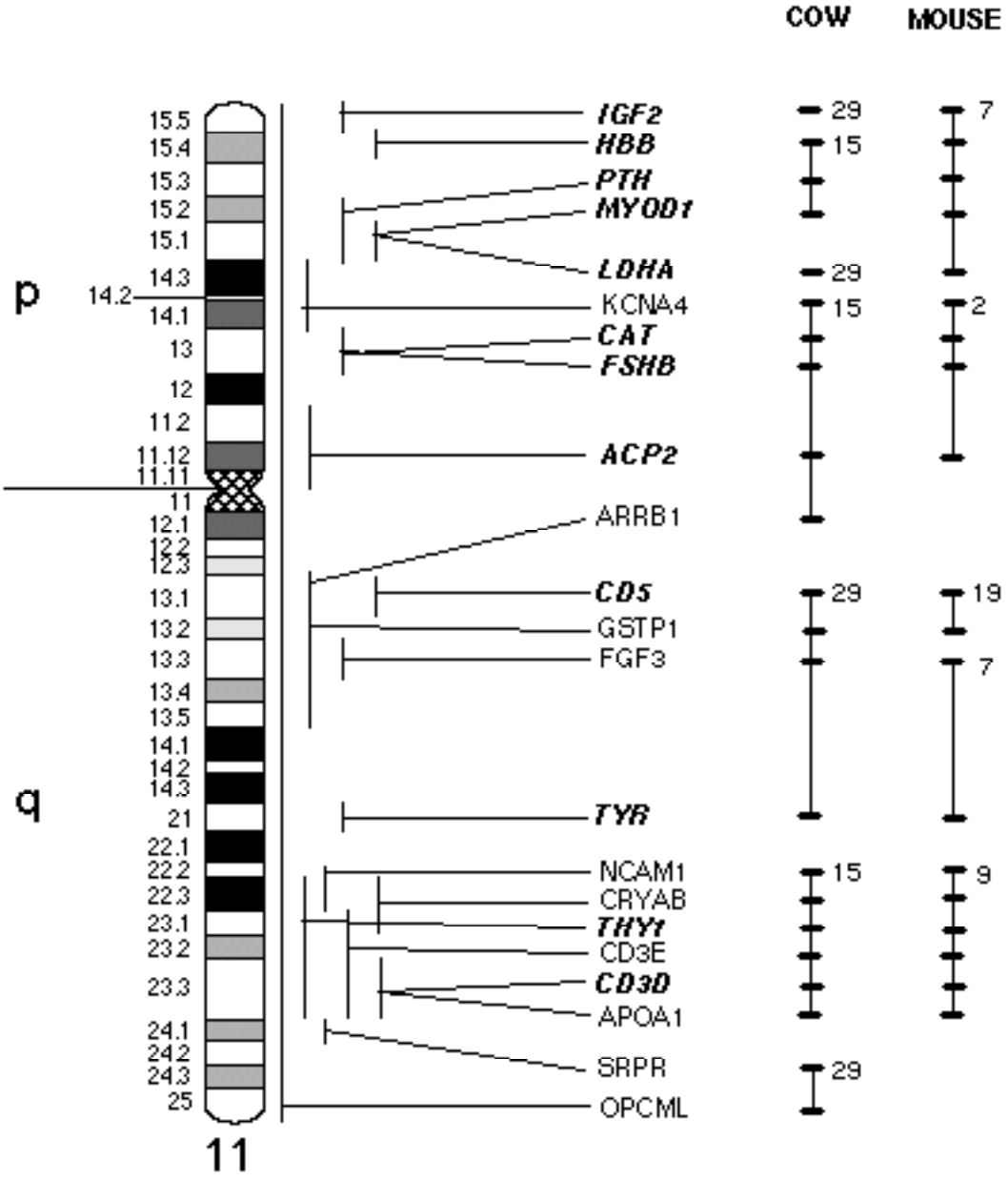

Figure 10.9: Source:[5]. A Comparison of cow and mouse to human X chromosome.

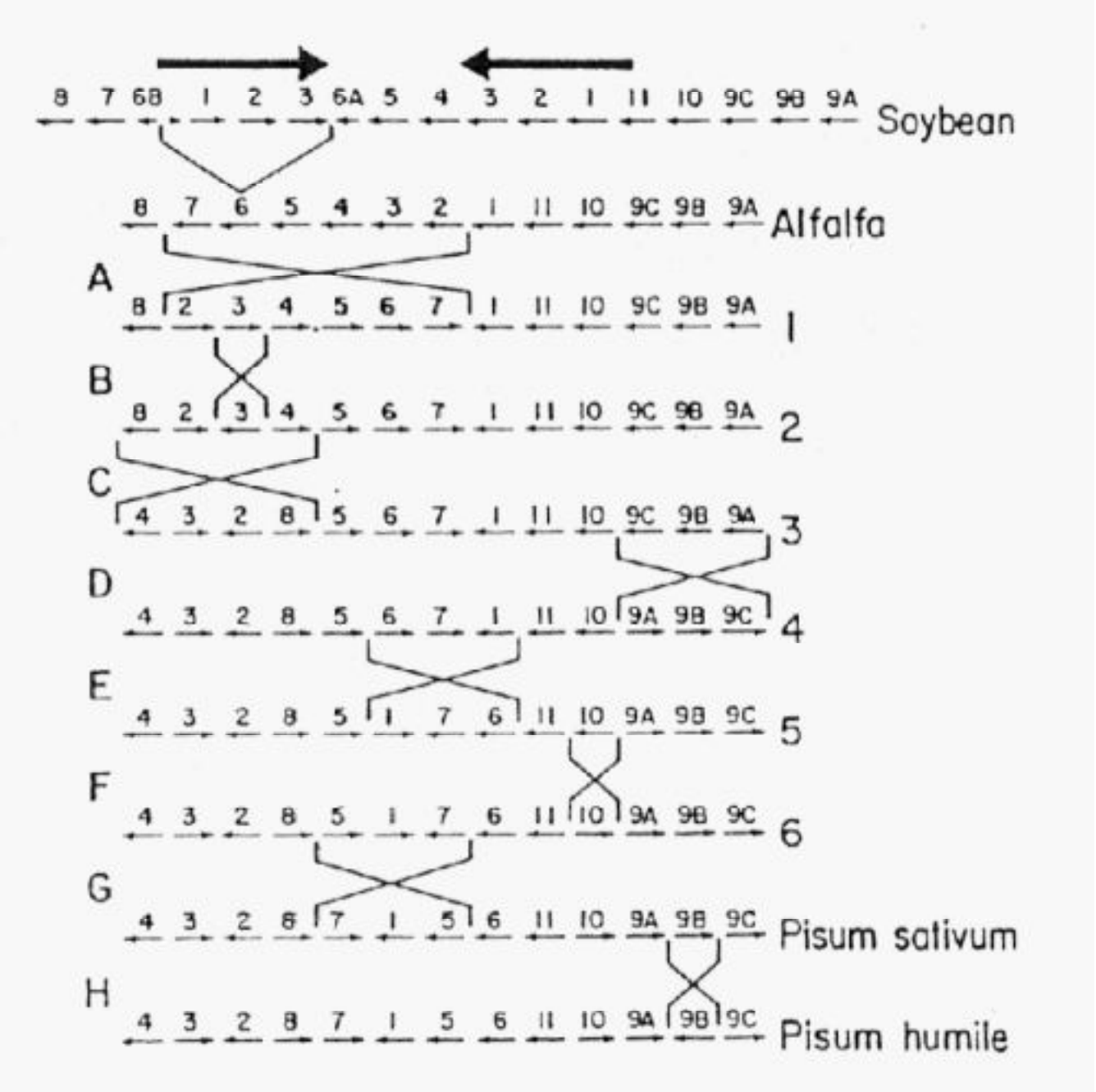

Figure 10.10: Source:[23]. A model for the evolution of the pea chloroplast genome. The horizontal arrows represent conserved sequence blocks.

Sturtevant have shown in 1938 [9] evidence of inversions in chromosomes of *Drosophila*. In the 1980's, Palmer [20, 21, 22, 23, 15] 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, signs indicate to their orientation. The difference in gene order between genomes can be explained by some reversals between them. These reversals correspond to evolutionary changes throughout 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.

In contrast to problem 10.1, in 1995, Hannenhalli and Pevzner [14] have shown that the problem of sorting a *signed* permutation by reversals is polynomial, and can be solved in $O(n^4)$ time. In 1996, Berman and Hannenhalli [4] 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 an algorithm developed by Kaplan, Shamir and Tarjan for sorting a signed permutation of n elements that is partly based on Berman and Hannenhalli's algorithm but is simpler and runs in $O(n^2)$ time, thereby improving upon the previous bound. In fact, if the reversal distance is r, the algorithm requires $O(n \cdot r + n\alpha(n))$ time [17].

10.4.2 Group Theory Viewpoint

From a group theory point of view, sorting signed permutations can be viewed as follows: Consider S_n, the symmetric group of order n (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 [10]. Jerrum generalized this result by proving it is PSPACE-complete [16]. Another related problem is:

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

10.4.3 Transforming to unsigned permutations

Let us define a one-to-one mapping u from the set of signed permutations of order n into the set of unsigned permutations of order $2n+2$. 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 $2x-1$. Also, augment the permutation by setting $\pi_0 = 0$ and $\pi_{2n+1} = 2n+1$. For example:
$(4,-3,1,-5,-2,7,6) \rightarrow (0,7,8,6,5,1,2,10,9,4,3,13,14,11,12,15)$.

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

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

It can be easily seen that we may now limit our discussion to the aforementioned kind of unsigned permutations, given that the reversals we perform on them have a one-to-one correspondence 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.

For example, in the permutation $(0,7,8,6,5,1,2,10,9,4,3,13,14,11,12,15)$, $\rho(1,6)$ is an even (legal) reversal, while $\rho(3,5)$ and $\rho(2,5)$ aren't.

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 a reversal we will always mean an even reversal. We shall also identify the signed permutation with its unsigned counterpart.

10.4.4 Definitions

Let $\pi = (0, \pi_1, \ldots, \pi_n, n+1)$ denote an unsigned permutation of the above type. 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. We say that a reversal $\rho(i,j)$ *acts on* the gaps (π_{i-1}, π_i) and (π_j, π_{j+1}).

10.4.5 The Breakpoint Graph

Definition The *breakpoint graph* $B(\pi)$ of a permutation $\pi = (0, \pi_1, \ldots, \pi_n, n+1)$ is an edge colored graph on $n+2$ vertices $\{0, \pi_1, \ldots, \pi_n, n+1\}$. 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}. (For an example see Figure 10.11(a)).

Note that in $B(\pi)$ every vertex has either exactly one black edge and one gray edge incident on it, or no incident edges at all. Therefore, there is a unique decomposition of $B(\pi)$ into disjoint cycles. The edges of each cycle are alternating gray and black. Our goal is to remove all edges from a given graph $B(\pi)$. Let $c(\pi)$ be the number of cycles in $B(\pi)$.

Figure 10.11(a) shows the breakpoint graph of $\pi = (0, 7, 8, 6, 5, 1, 2, 10, 9, 4, 3, 13, 14, 11, 12, 15)$, which results from the permutation $(4, -3, 1, -5, -2, 7, 6)$. 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 10.11(b).

For an arbitrary reversal ρ on a permutation π, define $\Delta b(\pi, \rho) = b(\pi \cdot \rho) - b(\pi)$ and $\Delta c(\pi, \rho) = c(\pi \cdot \rho) - c(\pi)$. 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 Δc.

Bafna and Pevzner[2] observed that:

Claim 10.4 ([2]) *[The following values are taken by Δb and Δc depending on the types of the gaps $\rho(i, j)$ acts on. Case analysis shows that only these values are possible (see Figure 10.12):*

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 10.5 ([2]) $d(\pi) \geq b(\pi) - c(\pi)$.

Proof: From the last observation we see that, for any reversal, the best improvement we can get is decreasing $\Delta b(\pi) - \Delta c(\pi)$ by 1 (this happens in cases 4a, 4b and 4d), and since $\Delta b(id) - \Delta c(id) = 0$, we need at least as many steps to get to the identity permutation. ■

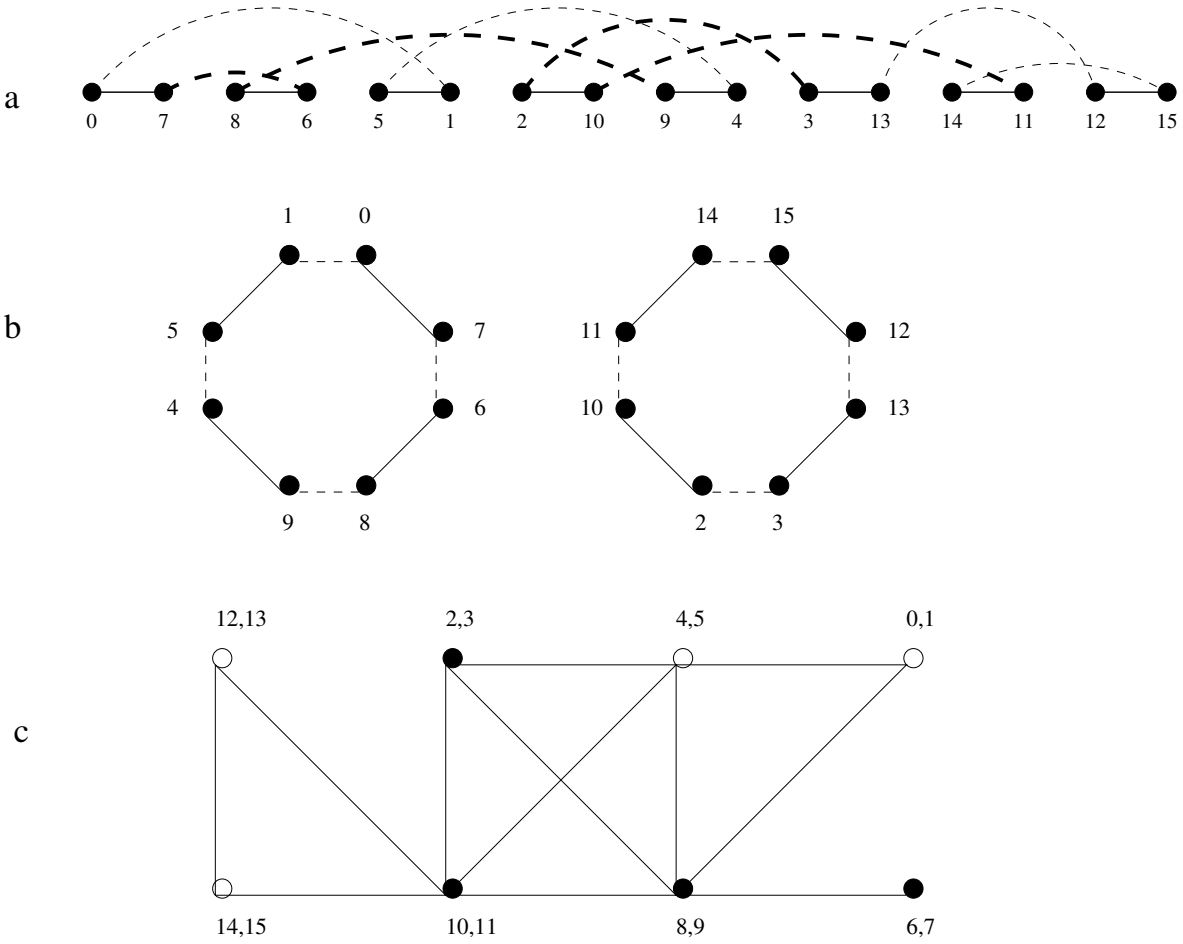

Figure 10.11: Source:[17]. (a) The breakpoint graph $B(\pi)$ for the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. Black edges are solid; gray edges are dashed; oriented edges are bold. (b) $B(\pi)$ decomposes into two disjoint alternating cycles (c) The overlap graph $OV(\pi)$. Black vertices correspond to oriented edges.

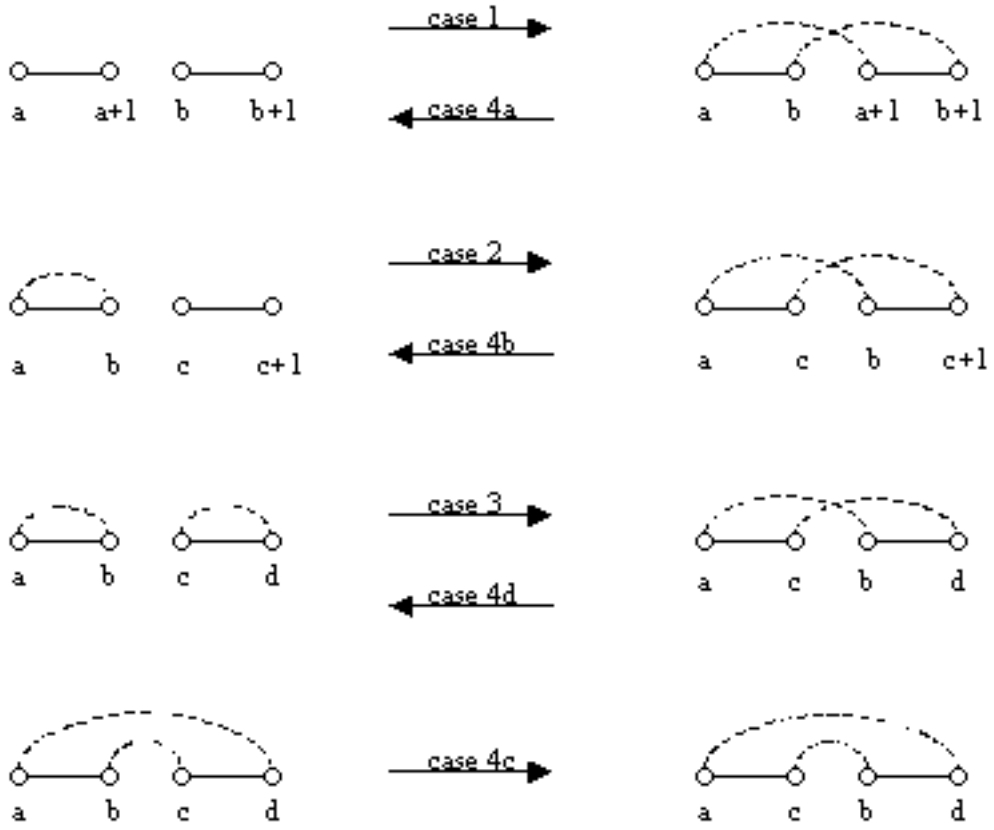

Figure 10.12: All possible cases of changes to Δb and Δc by applying a reversal (see Section 10.4.5).

Definition 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 10.11(a) is unoriented, while the gray edge $(7, 6)$ is oriented.

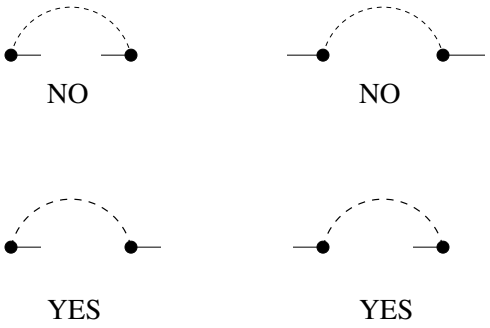

Figure 10.13: A gray edge (π_k, π_l) is oriented iff k+l is even.

10.4.6 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.

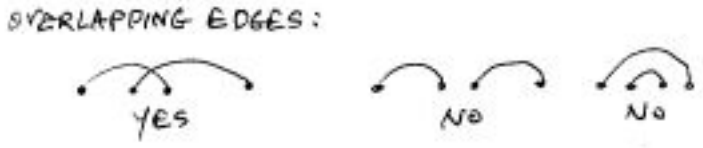

Figure 10.14: Two edges are said to overlap when their corresponding intervals overlap.

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 10.11(c) shows the interval overlap graph for $\pi = (4, -3, 1, -5, -2, 7, 6)$. It has only one oriented component. Figure 10.15(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.

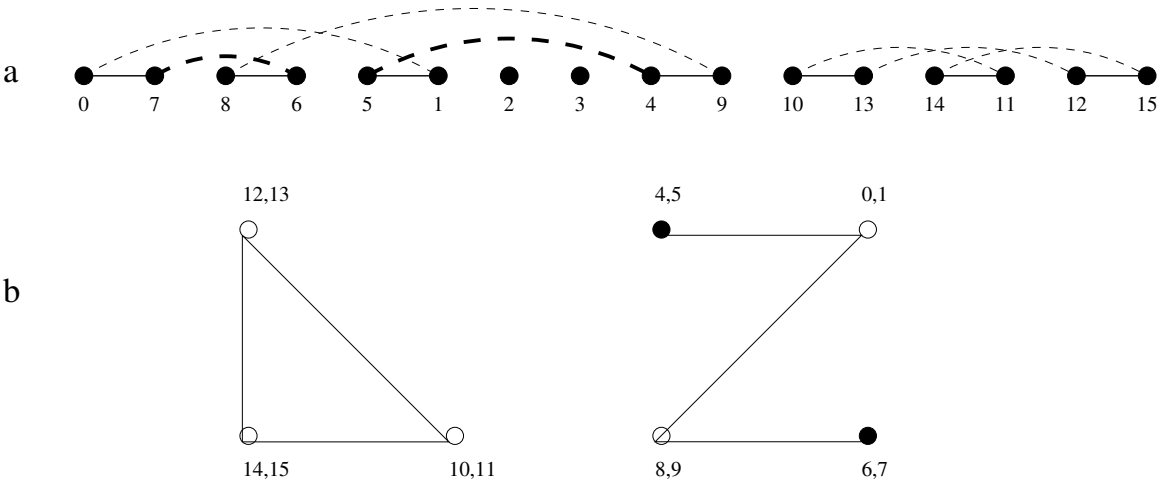

Figure 10.15: (a) The breakpoint graph $B(\pi')$ of $\pi' = (4, -3, 1, 2, 5, 7, 6)$. π' was obtained from π of Figure 10.11 by the reversal $\rho(7, 10)$; or, equivalently, by the reversal defined by the gray edge $(2, 3)$. (b) The overlap graph of π'.

10.4.7 The Algorithm

In '95 Hannenhalli and Pevzner [13] proved:

Theorem 10.6 ([13]) *For a signed permutation π, $d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi)$*, where:

- h is a function of the unoriented components of the overlap graph.

- f is either 1 or 0 depending on whether the unoriented components have a particular structure. If there are no unoriented components then f is always 0.

Corollary 10.7 *If there are no unoriented components in $OV(\pi)$ then $d(\pi) = b(\pi) - c(\pi)$.*

Hannenhalli and Pevzner [13] gave a constructive proof of this fact that gives us a $O(n^4)$ algorithm. Berman and Hannenhalli [4] improved the implementation of this to $O(n^2\alpha(n))$

We will show how to solve the problem in $O(n^2)$ in the case where $h = 0$ (so that $d = b-c$). The general case can quite easily be reduced to this by the method for clearing hurdles described in [17]. Our key idea is to prove (constructively) that the following condition is fulfilled for every step of the algorithm:

Condition 10.8 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.

The first part holds for the oriented reversals. Therefore, we want to find an oriented reversal r for which the second part holds. First, let us look at what effect performing an oriented reversal has on the overlap graph: 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. We can formulate effect of such a reversal r on the overlap graph:

Claim 10.9 *The effect of an oriented reversal r represented by node n on the overlap graph is as follows:*

- *Delete any node whose corresponding oriented gray edge defines r. This includes n and sometimes another node.*
- *Complement the subgraph induced by n's neighbors.*
- *switch the orientation of n's neighbors - edges that were oriented are now unoriented, and vice versa.*

For example - see what happens when we move from Figure 10.11 to Figure 10.15 - by executing reversal $\rho(7, 10)$. To get an optimal sequence, the choice of a reversal needs to be a good one, e.g., one that maintains condition 10.8. We must therefore make sure that 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 so that for all *oriented* $y \notin C$, if $(x, y) \in A$ and $x \in C$, there exists an *oriented* node $z \notin C$ where $(z, y) \in A$ and $(z, x) \notin A$.

For example, in the overlap graph shown in Figure 10.11(c) $\{(2,3),(10,11)\}$ and $\{(6,7)\}$ are both happy cliques, but $\{(2,3),(10,11),(8,9)\}$ is not. There are, typically, many happy cliques in a permutation's overlap graph.

Claim 10.10 *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 (and therefore is safe and fulfills Condition 10.8).*

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:
  It is obvious - otherwise M, an unoriented component, would have been present *beforehand*, since a reversal changes only it's neighborhood.

- 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 (see Figure 10.16(a)). Otherwise, e was oriented, and by the definition of the happy clique C, e has an oriented neighbor g, non-adjacent to x. Therefore $g \in M$, and its orientation remains unchanged by applying r, thus M is oriented - a contradiction (see Figure 10.16(b))

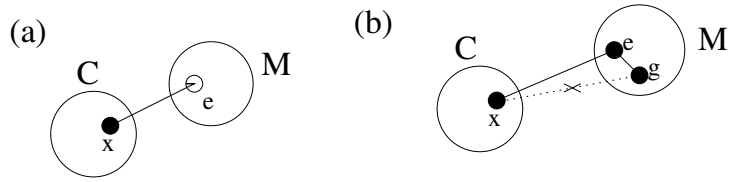

Figure 10.16: Two impossible scenarios for Claim 10.10

- Every unoriented neighbor of x is adjacent to y:
  Suppose to the contrary that z is an unoriented neighbor of x, nonadjacent 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 π_{2i+2} appears between π_{2i} and π_{2j-1}, and so is π_{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. Hence y has more unoriented neighbors than x, a contradiction to the choice of x.

■

Claim 10.10 implies, for example, that the reversal defined by the gray edge $(10, 11)$ is a safe proper reversal for the permutation of figure 10.11 (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 10.15.

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 $\Omega(n^2)$ operations 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 10.4.5 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 $OV(\pi)$. The 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

How do we even know that there is always a happy clique, let alone find one? In their paper, Kaplan, Shamir and Tarjan [17] have proven a general result about the existence of happy cliques:

Theorem 10.11 ([17]) *The oriented neighborhood of every oriented node contains a happy clique.*

We will not prove it here. The interested reader is referred to [17]. Here, we will show an algorithm for finding one, specific, happy clique (and therefore give an alternative proof to a weaker version of the non-constructive Theorem 10.11):

Theorem 10.12 *Given $OV(n)$ in its implicit representation, One can find a Happy Clique in $O(n)$ time.*

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 $1 \leq i \leq k$, the algorithm maintains a happy clique C_i in the subgraph of $OV(\pi)$ induced by these vertices. Assume $|C_i| = j$, $j \leq i$ 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 10.17(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 10.17(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 10.17(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 10.17(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 10.17(e).

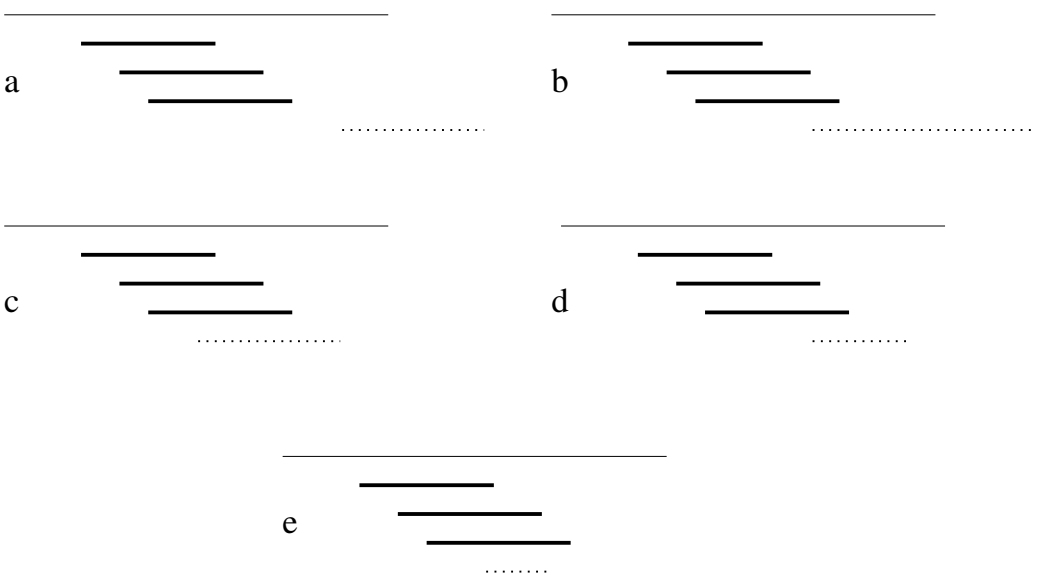

Figure 10.17: The various cases of the algorithm for finding a happy clique. The topmost interval is always t_i. The three thick intervals comprise C_i. The dotted interval corresponds to n_{i+1}.

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 in linear time.

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 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 $f = arg\max_l\{\sum_{i=1}^{l} o[i] | 1 \leq l \leq j\}$. 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 10.18 gives a schematic description of the algorithm.

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 [4]. Step 2 takes $O(n)$ time. Step 3 takes $O(n)$ time per reversal, by the previous discussion. This sums up to $O(n^2)$ because $r \leq n$. ■

This algorithm presents improvements over previous algorithms in both the asymptotic running time, the constants, and the simplicity of the code (because the theory is simpler).

10.4.13 An Additional Improvement

In [3] Anne Bergeron significantly simplified the theory for sorting signed permutations, and made it much more intuitive. She disposed of the concept of breakpoint graphs and overlap graphs, and made the concepts of oriented reversals and hurdles more intuitively appealing. She also gave a very simple $O(n^3)$ algorithm for the hurdle-free case. Here are some of her

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 10.18: Sorting signed permutations.

results, working on the original, signed, permutation, only augmenting it with $\pi_0 = +0$ and $\pi_{n+1} = n+1$:

Definition ([3]) Let $\pi = 0, \pi_1, \ldots, \pi_n, n+1$ be a signed permutation. An *oriented pair is a pair of consecutive numbers with opposite signs, that is a pair* (π_i, π_j) *so that:*

- $i < j$
- π_i *and* π_j *have opposite signs (0 is considered positive)*
- $||\pi_j| - |\pi_i|| = 1$

For example, in the permutation $\pi = (0, 3, 4, -2, -5, 1, 6)$ the oriented pairs are (1,-2), (-2,3), (4,-5) and (-5,6). It is easily seen that there is a one-to-one correspondence between oriented pairs and oriented gray edge, and that they are, essentially, the same thing.

Each oriented pair induces a reversal that eliminates a positive number of breakpoints. For example, in $\pi = (0, 3, 4, -2, -5, 1, 6)$ the oriented pairs induce the reversals $\rho(4, 5)$, $\rho(1, 2)$, $\rho(3, 4)$ and $\rho(4, 5)$, respectively. Note that the first and last oriented pairs induce the same reversal. This is because each oriented pair induces the oriented reversal that matches its oriented gray edge, and, as we have seen in Section 10.4.5, two oriented gray edges may define the same oriented reversal.

We observe that:

Claim 10.16 *If there is a sequence of oriented reversals that sorts a permutation than it is an optimal sorting sequence.*

This holds because we could never have created a hurdle (because there is no way to remove a hurdle using oriented reversals), and therefore all our oriented reversals were safe oriented reversals - giving us an optimal sorting. Hence, we would like to have at least one oriented pair at all times. Therefore, intuition suggests that we try a heuristic that maintains the maximal number of oriented pairs at all times.

Bergeron defined the score of a permutation, and provided the subsequent algorithm:

Definition The *score of an oriented reversal ρ in a permutation π is the number of oriented pairs in the resulting permutation $\pi \cdot \rho$.*

algorithm BERGERONSBR(π);
/* π is a signed permutation without hurdles*/
while π has an oriented pair,
    choose the oriented reversal that has maximal score and execute it.

Figure 10.19: Bergeron's Method for Sorting a Signed Permutation.

Surprisingly, Bergeron proves that this algorithm ends when it gets the identity permutation - that is, it always sorts the permutation (and, according to Claim 10.16, does so optimally). See Figure 10.20 for an example.

This algorithm can be trivially implemented in time $O(n^3)$ (it is possible to do better), and Bergeron shows in [3] how to implement it on a vector-machine (which is a RAM that operates on size-n vectors) in time $O(n^2)$.

10.4.14 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 approximation algorithm to find π that minimizes $\sum_{i=1}^{3} d(\pi, \pi_i)$ (finding an exact solution is NP-hard [6]).
- 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.

π_1 = (+0,+3,+1,+6,+5,-2,+4,+7)
score(+1,-2)=2 score(+3,-2)=4.
π_2 = (+0,-5,-6,-1,-3,-2,+4,+7)
score(+0,-1)=2, score(-3,+4)=4, score(-5,+4)=2, score(-6,+7)=2.
π_3 = (+0,-5,-6,-1,+2,+3,+4,+7)
score(+0,-1)=0, score(-1,+2)=2, score(-5,+4)=2, score(-6,+7)=2.
π_4 = (+0,-5,-6,+1,+2,+3,+4,+7)
score(-5,+4)=2, score(-6,+7)=2.
π_5 = (+0,-5,-4,-3,-2,-1,+6,+7)
score(+0,-1)=0, score(-5,+7)=0.
id = (+0,+1,+2,+3,+4,+5,+6,+7).

Figure 10.20: Example run of Bergeron's Algorithm. Note that in π_3, if we had executed the reversal (+0,-1), it would have produced a permutation with no oriented pairs, and the algorithm would have stopped unsuccessfully.

- 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] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *Proc. 12th Annual Symposium on Combinaotrial Pattern Matching (CPM '01)*, pages 106–117, 2001.

[4] P. Berman and S. Hannenhalli. Fast sorting by reversals. In *Proc. 7th Symp. on Combinatorial Pattern Matching (CPM)*, pages 168–185, 1996. LNCS 1075.

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

[6] 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.

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

[8] 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.

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

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

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

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

[13] S. Hannenhalli and P. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proc. 36th IEEE Symp. of the Foundations of Computer Science*, pages 581–592, 1995.

[14] S. 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.

[15] 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.

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

[17] H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. In *SIAM J. on Computing 29(3)*, pages 880–892, 1999.

[18] 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.

[19] US Department of Energy human genome program. http://www.ornl.gov/hgmis/.

[20] 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.

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

[22] 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.

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