CSC2427H, Winter 2006
Algorithms in Molecular Biology
Lecture #10

Lecturer: Michael Brudno
Scribe: Ali Juma

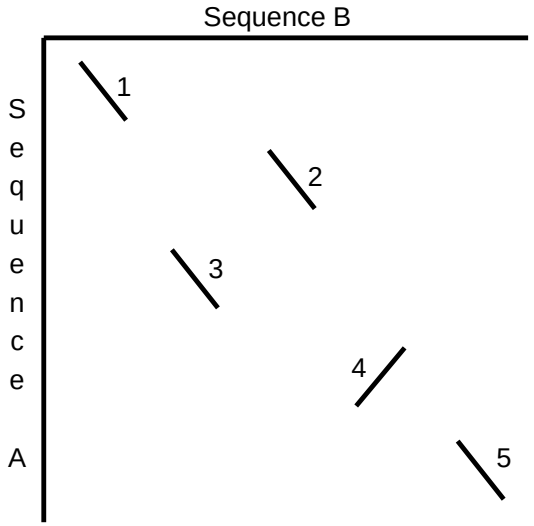
February 10, 2006

1 Introduction

In previous lectures we've looked at comparing two sequences to find the optimal global alignment and the optimal local alignment. It turns out that the best local alignments usually correspond to genes that the two sequences have in common. By finding the best local alignments, we can identify genome rearrangement events that have taken place. This information can be used as a measure of the distance between the two sequences.

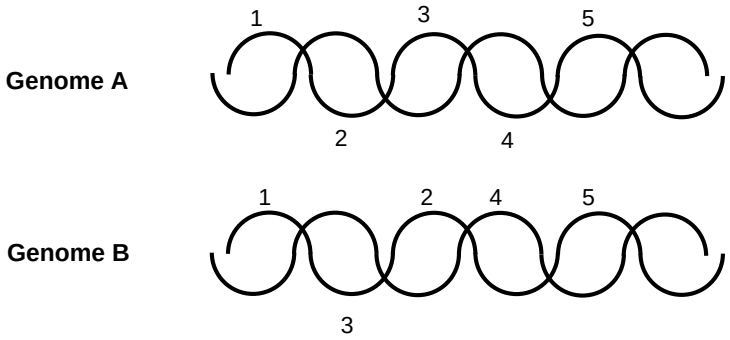
2 Using local alignment to identify genes

Consider the following local alignment of two sequences.

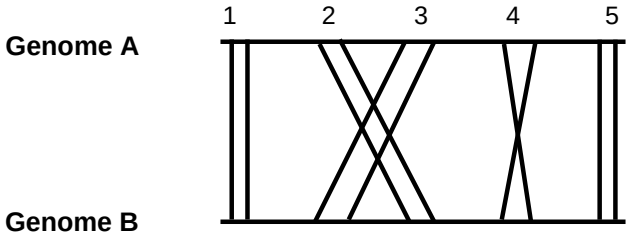


In the above diagram, five alignments are identified. Note that alignment 4 is found by computing the local alignment of sequence A and the reverse complement of sequence B. We consider each of the five alignments as genes that the sequences have in common. However, we can't always know for sure that these are in fact genes – we just know that their length and alignment score strongly suggest that they are genes.

In genome A, the genes appear in the order 1,2,3,4,5, while in genome B, the genes appear in the order 1,3,2,-4,5. We use -4 to represent the reverse of gene 4. This is used to indicate that gene 4 is on the opposite orientation (that is, the opposite strand) of genome B, as illustrated below:



Given homologous portions of two genomes, we are interested in determining how the order of genes in one genome relates to the order in the other. For genome A and genome B, this can be visualized as follows:

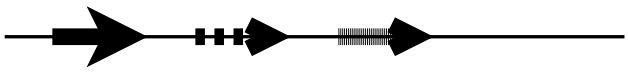


This tells us that genes 1 and 5 appear in the same positions in genomes A and B, while genes 2 and 3 have their positions interchanged, and gene 4 appears in the same position in both genomes but on opposite strands.

3 The evolution of genomes

We now turn to the question of how genomes evolve. Genomes evolve not just by insertions and deletions at individual positions, but also by the movement of larger pieces – that is, by the rearrangement of genes.

Consider the following genome, where each directed arrow represents a gene:



The rearrangement events that can take place are:

- Deletion (the loss of a gene):

- Translocation (where two adjacent genes swap positions):

- Duplication (where we get two copies of the same gene):

- Inversion (where a gene changes orientation):

In multi-chromosomed organisms, like humans, two additional events can occur: chromosomal fusion (where two distinct chromosomes fuse to become a single chromosome) and chromosomal fission (where a single chromosome splits into two distinct chromosomes).

A "whole genome duplication" occurs when every gene in a genome is duplicated, resulting in the organism having two copies of each gene. This is thought to be a very important biological event. It is believed that humans have undergone two rounds of whole genome duplication, but this is controversial. After a whole genome duplication, some of the redundant gene copies are lost, some mutate, and some remain the same. So it is not the case that after a whole genome duplication, an organism will continue to maintain multiple copies of each gene. It is interesting to note that our crops, which are bred to be big, actually have many copies of their genome. In this sense, due to selective breeding practices that long predate the study of genetics, we can consider all of our food to be genetically engineered.

4 Measuring the distance between two genomes

One way to measure the distance between two genomes is to count the number of rearrangement operations needed to transform one genome into the other. Since we don't know how to count duplication and deletion events, we instead restrict ourselves to considering only inversion and translocation events.

Consider the sequence 1 2 3 and 1 3 2. Note that we can transform the first sequence into the second sequence using a single translocation. We can also transform the first sequence

into the second sequence using only inversions as follows (where we use a bar to indicate the element(s) being inverted):

$$\begin{aligned} 1\ \overline{2}\ 3 &\rightarrow 1\ {-2}\ 3 \\ 1\ {-2}\ \overline{3} &\rightarrow 1\ {-2}\ {-3} \\ 1\ \overline{{-2}\ {-3}} &\rightarrow 1\ 3\ 2 \end{aligned}$$

Observe that in this example, a single translocation can be modeled using three inversions. This is also true in general. However, it is not clear if the modeling of a translocation using three inversions makes biological sense. Some biologists claim that the actual ratio of inversions to translocations is 2:1, not 3:1. If this is indeed the case, then our model overestimates the number of inversion events.

Note that our inversions are *signed* – they change the sign of the inverted elements. This turns out to be very important. Given a permutation of the set $\{1, 2, \ldots, n\}$, the problem of counting the number of *unsigned* inversions needed to transform the permutation into the identity permutation ($1\ 2\ \ldots\ n$) is NP-hard. On the other hand, given a *signed* permutation (where each element is allowed to be either positively or negatively signed), the problem of counting the number of *signed* inversions needed to transform the permutation into the identity permutation can be solved in polynomial time using the Hannenhalli-Pevzner algorithm [HP99].

4.1 Counting breakpoints

Before we look at the Hannenhalli-Pevzner algorithm, we first consider a simpler way of measuring the distance between two genomes – counting the number of *breakpoints*. In our examples, we always assume that the genes are labeled so that their order in the first genome is the identity permutation. Then, a breakpoint occurs between two consecutive elements a and b in the second genome iff $b \neq a + 1$. For example, suppose the order of genes in the second genome is

$$-2\ 7\ 6\ 5\ {-4}\ 3\ 1$$

Then, there are six breakpoints. These occur after -2, 7, 6, 5, 4, and 3.

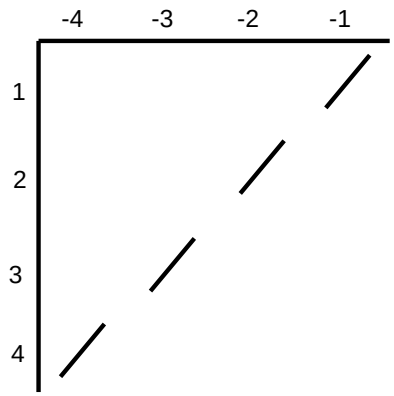
Now suppose the order of genes in the second genome is

$$1\ 2\ 3\ 6\ 7\ {-5}\ {-4}$$

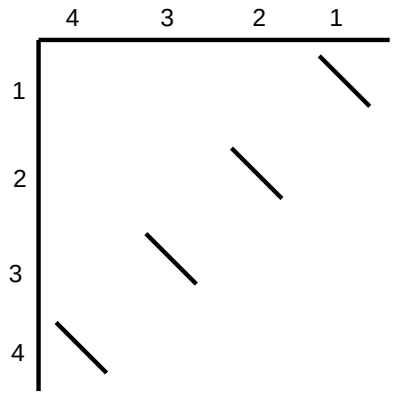
Then, there are two breakpoints. These occur after 3 and after 7.

Breakpoints can be visualized by looking at the dot-plot resulting from the local alignment of the two genomes.

Consider the case where the order of genes in the second genome is -4,-3,-2,-1. We have the following dot-plot:



Informally, we see that the genes in the above dot-plot occur consecutively, with no gene "out-of-position" with respect to the previous one. From this we can see that there are no breakpoints. On the other hand, consider the case where the order of genes in the second genome is 4,3,2,1. We have the following dot-plot:



Here we see that each gene is not correctly positioned with respect to the previous one. From this we can see that there are three breakpoints (one between each pair of consecutive genes).

Note that the number of breakpoints between two genomes can be computed quickly. For three or more genomes, the number of breakpoints can be computed quickly if we use the sum-of-pairs idea (and take the sum of the number of breakpoints between each pair of genomes). While breakpoints are easy to compute, they don't tell us anything about how the genomes evolved.

4.2 Counting inversions using the Hannenhalli-Pevzner algorithm

A more informative metric is obtained using the Hannenhalli-Pevzner algorithm, which counts the number of inversions needed to transform one genome into the other. Again, in our examples we assume that the genes are labeled so that their order in the first genome is the identity permutation. To see the main idea in the algorithm, consider the following example:

$$5\ 2\ 8\ {-1}\ 6\ 7\ {-4}\ 3$$

Note that if we invert elements 2 and 8 (and obtain 5 -8 -2 -1 6 7 -4 3), then element 2 becomes correctly positioned and signed with respect to element -1. Similarly, if we invert element 3, then this element becomes correctly positioned and signed with respect to element -4. Or, if we invert 5 2 8 -1 6 7 (and obtain -7 -6 1 -8 -2 -5 -4 3) then 5 becomes correctly positioned and signed with respect to element -4.

The general idea is to look for pairs of the form $k \ldots -(k+1)$, since inverting all the elements between k and $-(k+1)$ not including k results in $(k+1)$ signed identically to k and positioned immediately to the right of k. Similarly, we look for pairs of the form $(k+1) \ldots -k$, since inverting all the elements between $(k+1)$ and $-k$ not including $-k$ results in $(k+1)$ signed identically to $-k$ and positioned immediately to the left of $-k$. We refer to pairs of either type as *oriented pairs*. Note that as long as a permutation has elements of different signs, it is guaranteed to have at least one oriented pair.

The algorithm works in stages. At each stage, the algorithm identifies an oriented pair, and performs the corresponding inversion. If there is more than one such pair, the algorithm behaves greedily and chooses the pair that maximizes the number of oriented pairs at the next stage. Consider again the example

$$5\ 2\ 8\ {-1}\ 6\ 7\ {-4}\ 3$$

We have three oriented pairs to choose from: (2, -1), (-4, 3), and (5, -4). For each such pair, we count how many oriented pairs we will have at the next stage if this pair is chosen:

- If we choose (2, -1), the resulting permutation is

  $$5\ {-8}\ {-2}\ {-1}\ 6\ 7\ {-4}\ 3$$

  This has 4 oriented pairs: (5, -4), (-8, 7), (-2, 3), and (-4, 3).

- If we choose (-4, 3), the resulting permutation is

  $$5\ 2\ 8\ {-1}\ 6\ 7\ {-4}\ {-3}$$

  This has 3 oriented pairs: (5, -4), (2, -1), and (2, -3).

- If we choose (5, -4), the resulting permutation is

  $$-7\ -6\ 1\ -8\ -2\ -5\ -4\ 3$$

  This has 3 oriented pairs: (1, -2), (-2, 3), and (-4, 3).

Therefore, the algorithm chooses the pair (2, -1).

It can be shown that if this greedy algorithm eventually obtains the identity permutation, then the number of inversions performed by the algorithm is in fact the minimum number needed. However, the algorithm may also reach a point where all elements have the same sign. In this case, there are no oriented pairs to choose from. We do not discuss how to handle this case here, but details can be found in [Ber05]. Nevertheless, the greedy algorithm on its own works in most cases.

References

[Ber05] Annie Bergeron. A very elementary presentation of the Hannenhali-Pevzner theory. *Discrete Applied Mathematics*, 146(2):134–145, 2005.

[HP99] 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, 1999.