Lectures 21, 22, and 23: Phylogenic Trees and Evolution

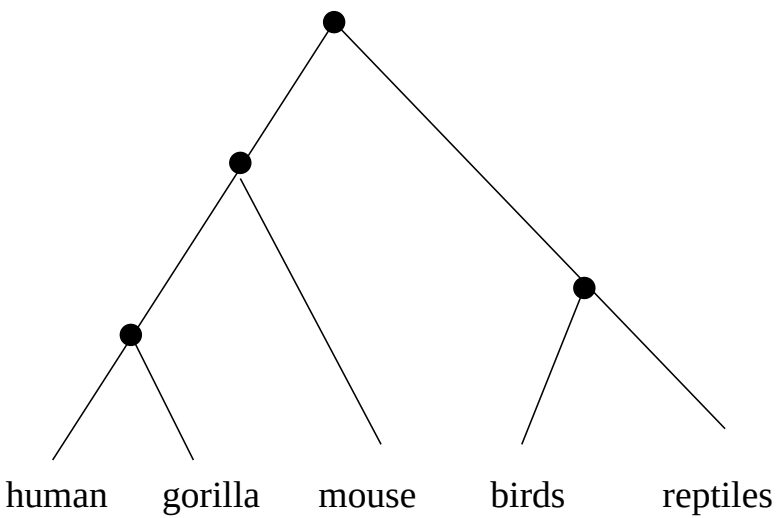
Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.sunysb.edu/∼skiena

Phylogenic Trees

In any evolutionary process, *speciation* events cause a new species to split off from an existing one, thus creating the diversity of life forms we know today.



Reconstructing Evolution

A key issue in evolutionary biology is to reconstruct the history of these speciation events. Given the properties of the leaf nodes, reconstruct what the tree is.

Much of the current interest in phylogenic trees follows from the increasing availability of DNA sequence data.

Biological applications include evolution studies (e.g. the *out of Africa debate*) and medical research (tracing HIV infection).

However, phylogenic trees play an important role in analyzing the history of languages, religions, chain letters, and medieval manuscripts, as well as biology.

What Data is Available?

- *Distance data* measures (directly or indirectly) d_{ij}, the time since species i and j diverged. This can be estimated from the distance between DNA sequences, assuming a 'molecular clock' governing the frequency of mutations.
- *Feature/character data* measures taxonomical properties such as 'warm blooded', 'has wings', or 'walks upright'. Hard-to-develop features describe branch points in the phylogenic tree.

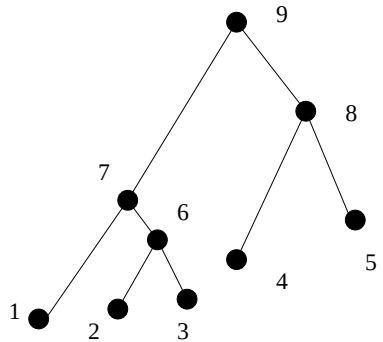
Different types of reconstruction algorithms are necessary for such distinct types of data.

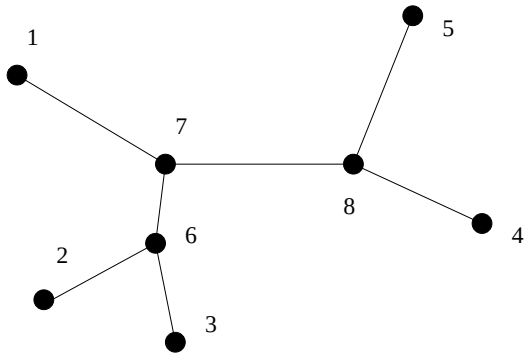
Counting Trees

Observe that there are many tree *topologies* possible for any set of n leaves.

Every *binary* tree on n leaves has $n-1$ *internal* nodes, and thus $2n-1$ vertices and $2n-2$ edges.

Every *unrooted* binary tree n leaves has $2n-2$ vertices and $2n-3$ edges, since the in-degree 0 root can be contacted to a single edge.





Since the root can be positioned at any edge, there are $2n - 3$ more rooted trees than unrooted trees on n leaves.

Further, any rooted tree on n leaves corresponds to an unrooted tree on $n + 1$ leaves, since we can take the highest numbered leaf to be the root.

Tree Counting Recurrence

Thus

$$Rooted[n] = (2n-3)Unrooted[n]$$

$$Rooted[n] = Unrooted[n+1]$$

yielding that

$$Rooted[n] = (2n-3)(2n-5)\ldots 1$$

For $n = 10$, there are about 2,000,000 unrooted trees, and for $n = 20$ about 2.2×10^{20}, so the number of possible topologies grows very fast.

This makes it hopeless to exhaustively search for the best possible tree beyond 10 or so species.

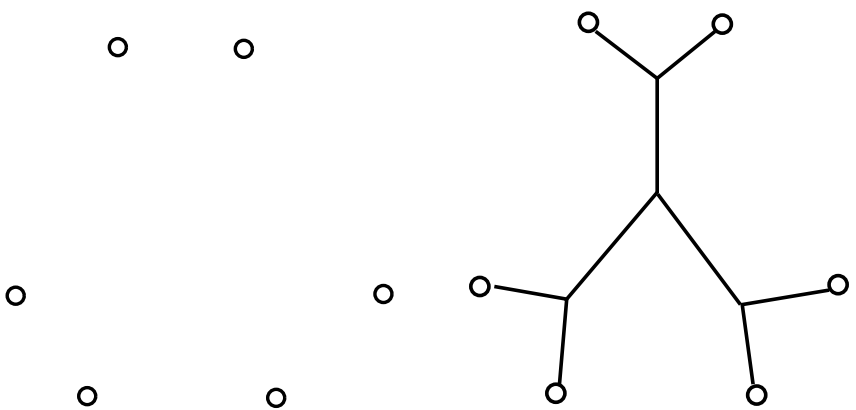
Why is it Difficult?

The business of reconstructing trees is very messy for several reasons:

- Nobody knows the correct underlying trees, but many people have strong and contradictory opinions.
- The data is inherently noisy, error-prone, and hard to interpret.
- Most precise definitions of optimal trees lead to NP-complete problems soon as errors creep into the data.
- Many different tree reconstruction algorithms and heuristics have been proposed, each of which yields different trees on the same data.

Reconstructing Intermediate Nodes

Phylogenic tree problems have the same flavor as *Steiner* tree problems in graphs, where we must deduce the positions of intermediate nodes to find the best possible fit.



Algorithms for Distance Data

Distance data tree construction algorithms bare a strong relationship to *clustering algorithms*, particularly agglomerative algorithms which explicitly construct a tree as they merge clusters together.

Representative algorithms include *nearest neighbor* joining, and repeatedly merging the nearest cluster centroids (the *unweighted pair group method using arithmetic averages* (UPGMA)).

Such algorithms can yield reasonable and informative trees, but there seems no good reason to believe that they yield *the* correct tree.

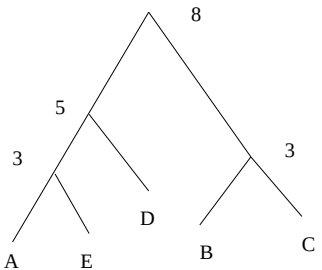
Defining mathematical properties which real trees obey enable us to define optimization criteria which make it plausible to define the *best* tree.

Ultrametric Trees

An *ultrametric* tree is a rooted tree where

- Each internal node is labeled by a number.
- Each internal node has at least two children
- Along any root to leaf path the labels strictly decrease.

Note that if the labels mean “time units ago”, this is true of any evolutionary tree.



	A	B	C	D	E
A		8	8	5	3
B			3	8	8
C				8	8
D					5
E					

Each pairwise distance $d(i, j)$ represents the label of the least common ancestor of i and j.

Reconstructing Ultrametric Trees

There is an efficient $O(n^2)$ algorithm to reconstruct an ultrametric tree, if one exists.

Observe that the labels on the path from a leaf a to the root follow from sorting the distances in row a, since a branching point corresponds to each unique distance.

Shared distances on the path partition the other leaves into groups in the other subtrees.

Thus each of the resulting partitions is fixed, and can be refined by considering other rows.

Unless we get a contradiction, we get an ultrametric tree. Further, this tree must be unique.

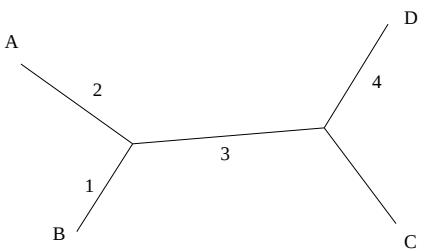
What about Noise?

The problem becomes hard when you seek the “most” ultrametric tree in noisy data. A little noise can throw the topology off considerably.

Additive Trees

A weaker but well defined condition assumes that we have the *distance* between all pairs of leaves, and seek an unrooted, edge-weighted tree such that the sum of the distances along each path adds up to the defined distance.

	A	B	C	D
A		3	7	9
B			6	8
C				6
D				



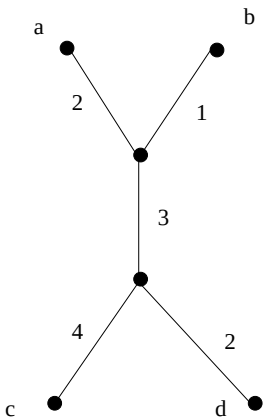
Building Additive Trees

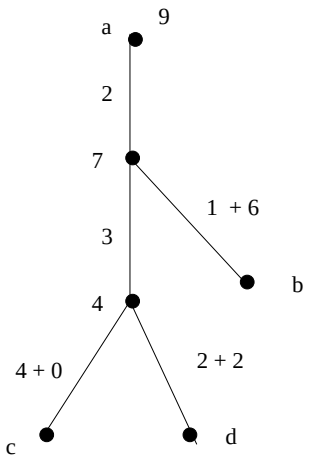
An algorithm for additive tree reconstruction follows from a reduction to ultrametric trees.

First, lift one of the nodes v with a maximum distance entry M_v to the root.

Second, add weight to each leaf edge i so that the total distance from root to each leaf is M_v.

Finally, assign each node the largest weight below it to give us matrix D':

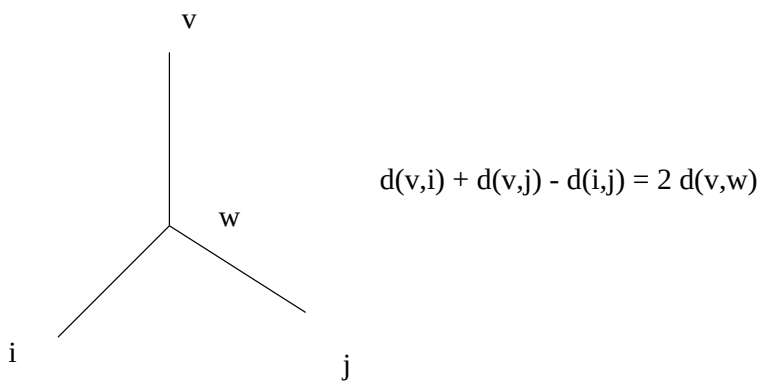




These node labels of D' define an ultrametric tree, since they decrease along each root-to-leaf path.

Thus if we could construct this matrix D' without the knowledge of the tree, we could solve the additive tree problem as an ultrametric tree problem by reversing this construction.

Since the additive input matrix D determines v and M_v, we need to compute the distance in the unknown tree from v to the *least common ancestor* $LCA(i, j)$.



Thus if D is an additive matrix, then D' is an ultrametric matrix, where

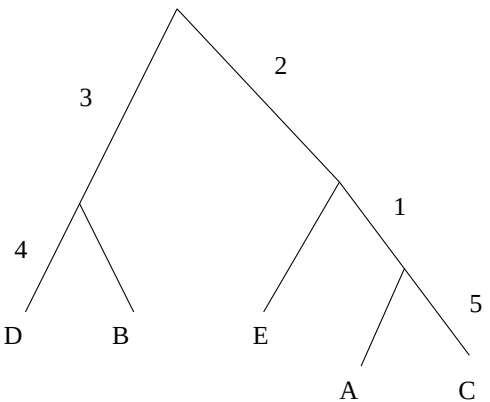
$$D' = M_v(D) + (D(i, j) - D(v, i) - D(v, j))/2$$

The technical assumption that all species in an ultrametric tree are leaves can be restored by hanging v directly off the root.

Perfect Phylogenies

Suppose we have n species, and m features each of which only evolved once (*parsimony*).
Each feature can be represented by an n-element $\{0, 1\}$-vector where $f_i(j) = 1$ iff species j has feature i.
The *perfect phylogeny problem* asks for a phylogenic tree given an $n \times m$ binary feature matrix, if one exists.

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0



Note that this is an edge-labeled tree, and certain features do not necessarily contribute to a split (e.g. 3), particularly when $m > n$.

Reconstructing Perfect Phylogenies

Suppose we know that all features evolve from 0 to 1, i.e. the characters are ordered.

Claim: Matrix M has a perfect phylogenic tree iff for every pair of columns i, j the set of 1s are either (1) disjoint, or (2) i contains j.

If column i *contains* all the 1s of column j, then feature j evolved *before* feature i.

If two columns are disjoint, then the features evolved *independently*.

If species x has feature i but not j, and y has a feature j but not i, then *no perfect phylogeny can exist*.

Perfect Phylogeny Algorithm

This immediately gives an $O(m^2n)$ algorithm to test the matrix for this property, which can be improved to $O(nm)$ using radix sorting.

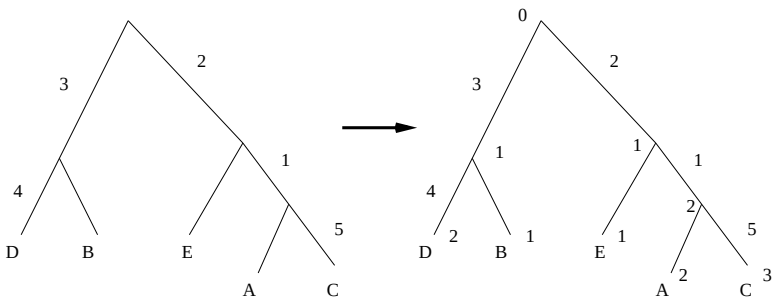
Perfect Phylogeny via Ultrametric Trees

Define an $n \times n$ matrix where $D[i, j]$ equals the number of characters which species i shares with j for a given feature matrix M.

	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	0
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	0

	A	B	C	D	E
A	2	0	2	0	1
B		1	0	1	0
C			3	0	1
D				2	0
E					1

Given a perfect phylogeny for M, we can label each node of the tree with the number of labels encountered from the root:



These numbers are increasing along each root-to-leaf path, so negating them given an ultrametric matrix.

Thus if M has a perfect phylogeny, then D must be ultrametric.

This gives a reconstruction algorithm since ultrametric trees are unique.

Generalizing Perfect Phylogeny

The perfect phylogeny problem can be made more general by allowing non-binary features, e.g. the locomotion feature might be ‘fixed’, ‘crawling’, or ‘walking’.

In general, each feature is one of r states. In *ordered* phylogeny problems, we know the directed sequence of transitions for each character as we move down the tree.

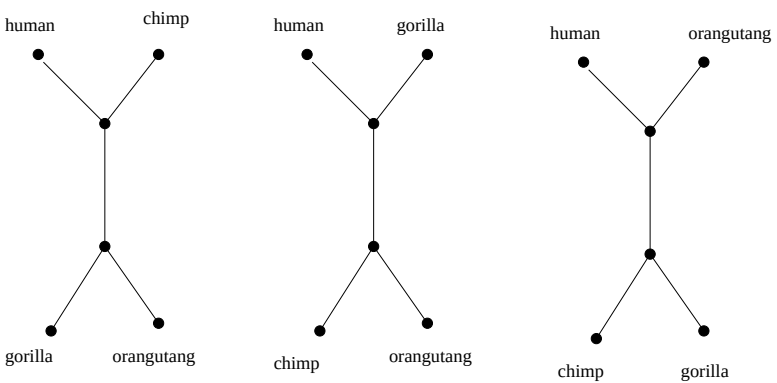
There are polynomial *ordered* perfect phylogeny algorithms for any constant r, i.e. r is in the exponent of the running time.

For unordered problems it is NP-complete for general r, but polynomial for $r = 2$.

Quartet Puzzling

Another approach to reconstructing phylogenic trees is to carefully analyze small subsets of species to reconstruct their relative history, and then integrate a set of resulting trees into a consistent whole.

The smallest interesting unrooted trees contain four species, and are called *quartets*.



There are three possible quartets on any set of species.
The set of $\binom{n}{4}$ quartets induced from any tree uniquely defines the topology of the tree.
Note that rooted triplets can be modeled as quartets if one species is ancestral.
In general, it is difficult to analyze the data to construct all possible quartets in a consistent manner. The problem of constructing the tree maximizing the number of satisfied quartets is NP-complete.

Consensus Trees

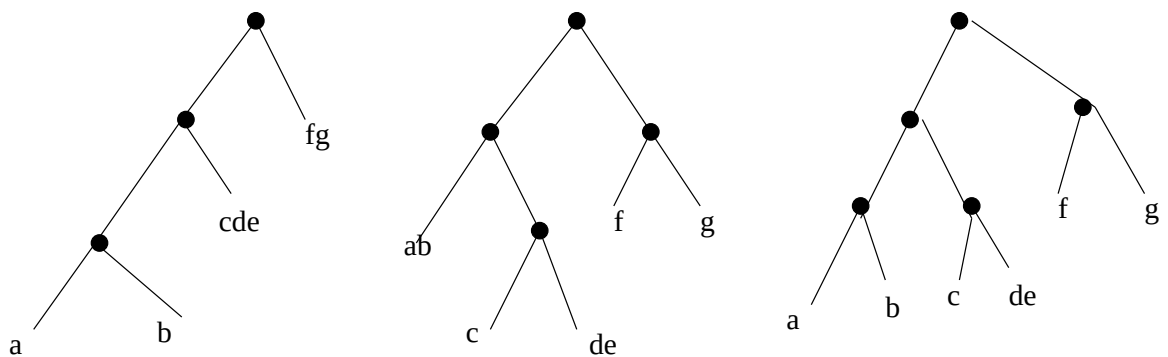
Because the various algorithms and heuristics give different trees on the same data, more evidence is needed than a single tree to define the history.

For this reason, there are suites of programs (e.g. Phylip) which contain implementations of many different tree construction algorithms and heuristics.

Thus there is a need for algorithms which find the *consensus* of a set of trees, i.e. the branch points that all (or most) of the trees share in common.

Tree Compatibility

Also there are *tree compatibility* problems, where we seek the most refined tree which do not contradict evidence from any of a set of partially specified trees:



Genome Rearrangements

Point mutations are only the simplest type of genetic modification event.

Duplication events happen when a second copy of a given gene is inserted into the chromosome. This allows for one copy of the gene to evolve a new function without preventing the production of the protein.

Reversal events happen when a portion of a chromosome is deleted and replaced with the reversed sequence. Such events occur during crossover operations, and can create entirely new genes as well as copies.

Translocation Events

Translocation events happen when genes moved to different locations or chromosomes. Translocated genes may be regulated in a different way in the new location. Genes can even jump across bacterial species (lateral transfer).

Reconstructing evolutionary history requires accounting for these large-scale events.

Where Babies Come From

Each sperm/egg cell contains 23 chromosomes, representing the parent's genetic contribution to their offspring.

Other human cells contain 23 *pairs* of chromosomes, one of each pair being inherited from each parent.

Gametes form a single chromosome from each pair through *recombination*, where a *crossover* operation randomly alternates between the two chromosomes to select which gene copy to pass on.

Such sexual reproduction provides explanations for many things, including why introns may be good, and sex-linked and dominant/recessive diseases.

Errors in this process plausibly account for many genome rearrangement events.

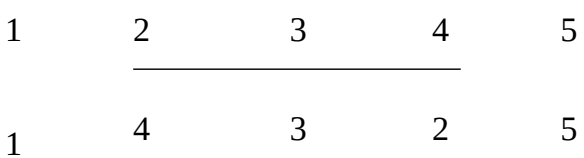
Because there are second copies of most genes, cells can be surprisingly robust in the face of large scale changes.

Cancer cells often lose parts of chromosomes, and certain interesting but non-fatal diseases occur when extra copies of chromosomes occur.

Genome Reversals

As we have seen, large sets of homologous genes exist between pairs of organisms.

Crossover mutations cause *reversals* of a sequence of contiguous genes.



The evolutionary history between two species should reflect the *shortest* sequence of reversals (crossovers) necessary to align all homologous pairs of genes.

Nadeau and Taylor estimate that 178 ± 39 crossovers occurred between mouse and man.

Algorithmic Problems

History reconstruction motivates the problem of *sorting with reversals*, since

- We can reduce each genome to a permutation on 1 to n, where n is the number of homologous genes between the species, and
- By appropriately labeling the genes, one of the permutations can be $1, 2, \ldots, n$.

Any reversal operation changes the *orientation* of all reversed genes. Thus a sorting problem can be *signed* or *unsigned* depending upon whether we know/care or do not know/care about the orientation of each gene.

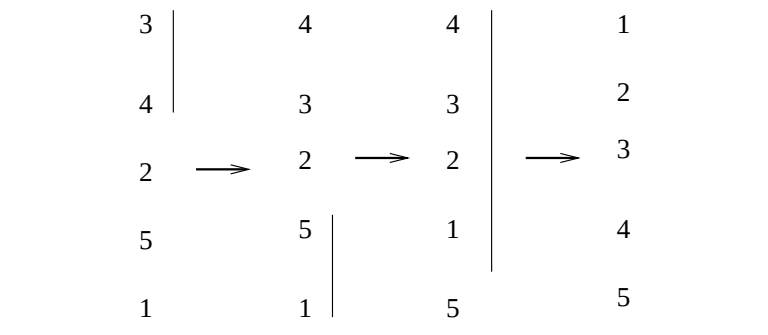
There are two distinct theoretical problems in sorting with reversal problems:

The *diameter* question asks which two length n permutations are farthest apart over all $\binom{n!}{2}$ pairs – how many reversals *always suffice* to sort.

The *distance* question asks, for a given pair p_1, p_2 of length n permutations what is the fewest reversals to bring p_1 to p_2 – how many reversals are necessary.

Pancake Sorting

Suppose we seek to sort a stack of pancakes by size using a spatula. How many *prefix* reversals do we need to sort n pancakes.



By flipping the largest unsorted pancake to the top, and then into position (i.e. selection sort) then any permutation can be sorted in at most $2n$ reversals.

Any strategy must use at least $n - 1$ reversals in the worst case, since each reversal removes at most one *breakpoint*, i.e. (1,3,5,7,2,4,6,8).
Thus the diameter d is between $2n \geq d \geq n - 1$, but tighter bounds are well known...

Approximating Pancake Distance

Any permutation can be partitioned into *strips* of consecutive increasing or decreasing elements.

The gaps between strips are *breakpoints*, and the number of breakpoints is a *lower bound* on the number of reversals needed to sort.

Suppose we modify the selection sort strategy to move the strip containing the largest unpositioned element into position.

One reversal brings the strip to the top, another brings the biggest element in the strip to the top (if necessary), and one last flip puts it into position.

Thus this strategy uses at most three times the number required flips and *approximates* the actual pancake distance for every pair of permutations.

A Better Pancake Heuristic

Suppose instead we just want to merge the strip on top with one of its two neighbors.

If the orientation is correct, one flip suffices; if not two flips suffice.

Since the number of breakpoints is equivalent to the number of strips, this gives a factor 2 approximation.

More careful analysis can lower the constant somewhat, but no polynomial algorithm is known to compute the minimum pancake distance.

Lower bounds on the length of the optimal solution are helpful in reasonably efficient *branch and bound* exhaustive search algorithms.

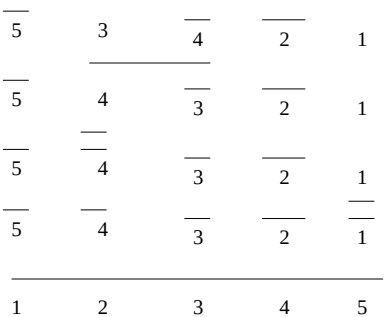
Computing the Reversal Distance

The motivating biological problem demands that we find the shortest sequence of *general* (not prefix) reversals.

Note that there are $\binom{n}{2}$ general reversals but only n prefix reversals, so we have much more freedom.

Caprera proved that computing the reversal distance of *unsigned* permutations is NP-complete.

However, Hannenhali/Pevzner gave a polynomial algorithm for computing reversal distance for *signed* permutations.



The signed case is biologically important since the orientation of genes can be determined from the DNA sequence.
The exact algorithm for signed permutations requires careful, technical combinatorial arguments.

Approximating Unsigned Reversal Distance

Since this problem is NP-complete, we seek an approximation algorithm.

A breakpoint occurs whenever neighboring elements are not consecutive, or with out of position endpoints.

$(3, 2, 4, 5, 1)$ has breakpoints 03, 24, 51, and 16.

Note that the number of breakpoints/2 is a *lower bound* on reversal distance, since any reversal can erase at most two breakpoints.

A strip of consecutive numbers between two breakpoints is *increasing* if the numbers strictly increase. Otherwise it is *decreasing*. A singleton strip will be defined as decreasing.

Claim 1: Any non-identity permutation without decreasing strips has a reversal which does not increase breakpoints but leaves a non-trivial decreasing strip.
Proof: Flip the end strip, e.g. 45123 goes to 45321.
Claim 2: If a permutation has a decreasing strip, then there exists a reversal which decreases the number of breakpoints.
Proof: Match up the smallest endpoint of a decreasing strip.
E.g: $543...12..$ goes to $54321.....$ and $12...543..$ goes to $12345.....$
The following examples don't work – but the decreasing strip isn't smallest: $543...21..$ and $21...543..$

Heuristic and Analysis

If there is a decreasing strip, use one reversal and remove a breakpoint.

If not, use one reversal to create a decreasing strip.

Since one breakpoint is removed every other iteration (at least), twice the number of breakpoints is an *upper* bound on the reversal distance.

Since this is 4 times the lower bound we have a factor 4 approximation.

The approximation factor can be lowered to 1.5 with more careful structural analysis.