1 General Formula

2 Sequence Alignment

2.1 DNA Errors

Substitution: Flip one base for another (Mismatch Error)
Indel: Insert/Delete base in sequence (Gap Error)
Transition: Purine to purine, pyrimidine to pyrimidine.
Transversion: Purine to pyrimidine, vice versa.
Transposon: Delete/Insert whole region from sequence and insert to another

2.2 Similarity Heuristics

Hamming Distance: Number of different positions within string *Ex*: ATCTCA vs ATCACA is 1
Normalized Hamming Distance: $\frac{\text{Hamming Distance}}{\text{len(Seq)}}$
Edit Distance: Minimum-weight series of edit operations transforming a into b
Ex: TGCAGT TCACAGT (Switch G to C, remove A, d= 2)
Substitution Matrix: Cost of switching one base to another (since transitions and transversions cost different amounts)
Levenshtein Distance: Number of edits (indels + substitutions)

2.3 Pairwise Alignment

Pairwise Alignment: Arrange two sequences so similar regions are aligned to each other.

2.3.1 Global Alignment

Solves Global alignment (Overlap two entire sequences completely)
Time & Space Complexity: $O(mn)$
Needleman-Wunsch:

```
Input: Sequences seq1, seq2, match score ($\alpha$), gap penalty ($\mu$), mismatch penalty ($\sigma$)
Output: Alignment score, backtrack array
Initialize alignment + backtrack array of size seq1, seq2
Initialize top row and left column base cases
For every cell in matrix
    if matching bases then add match score to score of left diagonal
    if top/left cell + gap penalty is lowest score, add it
    if left diagonal + mismatch penalty is lowest score, add it
    put direction taken in backtrack array
output alignment score at bottom right cell, and use backtrack array to derive original alignment
```

2.3.1.1 NCBI BLASTn Scoring scheme

Because whole regions get inserted/deleted, having n bases indeled is less costly than n indels. So we make gap opening a high cost, and continuing gap lesser cost.

Consant: Gap Penalty = Constant for all size n
Linear: Gap Penalty = Gap Length x Gap opening
Affine: Gap Penalty = Gap Opening + Gap Extension x (Gap Length - 1)
Convex: Gap Penalty <= Affine Gap Penalty

2.3.2 Local Alignment

Smith-Waterson: Same as **Needleman-Wunsch** but we can start and end anywhere, and the goal is to find the lowest cost path overall. 0 is minimum, so initialize every cell as minimum of a continuation of path and 0, and the highest score is the ending of the alignment. To retrieve, we start at highest cell and go back until we reach 0.

2.3.3 Database Search

k-mer: Continuous block of k characters (def. 11 for nucl and 3 for prot)
High-Scoring Segment Pair: Alignment found by BLAST word match

BLAST:

```
Build sequence into k-mers (continuous block of $k$ characters)
For each k-mer build index hash table of occurrences of all k-mers in target string
For a query, break into k-mers and lookup each k-mer as seed for sequence alignment
    In lookup of k-mers, allow some deviation along threshold of difference
For each seed, extend seq until alignment falls below threshold
```

2.3.3.1 Variations

Gapped BLAST: Extensions are in BLAST are allowed to use gaps.
Two-hit Seeding: You can only extend a sequence if there is another sequence nearby it. Finding two-length words is more likely than a full word, so it's more sensitive.

2.3.3.2 Scoring

Raw Score: Alignment score (S) of string
P value: Probability of alignment score $\gtrless S$ given random strings
E value: Expected number of alignments with score $\gtrless S$ given random strings

2.3.3.3 Alignment Scoring

- Used to compare alignments of same query-reference pair
- Used to compare alignments of diff query-reference pair
- Used to indicate confidence of alignment

2.3.3.3.1 Comparing Alignment of same query-reference pair

Define match (p), substitution (q), indel (r) probability. Probability is $p^m q^s r^i$, log-prob = $m\log(p) + s\log(q) + i(\log)r$

Problems:
- Longer spotty seq's are penalized more than shorter perfect seq's
- Repeated information is penalized (even though it's functionally the same)

2.3.3.3.2 Comparing Alignment of diff query-reference pair

Homology Model: Assume alignment reflects evolution
Random Model: Assume alignment spurred randomly

2.3.3.3.2.1 Homology Model

Probability: $\Pr(x, y \mid H)$
Assume p_{ab} is probability evolution gave char a in x and char b in y $P(x, y \mid H) = \Pi_{i=1}^{n} p_{x_i y_i}$

2.3.3.3.2.2 Random Model

Probability: $\Pr(x, y \mid R)$
Assume p_a is probability to randomly get char a $P(x, y \mid R) = \Pi_{i=1}^{n} p_{x_i} \Pi_{j=1}^{n} p_{y_j}$

2.3.3.3.2.3 Likelihood Ratio

Likelihood Ratio: $P = \frac{\Pr(x,y \mid H)}{\Pr(x,y \mid R)}$
$\sum_i \log\left(\frac{p_{x_i y_i}}{p_{x_i} p_{y_i}}\right)$
Alignment score: $\log(P)$

2.3.3.3.2.4 Substitution Matrix

A matrix used to contain the unique probabilities of switching out one character for another.
BLOSUM62 is empirically found values on actual probabilities of switching out amino acids.

To calculate $\frac{p_{x_i y_i}}{p_{x_i} p_{y_i}}$ take a toy database of values, and calculate probability of getting pair (x, y) over all possible pairs in sequences (Number of pairs (x, y) over number of all pairs) and then divide that by probability of getting x and y individually (number of $\frac{x}{y}$ over all base pairs in list). Then substitution matrix only takes log values, so take the log.

We can use substitution matrix to simplify calculating values $\sum_{x_i, y_i \in \text{LCS}} s(x_i, y_i) = \log(P)$

2.3.3.4 Open Vocabulary Problem (Byte-Pair encoding)

To increase ability to scan sequence, take index of k-mers, sorted by frequency, and add that as a new "character". Iteratively, we reduce the number of tokens to scan.

2.4 Multi-sequence Alignment

Align k sequences

2.4.1 Sum-of-Pairs

Do pairwise sequence alignment for every pair of sequences, and take the sum for a global alignment score, then optimize over that MSA for the alignment. Takes $O(n^2 k^2)$ time

2.4.2 Centre-Star Method

Choose one sequence as center sequence by doing PSA on all pairs and find one sequence that has maximum of all alignment scores. Then combine all the sequences by determining the gaps in each sequence at which position, and then insert them in every sequence. Ex: AC-GA, TC-AG, A-TCG => A-C–GA, T-C–AG, A-T–CG

This doesn't really concern alignment between non-centre sequences.

2.4.3 Progressive Alignment

Greedy algorithm that is a heuristic that doesn't visit all solutions but still gives good answer.

Do pairwise alignment on all sequence pairs, $O(k^2 n^2)$. Then build distance matrix among sequences via some heuristic. Then build a guided tree by some clustering algorithm and then select some starting sequence according to some optimal criteria, and align between node pairs, merging recursively for $k - 1$ alignments.

2.4.3.1 Distance Matrix

Matrix with zero on diagonal, and non-negative values, symmetric, and satisfies triangle inequality

2.4.4 Aligning new sequence

Then, if we have a new sequence to add, we gather our alignment of all the sequences, and capture the frequency of each possible character at each position for each character. Then the new "sequence" is the greatest frequency at each position, which you align to.

3 Phylogeny

MRCA: Most Recent Common Ancestor **Internal Nodes**: Ancient hypothetical species **Leaf Nodes**: Extant species
Neighbour: Two species that are most closely related on tree
Outgroup: **Clade**: Group of organisms with common ancestor
Outgroup: Set of organisms that are not in the "ingroup", and are distant in the tree. **Homolog**: Descendant of common ancestor **Analog**: Similar but don't share ancestry **Ortholog**: Speciation event, where common ancestors break into two species **Paralog**: One gene breaks into two separate versions in same genome **Phylogeny**: Construction of evolutionary tree by evolutionary relationship **Morphology**: Construction of evolutionary tree by physical featuers

3 problems: **Topology**, **Root**, **Branch Lengths**

Cladogram: Tree where branch length has no meaning
Phylogram: Tree where branch length conveys genetic change **Ultrametric tree**: Tree where branch length conveys time

Hierarchical Clustering: Arrange sequences based on pairwise distance in hierarchical manner

Well-behaved tree: All nodes have equal distance from root to leaves (ultrametric), the distance is defined as "time"

3.1 Distance-based methods

3.1.1 UPGMA

UPGMA: Unweighted Pair Group Method with Arithmetic Mean

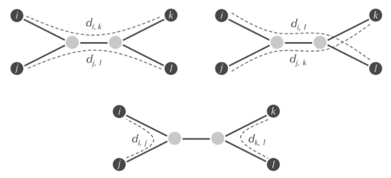
Linkage Criterion: Distance metric between two groups (not just sequences, but 2 separate groups of sequences)

Given distance matrix, find closest pair of species, then combine them to a "new node" and recalculate the values for it and the remaining sequences by $\text{UPGMA} = \frac{\text{Sum of pairwise distances}}{\text{Number of pairs}}$ and continue until you're done.

This best describes topology and distance of species.

3.1.2 Neighbour Joining

Additive Matrix: There exists an additive tree fitting the matrix
Additive Tree: Given symmetric $n \times n$ 0-diagonal matrix, for any neighbouring leaves i, j, with parent m, the distance from m to any other leaf k is $d_{k,m} = \frac{d_{i,k} + d_{j,k} - d_{j,l}}{2}$ **Four-Point Theorem**: Distance matrix is additive iff two of the sums are equal and third is less than or equal to other sums:



3.1.2.1 Procedure

Create auxiliary matrix where $Q(i, j) = (n-2)d_{i,j} - \sum_{k \in L} d_{i,k} - \sum_{k \in L} d_{j,k}$. Then find elements with a minimum non-diagonal value. Then, let $\Delta = \frac{\text{Total Distance}(i) - \text{ Total Distance}(j)}{n-2}$, then $\text{limbLength}_i = \frac{1}{2}(D_{i,j} + \Delta)$ and $\text{limbLength}_j = \frac{1}{2}(D_{i,j} - \Delta)$ Then, add a new row/column to D, so that $D_{k,m} = D_{m,k} = \frac{1}{2}(D_{k,i} + D_{k,j} - D_{i,j})$ and remove rows i and j from D. Then repeat recursively and add to the graph as needed.

Given set of leaf nodes, for every pair, define new q-metric relative to other species, as $D_{ij} - (r_i + r_j), r_i = \frac{1}{|L|-2} \sum_{k \in L} d_{ik}$

Then, find the smallest q-metric and combine the two species to make a new internal node as average of the two.

3.1.3 Jukes-Cantor

Jukes-Cantor is a tool to recompute distances based on back-mutations for more accurate phylogenetic trees.

Assumptions:
1. All nucleotide bases occur with equal probability
2. Ech base has equal probability of mutating into any other
3. Substitutions occur independently

Let $p_{A(t)}, p_{C(t)}, \ldots$ is probability that a given site contains nucleotide $A, C, \ldots$ at time t.

Since we assume all nucleotides are equally likely, $p_{X(t)}$ is probability that a site is $X(C, G, T)$ at time t

$\sum_{\delta \in L} p_\delta(t) = 1$ and if a nucleotide is A at time $t = 0$, then $p_{A(0)} = 1, p_{C(0)} = p_{G(0)} = \ldots = 0$

Define mutation velocity away from A as α, then mutaiton velocity from A to C is $\frac{\alpha}{3}$, same with all other nucleotides.

For infinitesimal time step dt, $p_{A(t+dt)} = \text{probability of staying A} = p_{A(t)} \cdot (1 - \alpha dt) + p_{C(t)} \cdot \left(\frac{\alpha}{3}\right)dt + \ldots + p_{T(t)} \cdot \left(\frac{\alpha}{3}\right)dt$

Similarly, $p_{X(t+dt)} = \text{probability of becoming C, G, or T} = p_{X(t)}(1 - \alpha dt) + p_{A(t)}\left(\frac{\alpha}{3}\right)dt + 2p_{X(t)}\left(\frac{\alpha}{3}\right)dt$

Taking the limit as $dt \to 0$, $\frac{dp_A}{dt} = -\frac{4\alpha}{3}p_A + \frac{\alpha}{3}$ and $\frac{dp_X}{dt} = \frac{\alpha}{3}p_A - \frac{\alpha}{3}p_X$

Then we solve to get $p_{A(t)} = \frac{1}{4} + \frac{3}{4}e^{-\frac{4\alpha}{3}t}$ and $p_{X(t)} = \frac{1}{4} - \frac{1}{4}e^{e^{-\frac{4\alpha}{3}t}}$

Thus, the expected number of substitutions per site is: $d = -\frac{3}{4}\ln\left(1 - \frac{4}{3}p\right)$ which is our new distance function. $p = 3P_x$ as the observed fraction of nucleotide differences between sequences, and d is assumed to be αt

3.2 Character-based Method

3.2.1 Maximum Parsimony

Score trees based on the minimum number of substitutions required to convey the found tree.

Pseudocode: Given tree and alignment column u, label interal nodes to minimize substitutions. **Initialization**: Set $C = 0$ and root node $k = 2N - 1$ **Iteration**: If k is a leaf, set $R_k = \{x_{k(u)}\}$ If k is an internal node with children i, j: If $R_i \cap R_j \neq \emptyset, \text{set } R_k = R_i \cap R_j$ Else

Note that $x_{k(u)}$ means the u'th base pair for node k. Then, for internal nodes, you take the intersection as the list of possible values that wouldn't require a substitution. If there is nothing, then join them, so that you only require one substitution.

3.2.2 Maximum Likelihood

The likelihood of a tree T with branch lengths t and given sequence alignment is $L(T, t) = P(x_1, \ldots, x_n \mid T, t)$

We want to find the most likely tree given sequence alignments. Note, $L(x_1, x_2) = \sum_X \pi_X P(x_1 \mid X, t_1) P(x_2 \mid X, t_2)$

π_X is stationary probability of nucleotide X, X is possible ancestral states at root. $P(x_i \mid X, t_i)$ is probability of transition from X to x_i in time t_i

Pseudocode: **Initialization**: At each leaf node, define $L_{i(X)} = 1$ if $X = x_i$, 0 otherwise. **Recursion**: For each internal node with children i, j $L_{k(X)} = \sum_{Y,Z} P(X \to Y, t_i) L_{i(Y)} \cdot P(X \to Z, t_j) L_{j(Z)}$ **Final Step**: $L(T) = \sum_X \pi_X L_{\text{root}}(X)$

3.2.3 Tree Space Search

Stepwise Addition: Start from minimum tree (3 taxa) and add taxa, then optimize topology. **Use starting tree**: A tree with all taxa but is less accurate. Either use a cheap method for construction or randomly create.

3.3 Summary

Methods	Optimality Criterion
Distance Methods	Distance
Maximum Parimsony	Parimsony

Methods	Optimality Criterion
Maximum Likelihood	Likelihood

Note top-to-bottom is increasing accuracy and decreasing efficiency.

4 Genome Mapping

4.1 Sequencing

We get sequencing from either Sanger sequencing, Next Generation Sequencing, to produce DNA fragments.

Long Reads (Pacbio/Nanopore) typically have a high error rate, and short reads (Illumina) have a bigger computational task.

4.2 Assembly

The goal is to derive the original string from a collation of fragments.

Coverage: Number of fragments overlapping for a specific position of the original sequence.

Note: If there is a lot of overlap between the end of one fragment and start of another, it is likely that they were overlapping.

4.2.1 Long Read

1. Reads
2. Build overlap graph
3. Bundle stretches of the overlap graph into **contigs**
4. Pick most likely nucleotide sequence for each contig
5. Contig

Contig: Set of DNA segments that overlap in a way that provides contiguous representation of genomic region.

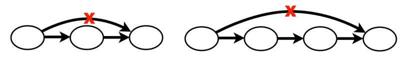
k-mer composition: Collection of all k-mer substrings of the Text, including repeats.

k-mer Prefix/Suffix: First/Last $k-1$ nucleotides of k-mer

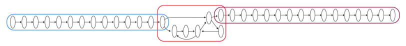
Then connect all k-mers of the text in a graph, where for given (Text) nodes with Suffix(Text) = Prefix(Text') should have an edge between them.

Then, to solve this problem, we want a Hamiltonian path (going through every node) to solve, so we can use heuristics to solve.

For instance, remove transitively-inferrible edges, edges skipping one or two nodes.



Then, create a contig for all non-branching stretches.



Then, derive a consensus by using sequence alignment.

4.2.2 Short Read

1. Error Correction
2. Graph Construction
3. Graph Cleaning
4. Contig Assembly
5. Scaffolding
6. Gap Filling

Error Correction means preprocessing reads to find errors and only accepting exact matches and try to replace rare k-mers with common k-mers.

4.2.2.1 De Brujin

Graph Consruction: Given collection of k-mers P, define nodes as all unique $k-1$-mers that are prefixes or suffixes of k-mers in P and for each k-mer in P create a directed edge from its prefix to its suffix.

Euler's Theorem: Every balanced, strongly connected directed graph has a cycle that visits every edge exactly once.

Form a cycle by randomly walking in a graph, and not revisiting edges. While unexplored edges remain, select a node, newStart, in Cycle with unexplored edges and form a new cycle by traversing Cycle starting at newStart, then randomly walk to include new unexplored edges.

Note that you can turn an Eulerian path into an Eulerian cycle by adding an edge between two unbalanced nodes making it a balanced graph.

Graph Cleaning: Remove "sink" nodes in De Brujin graph and combine divergence structures that converge to a single node later through the graph.

Contig Assembly: Use Euler algorithm to identify strongly connected regions in the genome and use thae path to determine the full contig.