Bioinformatics

University of Cambridge

Ashwin Ahuja

Computer Science Tripos
Part II

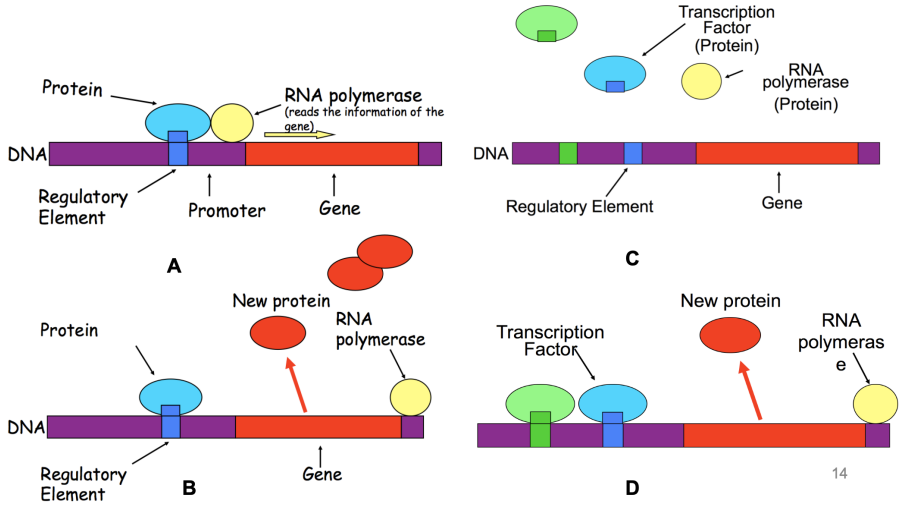
January 2020

Contents

1 Purpose

Main purpose is to help doctors and help to understand the biology and computing with DNA and other biological molecules. Can use DNA as storage information. Today, through *Oxford Nanopore* and *Bento Lab* sequencing has got much cheaper and quicker. Idea of **Garage Genomics** where it often only takes an hour to get a portion of the DNA.

- DNA made of four-letter alphabet: Adenine, Thymine, Cytosine and Guanine - NB. it contains information for it's self-assembly
- RNA is the same with Uracil instead of Thymine.
- **Amino Acid**: Made of three letters (codon) of DNA - can be represented as a 3D labelled graph. 20 types - some combinations of three are repeated in defining the amino acids
  - DNA $\xrightarrow{\text{transcription}}$ RNA $\xrightarrow{\text{translation}}$ Protein
- Protein: composed of multiple amino acids
- Genome: organism's genetic material - for humans = 46 strings (chromosomes) with length 3 x 10e9
- Gene: hereditary information located on the chromosomes and consisting of DNA. These are activated or repressed by regulatory proteins which bind to gene flanking sequences (promoter) and are coded by the same or other genes. We can track expression levels using chips.



DNA can be used for a number of tasks - with DNA based techniques being used by Adleman for simple tasks. In particular, can use it for:

1. Nanocommunications
2. To make circuits and new injectable devices

1.1 Adleman Travelling Sales Problem

1. Generate all possible routes
2. Select itineraries that start with the proper city and end with the final city
3. Select itineraries with the correct number of cities
4. Select itineraries that contain each city only once

Idea is to sort DNA by length and select DNA whose length corresponds to 7 cities. Use gel made of polymer with meshwork - DNA forced to thread way through tiny spaces between strands. Speeds and slows DNA at different rates depending on length - end up running gel with each band corresponding to a certain length. Then cut out band of interest. Amplify this using Polymerase Chain Reaction. Then use complementary primers to start and stop cities, which all encode different itineraries. To isolate a single specific sequence, we use a technique called affinity purification - by attaching compliment of a sequence to a substrate

- Memory is much easier to get lots of in DNA
- Parallel operations
- Would require too much DNA still
- Expensive

1.2 DNA as Information Storage

Read write speed is still rather low but has a very high data density. Can also use Synthetic DNA instead of actual DNA. Principles of DNA Information Storage:

1. Two files stored by encoding each file in a set of different DNA sequences. Redundant information added to enable error recovery at retrieval and distanct primer s appended to each set of sequences.
2. Specific file is retrieved by amplifying molecules with ePCR

Organick et al. designed a clustering and consensus algorithm that aligns and filters reads before error correction. Also takes into account reads that differ from the correct length. It described large-scale random access, low redundancy and robust encoding and decoding of information. Allows 200MB data.

1. **Encoding**: Break into large set of 150-nucleotide DNA sequences. Uses Reed-Solomon code redundancy to overcome errors in synthesis and sequencing. Resulting collection synthesized. Random access starts by amplifying subset using PCR which are then sequenced. Finally sequencing reads are decoded using clustering, consensus and error correction algorithms
2. Starts by randomising data to reduce chances of secondary structures, primer-payload non-specific binding and improved properties during encoding. Breaks into fixed-length payloads, adds addressing information and applies outer coding. Reed-Solomon to increase robustness to missing sequences and errors. Next applies inner coding
3. Decoding process starts by clustering reads based on similarity and finding consensus between the sequences in each cluster to reconstruct the original sequences, which then decoded back to digital data

2 Alignment

Alignment is a problem for both DNA and Protein Sequences. Input is DNA or protein sequences and the output is a set of aligned positions that makes it easy to identify patterns. Following things can happen to DNA:

1. Matches
2. Insertions
3. Deletions
4. Mismatches - mutations

2.1 Longest Common Subsequence

Allows only insertions and deletions (no mismatches). Idea is that every common subsequence is a path in a 2D grid, path with maximum number of diagonal edges is the Least Common Subsequence.

Edit Distance: Minimum number of operations to transform a string into another

$$s_{i,j} = \max \begin{cases} s_{i-1,j}+0 \\ s_{i,j-1}+0 \\ s_{i-1,j-1}+1, & \text{if } v_i = w_j \end{cases}$$

```
LCS(v, w)
1  for i ← 0 to n
2      s_{i,0} ← 0
3  for j ← 1 to m
4      s_{0,j} ← 0
5  for i ← 1 to n
6      for j ← 1 to m
7          s_{i,j} ← max { s_{i-1,j}
                         { s_{i,j-1}
                         { s_{i-1,j-1} + 1,   if v_i = w_j
8          b_{i,j} ← { "↑"   if s_{i,j} = s_{i-1,j}
                     { "←"   if s_{i,j} = s_{i,j-1}
                     { "↖",  if s_{i,j} = s_{i-1,j-1} + 1
9  return (s_{n,m}, b)
```

```
PRINTLCS(b, v, i, j)
1  if i = 0 or j = 0
2      return
3  if b_{i,j} = "↖"
4      PRINTLCS(b, v, i − 1, j − 1)
5      print v_i
6  else
7      if b_{i,j} = "↑"
8          PRINTLCS(b, v, i − 1, j)
9      else
10         PRINTLCS(b, v, i, j − 1)
```

Adding in costs:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \end{cases}$$

However, as length of sequences go up, finding this alignment is very time intensive

2.2 Needleman-Wunsch

Idea is that we have a penalty such that we are pushed towards using the match case rather than insertion and deletion cases, where there are three possible cases:

1. x_i aligns to y_j - match => F(i, j) = F(i-1, j-1) + m if $x_i == y_j$ else F(i, j) = F(i-1, j-1) - s
2. x_i aligns to a gap => F(i, j) = F(i-1, j) - d
3. y_j aligns to a gap => F(i, j) = F(i, j-1) - d

In order to decide which of the cases is correct, we use induction:
Inductive Assumption: F(i, j-1), F(i-1, j), F(i-1, j-1) are optimal, then:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

Where:

$$s(x_i, y_j) = if\, x_i = y_j : m, else : -s$$

2.2.1 Global Alignment Problem

Find the longest path between vertices (0,) and (n, m) in edit graph OR find highest-scorind alignment between two strings using a scoring matrix. Needleman-Wunsch algorithm used to solve Global Alignment is as follows:

1. Initialization.
   a. F(0, 0) = 0
   b. F(0, j) = - j × d
   c. F(i, 0) = - i × d

2. Main Iteration. Filling-in partial alignments — d is a penalty
   a. For each i = 1……M
      For each j = 1……N

$$F(i,j) = \max \begin{cases} F(i-1,j) - d & \text{[case 1]} \\ F(i,j-1) - d & \text{[case 2]} \\ F(i-1,j-1) + s(x_i, y_j) & \text{[case 3]} \end{cases}$$

$$Ptr(i,j) = \begin{cases} UP, & \text{if [case 1]} \\ LEFT & \text{if [case 2]} \\ DIAG & \text{if [case 3]} \end{cases}$$

3. Termination. F(M, N) is the optimal score, and from Ptr(M, N) can trace back optimal alignment

It has complexities:

- Space = O(mn)
- Time = O(mn)

  Filling matrix = O(mn)

  Backtrace = O(m+n)

Overlap Detection Variant: When it's acceptable to have an unlimited number of gaps in the beginning and end, changes:

- **Initialisation** to: $\forall i,j : F(i,0)=0, F(0,j)=0$
- **Termination** to: $F_{OPT} = \max \begin{cases} \max_i F(i,N) \\ \max_j F(M,j) \end{cases}$

2.3 Smith-Waterman

2.3.1 Local Alignment Problem

Find the longest path (or path with max or a certain matrix score) among paths between arbitrary vertices in the edit graph - effectively is global alignment in a subrectangle.

- **Initialisation**: $F(0,0)=F(0,j)=F(i,0)=0$
- **Iteration**: $F(i,j) = \max \begin{cases} 0 \\ F(i-1,j)-d \\ F(i,j-1)-d \\ F(i-1,j-1)+s(x_i,y_j) \end{cases}$
- **Termination**
  - If we want best local alignment
    $F_{OPT} = max_{i,j} F(i,j)$
  - If we want all local alignments scoring > t
    $\forall i,j$ find F(i, j) > t and trace back

2.3.2 Scoring Gaps

Change the penalty and the gain based on the challenge in certain mutations, eg, can use the likelihood of mutation from one protein to another.

Affine Gap Penalty for gap of length k: $\sigma + \dot{\epsilon}(k-1)$ where:

- σ = Gap Opening Penalty
- ϵ = Gap Extension Penalty - $\sigma > \epsilon$

Alignment with Gaps

- **Initialisation**: same
- **Iteration**:

$$F(i,j) = \max \begin{cases} F(i-1,j-1)+s(x_i,y_j) \\ \max_{k=0\ldots i-1} F(k,j)-\gamma(i-k) \\ \max_{k=0\ldots j-1} F(i,k)-\gamma(j-k) \end{cases}$$

$$\gamma(n) = d+(n-1)\times e$$

- **Termination**: same
- **Running Time**: O(N^2M)
- **Space**: O(NM)

2.4 Affine Gap

- $\gamma(n) = d + (n-1) \times e$
- $F(i,j)$ is the score of the alignment $x_1,...,x_i$ to $y_1,...,y_j$ if x_i aligns to y_j
- $G(i,j)$ is the score if x_i or y_j aligns to a gap
- **Initialisation**: F(i, 0) = d + e(n-1), F(0, j) = d + e(j-1)
- **Iteration**

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x_i,y_j) \\ G(i-1,j-1) + s(x_i,y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

$$G(i,j) = max \begin{cases} \text{G(i,j}-1) - \text{e} \\ \text{G(i}-1,\text{j}) - \text{e} \end{cases}$$

2.5 Banded Dynamic Programming

If x and y are similar - path of alignment close to diagonal if few gaps: Assuming gaps(x, y) < k(N) - Time, Space = O(N × k(N)) « O(N^2)

Initialisation: F(i, 0), F(0, j) undefined for i, j > k

Iteration: For i = 1, ..., M and j = max(1, i-k), ..., min(i+k)

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i,y_j) \\ F(i,j-1) - d, \text{ if } j > i - k(N) \\ F(i-1,j) - d, if\, j < i + k(N) \end{cases}$$

Space complexity of computing just the score itself is O(n) - only need previous column to calculate current column and can then throw away that previous column once we're done using it.

2.5.1 Prefix

Prefix(i) is the length of the longest path from (0, 0) to (i, m/2) - simply done by doing dynamic programming in the left half of the matrix

2.5.2 Suffix

Suffix(i) is the length of longest path from (i, m/2) to (n, m). Dynamic Programming in right half of reversed matrix is the main way of doing this. N.B. for both this and prefix, only need to store two columns of scores at any one point.

Can combine the two prefix and suffix to therefore have a middle vertex. Length(i) is the length of the longest path from (0, 0) to (n, m) that passes through vertex (i, m/2)

Can use a **divide and conquer approach to sequence alignment** by:

1. find MiddleNode - node where optimal alignment path crosses the middle column = $max_{0 \leq i \leq n} length(i)$
2. Recurse on first and second half

NB: finding longest path in alignment graph requires storing all backtracking pointers - O(mn) memory. But, finding length of longest path in the alignment graph is O(n) memory, hence the entire solution can be done in linear space.

Linear Space Alignment

```
LinearSpaceAlignment(top,bottom,left,right)
  if left = right
    return alignment formed by bottom-top edges "↓"
  middle ← ⌊(left+right)/2⌋
  midNode ← MiddleNode(top,bottom,left,right)
  midEdge ←  MiddleEdge(top,bottom,left,right)
  LinearSpaceAlignment(top,midNode,left,middle)
  output midEdge
  if midEdge = "→" or midEdge = "↘"
    middle  ← middle+1
  if  midEdge = "↓" or midEdge = "↘"
    midNode  ← midNode+1
  LinearSpaceAlignment(midNode,bottom,middle,right)
```

N.B. We can compute the edit distance faster than O(nm) - O($\frac{n^2}{logn}$)

2.6 Nussinov RNA Folding

Secondary Structure is the topology of local segments. It is represented as a set of paired positions, telling us which bases are paired. Four possibilities:

1. i and j are a pair
2. i is unpaired
3. j is unpaired
4. bifurcation - they belong to two different loops

Assumption is that there are no pseudo-knots

2.6.1 Algorithm

1. **Initialisation**:
   (a) $\gamma(i,i-1)=0$ for i = 2 to n
   (b) $\gamma(i,i)=0$ for i = 1 to n
2. $$\gamma(i,j)=\max\begin{cases}\gamma(\mathrm{i+1,j})\\ \gamma(\mathrm{i,j-1})\\ \gamma(\mathrm{i+1,j-1})+\delta(\mathrm{i,j})\\ \max_{\mathrm{i<k<j}}[\gamma(\mathrm{i,k})+\gamma(\mathrm{k+1,j})]\end{cases}$$
   d(i, j) = 1 if x_i and x_j are a complementary base pair and d(i, j) = 0 otherwise
3. Take highest value and then traceback

Algorithm: Nussinov RNA folding, traceback stage
Initialisation: Push $(1, L)$ onto stack.
Recursion: Repeat until stack is empty:
- pop (i, j).
- if $i >= j$ continue;
  else if $\gamma(i+1, j) = \gamma(i, j)$ push $(i+1, j)$;
  else if $\gamma(i, j-1) = \gamma(i, j)$ push $(i, j-1)$;
  else if $\gamma(i+1, j-1)+\delta_{i,j} = \gamma(i, j)$:
    - record i, j base pair.
    - push $(i+1, j-1)$.
  else for $k = i+1$ to $j-1$: if $\gamma(i,k)+\gamma(k+1, j) = \gamma(i, j)$:
    - push $(k+1, j)$.
    - push (i, k).
    - break.

N.B. There can be multiple optimal substructures

Complexities

- O(n^3) in time
- O(n^2) in space

3 Trees and Phylogeny

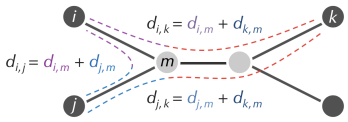
3.1 Distance Matrices to Evolutionary Trees

Phylogeny is comparison of sequences to determine the hierarchical structure. Distance matrix if the number of differing symbols of a multiple alignment.

Idea for trees is that the leaves (degree = 1) are the present-day species and internal nodes (degree $\geq$ 1) are the ancestral species. Most recent common ancestor is the root of the tree. **Simple Tree**: tree with no nodes of degree 2 - every simple tree with at least two nodes has at least one pair of neighbouring leaves

Distance-Based Phylogeny: Construct evolutionary tree from distance matrix. There is a unique simple tree fitting an additive matrix (distance matrix st there exists an unrooted tree fitting it)

However, distance-based algorithms for evolutionary tree reconstruction says nothing about ancestral states - we lost information when we converted a multiple alignment to a distance matrix.



$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$
$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j}) / 2$
$d_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$
$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j}) / 2$
$d_{i,m} = (D_{i,k} + D_{i,j} - D_{j,k}) / 2$

$$d_{i,m} = \frac{1}{2}(D_{i,k} + D_{i,j} - D_{j,k})$$

BUT, this does not always work when we don't select nodes which are actually neighbours! Hence, switch to considering limbs

3.2 Additive Phylogeny

Limb Length Theorem: LimbLength(i) = 1/2 min($D_{i,k} + D_{i,j} - D_{j,k}$) over all leaves j and k. Additive Phylogeny done with a four point condition. Aim is to find two things that aren't neighbours with i

3.2.1 Algorithm

1. Pick arbitrary leaf j
2. Compute its limb length
3. Subtract LimbLength(j) from each row and column
4. Remove that leaf from matrix
5. Recurse on this and attach j in the correct place with length LimbLength(j)

3.3 Least Squares Distance Based Phylogeny

Sum of Squares Error

$$\text{Discrepancy}(T, D) = \Sigma_{1 \leq i < j \leq n} (d_{ij}(T) - D_{ij})^2$$

Aim: Given distance matrix find the tree that minimises the sum of squared errors - NP-Complete Problem!

3.4 Ultrametric Evolutionary Trees

Ultrametric Tree: distance from root to any leaf is the same. This plays well with the idea that the distance to the most recent common ancestor is the same for each current species.

3.4.1 UPGMA

UPGMA produces an approximation - but creates a rooted ultrametric tree

1. Form cluster for each present species containing a single leaf
2. Find two closest clusters according to (|C| is the number of elements in C):

$$D_{\text{avg}}(C_1,C_2) = \Sigma_{i \text{in} C1, j \text{ in } C2} D_{ij} / |C_1| \cdot |C_2|$$

3. Merge C_1 and C_2 into one cluster C
4. Form new node for C and connect to C_1 and C_2 by an edge. Age of C is $0.5 D_{avg}(C_1,C_2)$
5. Update distance matrix by computing the average distance between each pair of clusters
6. Iterate until a single cluster contains all species

Complexities

- Space: O(n^2)
- Time: O($n^2 log(n)$) as we need to the sorting of the distances of the clusters

3.5 Neighbour Joining

Given n x n distance matrix D, neighbour-joining matrix is:

$$D^*_{i,j} = (n-2) \cdot D_{i,j} - \text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)$$

Given $TotalDistance_D(i)$ is the sum of distances from i to all other leaves. This gives the property that the if D is additive then the smallest element of D* corresponds to neighboring leaves in Tree(D)

3.5.1 Algorithm

1. Construct neighbour-joining matrix D* from D
2. Find minimum element $D*_{i,j}$ of D*
3. Compute:
$$\Delta_{i,j} = (\text{TotalDistance}_D(i) - \text{TotalDistance}_D(j))/(n-2)$$
4. LimbLength(i) = $0.5(D_{i,j} + \Delta_{i,j})$ and LimbLength(j) = $0.5(D_{i,j} - \Delta_{i,j})$
5. Form D' by removing i, j row / column and adding row and column m, st $\forall k, D_{k,m} = 0.5(D_{i,k} + D)j,k - D_{i,j})$
6. Recurse to produce Tree(D')
7. Reattach limbs of i and j

Works precisely with additive matrices, producing an approximation (as expected) for non-additive matrices.

3.5.2 Complexities

Space: O(n^2)
Time: O(n^3)

3.6 Character-Based Tree Reconstruction

Evolves from the realisation of loss of information from conversion of multiple alignment to a distance matrix. This time we produce a tree of the exact character changes every level we move up.

Parsimony Score: Sum of Hamming Distances as we go along each edge

3.7 Small Parsimony Problem

Hence, **Small Parsimony Problem**: Find most parsimonious labelling of the internal nodes of a rooted tree. Simplify by considering for each symbol in turn.

Dynamic Programming Algorithm

1. Let T_v denote sub-tree of T whose root is v
2. $s_k(v)$ as minimum parsimony score of T_v over all labelling of T_v assuming that v is labelled by k
3. Minimum Parsimony Score for the tree is equal to minimum value of $s_k(root)$ over all symbols k

$\delta_{i,j}=0$ if i = j, else: $\delta_{i,j}=1$

$$s_k(v)=\min_{\text{all symbols } i}\{s_i(\text{Daughter}(v))+\delta_{i,k}\}+\min_{\text{all symbols } i}\{s_i(\text{Son}(v))+\delta_{j,k}\}$$

Complexity if O(mnk^2) where m species, n characters and k states

3.8 Large Parsimony Problem

Given set of strings, find tree having minimum parsimony score - this is an NP-Complete problem.

3.8.1 Greedy Heuristic

By removing an internal edge, an edge connecting two internal nodes produces four subtrees - rearranging these subtrees is called nearest neighbour interchange

Algorithm

1. Set current tree equal to arbitrary binary rooted tree structure
2. Go through internal edges and perform all possible nearest neighbours interchanges
3. Solve Small Parsimony Problem on each tree
4. If any tree has parsimony score improving over optimal tree, set equal to current tree else return current tree

3.8.2 Tree Validation: Bootstrap Algorithm

- If there are m sequences, each with n nucleotides, a phylogenetic tree can be reconstructed using some tree building methods.
- From each sequence, n nucleotides are randomly chosen with replacements, giving rise to m rows of n columns each. These now constitute a new set of sequences.
- A tree is then reconstructed with these new sequences using the same tree building method as before.
- Next the topology of this tree is compared to that of the original tree. Each interior branch of the original tree that is different from the bootstrap tree is given a score of 0; all other interior branches are given the value 1.
- This procedure of resampling the sites and tree reconstruction is repeated several hundred times, and the percentage of times each interior branch is given a value of 1 is noted. This is known as the bootstrap value. As a general rule, if the bootstrap value for a given interior branch is 95% or higher, then the topology at that branch is considered "correct".

We can generalise pairwise alignment to multiple alignment, changing a 2-row matrix into an n-row matrix. The scoring function should score alignments with conserved functions higher.

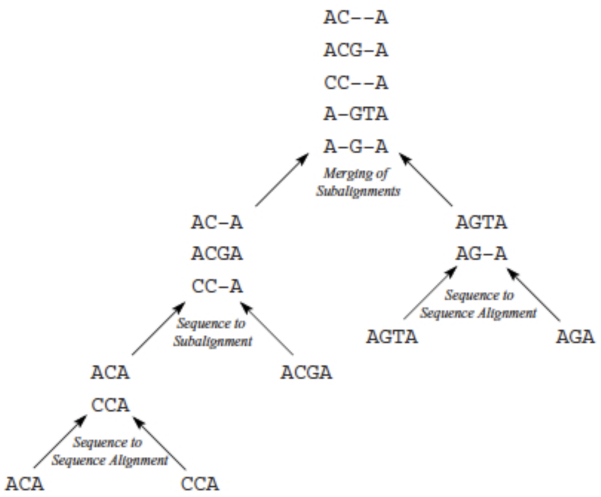
$$s_{i,j,k}=\max\begin{cases} s_{i-1,j-1,k-1}+\delta(v_i,w_j,u_k) \\ s_{i-1,j-1,k}+\delta(v_i,w_j,-) \\ s_{i-1,j-1,k-1}+\delta(v_i,-,u_k) \\ s_{i,j-1,k-1}+\delta(-,w_j,u_k) \\ s_{i-1,j,k}+\delta(v_i,-,-) \\ s_{i,j-1,k}+\delta(-,w_j,-) \\ s_{i,j,k-1}+\delta(-,-,u_k) \end{cases}$$

where $\delta(x,y,z)$ is an entry in the 3D scoring matrix. For a k-way alignment, we build a k-dimensional Manhattan Graph with n^k nodes where most nodes have 2^k-1 incoming edges. It has a runtime of $O(2^k n^k)$

3.9 Progressive Alignment

Idea is that given a set of arbitrary pairwise alignments, we can construct a multiple alignment that induces them. The methods are heuristics in nature - most used algorithm is **CLUSTALW**:

1. Given N sequences, align each sequence to each other
2. Use score of the pairwise alignments to compute a distance matrix
3. Build guide tree (shows the best order of progressive alignment)
4. Progressive Alignment guided by the tree



Not all pairwise alignments build well into a multiple sequence alignment

4 Genome Sequencing

Sequencing has got much cheaper and easier in the last ten years - useful to be able to detect and start to correct for genetic conditions. However, can only shred genome and generate short reads, hence need to reconstruct the genome.

4.1 Assumptions - largely untrue

1. Perfect coverage of genome by reads
2. Error-free reads - errors lead to bubbles in de Bruijn Graph
3. Multiplicities of k-mers are known
4. Distance between reads in read-pairs is contant

String Reconstruction Problem: Reconstruct string from its kmer composition. First step is to connect all possible coinciding nodes and convert it into a graph of some kind. Then, need to find the Hamiltonian path (path which visits each node in a graph exactly once) - this is a hard problem though (NP-Complete). Example problem is 3-mers, in this case (when the genome is known):

- Use 3-mers as edges of the graph and 2-mers as nodes. We then connect identically labelled nodes recursively to produce the De Bruijn Graph. This is O(n) To find the genome from this, we find the Eulerian Path - the path that visits each edge exactly once - there exists an efficient solution to this
- Why Eulerian?
  - Node from left end is semi-balanced with one more outgoing node than incoming, node at right is one more incoming than outgoing. Others are balanced.

4.2 Compute de Bruijn Graph given unknown genome

- Represent composition as a graph of isolated edges
- Make two nodes from each 3-mer and add edge
- Attach identically labelled nodes over and over again firstly to make a chain
- Then attach identically labelled nodes, producing de Bruijn graph

```
DeBruijn(k-mers):
    Form node for each (k-1)-mer from k-mers
    for each k-mer in k-mers:
        Connect prefix node with its suffix node by an edge
```

4.3 Eulerian Graph Problem

```
EulerianCycle(BalancedGraph)
    form a Cycle by randomly walking in BalancedGraph (avoiding already visited edges)
    while Cycle is not Eulerian
        select a node newStart in Cycle with still unexplored outgoing edges
        form a Cycle' by traversing Cycle from newStart and rando-
        ml walking afterwards Cycle <- Cycle'
    return Cycle
```

Hence have broken the genome into contigs - or these sections of cycles

4.4 DNA Sequencing with Read-pairs

Sequence read by two readers at set distance from one another (fixed) along a fragment of DNA. **Paired K-mer** is a pair of k-mers at a fixed distance d apart in the genome.

String Reconstruction from Read-Pairs Problem: Reconstruct string from paired k-mers.

To construct the de Bruijn graph, first do as before, just with two sequences per node and per edge and then attach identical nodes as before

4.5 Errors

1. Errors at end of read - trim off dead-end tips
2. Errors in middle of reads - pop bubbles
3. Chimeric Edges - clip short, low coverage nodes

5 Clustering

Good Clustering Principle: Elements within the same cluster are closer to each other than the elements in different clusters.

Can use the idea of finding center points of each of the cluster to then cluster the data:

$$d(DataPoint,Centers) = min_{all-points-x-from-Centers} d(DataPoint,x)$$

k-Center Clustering Problem: Given set of points Data, find k centers minimizing MaxDistance(Data, Centers). FarthestFirstTraversal is a method that minimises the MaxDistance from the data to the centers, however, this includes outliers, hence instead want to minimise the squared error distortion between data and centers

5.1 Lloyd Algorithm

Centre of Gravity Theorem: centre of gravity of points Data is the only point solving the 1-Means Clustering Problem

$$\text{Center of Gravity} = \Sigma_{\text{all points DataPoint in Data}} DataPoint/number(pointsInData)$$

1. Select k arbitrary data points as centers
2. Assign each data point to its nearest center (**Centers to Clusters**)
3. Set new centers as each clusters' center of gravity (**Clusters to Centers**)
4. Repeat until convergence (which always happens)
    - If data point assigned to new center during centers to clusters, distortion is reduced as center must be closer to the point than previous center was
    - If center moved during clusters to centers, since center of gravity is the only point minimising the distortion.

5.2 Soft Clustering

When we can label the midpoint between two clusters as half of each cluster - hence, each point receives a proportion of each of the

Can use Probability Theory to go from Data and Parameters to HiddenVector (assignments of data points to k centers) given it is in the middle of two clusters

$$\theta_A = \text{ fraction of heads generated in all flips with coin A}$$

$$\theta_B = \text{ fraction of heads generated in all flips with coin B}$$

Given choice of H:

$$Pr(sequence|\theta_A) = \theta_A^H (1-\theta_A)^H$$
$$Pr(sequence|\theta_B) = \theta_B^H (1-\theta_B)^H$$

See notes for rest of discussion on this topic.

Responsibility of star i for a planet j is proportional to the pull (Newtonian Law of Gravitation)

$$Force_{i,j} = 1/distance(Data_j, Center_i)^2$$

$$HiddenMatrix_{ij} = Force_{i,j}/\Sigma_{\text{all centers j}} Force_{i,j}$$

5.3 Hierarchical Clustering

Firstly means creating a similarity matrix or distance matrix.

1. Identify two closes clusters and merge them
2. Recompute distances
3. Repeat

Choice of distance measure is important, can be:

1. Average Distance between points of clusters
2. Minimum Distance between clusters

5.4 Markov Clustering Algorithm

Take random walk on graph described by similarity matrix, but after each step we weaken the links between distant nodes and strengthen links between nearby nodes. Random walk has higher probability to stay inside cluster than to leave it soon. Crucial point lies in boosting this effect by an iterative alternation of expansion and inflation. Inflation parameter responsible for both strengthening and weakening of current.

1. Input is an un-directed graph, with power parameter e (usually =2), and inflation parameter r (usually =2).
2. Create the associated adjacency matrix
3. Normalize the matrix; $M'_{pq} = \frac{M_{pq}}{\sum_i M_{iq}}$
4. Expand by taking the e-th power of the matrix; for example, if $e = 2$ just multiply the matrix by itself.
5. Inflate by taking inflation of the resulting matrix with parameter r : $M_{pq} = \frac{(M_{pq})^r}{\sum_i (M_{iq})^r}$
6. Repeat steps 4 and 5 until a steady state is reached (convergence).

This is experimentally shown to converge after 10 to 100 steps.

Complexity:

- Expansion = $O(n^3)$
- Inflation = $O(n^2)$
- But since matrices are sparse, can improve the speed of the algorithm a lot

Input : A weighted undirected graph $G = (V, E)$, expansion parameter e, inflation parameter r
Output : A partitioning of V into disjoint components
$M \leftarrow M(G)$
while M *is not fixpoint* **do**
  $M \leftarrow M^e$
  forall $i \in V$ **do**
    forall $j \in V$ **do**
      $M[i][j] \leftarrow M[i][j]^r$
    forall $j \in V$ **do**
      $M[i][j] \leftarrow \frac{M[i][j]}{\sum_{k \in V} M[i][k]}$
$H \leftarrow$ graph induced by non-zero entries of M
$C \leftarrow$ clustering induced by connected components of H

5.5 Stochastic Neighbour Embedding

Creates 2D maps from data with hundreds or thousands of dimensions. Algorithm is non-linear and adapts to the underlying data, performing different transformations on different regions of the data. Tuned by changing the **perplexity** - which defines how to balance attention between local and global aspects of the data (ie a guess about the number of close neighbours each point has).

Key Points

1. Convert each high-dimensional similarity into probability that one data point will pick the other data point as a neighbour
2. Evaluate the map:
   (a) Use pairwise distances in low-dimensional map to define prob that a map point will pick another map point as its neighbour
   (b) Compute Kullback-Leibler divergence between probabilities in the high-dimensional and low-dimensional spaces

(c) Each point in high-dimension has conditional probability of picking each other point as its neighbour

(d) Distribution over neighbours is based on high-dimension pairwise distances

SNE is the process of constructing conditional probabilities representing the similarity between high dimensional data points using their Euclidean distances - defined by (for points x_j and x_i:

$$p_{j|i} = \frac{\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2}\right)}$$

Large $p_{j|i}$ is indicative of similar data points.

5.5.1 t-SNE Algorithm

Improves on original SNE by implementing cost function with simpler gradient that uses Kullback-Leibler divergence between high-dimensional join probability distribution P and low-dimensional student-t based joint probability distribution Q:

$$q_{ij} = \frac{\left(1 + \|x_i - x_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

Explicitly is explicitly defined hence:

$$\frac{\delta C}{\delta Y} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

Increases distances such that can definitely map dissimilarities between points that is not otherwise possible to show.

Algorithm 1: Standard t-distributed Stochastic Neighbor Embedding Algorithm.

Data: : data set X = $x_1, x_2, ..., x_n$,
cost function parameters: perplexity *Perp*;
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$;
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = y_1, y_2, y_n$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (Equation 1)
  set $p_i j = \frac{p_{j|i} + p_{i|j}}{2n}$;
  sample initial solution $\mathcal{Y}^{(0)} = y_1, y_2, y_n$ from $\mathcal{N}(0, 10\_4I)$;
  for $t = 1$ **to** T **do**
    compute low-dimensional affinities q_{ij} (Equation 2)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (Equation 3)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{t-1}) - \mathcal{Y}^{t-2}))$;
  end
end

5.6 Burrows Wheeler Transform

1. Form N*N matrix (Burrows-Wheeler Matrix) by cyclically rotating the given text to form the rows of the matrix
   Sentinel: lexicographically greatest character in the alphabet and occurs exactly once in text
2. Sort matrix according to alphabetical order
3. Last column of matrix is BWT(T) and also need to knwo the row number where the original string ends up

It is reversible as the ith occurence of a character in the last column is the same text occurence as the ith occurence in the first column.

To recreate T from BWT(T), repeatedly apply the rule: T = BWT[LF(i)] + T; i = LF(i) where LF(i) maps row i to row whose first character corresponds to i"s last per LF Mapping. First step: $S = 2$; $T = \$$. Second step: s = LF[2] =6; $T = g\$$. Third step: s = LF[6] = 5; $T = cg\$$.

6 Genome Assembly and Pattern Matching

- **Reference Genome**: Database genome used for comparison
- Assembly has the issue of constructing the de Bruijn graph - this takes a lot of memory
- **Read Mapping**: Determine where each read has high similarity to the reference genome
- Why not Alignment?
  - **Fitting Alignment**: Align each read Pattern to the best substring of Genome
  - Runtime O(|Pattern| * |Genome|) for each Pattern
  - Runtime O(|Patterns| * |Genome|) for a collection of Patterns
- **Single Pattern Matching Problem**: Find all positions in Genome where Pattern appears as a substring
- **Multiple Pattern Matching Problem**: Given collection of strings Patterns, find all positions where a string from Patterns appears as a substring
  - Can brute force this, but it's far too slow
  - **Process Patterns into a Trie**: Combine reads into a graph. Each substring of genome can match at most one read. So each read will correspond to a unique path through this graph (the resulting graph is called a trie)
  - **TrieMatching**: slide the trie down the genome. At each position, walk down the trie and see if we can reach a leaf by matching symbols

    Runtime of Trie Construction: O(|Patterns|) and **Runtime of Pattern Matching**: O(|Genome| * |LongestPattern|). Space Complexity = O(|Patterns|)

In order to preprocess the genome, we split the genome into all its suffixes. Can combine suffixes into a data structure using a trie. For each Pattern, we see if Pattern can be spelled out from the root downward in the suffix trie. Memory runtime = O(|Suffixes|). However, can reduce memory by compressing each nonbranching path of the tree into an edge

Runtimes

- **Time**: O(|Genome|) to construct the suffix tree directly and O(|Genome| + |Patterns|) to find the pattern matches.
- **Memory**: O(|Genome|) to construct suffix tree directly and O(|Genome|) to store the suffix tree
- However, constants are high - hence, it takes a reasonably long time

6.1 Genome Compression

1. Run-Length Encoding
   - Genomes don't have a lot of runs
   - But can convert repeats into runs using the Burrows-Wheeler Transform. We showed the Burrows-Wheeler Transform is reversible so this works quite well but old method requires us to store |Genome| copies of |Genome| to find BWT(Genome). Can however be sped up:
   - **First-Last Property**: k-th occurence of symbol in FirstColumn and k-th occurence of symbol in LastColumn correspond to the same position of symbol in Genome - hence memory = O(|Genome|)

- Can use BWT as the data structure for Suffix Pattern Matching by adding a suffix array (holds the starting position of each suffix beginning a row). Memory of suffix array is 4 x |Genome|. But can store part of the suffix array by adding checkpointing (though this increases the runtime by a constant factor)

Approx Pattern Matching Problem: all positions in Genome where a string from Patterns appears as a substring with at most d mismatches. Methods are:

1. **Seeding**: If Pattern occurs in Genome with d mismatches, then we can divide Pattern into d+1 equal pieces (seeds) and find at least one exact match

   Check if each Pattern has a seed that matches Genome exactly.

   If so, check entire Pattern against Genome

2. **BWT**: Use BWT instead - this is much more efficient

7 Hidden Markov Models

7.1 Genomic Sequencing

From 1990-2003: Human Genome Project provides a complete and accurate sequence of all DNA base pairs of human genome

7.1.1 Genome Analysis

1. Sequencing
2. Read Mapping - this is the bottleneck: Sequencer can sequence 300 million bases/minute, can only do read mapping at 2 million bases / minute
3. Variant Calling
4. Scientific Discovery

Identifying Genes and Gene Parts:

1. Information starts with promoter followed by a transcribed but non-coding region called the 5' untranslated region.
2. Initial exon contains start codon
3. Then alternating series of introns and extrons followed by terminating exon which contains stop codon
4. Followed by another non-coding region - 3' UTR - end has a polyA signal.
5. intron/exon and exon/intron boundaries are conserved short sequences and called acceptor and donor sites

Identifying protein structural parts: want to predict the position in the amino acids with respect to the membrane.

7.2 HMM Definition

- Alphabet - $\Sigma = b_1, b_2, ..., b_M$
- Set of States - $Q = 1, ..., K$
- Transition Probabilities: a_{ij} = transition prob from state i to state j
- Start probabilities: a_{0i}
- Emission Probabilities within each state: $e_i(b) = P(x_i = b | \pi_i = k)$

Notes:

- Parse of a sequence is a sequence of states that has passed through

  Likelihood of a parse is:

$$P(x, \pi) = P(x_1, ..., x_N, \pi_1, ..., \pi_N) = P(x_N, \pi_N | \pi_{N-1}) P(x_{N-1}, \pi_{N-1} | \pi_{N-2}) ... P(x_2, \pi_2 | \pi_1)$$

- HMM is memory less

Main questions are:

1. **Evaluation**: Given HMM and sequence, find its prob

   Forward Algorithm

   - $$P(x) = \sum_{\pi} P(x,\pi) = \sum_{\pi} P(x|\pi)P(\pi)$$
   - Forward Probability:
   $$f_k(i) = P(x_1 ... x_i, \pi_i = k)$$
   - **Initialisation**:
   $$f_0(0) = 1$$
   $$f_k(0) = 0, \text{ for all } k > 0$$
   - **Iteration**:
   $$\mathrm{f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl}}$$
   - **Termination**:
   $$P(x) = \sum_k f_k(N) a_{k0}$$

   where a_{k0} is the probability that the terminating state is k

2. **Decoding**: Given HMM and sequence, find sequence of states that maximises prob

   - Solve using dynamic programming - Viterbi Algorithm (time: O(K^2N), space: O(KN))
     (a) **Initialisation**: $V_0(0) = 1$, $V_k(0) = 0 \forall k > 0$
     (b) **Iteration**:
     $$\mathrm{V_j(i) = e_j(x_i) \max_k V_k(i-1) a_k}$$
     $$\mathrm{Ptr_j(i) = argmax_k a_{kj} V_k(i-1)}$$
     (c) **Termination**:
     $$\mathrm{P(x,\pi^*) = \max_k V_k(N)}$$
     (d) **Traceback**:
     $$\pi^*_N = \mathrm{argmax_k V_k(N)}$$
     $$\pi^*_{i-1} = Ptr_{\pi i}(i)$$

   Algorithm: Given HMM, generate sequence of length n as follows

   (a) Start at state π_1 according to prob $a_{0\pi 1}$
   (b) Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
   (c) Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
   (d) Repeat until emitting x_n

3. **Learning**: Given HMM, unspecified transition / emission probs and sequence x, find maximising parameters that maximises the prob of sequence

Backward Algorithm: Compute $P(\pi_i = k|x)$, start by computing:

$$P(\pi_i = k, x) = P(x_1 ... x_i, \pi_i, \pi_i = k) P(x_{i+1} ... x_N | \pi_i = k)$$

$$= \text{FORWARD x BACKWARD}$$

Backward Prob:

$$\mathrm{b_k(i) = P(x_{i+1} ... x_N | \pi_i = k) = \sum_l e_l(x_{i+1}) a_{kl} b_1(i+1)}$$

Initialisation: $\mathrm{b_k(N) = a_{k0}}$, for all k
Iteration: $\mathrm{b_k(i) = \sum_l e_l(x_{i+1}) a_{kl} b_1(i+1)}$
Termination: $P(x) = \Sigma_l a_{0l} e_l(x_1) b_l(1)$
Complexities: Time: O(K^2N), Space: O(KN)

7.3 GeneScan

For given sequence, parse is an assignment of gene structure to that sequence. In a parse, every base is labelled, corresponding to the content it belongs to. In simple model, parse contains only I (intergenic) and G (gene). More complete model contains '-' for intergenic, 'E' for exon and 'I' for intron

1. **Duration of states**
   - Exons - coding
   - Introns - non coding
2. **Signals at state transitions**
   - ATG
   - Stop Codon
   - Exon/Intron and Intron/Exon Splice Sites
3. **Emissions**
   - Coding potential and frame and exons
   - Intron emissions

7.3.1 Features

1. Model both strands at once
2. Each state may output a string of symbols
3. Explicit intron / exon length modeling
4. Advanced splice site modeling
5. Complete intron / exon annotation for sequence
6. Able to predict multiple genes and partial / whole genes
7. Parameters learned from annotated genes
8. Separate parameter training for different CpG content groups

7.4 Transmembrane HMM

Model consists of submodels for:

- Helix core and cap regions
- Cytoplasmic and extracellular loop regions
- Globular domain regions

Sensitivity: fraction of known genes correctly predicted Sn = $N_{true-positives}/N_{all-true}$
Specificity: fraction of predicted genes that correspond to true genes: Sp = $N_{true-positives}/N_{all-positives}$
Correlation Coefficient

$$CC = \frac{[(TP)(TN)-(FP)(FN)]}{\sqrt{(AN)(PP)(AP)(PN)}}$$
$$AN = TN+FP; AP = TP+FN$$
$$PP = TP+FP; PN = TN+FN$$