Laboratory of Bioinformatics 1 part B - Capriotti

Saul Pierotti

April 4, 2020

Introduction

- There will be a project for the final, to be submitted for May 18
- Protein structure is more conserved than sequence
- When sequence identity is sufficiently high, we can tranfer structural information
- A structural alignment is a rigid body transformation of 2 subsets from 2 sets of points that maximizes a given distance metric
  - The subsets need to have the same number of elements and define the corrispondence set
  - Finding the corrispondece set is an NP-hard problem
  - Finding the optimal rigid transformation of the corrispondence set is $\Theta(n)$
- The distance of 2 sequences can be evaluated with a substitution matrix and a gap penalty
- Global alignments are computed with the NW algorithm, local alignments with the SW
- Local alignments are useful for multidomain proteins, when only some domains are conserved
- The significance of an alignment score can be evaluated by comparing with the score distribution for random alignments
- Sanders and Schneider developed a twilight curve before Rost
  - Sander's curve becomes straight after 80 residues
  - Rost's curve does never become straight
- Over 100 residues, under 25% of sequence identity only 10% of the sequences are homologous, while above 20% 90% of them are
- Over 30% identity sequences longer than 100 residues have similar structures, but this does NOT mean that under 30% the structure is necessarily different!
- Proteins with low sequence identity but high structural similarity are referred to as remote homologs
- Structures can be predicted by comparative modeling, threading, ab initio
- If I want to use sequence identity for trasferring annotation features, I need to identify the problem-specific twilight region
  - For subcellularlocalization, the twilight zone is 50% (!)
- The sequence identity needed for transferring subcellular localization is higher than that required for structure
- Function of proteins with really high sequence identity can be completely different
- In remote homologs the sequence alignment is often wrong
- Important residues in a sequence can be identified by comparing conservation levels

Structural alignment

- Structural alignment is different from superimposition
- Superimposition assumes that I already have the correspondence set, and it is relatively easy
- Structural alignment requires the identification of the correspondence set, which is hard
- The definition of domain is often heuristic and questionable
- Proteins with similar spatial distribution but different topology are difficult to align
- Alignment methods can be classified in different ways

  - Pairwise or multiple
  - Depending on the descriptor used
    * Backbone
    * All atoms
    * Sequence-based
    * Contact map
    * Surface
  - Rigid body or flexible
- The comparison of torsion angles is $\Theta(n)$
  - They are invariant for rotation and translation
  - It is good for local regions but problematic for whole structures
- A distance matrix is also invariant for rotaion and translation
  - Comparing matrices is hard, $\Theta(n^2)$
  - It is not sensitive to chirality
- At the moment, all methods are able to identify obvious similarities
- Remote similarities are detected by a subset of methods, and different methods recognize different similarities
- Speed is an issue in many algorithms
- We want our method to be biologically meaningful, not only geometrically
- The expected score or random pairwise alignments is an extreme value distribution
  - I would have a gaussian if there was no evolution
  - In real databases I have an excess of good-scoring pairs
- When I want to determine the distribution of scores, it is better to have an analitycal distribution than an empirical one
  - I don't have tools for working with empirical distributions (!)

CE algorthm

- Compares AFPs composed of 8 residues, stiches them together and finds an optimal path trough them with dynamic programming
- It gives a statistical score
- The alignment is the longest continuous path of AFPs in a similarity matrix S
- The similarity matrix S is composed represent all AFPs conforming to a similaritt criterion
- The dimensions of S are (na-m)(nb-m), where na and nb are the length of the sequences and m the size of the AFPs
- The matrix is large to compute, therefore we need constraints
- Two consecutive AFPs can be aligned with a gap in protein A, a gap in protein B or without gaps
- The AFP lenght is set to 6 and the maximum possible gap to 30
- Similarity measures are RMSD, full set of distances, and others
- The best 20 alignments with Z score above 3.5 are compared based on RMSD and the best one is kept
  - I get an error in 1000 comparisons
- Each gap is assessed for relocation up to m/2 times
- Iteritive optimization with dynamic programming
- It cannot find non-topological alignments
- The unit of comparison was originally the protein chain, but domains are optimal
  - Domains are difficult to define (!)
- The statistical distribution of alignment scores can be used to evaluate the Z score of an alignment

PDBe Fold

- It uses secondarys structure elements (SSEs)
- Secondary structure is typically conserved
- SSE are represented as vectors that connected in a graph by edges
  - 2 vertices and an edge describe position and orientation of the SSEs

- SSEs are helices and strands
- Each edge is labelled by a property vector containing information on edge-vertices angles, torsion angles between vertices, lenght of the edge
- The set of vertices, edges and labels defines the graph that is then matched with an algorthm
- Vertex and edge lenghts are compare both in absolute and relative terms
    - In relative terms, the same absolute difference is less significative for longer edges
- Torsion angles are used for distinguishing mirror simmetries
- The SSE matching gives correspondences among SSEs, and can be used to yeld an initial sequence alignment
- Connectivity (topology) can be neglected, considered but allow for any number of missing SSEs (soft connectivity) or allow only for an equal number of unmatched SSEs (strict connectivity)

MAMMOTH algorithm

- Matching molecular models obtained from theory (MAMMOTH) is one of the fastest algorithms
- The protein is represented as a set of unit vectors among Ca
- It is based on dynamic programming
- An unit vector is the normalized vector among Ca atoms
    - For each position, k consecutive vectors are mapped into a unit sphere that represents the local structure of k residues
- Each set of unit vectors is compare to all the sets in the other structure, building a matrix
- Each comparison yelds a unit root mean square distance (URMS)
    - This is compared against the expected random URMS
    - THe alignment score is obtained by normalizing the URMS with its expected value
- The path trough the matrix is found with dynamic programming by a global alignment without end-gap penalties

RNA structure

- Most RNAs are around 50 bp
- Secondary structure of RNAs is usually represented with parenteses
    - I cannot represent pseudo-knots in this way
- For RNA, the secondary strucutre is much more informative than for proteins
    - A certain secondary structure constraints a lot the tertiary structure
- There is less variability in RNA strucutures than in proteins
- The best atom for representing the backbone is C3', since it has the most constant inter-nuclotide distance
- The professor adapted MAMMOTH to work with RNA C3' atoms instead of Ca in proteins: SARA
    - The statistics of the score had to be re-evaluated
    - They still used the extreme value distribution, which is defined by μ and σ
    - They selected how the parameters change when RNA size changes
    - The set of unit vectors was 3 instead of 7
    - The method gives a -log(p-value) score
    - By comparing RNAs of known function, I can determine a score threshold that gives correct functional annotation
- Another method was developped in Israel: ARTS
- Few people are working in RNA: not so many methods
- The twilight zone of RNA sequence alignment is around 60%
- Secondary structure identity (PSS) correlates well with tertiary structure identity (PSI) but not with sequence identity

Multiple sequence alignment

- In MSA it is easier that in pariwise alignments to identify conserved regions, that could be functionally important
  - We can observe blocks of conservation in MSAs
- I can transform a MSA in a profile of the sequences
- A profile is a matrix with a row for each possible residue and a column for each position
  - The value of each element reflects the frequency of a residue in a specific position
  - Each position is therefore a vector of 20 elements
  - I represent a profile as a matrix containing as many vectors as the number of positions
  - I can also have a row for the presence of a gap in the position
- A sequence logo is a plot showing the entropy of each residue in each position
  - It is obtained from a profile and it is a way to represent it
- Shannon entropy: information content of a message
  - For a single colum $S(p) = \sum_{i=1}^{20} -p_i \ln p_i$
  - Total conservation: $S(p) = 0$
  - All residues are equally probable: $S(p) = ln(20)$
  - There are more sofisticated models that take into account the expected frequency of residues
  - The entropy of an alignment is obtained by summing the Shannon entropy over the all alignment
- Scoring an MSA: sum of pairwise scores or entropy score
  - Not all the position are equal in an MSA: some conservations are critical, others not
  - Scoring has necessarily to depend on the evolutionary history of the sequences
  - Almost all scoring functions assume positional independence
- I can score each pairwise alignment and sum it
  - $S = \sum_{i<j} S(A_i, A_j)$
- I can score an MSA depending on its entropy
  - The best alignment is the one with the lowest entropy (i.e. the most conserved one)
  - It is the sum over the alignment of the entropy in each position
  - $S = \sum_{j=1}^{Ncols} \sum_{i=1}^{20} -p_i \ln p_i$
- I can align a sequence to a profile
  - Each position is aligned to a vector for the position
  - The score for the position of the residue in the sequence with every possible residue is summed and weighted for the frequency encoded in the vector
    - This is a matrix by vector multiplication (!)
  - These scores can be used with a dynamic programming algortihm

Algorithms for MSAs

- Dynamic programming approaches exist, but they are $O(N^M)$ and they are np-hard
- An MSA method can be evaluated from the functionally important residues that are correctly aligned

Progressive MSA

- ClustalW is an example of progressive MSA
- I allign sequences in pairs, one after the other
- The result depends on the order of how I pair sequences (!)
  - I usually pair the most similar sequences first
- Similarity is measured by Kimura distance (see MUSCLE for more info)
- From each pairwise alignment, I build a profile
- I iterate until there are no sequences left, by aligning pairwise sequences and profiles
- In order to do this I need to be able to align profiles (!)
- I want to be conservative with gaps with the initial pairwise alignments, and introduce them later on profiles
  - When I get to profiles I have info about conservation (!)

- Errors in the first alignments are propagated
    - If I am not conservative I can become full of gaps
- I can improve the alignment by changing the sequence tree
    - By default Clustal uses NJ
    - Maybe I have a tree available (!)
- Adding gaps is tricky, since their penalty logically depends on the position and conservation
    - They are usually added in the fisrt alignments
- In ClustalW the penalty is multiplied by a factor which is context specific
    - Gaps in hydrophobic regions are more penalised
    - These coefficients were derived from gaps frequencies in a large number of structural alignments
    - Gaps are discouraged if there is another gap nearby in the MSA
- Low-scoring alignments are postponed for later by adjusting the tree
    - ClustalW aligns them when it has more information deriving from the profiles
- A profile-to-profile alignments involve the pairwise comparison of same-dimentional vectors
    - I do a double sum all against all elements weighted with a substitution matrix
    - This is done via a simple vector to matrix multiplication, followed by a multiplicatio for the remaining vector (!)
- ClustalW corrects for biased representation of subfamilies
- The scoring matrices used change depending on the similarity of the sequences to be compared
- In general, ClustalW uses an heavily crafted heuristics
- The main problem of progressive alignment: subalignments are frozen in place
    - Once aligned, a group of sequences cannot be re-aligned by taking advantage of the new information deriving from other sequences

Iterative MSA

- Iterative MSA tries to overcome the problem of frozen subalignments
- MUSCLE: multiple sequence comparison by log expectations
- It is based on 3 steps: draft progressive, improved progressive, and refinement
- Draft progressive: create a first progressive MSA
    - Sequence similarity is defined by k-mer distance, not pairwise alignment score
        * If a rare kmer is present in 2 sequences maybe they are related
    - It creates a distance matrix with all sequences against each other
    - It uses UPGMA instead of NJ for building the tree from the matrix
    - The score is based on log expectations, not pairwise score for profile to profile alignments
        * It is the entropy score
- Improved progressive: from the draft create a new matrix and from that a new tree and a new alignment
    - The pairwise distances are calculated from the Kimura distance
        * $K_{dist} = -\ln 1 - D - D^2/5$, where D is the pairwise identity
- Refinement: cut and re-align the tree
    - I edge is deleted at random from the tree
    - The 2 resulting profiles are re-aligned to each other to get the full MSA
    - If the score improves, keep the new MSA otherwise keep the previous one
    - This is iterated until convergence on a local minimum

Consistency-based MSA

- Consistency: if residue X is aligned with Y and Y is aligned with Z, then X is aligned to Z
    - This is necessarily true in an MSA
- In reverse, I can use consistency to align two sub-alignments: I take advantage of transitivity of alignments
- MSA are not necessarily consistent with the repsective pairwise alignments
    - Progressive MSA methods frequently are not consistent with the pairwise alignments used for building the tree

- T-Coffe (tree-based consistency objective function for alignment evaluation) is an MSA method based on consistency
  - Build the primary library
    * I do all the possible pairwise alignments and I measure the pairwise sequence identity
    * Each pairwise alignment is equipped with a weight equal to the average identity of matched residues, ignoring gaps
  - Build the extended library
    * In order to align sequences A and B, I try all the possible alignment, direct and based on an intermediate sequence C
    * The weight of each alignment is the minimum of the pairwise weights for the intermediate alignments
    * The final weight of a position is the sum of the weights of all the possible alignments supporting it
  - Maximise the pairwise alignments from the extended library with dynamic programming
    * The score of each match corresponds to its weight
  - From the extended pairwise alignments, build a guide tree
  - Do a progressive MSA from this guide tree and the extended pairwise alignments
  - T-Coffe considers both global and local pairwise alignments and it can use information about domains and motifs

MSA benchmark

- BaliBASE was the first large-scale benchmark specifically designed for MSA
  - It is a dataset with manually refined alignments derived from structural superimposition
- BaliBASE is subdivided in several reference datasets
  - 1 - Small number of equidistant sequences
    * This is further subdivided by identity levels
  - 2 - Families with one or more orphan sequences
  - 3 - Pair of divergent subfamilies with less than 25% reciprocal identity
  - 4 - Sequences with large extensions at the N or C terminal
  - 5 - Sequences with large internal indels
- The evaluation of the benchmark is based on a series of scores
  - The scores are evaluated only for columns that are reliably aligned in the reference (core columns)
  - Sum of pairs score (SP score): proportion of correctly aligned residue pairs in the core columns
  - Total column score (TC score): proportion of completely correclty aligned core columns
  - TC and SP score both are a number between 0 and 1
  - In a pairwise alignment SP and TC score are necessarily equal
  - In an MSA with 3 or more sequences, SP >= TC
  - Both scores encourage sensitivity, but they do not test for specificity
    * There is no penalty for wrong alignments (!)
- BaliBASE also evaluates time of execution and peak memory usage
- SP, TC, memory and time are reported as Z-scores on a spiderweb plot for each alignment tool
- What comes out of the BaliBASE benchmark?
  - No single method is perfect in all cases (!)
  - On average, consistency-based methods are more accurate but slower
  - T-Coffe suffers with N and C terminal extension
  - ClustalW and MUSCLE are the least resource-heavy tools
  - T-Coffe and MAFFT are well suited for alignments larger that those in BaliBASE
  - Multi-threading can greatly speed-up these softwares, since there is a lot of parallel computing
  - Many algos take advantage of parallel processing

Probabilistic sequence models

- A model is an object producing different outcomes (sequences) from a probability distribution
- The probability distribution in sequence space determines the specificity of the model
- The probability for model M of generating sequence s is $p(s|M)$
- In the reverse, I can see a model as an object that given an outcome computes a probability value
- Models can be trained: I can adjust the probability density function over the sequence space from a set of known sequences
  - If I want to model the globin family, I can train my model with sequences that are know to belong to that family
- After training, I can use the model to compute the probability of an unknown sequence to belong to the globin family
- The model M given a sequence s returns the probability $p(s|M)$
  - This is the probability of the model generating the sequence, not the sequence coming from the model
- Most times I am interested in the probability of a given sequence s to come from the model M
  - This is the probability of a sequence being part of a specific family
  - This is $p(M|s)$
- In order to compute $p(M|s)$ from $p(s|M)$ I need to use Bayes theorem
  - $p(M|s) = p(s|M)p(M)/p(s)$
- The priors $p(M)$ and $p(s)$ needs to be estimated to do the conversion
  - $p(M)$ is the a priori probability of any sequence belonging to the model
    * This is the relative abundance of the class, relative to all possible classes
    * It can be estimated from the abundance of the known sequences in the family
  - $p(s)$ is the a priori probability of the sequence and cannot be estimated reliably
- In order to avoid specifing $p(s)$ I can compare the probabilities of 2 different models
  - Instead of looking for $p(M_1|s)$, I look for $p(M_1|s)/p(M_2|s)$
  - $\frac{p(M_1|s)}{p(M_2|s)} = \frac{p(M_1|s)p(M_1)}{p(s)}\frac{p(s)}{p(M_2|s)p(M_2)} = \frac{p(M_1|s)p(M_1)}{p(M_2|s)p(M_2)}$
  - In this way, the conditional probabilities of the sequences are easy to estimate from the models themselves
  - The ratio $p(M_1)/p(M_2)$ can be estimated from the relative abundance of the 2 classes
- To make the calculation more standard, I can systematically compare any model to the NULL model
- The NULL model N is a model that generates all the possible sequences with equal probabilities, only depending on the residue frequencies

Markov Models

- HMM have their most frequent application in speech recognition
- A simple Markov Model, or Markov chain is a collection of states associated with probabilities for all the possible transitions between them
- It is useful for modeling the probability of a sequence of states that only depend on the rpeciding state in the sequence
- I can consider each residue as a state, and I can assume that its state depends only on the previous residue
- The Markov model will contain all the possible residues and their transition probabilities
- In this framework, the transition probability is the probability that residue B follows residue A in position x_i of a sequence
- The trasition probability a_{AB} is the conditional probability of the position i+1 being B given that position i is A
  - $a_{A,B} = p(x_{i+1} = B|x_i = A)$
- The probability of a sequence x of lenght n is the product of all the transition probabilities at the various positions
  - Here I am assuming independence of each transition

- $p(x) = p(x_n|x_{n-1}) * p(x_{n-1}|x_{n-2}) * ... * p(x_2|x_1) * p(x_1)$
- $p(x) = p(x_1) * \prod_{i=2}^{n} a_{x_i-1,x_i}$
- I can also add a BEGIN and an END state to my model for avoiding irregularities
  - In this case the transition probability from BEGIN to a state is the probability of starting with that state
  - This is symmetrical for transitions from a state to the END state
  - We treat both BEGIN and END states as the same state 0 so a_{0k} and a_{j0} are transitions from BEGIN and to END
    - There is noambiguity since transitions are only from BEGIN and only to END
- Let's say I want to model the probability that a given sequence is a CpG island
  - In such sequence, a_C, G would be much higher than elsewhere
  - I can create 2 different Markov chains M_+ and M_- for modelling the 2 sequences: CpG island and non CpG island
  - The 2 models will have the same states but different transition probabilities
  - To determine the likelihood S of a sequence x being a CpG island, i can compare the log-odds of the 2 models
    - $S(x) = \log \frac{P(x|M_+)}{P(x|M_-)} = \sum_{i=1}^{n} \log \frac{a^+_{x_{i-1},x_i}}{a^-_{x_{i-1},x_i}}$
- I can define the probability of a sequence s_i to be generated by a family described by the model M
  - $p(s_i|M)$
- In a Markov model, the sum of probabilities going out of a state is always 1
  - It is certain that I will go out of the state
- When I have only 2 possible mutually exclusive models, I can have a measure for $p(s)$
  - $p(s|M_1) + p(s|M_2) = p(s)$
- From this, I can recover $p(M_1|s)$ and $p(M_2|s)$
  - $p(M_1|s) = \frac{p(s|M_1)p(M_1)}{p(s)} = \frac{p(s|M_1)p(M_1)}{p(s|M_1)+p(s|M_2)}$
  - $p(M_2|s) = \frac{p(s|M_2)p(M_2)}{p(s)} = \frac{p(s|M_2)p(M_2)}{p(s|M_1)+p(s|M_2)}$
- We always work with Markov models of order 1: every state depends only on the previous 1 state
  - There are also MM of order 0 or >1

Training the model

- The parameters for a model can be estimated from a set of training data
- For any sequence s and model M, I can express $p(s|M)$ as the Markov chain that can produce s
  - $p(s|M) = \prod_{j=0}^{n+1} \prod_{k=0}^{n+1} a_{jk}^{n^{jk}}$
  - In this representation 0 is the BEGIN state and n+1 the END state
  - The probability is the product of the transition probability for all the possible transitions to the power of how many times they do occur
- The model is always under the normalization constraint
  - $\forall j \ \sum_{k=1}^{n+1} a_{jk} = 1$
  - The sum of outgoing transitions from any stae must sum up to 1
- Maximum likelihood estimation: the value of the parameter θ is the one that maximises the probability of the dataset D given the model and the parameter
  - $\theta_{ML} = argmax_\theta P(D|M, \theta)$
  - The solution for any parameter θ can be obtained
    - $\theta = a_{ik} = \frac{n_{ik}}{\sum_j n_{ij}}$
    - The optimal value of the parameter is the frequence of occurrence of the transition in the dataset
    - The normalization constraint forces to divide the count of transitions for the total number of outgoing transitions
- Maximum a posteriori estimation: the Bayesian correction of the ML approach
  - $\theta_{MAP} = argmax_\theta(p(\theta|M, D))$
  - $p(\theta|M, D) = p(D|M, \theta)p(\theta)$

HMM

- Let's now try to model the presence of a CpG island in a larger sequence
  - I can integrate both models M_+ and M_- in a single model
  - I will have 2 states for each nucleotide, one for each model
  - The transition probabilities inside states of the + and - models will be similar to before
  - In addition I will have a small probability of going from a state of one model to any state of the other model
  - It will be more probable to go from - to + than vice versa
    - This means that I will be most of the time in -, so most of the sequence is not a CpG island
- This is an Hidden Markov Model since for every position the sequence itself I cannot no which state generated it
  - For each possible nucleotide I have 2 states, and I do not know which one it came from
- Differently from Markov chains, in HMM we need to distinguish the sequence of states from the sequence of observables (symbols here)
- The sequence of states, which is hidden to us, is called the path π and it is a simple Markov chain
  - The path has transition probabilities $a_{jk} = p(\pi_i = k|\pi_{i-1} = j)$
- A symbol can correspond to multiple states but also a state can generate different symbols (!)
  - In general, the outcome of a single state derives from a probability distribution
  - We define the emission probability of symbol b from state k as $e_k(b) = p(x_i = b|\pi_i = k)$
  - The sum of emission probabilities from a state is always 1, so the state always produces something
- I can write the probability of observing the sequence x of lenght L under the path π
  - $p(x, \pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$
  - In this equation $\pi_{L+1} = 0$, so the last transtition is to the END state
  - The probability of a carachter x_i being generated by the model is the product of the probability deriving from the markov chain and the emission probability for that carachter under the current state
  - The path is typically hidden, so this equation is not useful in practice
- The probability of the above equation under a model M can be rewritten as $p(x, \pi|M)$
  - This can be decomposed as $p(x, \pi|M) = p(x|\pi, M) * p(\pi|M)$
- Naive approach: if I want to obtain $p(x|M)$ under an HMM I need to sum over all the possible paths
  - $p(x|M) = \sum_{\pi} p(x, \pi|M)$
  - The number of possible paths is the number of states elevated to the lenght of the sequence
    - No way you can do that brute force
  - The time complexity is $O(tn^t)$ where n is sequence lenght an t the number of different symbols
    - Danger! NP-hard!
- There are different algorithms for computing $p(x|M)$ under an HMM
  - In general, my aim is to decode the path from the sequence, so that I can assess the true probability
- Viterbi algorithm: dynamic programming for finding the most probable path
  - If I need to choose just 1 path the most probable one is the most logical choice
  - Let's define the most probable path $\pi^* = argmax(p(x, \pi))$
  - I can find π^* recursively
    - I suppose that the probability of π^* having state k in position i is $v_k(i)$ and it is known for all the states k
      - This means that I know the probability of each state in each position of the most probable path
    - I can calculate recursively the probability of state l for position i+1
      - $v_l(i + 1) = e_l(x_{i+1}) * max_k(v_k(i)a_{kl})$
      - The first term is the emission probability of the state l for the observed symbol x_{i+1}
      - The second term is the probability of having state k in position i times the transition probability from k to l
      - In the second term I take the max in k, so I choose the k that maximises the quantity
      - The problem then recurses in calculating $v_k(i)$ and so on
    - All sequences need to start at some point: the recursion ends in $v_0(0) = 1$

        - It is certain that the beginning of the sequence comes from state 0
    - Given this framework, I can create a dynamic programming matrix that finds the optimal path
      * Initialization
        - $i = 0,\ v_0(0) = 1,\ v_k(0) = 0 \quad \text{for } k > 0$
      * Recursion with i = 1 to L (lenght of sequence)
        - $v_l(i) = e_l(x_i) * max_k(v_k(i-1)a_{kl})$
        - $pointer_l(i) = argmax_k(v_k(i-1)a_{kl})$
      * Termination
        - $p(x, \pi^*) = max_k(v_k(L)a_{k0})$
        - $\pi^*(L) = argmax_k(v_k(L)a_{k0})$
      * Traceback with i = L downto 1
        - $\pi^*(i-1) = pointer_i(\pi_i^*)$
    - The probabilities obtained with the Viterbi algorithm are really small and give underflow errors
      * It is better to operate in log space
        - I use $\log v_l(i)$
      * This makes also the products become sums
- Forward algorithm: why only the most probable path, if I can have all of them?
    - Using only π^* as in the Viterbi algotihm is a huge approximation, but it works surprisingly well
    - Actually we don't need to do so, since we can calculate the complete probability of x for all paths
    - We can just replace the maximizations of the Viterbi algorithm with sums
    - We can define $f_k(i)$ as the forward parallel of the Viterbi quantity $v_k(i)$
      * It is the probability of state k in position i under the forward algorithm
    - The probability of state k in position i is the joint probability of the sequence up to position i and the fact that the current state is k
      * $f_k(i) = p(x1..x_i, \pi_i = k)$
    - The recursion equation is therefore
      * $f_l(i+1) = e_l(x_{i+1}) * \sum_{k=0}^{i}(f_k(i)a_{kl})$
    - Like the Viterbi approach, the forward algorithm can give underflow errors
      * I can correct by operating in log space or scaling the probabilities
      * In log space the math is not as clean as with the Viterbi
    - The time complexity is $O(tn^2)$
- Backward algorithm: some as forward, but starting from the end
    - The quantities that I consider here is $b_k(i)$
    - It is the probability of state k in position i
    - I don't use the backword algorithm for calculating $p(x)$, since usually I get it with the forward algorithm
- The backward algorithm is useful for calculating posterior probabilities
- What I am really interested in is not $p(x, \pi_i = k|M)$, but $p(\pi = k|x, M)$
    - I want to now the probability of the hidden state k being at work given the sequence
    - $p(x, \pi_i = k)$ can be decomposed as $f_k(i)b_k(i)$
      * This is the joint probability of having $\pi_i = k$ when the sequence up to i and from i to the end is equal to the respective portions of x
    - From this I can get the posterior probabilities
      * $p(\pi = k|x) = p(x, \pi_i = k)/p(x) = f_k(i)b_k(i)/p(x)$
      * $p(x)$ is the result of the forward algorithm here (!)
- A posteriori decoding: when choosing the most probable path is not justified
    - In some situations just choosing the most probable path (Viterbi decoding) is not legitimate
      * I can have many paths with similar probabilities, and a posteriori decoding evaluates all of their contributions for any state
    - For position i the a posteriori estimate for state π_i is $\hat{\pi}_i$, as compared to π_i^* of Viterbi decoding
      * $\hat{\pi}_i = argmax_k(p(\pi_i = k|x))$
      * This probability includes all the possible paths that can bring me in position i at state k
    - This definition is not very useful for determining the path, I can only decode a single state
    - If I use this equation for the whole path, it can be non-sensical (!)

        * It could include forbidden transitions
        * It can give the most probable state in position i given path, and then a state in position i+1 given a different path, when the transition from the 2 is impossible

HMM for pairwise alignments

- I can see a gapped alignment as a finite state automaton (FSA) with a match state M, and two state for the respective insertions, X and Y
    - In this FSA I have score changes at every state transition
- I can similarly create a probabilistic HMM with the same states
    - It does not emit a sequence, but an alignment (!)
    - It is called pair HMM
- The pair HMM has the following properties
    - The state M has emission probabilities for all possible matches $x_i : y_i$
    - State X has emission probabilities for all possible single charachter insertions in X (or delitions in Y) such as $x_i : gap$
    - State Y has emission probabilities for all possible single charachter insertions in Y (or delitions in X) such as $y_i : gap$
    - We introduce a BEGIN and END state
    - The transition probabilities are called
        * δ for M->X,Y
        * ϵ for Y->Y and X->X
        * θ for M,Y,X -> END
        * The other probabilities can be derived from the complements to 1
    - Every position in the pair HMM has two indexes intead of 1
- The Viterbi path of the pair HMM is the optimal FSA alignment (!)
    - It is the one I would recover from the NW algorithm

Profile HMMs

- Profile HMMs are the most important application of HMMs to bioinformatics
    - They were first done by Krogh (the one from Denmark)
- In the ungapped case, I just want to model the propensity of the position for a symbol
    - The profile HMM will have for each state M emission probabilities deriving from the profile vector at that position
- Introducing gaps, we see how their penalty shouldn't be the same across the alignment
- I can introduce the insert (I) state for modelling insertions in my sequence with respecty to its family profile
- The I state has emission probabilities deriving from background distribution
    - We need a transition from M_i to I_i, a loop from I_i to itself and a transition from I_i to M_{i+1}
    - This model is essentially an affine gap
- The deletion could be modelled by a series of transitions from M_i to all M_{i+k}, all the subsequent states
    - This requires n(n-1)/2 transitions, and I need probabilities for all of them
- To avoid this I insert silent states D for modelling deletions
    - A silent state is a state that does not emit any symbol
    - Now I have transitions from M_i to D_{i+1} and from D_{i+1} to D_{i+k}
        * I can enter in a delete state (and not add anything to my sequence) and continue there
    - I have also transitions D_i to M_{i+1} for modelling when the deletion ends
    - In this way I have 4n-8 parameters
- This delete model has different transitions in different positions, so I can include a position-specific gap penalty (!)
- In my insert model I cannot do this, since I have loops for long insertions in the same I state
    - This makes sense since The insertion only matters where it starts and how long it is

    - For delitions it matters which residues are missing from the family profile (!)
- As a final refinement, I can include transitions between delete and insert states
    - They are quite unlikely and usually they do not affect much the alignment
    - These are D_i to I_i and I_i to D_{i+1}
- Any profile HMM is able to produce any possible seuqence in sequence space
- Parameterising a profile HMM means to make the probability distribution of the produced sequences peak around members of the modelled family
    - We can play with transition and emission probabilities, and with the lenght of the model itself
- Modelling the lenght of the model means to decide which MSA columns to assign to match states, and which to insert states
    - A heuristic rule is to assign to insert states the columns that have more than half gaps
- Probabilities can be estimated from the transition and emission frequencies of the sequences in the MSA
    - For this to be meaningful, I need a big training set
    - There can be transitions or emissions with 0 probability due to sampling limitations
        * We can add pseudocounts for coping with this
- I can evaluate the score of an alignment to a profile with the profile HMM
    - I can calculate $p(x, \pi^*|M)$ with Viterbi or $p(x|M)$ with the forward algorithm
- I can use the log-likelyhood $LL = -\log p(x|M)$ as the alignment score
    - It is strongly lenght dependent and in a not linear fashion (!)
    - I can normalize it obtaining a Z-score
    - For the normalization I need a μ and a σ for the length-dependent score distribution
- I can also use the log-odds against the NULL model
    - This has usually a 3 times better signal-to-noise ratio in discriminating families
    - The NULL model is obtained from the residue composition of the training set
- The accuracy of a prediction can be evaluated on the confusion matrix
    - It is a simple 2*2 matrix that relates true and false positives and negatives
    - The variables under consideration are the true condition and the test outcome
- The accuracy ACC can be evaluated from the confusion matrix
    - $ACC = (TP + TN)/(TP + TN + FP + FN)$
    - This measure can be biased if the classes (positive and negatives or prediction 1 and prediction 2) are highly unbalanced
- A better approach is to evaluate sensitivity (True positive rate , TPR) and specificity (Positive predicted value, PPV)
    - $TPR = TP/(TP + FN)$
    - $PPR = TP/(TP + FP)$
- The Matthews correlation coefficient (MCC) is the analogous of Pearson for categorical predictions
    - $MCC = \frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
    - It measures the correlation among predictions and real classes
    - It is not affected by class unbalance
- The ROC curve (receiver operating carachteristics) is a plot of FPR (false positive rate) against TPR (true positive rate) when a parameter varies
    - This can be a tuning parameter
    - The area under the ROC curve is 0.5 for random predictions ad 1 for a perfect predicition