# Lecture 7 Sequence Similarity

Lecturer: Prof. Serafim Batzoglou
Scribe: Abhishek Rathod

In this lecture we will see
1.>A multiple sequence alignment system called Probcons
2.>Profile HMMs
3.>The Sequence Analysis and Modeling system SAM
4.>A database of protein family alignments called Pfam

To provide background I have put some notes in the appendix on basics of HMMs and pairwise alignment using HMMs.

Index

1. Probcons……..……2
   1.1 Motivation for ProbCons…….2
   1.2 ProbCons Algorithm……5
   1.3 Test Results……..7

2. Profile HMMs…….8
   2.1 Motivation for Profile HMMs……..8
   2.2 Profile HMM details……10

3. SAM……...13

4. Pfam……15

Appendix A Markov Chains and Hidden Markov Models……...17

Appendix B Pairwise Alignment using HMMs……21

References…….23

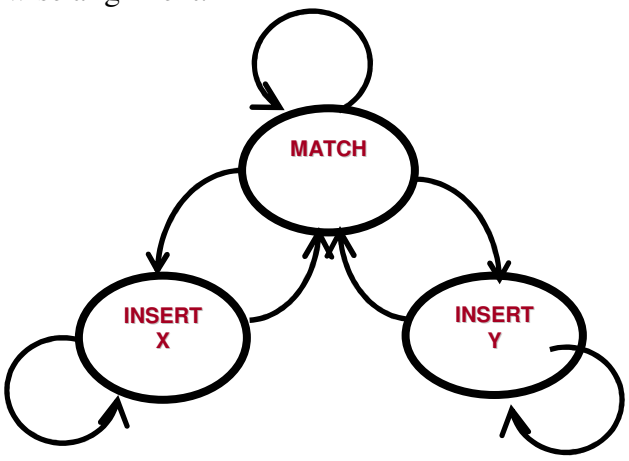
# 1.>Probcons

## 1.1>Motivation for ProbCons

Obtaining accurate alignments is a difficult computational problem because of not only the high computational cost but also the lack of proper objective functions for measuring alignment quality. Probcons introduced the notion of *probabilistic consistency*, a novel scoring function for multiple sequence comparisons. ProbCons is a practical tool for progressive protein multiple sequence alignment based on probabilistic consistency.

Direct application of dynamic programming is too inefficient for alignment of more than a few sequences. Instead, a variety of heuristic strategies have been proposed. By far, the most popular heuristic strategies involve tree-based *progressive alignment* in which groups of sequences are assembled into a complete multiple alignment via several pairwise alignment steps. As with any hierarchical approach, however, errors at early stages in the alignment not only propagate to the final alignment but also may increase the likelihood of misalignment due to incorrect conservation signals. Post-processing steps such as iterative refinement alleviate some of the errors made during progressive alignment.

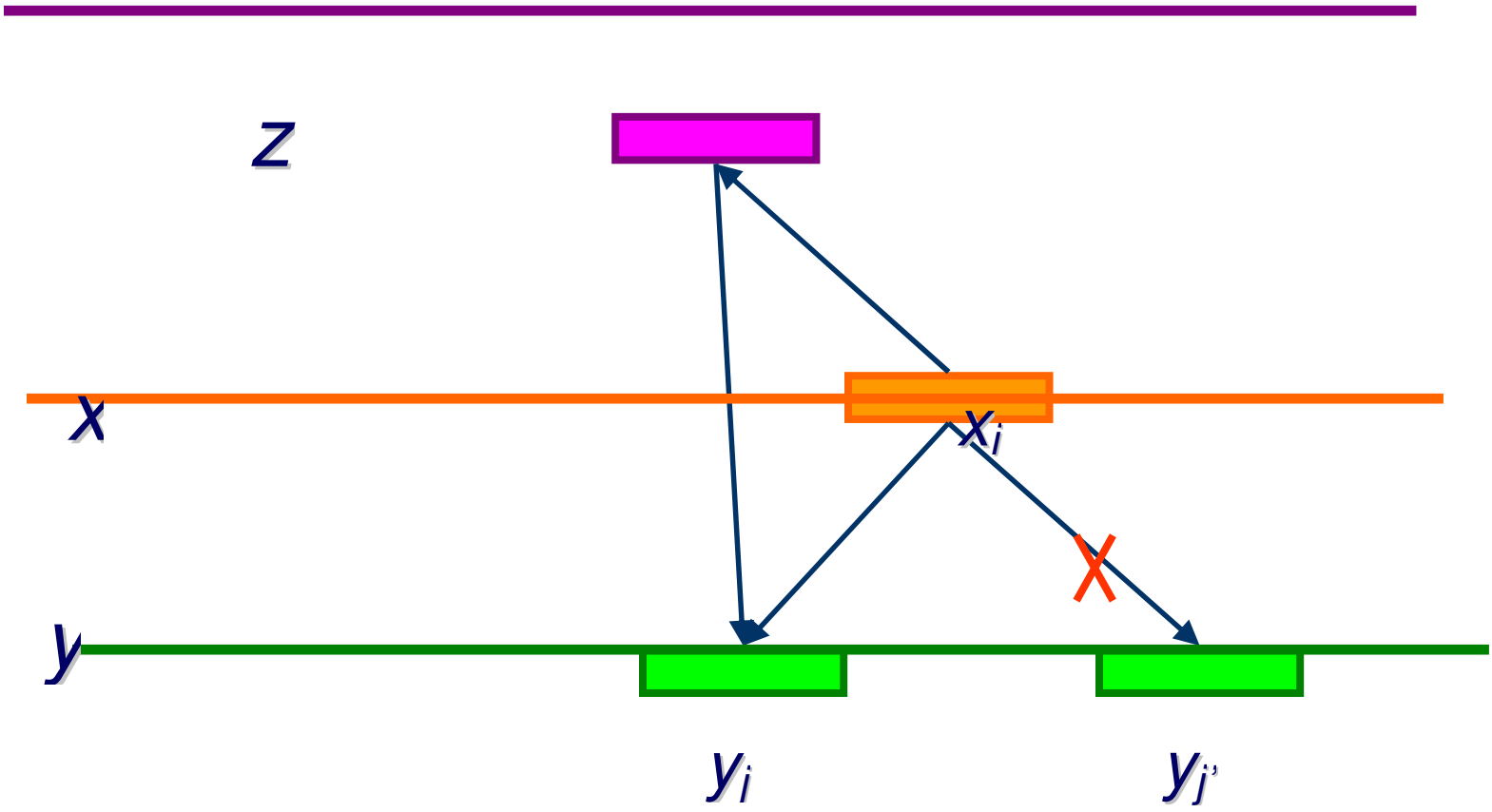
ProbCons is a pair-hidden Markov model-based progressive alignment algorithm that primarily differs from most typical approaches in its use of *maximum expected accuracy* rather than Viterbi alignment, and of the *probabilistic consistency transformation* to incorporate multiple sequence conservation information during pairwise alignment.




Emission probabilities, which correspond to traditional substitution scores, are based on the BLOSUM62 matrix. Transition probabilities, which correspond to gap penalties, are trained with unsupervised expectation maximization (EM).

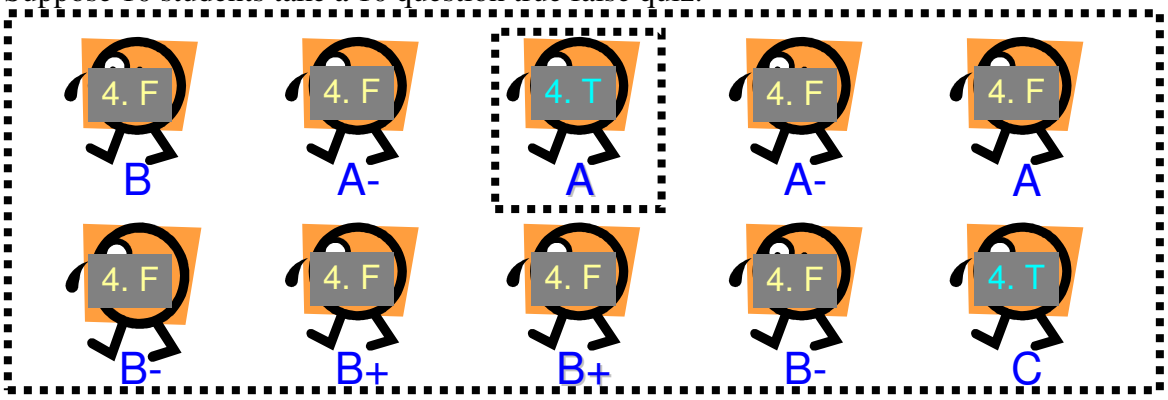
*Consistency-based* schemes take the view that "prevention is the best medicine." Note that for any multiple alignment, the induced pairwise alignments are necessarily *consistent*— that is, given a multiple alignment containing three sequence *x*, *y*, and *z*, if position *xi* aligns with position *zk* and position *zk* aligns with *yj* in the projected *x*–*z* and *z*–*y* alignments, then *xi* must align with *yj* in the projected *x*–*y* alignment. Consistency- based techniques apply this principle in reverse, using evidence from intermediate sequences to guide the pairwise alignment of *x* and *y*, such as needed during the steps of a progressive alignment. By adjusting the score for an *xi* ~ *yj* residue pairing according to support from some position *zk* that aligns to both *xi* and *yj* in the respective *x*–*z* and *y*–*z* pairwise comparisons, consistency-based objective functions incorporate multiple sequence information in scoring pairwise alignments.

The diagram below illustrates consistency:



The difference between Viterbi and Probabilistic Consistency can be understood from the lazy teacher analogy.

Suppose 10 students take a 10 question true false quiz.

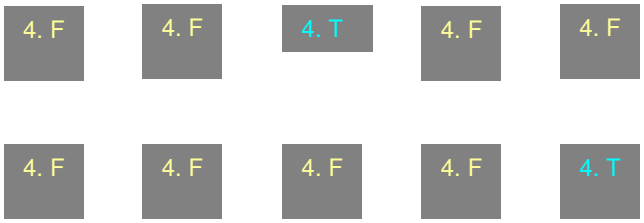



- The question is how do you make the answer key?
  - **<u>Approach #1</u>: Use the answer sheet of the best student! (Viterbi)**


- picks single alignment with highest chance of being completely correct
- mathematically, finds the alignment **α** that maximizes

  **$E_{\alpha^*}[1\{\alpha = \alpha^*\}]$**

  - **<u>Approach #2</u>: Weighted majority vote! (probabilistic consistency)**




- picks alignment with highest expected number of correct predictions
- mathematically, finds the alignment **α** that maximizes

  **$E_{\alpha^*}[\text{accuracy}(\alpha, \alpha^*)]$**

# 1.2.>ProbCons algorithm

Given $m$ sequences, $S = \{s(1), \ldots, s(m)\}$:

***Step 1: Computation of posterior-probability matrices***

For every pair of sequences $x, y \in S$ and all $i \in \{1,\ldots, |x|\}$, $j \in \{1,\ldots, |y|\}$, compute the matrix $P_{xy}$, where $P_{xy}(i, j) = \mathbf{P}(xi \sim yj \in a^* \mid x, y)$ is the probability that letters $xi$ and $yj$ are paired in $a^*$, an alignment of $x$ and $y$ generated by the model.

***Step 2: Computation of expected accuracies***

Define the expected accuracy of a pairwise alignment $a$ between $x$ and $y$ to be the expected number of correctly aligned pairs of letters, divided by the length of the shorter sequence:

$$\mathbf{E}_{a^*}(accuracy(a,a^*)|x,y) = \frac{1}{\min\{|x|,|y|\}} \sum_{x_i \sim y_j \in a} \mathbf{P}(x_i \sim y_j \in a^*|x,y).$$

For each pair of sequences $x, y \in S$, compute the alignment $a$ that maximizes expected accuracy by dynamic programming, and set

$E(x, y) = \mathbf{E}_{a^*}(accuracy(a, a^*) \mid x, y)$.

***Step 3: Probabilistic consistency transformation***

Reestimate the match quality scores $\mathbf{P}(xi \sim yj \in a^* \mid x, y)$ by applying the *probabilistic consistency transformation*, which incorporates similarity of $x$ and $y$ to other sequences from $S$ into the $x$–$y$ pairwise comparison:

$$\mathbf{P}'(x_i \sim y_j \in a^*|x,y) \leftarrow \frac{1}{|S|} \sum_{z \in S} \sum_{z_k} \mathbf{P}(x_i \sim z_k \in a^*|x,z)\mathbf{P}(z_k \sim y_j \in a^*|z,y).$$

In matrix form, the transformation may be written as

$$P'_{xy} \leftarrow \frac{1}{|S|} \sum_{z \in S} P_{xz} P_{zy}.$$

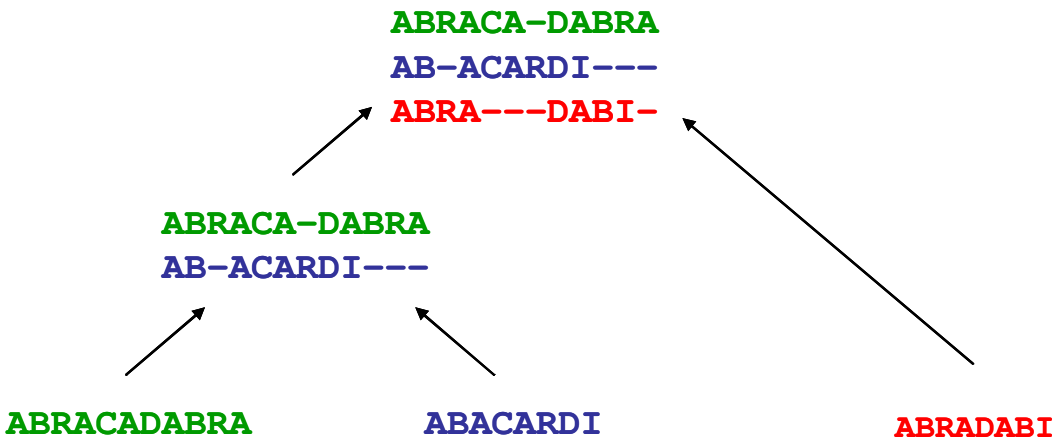
Since most values in the $P_{xz}$ and $P_{zy}$ matrices will be near zero, the transformation is computed efficiently using *sparse* matrix multiplication by ignoring all entries smaller than a threshold ω. This step may be repeated as many times as desired.

***Step 4: Computation of guide tree***

Construct a guide tree for $S$ through hierarchical clustering. As a measure of similarity between two sequences $x$ and $y$ use $E(x, y)$ as computed in Step 2. Define the similarity of two clusters by a weighted average of the pairwise similarities between sequences of the clusters.

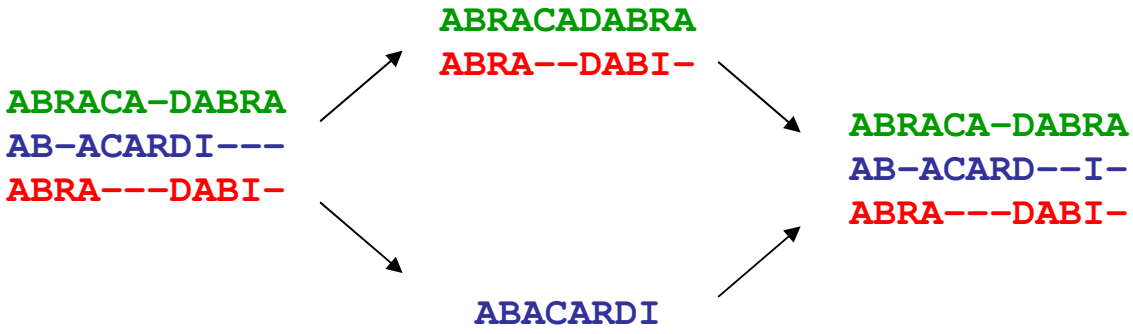
***Step 5: Progressive alignment***

Align sequence groups hierarchically according to the order specified in the guide tree. Alignments are scored using a sumof-pairs scoring function in which aligned residues are assigned the transformed match quality scores $\mathbf{P}\_(xi \sim yj \in a^* \mid x, y)$ and gap penalties are set to zero.




***Post-processing step: Iterative refinement***

Randomly partition alignment into two groups of sequences and realign. This step may be repeated as many times as desired. In this approach, the sequences of the existing multiple alignment are randomly partitioned into two groups of possibly unequal size by randomly assigning each sequence to one of the two groups to be realigned. Subsequently, the same dynamic programming procedure used for progressive alignment is employed to realign the two projected alignments. This refinement procedure can be iterated either for a fixed number of iterations or until convergence; for simplicity, only the former of these options is implemented in ProbCons, where 100 rounds of iterative refinement are applied in the default setting.



# 1.3>Test Results

Three reference benchmarks were used:

1. The BAliBASE 2.01 benchmark alignment database is a collection of 141 reference protein alignments, consisting of structural alignments from the FSSP and HOMSTRAD databases and hand constructed alignments from the literature.
2. The PREFAB 3.0 database is an automatically generated database consisting of 1932 alignments averaging 49 sequences of length 240.
3. The SABmark 1.63 database consists of two sets of consensus regions based on SOFI and CE structural alignments of sequences from the ASTRAL database.

- PROBCONS parameters trained via unsupervised EM on unaligned sequences from BAliBASE.
  Quality score is given by
  $Q(\alpha, \alpha^*)$ = (number of correct predicted matches)/(total number of true matches)

Results

| **Algorithm** | **BAliBASE (237)** | | **PREFAB (1932)** | | **SABmark (698)** | |
|---|---|---|---|---|---|---|
| | **Q** | **t** | **Q** | **t** | **Q** | **t** |
| Align-m | 0.804 | 19:25 | - | - | 0.352 | 56:44 |
| DIALIGN | 0.832 | 2:53 | 0.572 | 12:25:00 | 0.410 | 8:28 |
| CLUSTALW | 0.861 | 1:07 | 0.589 | 2:57:00 | 0.439 | 2:16 |
| MAFFT | 0.882 | 1:18 | 0.648 | 2:36:00 | 0.442 | 7:33 |
| T-Coffee | 0.883 | 21:31 | 0.636 | 144:51:00 | 0.456 | 59:10 |
| MUSCLE | 0.896 | 1:05 | 0.648 | 3:11:00 | 0.464 | 20:42 |
| PROBCONS | 0.910 | 5:32 | 0.668 | 19:41:00 | 0.505 | 17:20 |

It can be seen that Probcons gives the highest quality results while maintaining reasonable running times.

Some of the most famous resources for alignment are:
**Protein Multiple Aligners**
http://www.ebi.ac.uk/clustalw/
CLUSTALW – most widely used (1994)
http://phylogenomics.berkeley.edu/cgi-bin/muscle/input_muscle.py
MUSCLE – most scalable (2004)
http://probcons.stanford.edu/
PROBCONS – most accurate (2004)

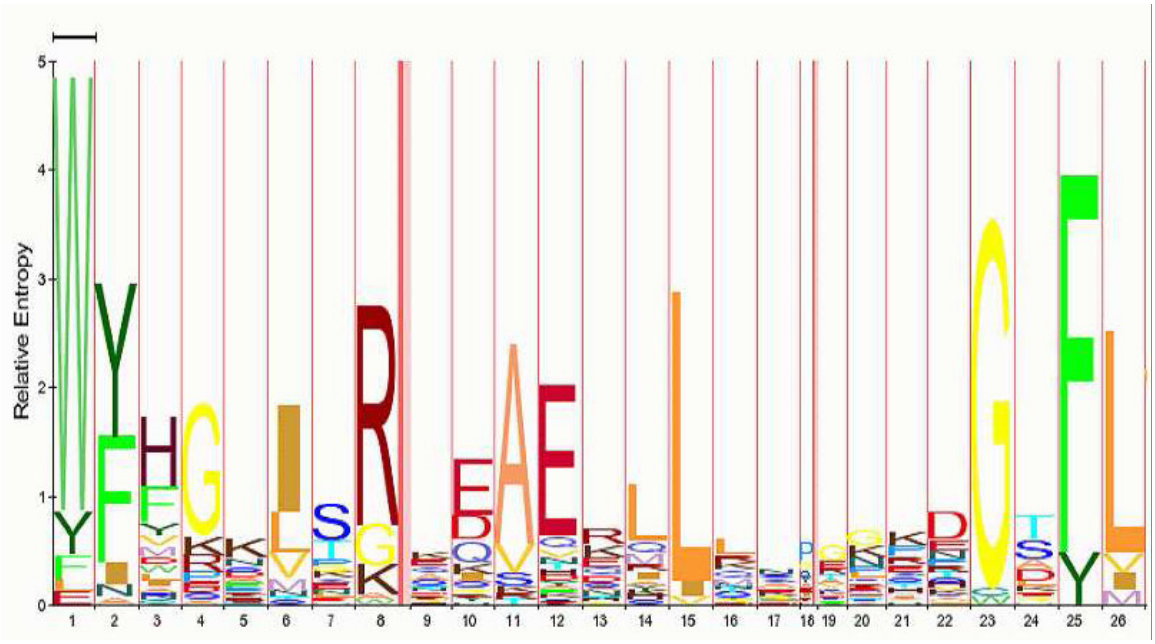
# 2.>Profile HMMs

## 2.1>Motivation for Profile HMMs

Number of known protein sequences and of solved 3D protein structures has been growing exponentially. However, the number of major distinct general shapes ("folds") that known proteins have has been converging to about 1000 (i.e. most newly discovered proteins fall into one of the already-known folds). Why is that? Perhaps because all proteins are related by evolution (new proteins can only come from long-established ones) so truly new shapes have little opportunity to arise. Or perhaps because there really only exist a few possible shapes that manage to meet the stringent requirements for a working protein, e.g. that it quickly old into native shape after being synthesized. Whatever the reason, this is good for biologists: when we find a new protein we can classify it into one of the 1000 known folds and already know a lot about it from the general properties of that fold.

Protein folds are divided into protein superfamilies which are divided into protein families. Proteins in a superfamily have faint evolutionary similarities detectable at the sequence level. Proteins in a family have sequence similarity of 25% or higher. Several protein classification systems exist. SCOP (scop.berkeley.edu) is entirely manual, has the highest classification quality, but covers fewest proteins. It is done mainly by one man (A. Murzin). CATH (www.biochem.ucl.ac.uk/bsm/cath ) is semi-manual, covers more proteins and is less precise than SCOP, and FSSP (www.ebi.ac.uk/dali/fssp/fssp.html) is completely automatic, covers more proteins than CATH and is less precise than CATH. Ideally we want an automatic and precise way to classify a new protein into an existing family, superfamily (if it belongs to no known families) or fold (if it belongs to no known superfamilies). Current approaches include: BLAST/PsiBLAST, Profile HMMs and Supervised machine learning methods. BLAST is simplest: you do one sequence-similarity search to find already-classified proteins with sequence similarity to the new protein. You assign to the new protein the classification from which you got the most BLAST hits to your protein. PsiBLAST is a more accurate method of doing the same thing: you first build a sequence profile of proteins that are similar to the new proteins, then BLAST the database of already-classified proteins with this profile instead of with the single sequence of the new protein. To build the profile, you 1) find pairwise alignments of your new protein to proteins in the database, which have scores above a given threshold; 2) build a sequence profile summarizing the alignments you found 3) repeat, now using the profile of proteins similar to the new protein sequence instead of the new protein sequence itself (i.e. find pairwise alignments of the profile to proteins in the database, etc). This approach is much more precise than BLAST, especially when your new protein is an outlier within its family. Precision of automatic protein classification methods can be measured by leaveone-out experiments (take an already-classified protein and see how the automatic method would have classified it based on all other known proteins – would we rediscover the correct classification?)

PSI Blast Profile

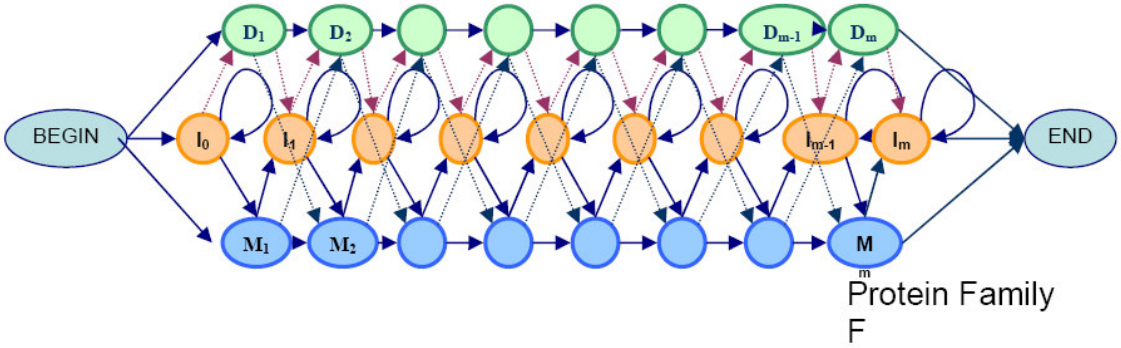



It gives the relative frequencies at each position of the various amino acids, as found in the protein sequences summarized by the profile. A limitation of using profiles to characterize protein families, is that a profile can't capture that "most proteins in this family have a gap at position 10". This can be a common variation across a family (e.g. if this is a point on the outside of the protein where loops of various lengths are easily added without disturbing protein function).Conversely, profiles also can't capture that "at position 10 there must NOT be a gap because this position is critical to protein function"; profiles can capture frequencies of amino acids at each position, but not frequencies of gaps. We need a model of protein families that lets us model the locations of gaps you would see if you did a multiple alignment of proteins in the family. Profile HMMs let us do just that. Krogh introduced an HMM architecture for representing profiles of multiple sequence alignments. Profile HMMs are strongly linear, left-right models, unlike the general HMM case.

Functional Biological sequences typically come in families, and many of the most powerful sequence analysis methods are based on identifying the relationship of an individual sequence to a sequence family. Sequences in a family will have diverged from each other in their primary sequence during evolution, having separated either by duplication in the genome or by speciation giving rise to corresponding sequences in the related organisms. In either case they normally maintain the same or related function. Therefore, identifying that a sequence belongs to a family and aligning it to other members often allows inferences about its function. Pairwise searching with any one of the members may not find sequences distantly related to ones you already have. Just as pairwise alignment captures much of the relationship between two sequences, a multiple alignment can show how the sequences I a family relate to each other. This is the primary motivation for studying profile HMMs.

# 2.2>Details about Profile HMMs

A profile HMM looks like this:




Note that

- Each M state has a position-specific pre-computed substitution table
- Each I state has position-specific gap penalties (and in principle can have its own emission distributions)
- Each D state also has position-specific gap penalties
  - In principle, D-D transitions can also be customized per position

**Notation:**

- transition between match states – **αM(i)M(i+1)**
- transitions between match and insert states – **αM(i)I(i), αI(i)M(i+1)**
- transition within insert state – **αI(i)I(i)**
- transition between match and delete states – **αM(i)D(i+1), αD(i)M(i+1)**
- transition within delete state – **αD(i)D(i+1)**
- emission of amino acid b at a state S – **eS(b)**

The emission alphabet of this HMM is the set of amino acids. This HMM is a generative model; the probability that it generates a given protein depends on how "typical" the protein is for the protein family represented by the HMM – or equivalently, how likely a given protein is to be part of this family. So, how is this HMM constructed? States in the bottom row correspond to profile positions with letters. If a profile position almost always contains a given amino acid, the corresponding M state almost always emits that amino acid; in general, the emission probability of an amino acid depends on the relative frequency of that amino acid in that position of the profile. The middle row states are insertion states. Their self-transition probabilities model lengths of gaps at that profile location and their emission probabilities can in principle model gap contents. The top row states are deletion states; transitions to these states model the likelihood of deletions at a given profile location.
The profile HMM is a concise summary of a multiple alignment of sequences in a protein family, and can be learned from a multiple alignment by converting the letter and gap frequencies at each position into emission and transition probabilities as suggested above. Given a protein sequence and a profile HMM, you can "align" the sequence to the HMM by finding the most likely path through the HMM that would generate the sequence. This can be computed using a DP algorithm.
Classification using profile HMMs can be done by computing a multiple alignment of sequences in each known protein family, summarizing this multiple alignment with a profile

HMM, then for an unknown protein sequence computing the probability that this sequence is produced by each family's protein HMM and assigning the protein the family whose profile HMM has the highest likelihood of producing the protein. Profile HMMs are generative models, and classification using generative models suffers from the problem that if there are not enough data points for some protein family, new proteins that are outliers in that protein family may not be correctly classified.

**Basic profile HMM parameterization:**
Like all HMMs, Profile HMMs have emission and transition probabilities. Assuming that these probabilities are nonzero, a profile HMM can model any possible sequence of residues from the given alphabet. It therefore defines the probability distribution over the whole space of sequences. The aim of the parameterization process is to make this distribution peak around the members of the family.
To do this we just count up the number of times each transition or emission is used, and assign probabilities according to

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \qquad e_k(a) = \frac{E_k(a)}{\sum_{a'} E_k(a')}$$

Where k and l are indices over states and $a_{kl}$ and $e_k$ are the transition and emission probabilities, and $A_{kl}$ and $E_k$ are the corresponding frequencies.
A major difficulty is that some transitions or emissions may not be seen in the training alignment, and so would acquire zero probabilities, which would mean they would never be allowed in future. As a minimal approach to avoid zero probabilities we can add pseudocounts to the observed frequencies.

**Alignment of a protein to a profile HMM:**
To align sequence $x_1 \ldots x_n$ to a profile HMM:

We will find the most likely alignment with the Viterbi DP algorithm

- Define
  - $V_j^M(i)$: score of best alignment of $x_1 \ldots x_i$ to the HMM ending in $x_i$ being emitted from $M_j$
  - $V_j^I(i)$: score of best alignment of $x_1 \ldots x_i$ to the HMM ending in $x_i$ being emitted from $I_j$
  - $V_j^D(i)$: score of best alignment of $x_1 \ldots x_i$ to the HMM ending in $D_j$ ($x_i$ is the last character emitted before $D_j$)
- Denote by $q_a$ the frequency of amino acid *a* in a 'random' protein

The Viterbi equations are given by:

- $V_j^M(i) = \log (e_{M(j)}(x_i) / q_{xi}) + \max \begin{matrix} V_{j-1}^M(i-1) + \log \alpha_{M(j-1)M(j)} \\ V_{j-1}^I(i-1) + \log \alpha_{I(j-1)M(j)} \\ V_{j-1}^D(i-1) + \log \alpha_{D(j-1)M(j)} \end{matrix}$

- $V_j^I(i) = \log (e_{I(j)}(x_i) / q_{xi}) + \max \begin{matrix} V_j^M(i-1) + \log \alpha_{M(j)I(j)} \\ V_j^I(i-1) + \log \alpha_{I(j)I(j)} \\ V_j^D(i-1) + \log \alpha_{D(j)I(j)} \end{matrix}$

- $V_j^D(i) = \max \begin{matrix} V_{j-1}^M(i) + \log \alpha_{M(j-1)D(j)} \\ V_{j-1}^I(i) + \log \alpha_{I(j-1)D(j)} \\ V_{j-1}^D(i) + \log \alpha_{D(j-1)D(j)} \end{matrix}$

These are the general equations. In a typical case, there is no emission score $e_{I(j)}(x_i)$ in the equation for $V_j^I(i)$ because we assume that emission distribution from the insert states $I_J$ is the same as the background distribution, so the probabilities cancel in the log odds form. Also D ➔ I and I ➔ D transition terms may not be present.

**Scoring system for Profile HMM**

The probability parameters in a profile HMM are usually converted to additive log-odds scores before aligning and scoring a query sequence. The scores for aligning a residue to a profile match state are therefore comparable to derivation of BLAST or FASTA scores: if the probability of the match state emitting residue x is $p_x$, and the expected background frequency of residue x in the sequence database is $f_x$ the score for residue x at this match state is $\log p_{x/} f_{x.}$

For other scores, profile HMM treatment diverges from standard alignment scoring. In traditional gap scoring alignment, an insert of x residues is typically scored with an affine gap penalty a + b(x-1), where a is the score for the first residue and b is the score for each subsequent residue in the insertion. In a profile HMM, for an insertion of length x there is a state transition into an insert state which costs $\log t_{MI}$ (where $t_{MI}$ is the state transition probability for moving from the match state to the insert state), (x-1) state transitions for each subsequent insert state that costs $\log t_{II}$ , and a state transition for leaving the insert state that costs $\log t_{IM}$ . This is akin to traditional affine gap penalty, with the gap open cost as $a = \log t_{MI} + \log t_{IM}$ and the gap extend cost as $b = \log t_{II.}$

# 3.>SAM

|The Sequence Analysis and Modeling system SAM is a collection of software tools for creating, refining and using linear Hidden Markov model for biological sequence analysis.

**Overview of SAM**

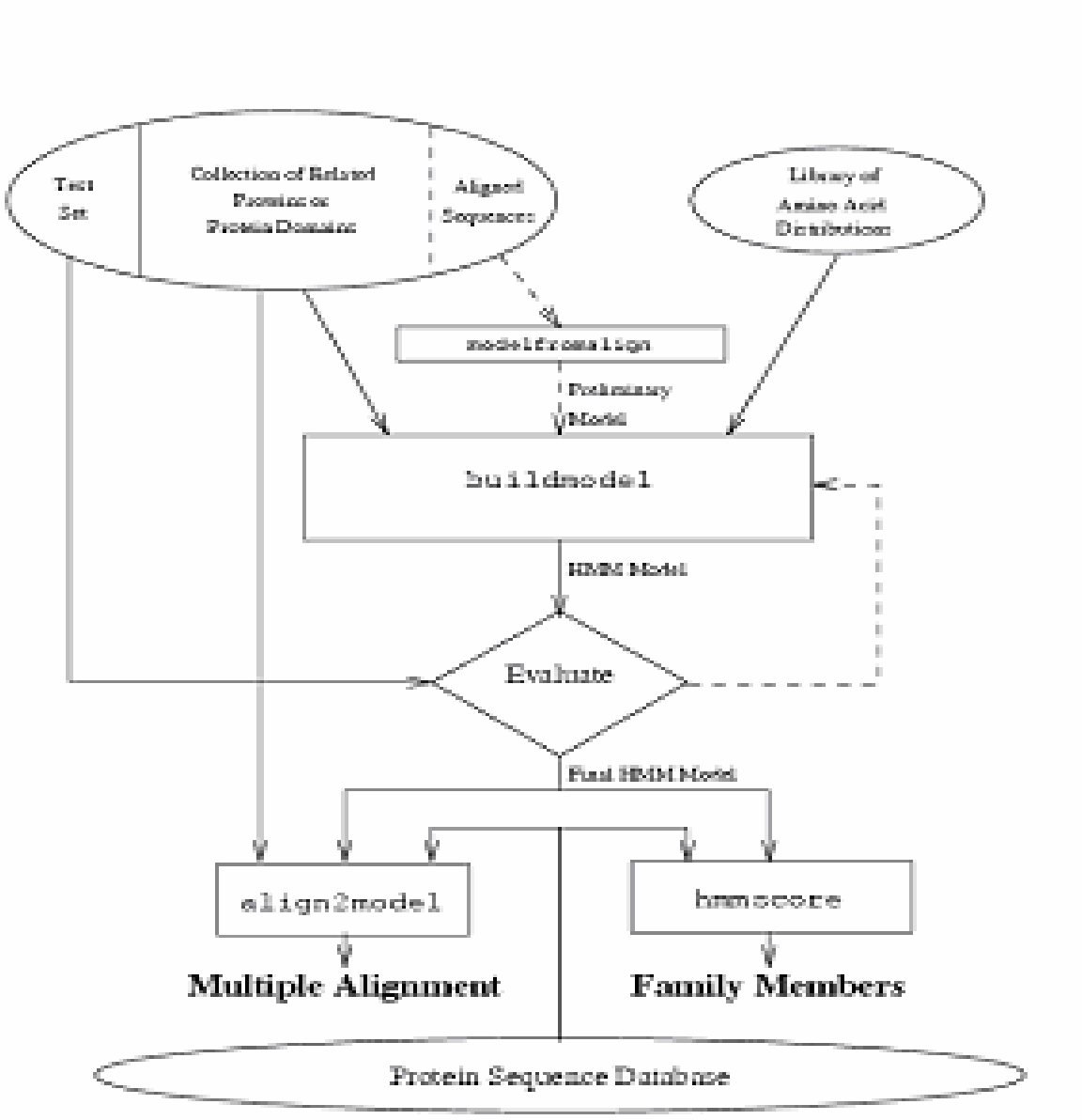
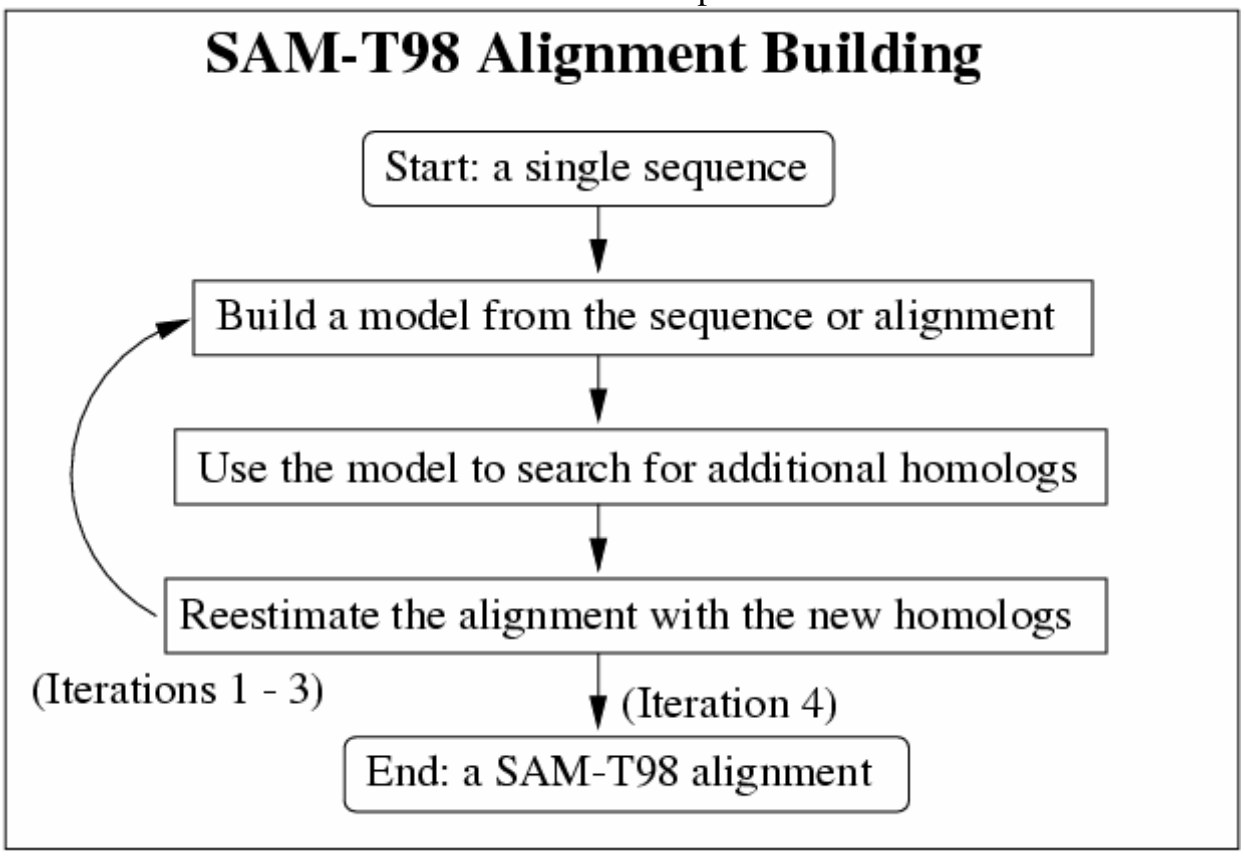



Figure 2: Overview of SAM.

UCSC's SAM-T02 method for iterative SAM HMM construction and remote homology detection and protein structure prediction updates SAM-T99 by using predicted secondary structure information in its scoring functions. Both SAM-T99 and SAM-T02 are "automatic" entries to CASP-5. SAM-T98 is obsolete.
Below we see how SAM-T98 was used to build profile-HMM



# 4.>Pfam

Pfam stands for **P**rotein **FAM**ilies database of alignments.
Each protein family is described by profile HMMs.
For each family in Pfam you can:

- Look at multiple alignments
- View protein domain architectures
- Examine species distribution
- Follow links to other databases
- View known protein structures

Pfam is a large collection of protein multiple sequence alignments and profile hidden Markov models. Structural data, where available, have been utilized to ensure that Pfam families correspond with structural domains, and to improve domain-based annotation. Predictions of non-domain regions are included. Pfam multiple sequence alignments contain active site residue markup. New search tools, including taxonomy search and domain query, greatly add to functionality and usability of Pfam resource.

Genome projects have used Pfam extensively for large scale functional annotation of genomic data. The multiple sequence alignments around which Pfam families are built are important tools for understanding protein structure and function and form the basis for techniques such as secondary structure prediction, fold recognition, phylogenetic analysis and mutation design.

Pfam consists of Pfam-A and Pfam-B
**Pfam-A:** Each curated family in Pfam is represented by a seed and full alignment. The seed contains representative members of the family as detected with a profile hidden Markov model (HMM) constructed from the seed alignment using the HMMer software.
**Pfam-B:** In an effort to be comprehensive the curated families in Pfam A are augmented by Pfam-B an automatically generated supplement derived from the PRODOM database.

**Pfam Annotation:**
Pfam contains annotation of each family in the form of textual descriptions, links to other resources and literature references. Pfam is a member of the InterPro consortium and has like the other members contributed annotation and families to InterPro Project.

**Domain Annotation:**
Domains are structural and functional building blocks of proteins and so where the data are available, structural information has been used to ensure that Pfam families correspond to single structural domains. The domain boundaries used are currently those defined by the SCOP database and a web-based tool allows direct cross-linking from domains on the SCOP website to the corresponding Pfam families. The

matching of families and domains enables enhanced understanding of the function of multi-domain proteins.

**Non-Domain Annotation:**
Although Pfam attempts to classify proteins into domains where possible, some regions of proteins are not expected to form stable globular domains. The include regions of biased amino acid composition, coiled coils, transmembrane regions and signal peptides. These regions are of considerable interest and so predictions are reported on the UK website.

**Taxonomy:**
The taxonomy search tool allows user to find Pfam entries to a specific to a group of organisms using a taxonomic query language.

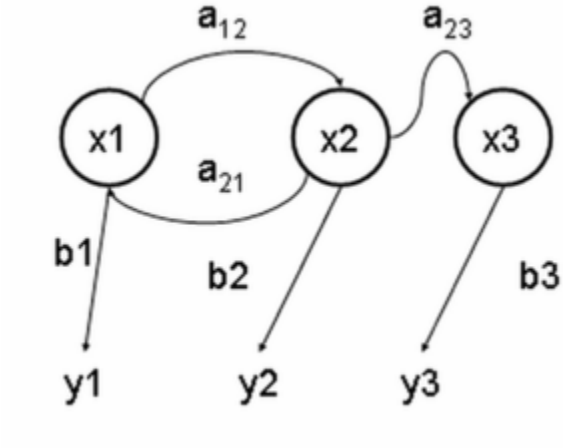
**The Pfam Relational Database:**
PfamRDB is a mySQL relational database consisting of approximately 10 tables adhering to a tight relational schema. It is updated in phase with the live Pfam database to maintain absolute consistency.

To summarize, some of the famous resources available on the web are

- HMMer – a free profile HMM software
  - http://hmmer.wustl.edu/
- SAM – another free profile HMM software
  - http://www.cse.ucsc.edu/research/compbio/sam.html
- PFAM – database of alignments and HMMs for protein families and domains
  - http://www.sanger.ac.uk/Software/Pfam/
- SCOP – a structural classification of proteins
  - http://scop.berkeley.edu/data/scop.b.html

# Appendix A
# Markov Chains and Hidden Markov models

Hidden Markov Models (HMMs) are a powerful tool for probabilistic analysis of existing data. Search and alignment constitute probably the best use of HMMs for biological sequence analysis. HMMs allow us to develop methods for sequence alignment that, although probabilistic in nature rather than optimal, run much faster and prove very useful.




State transitions in a hidden Markov model (example)
*x* — hidden states
*y* — observable outputs
*a* — transition probabilities
*b* — output probabilities

Definition of a Hidden Markov model:
The formal definition of a HMM is a system consisting of:

- **An alphabet** $\Sigma = \{b_1, b_2 \ldots b_M\}$
  These are the "letters" emitted by the system. A HMM is also called a Generatic Model since it generates a sequence of letters.
- **A finite set of states** $Q = \{1, 2 \ldots K\}$
  The system is in a particular state at any time, affecting what letters it is likely to emit.
- **Transition probabilities**
  $a_{ij}$ is the probability that if the system is in state i, the next state it jumps to is state j
  Note that the system can jump back into the same state, i.e. $a_{ii}$ can be non-zero.
  Also the graph of states and transitions need not be complete, i.e. some $a_{ij}$ can be zero
  Also note that the system must jump from its current state into *some* state, so:
  $a_{i1} + a_{i2} + \ldots + a_{iK} = 1$ for any state *i*
- **Start probabilities**
  $a_{0i}$ is the probability that the system starts in state i
  Again we must have $a_{01} + a_{02} + \ldots a_{0K} = 1$ since the system has to start in some state
- **Emission probabilities (for each state)**
  Before jumping to the next state, the system emits a letter

The emission probability $e_i(b)$ is the probability of emitting letter *b* given that the system is in state *I*, i.e. $e_i(b) = \Pr(x_i = b \mid \pi = i)$
Since the system *always* emits a letter before jumping, $e_i(b_1) + e_i(b_2) + \ldots + e_i(b_M) = 1$

A Hidden Markov Model is referred to as memory-less. This is because the future of the system (i.e. what letters will be output in the future) depends only on the current state of the system and not any previous states. This follows fairly straightforwardly from the definition since the transition and emission probabilities depend only on the current state. Thus it is as if the HMM has no "memory" of what has happened in the past. The formal way of stating this precisely and mathematically is:

"At each time step t, the only thing that affects future states is the current state $\pi_t$"

$P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_k, x_1, x_2, \ldots, x_k)$ whatever states and letters we had in the past

$= P(\pi_{t+1} = k \mid \pi_k)$ since only the current state matters

In general, this is seen as an advantageous property since it simplifies calculation and increases speed when generating these sequences. The disadvantage is that this limitation means that certain problems cannot be modeled by a HMM.

The 3 fundamental questions of HMMs

**1) Evaluation**

How likely is a given sequence of letters given our choice of model?
Formally, given a HMM M and a sequence of letters *x*, find $\Pr[x \mid M]$.

**The forward algorithm**

We can formulate the evaluation problem as a Dynamic Program if we define the forward probability $f_k(i)$ to be the probability of emitting the first *i* letters and ending in state *k*, i.e.

$f_k(i) = \Pr(x_1 x_2 \ldots x_i, \pi_i = k)$

$= \sum_{\pi} \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1}, \pi_i = k) e_k(x_i)$ summing up over all paths π

$= \sum_{l} \sum_{\pi} \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-2}, \pi_{i-1} = l) a_{lk} e_k(x_i)$ summing over states $\pi_{i-1}$

$= e_k(x_i) \sum_{l} f_l(i-1)\, a_{lk}$ by our definition

Since we can calculate $f_k(i)$ if we know $f_l(i-1)$ for all states *l*, we can set up a Dynamic Program:

Initialization: $f_0(0) = 1$ (0 is the imaginary state before beginning)

$f_k(0) = 0$ for all $k > 0$ (since no letters have been emitted yet)

Iteration: $f_k(i) = e_k(x_i) \sum_{l} f_l(i-1)\, a_{lk}$

Termination: $\Pr(x) = \sum_{k} f_k(n) a_{k0}$ (since the system has to finish in some state) where $a_{k0}$ is the probability that the terminating state is k (usually = $a_{0k}$)

This algorithm effectively fills in a table of $Kn$ values. The total time complexity is $O(K^2 n)$ and the space complexity is $O(Kn)$.

2) **Decoding**

Given a sequence of letters, trying to determine what state the system was in at each point, i.e. trying to assign a state to each letter. A parse is a sequence of states of the HMM such that at state s $i$ the letter x $i$ of the sequence was emitted. Formally, given a HMM M, and a sequence of letters $x$, find the sequence $\pi$ of states that maximizes $\Pr(x, \pi \mid M)$.

We define $V_k(i)$ to the probability of the best parse ending at state $\pi_i = k$.

Formally, $V_k(i) = \max_{\pi} \Pr(x_1 x_2 ... x_i, \pi_1, \pi_2, ..., \pi_{i-1}, \pi_i = k)$

To phrase as a Dynamic Program, we would like to be able to calculate $V_k(i)$ recursively. The intuition behind this is that if we already know the probability of the best parse ending at state $\pi_{i-1} = l$ for each state $l$, we can determine what state for $\pi_i$ is most likely since the transition probabilities and emission probabilities are known. As for the math:

$$
\begin{aligned}
V_k(i) &= \max_{\pi} \Pr(x_1 x_2 ... x_i, \pi_1, \pi_2, ..., \pi_{i-1}, \pi_i = k) \\
&= \max_{\pi} \Pr(x_i, \pi_i = k \mid x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1}) \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1}) \\
&= \max_{\pi} \Pr(x_i, \pi_i = k \mid \pi_{i-1}) \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1}) \text{ since memory-less} \\
&= \max_{l} \left( \max_{\pi} \Pr(x_i, \pi_i = k \mid \pi_{i-1} = l) \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1} = l) \right) \\
&\qquad \text{summing over all possible states for } \pi_{i-1} \\
&= \max_{l} \left( \Pr(x_i, \pi_i = k \mid \pi_{i-1} = l) \max_{\pi} \Pr(x_1 x_2 ... x_{i-1}, \pi_1, \pi_2, ..., \pi_{i-1} = l) \right) \\
&= \max_{l} \left( \Pr(x_i, \pi_i = k \mid \pi_{i-1} = l) V_l(i-1) \right) \qquad \text{by our definition} \\
&= \max_{l} \left( a_{lk} e_k(x_i) V_l(i-1) \right) \\
&= e_k(x_i) \max_{l} \left( a_{lk} V_l(i-1) \right)
\end{aligned}
$$

The Dynamic Program to solve this recurrence is known as the Viterbi Algorithm:

Initialization: $V_0(0) = 1$

$V_k(0) = 0$ for all $k > 0$ (just like the Forward Algorithm)

Iteration: $V_j(i) = e_j(x_i) \max_k \left( a_{kj} V_k(i-1) \right)$

Let $Ptr_j(i)$ store the value of $k$ that was used to maximize the above value

Termination: $\Pr(x \mid \pi^*) = \max_k V_k(n)$

Traceback: $\pi_n^* =$ the value of $k$ that was used to maximize $\Pr(x \mid \pi^*)$ in the termination

$\pi_{i-1}^* = Ptr_{\pi_i^*}(i)$

Just as in the Forward Algorithm, we are creating a table of $Kn$ entries, where each entry is constructed by maximizing (rather than summing) over $K$ values, resulting in a time complexity of $O(K^2n)$ and a space complexity of $O(Kn)$.

### 3) Learning

Given a sequence of states the system was in, trying to determine the parameters (transition and emission probabilities) of the system.
Formally, given a HMM with unspecified transition and emission probabilities, and a sequence of letters $x$, find parameters $\theta$ that maximize $\Pr(x \mid \theta)$

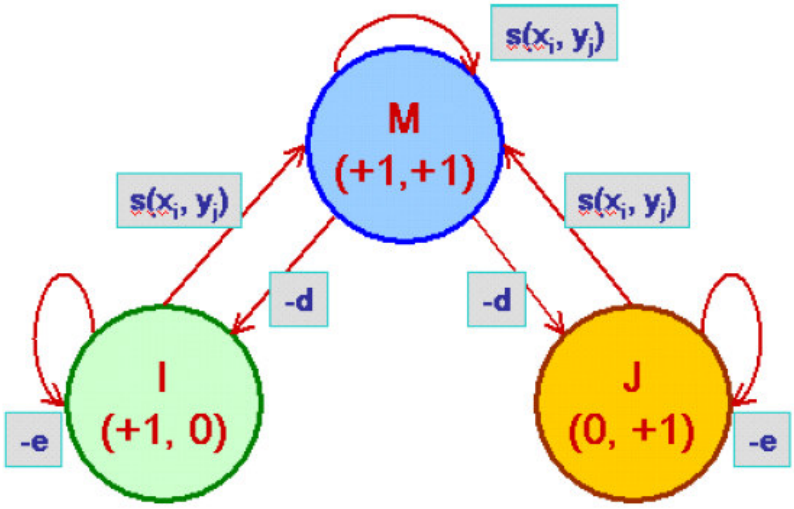
**Learning Scenarios**

There are two different type of learning scenarios depending on the type of data available with us. These are in some ways related to supervised and unsupervised learning scenarios in machine learning. I will not go in detail about this, since it is not pertinent to the lecture I am scribing.

# Appendix B
# Pairwise Alignment using HMMs

**Finite State Automaton for Alignment**

We can create a state model that corresponds to generating an alignment between two sequences x and y. There is a one-to-one correspondence between alignments and strings composed of M, I, and J, which is also a path through the automaton, such that the sum of +1 terms for x is equal to the length of sequence x and the sum of +1 terms for y is equal to the length of sequence y.




```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC-GGTCGATTTGCCCGACC
IMMJMMMMMMMJJMMMMMMJMMMMMMMIIMMMMMIII
```

We have labeled every transition in the model with a score. Transitions to state M indicate letter-to-letter correspondences, so they are labeled with $S(x_i, y_j)$ corresponding to the substitution score for replacing $x_i$ with $y_j$. We know which i and j to use, based on the current sum of +1's for x and y thus far. We label every transition from M to a gap state (I or J) with the gap initiation penalty -d, and we label each transition from a gap state to itself with the gap extension penalty –e.

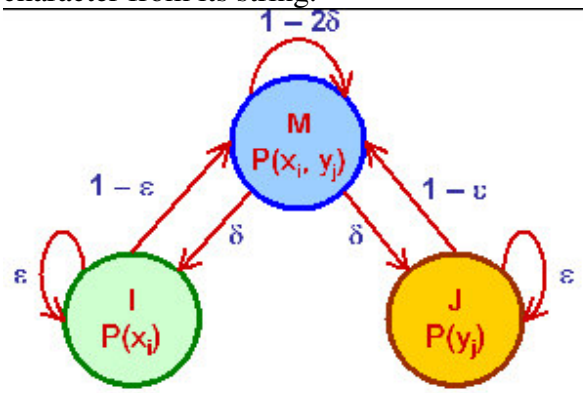
Every path through this model corresponds to an alignment. If we sum every score on the transitions of a path, we will have the same score as a global alignment dynamic programming problem with affine gap penalties.

**Probabilistic Interpretation of Alignment**

An alignment is a hypothesis that two sequences are related by evolution. We would like to find the *most likely* alignment rather than using arbitrary scoring parameters. We can then assert the likelihood that the sequences are related.

We will now consider this automaton as a pair Hidden Markov Model that produces an alignment between two sequences (again we are ignoring possible transitions between I and J, or possible start and end states). We have three states M, I, and J, as

shown in Model M on the left, below. Each state emits a pair of characters. State M emits one character from X and one character from Y. State I emits a character in X and a gap in Y. State J emits a gap in X and a character in Y. The probabilities of transition are described in the diagram (on edges) – delta affects how often gaps are opened, and epsilon affects how long gaps last.
To be more precise:
**delta:** set so that $1/ 2\delta$ is avg. length before next match
**epsilon:** set so that $1/(1 - \epsilon)$ is avg. length of a gap
Emission probabilities are listed inside the nodes. The emission probability in the model reflects likelihood of aligning $x_i$ and $y_j$, which could be calculated using BLOSUM matrices. The emission probabilities in I and J reflect the composition of the sequences X and Y respectively. They model the likelihood of emitting each character from its string.




This model assumes that the two sequences are related, since we generate them simultaneously.

We will first examine how each pair of letters contributes to the likelihood of the alignment:
$(1 - 2\delta)$ P(xi, yj) when aligned
$\epsilon$ P(xi) P(yj) when gapped
The maximum likelihood alignment is the alignment that will be produced by Needleman-Wunsch global alignment with affine gaps. Therefore, we have described an HMM where computing the maximum likelihood alignment is the same as computing global alignment with affine gaps. This means that running the Viterbi algorithm on a Pair HMM is equivalent to running Needleman-Wunsch dynamic programming. The equivalence makes sense when we examine the Viterbi recurrences:
$V_M(i, j) = \max \{ V_M(i - 1, j - 1), V_I(i - 1, j - 1), V_J(i - 1, j - 1) \} + s(x_i, y_j)$
$V_I(i, j) = \max \{ V_M(i - 1, j) - d, V_I(i - 1, j) - e \}$
$V_J(i, j) = \max \{ V_M(i, j - 1) - d, V_J(i, j - 1) - e \}$

References

1.>Lecture slides from CS 273 class.
2.>Biological Sequence Analysis – Durbin et.al.
3.>Profile Hidden Markov Models – Sean Eddy
4.>The many faces of sequence alignment – Serafim Batzoglou
5.>ProbCons: Probabilistic consistency-based multiple sequence alignment. Do et.al.
6.>The Pfam protein Families database – Bateman et.al.
7.>SAM Sequence Alignment and Modeling System – Richard Hughey et.al.