

Sequence Alignment and Stochastic Processes

Daniel F. Simola

28 May 2005

1 Sequence alignment algorithms

Pairwise and multiple sequence alignment algorithms are framed in the typical dynamic programming scheme (combinatorial optimization), using affine gap penalty functions. All of this can be generalized in terms of hidden markov models, as seen in Durbin's *BSA*. The goal of alignment algorithms is to determine the amount of homology between sequences (positional homology).

1.1 Pairwise alignment

1.1.1 Dynamic Programming algorithm based on a 3-state FSA using an affine gap function

- Initialize using affine penalty
 - $V(i, 0) = -W_o - (i - 1)W_e$
 - $V(0, j) = -W_o - (j - 1)W_e$
 - W_o is the gap opening penalty
 - W_e is the gap extension penalty
- In a bottom-up fashion, choose the best alignment at the current index pair $(M(i, j))$ based on the previous best alignment path plus the score of the alignment at the current position
 - $M(i, j) = \max[M(i - 1, j - 1), I_x(i - 1, j - 1), I_y(i - 1, j - 1)] + s(x_i, y_j)$
 - $I_x(i, j) = \max[I_x(i - 1, j - 1) - W_e, M(i - 1, j) - W_o]$ is the best score given x_i is aligned to a gap in y
 - $I_y(i, j) = \max[I_y(i - 1, j - 1) - W_e, M(i, j - 1) - W_o]$ is the best score given y_j is aligned to a gap in x
 - $s(i, j)$ is the alignment score (match/mismatch) of the base at index i to the base at index j

NB, W_o , W_e , and $s(\bullet, \bullet)$ are fixed values, whereas you need to store the values of I_x , I_y , and M each time you run the algorithm. This runs in $O(nm)$ time.

1.1.2 Basic alignment procedure

Goal: find positional homologies relating ≥ 2 sequences

Edit operations: insert, delete, substitute

Objective function: formal description relating sequence input to a metric score

- eg, $s(x, y) = 1$ if $x = y$, 0 otherwise is a similarity objective function relating two nucleotides x and y .
- can also compare position profiles: $S(X, Y) = f(s(x_1, y_1), s(x_2, y_2), s(x_3, y_3), s(x_4, y_4))$ (f is typically an average)

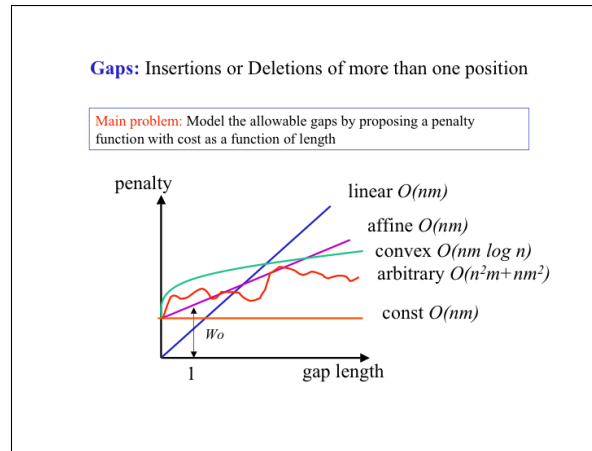


Figure 1: From Junhyong Kim's lecture notes 10.05

Optimization: typically arrive at a solution by optimizing the objective function (algorithmic/computational solution). This is typically done using a dynamic programming type algorithm.

Establish a recurrence relation: given position indices into X and Y , what are the possible ways to align x_i and y_j ?

Score these options: best score for an option is the best score so far plus the score for the alignment relation for i, j

Solve: elegant solution is recursion, practical solution is memoization (store computations progressively)

Read solution: As you solve the problem, remember the best paths from start to finish. At the end, traceback through the possible paths choosing the shortest path from finish to start.

1.1.3 Generalizations of the scoring function $s(x, y)$

- Assign different weights to insertion (I), deletion D , and substitution (S) edit moves (NB $w(S) < w(I) + w(D)$, since every substitution is a combination of insertion and deletion events)
- Assign alphabet-specific substitution weights, in the form of a matrix of (i, j) pairs. (eg PAM and BLOSUM cost matrices, computed from genomic data. NB these commonly used for amino acid alphabets.)

1.1.4 Gap penalty functions

- How to model multiple adjacent gaps? Most often use **affine** penalty function, which simply uses one penalty value for starting a gap and another for extending a gap, where this latter penalty is proportional to the length of the gap.
- NB affine means “connected with”, and geometrically, an affine transformation consists of a linear transformation (the gap opening penalty) followed by translation (extension penalty, which is linearly proportional)
- Only need to remember whether a gap was already opened at each index, thus remember the best alignment score that ended in a gap for each of the possible ways to align two indices, and the best overall score so far at an index pair.

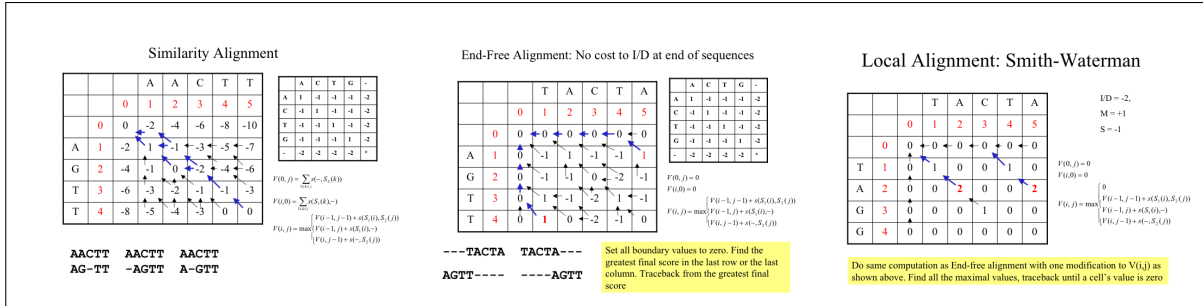


Figure 2: From Junhyong Kim's lecture notes 10.05

1.1.5 Sequence alignment variants

- **Global alignment** is the procedure described above - simply align two strings in their entirety, producing a single, globally optimal alignment (Needleman-Wunsch algorithm). This is most useful for sequence sets which have conserved segments as maximal similarity.
- **End-free alignment** is a variant of global alignment which simply does not penalize for gaps at sequence termini. This is useful if you expect one sequence to be contained within or overlap with another sequence, eg if you are aligning DNA sequence fragments for shotgun genome sequencing. Algorithmically just set the first row and column values to 0 in the score matrix.
- Use **local alignment** when you expect that only substrings of the input sequences should actually align well, eg if two sequences have diverged substantially. Thus the idea is to compute many alignments between substrings, and return a list of the best alignments. To modify global alignment to be local, choose the best alignment score or 0 at each position:

$$V(i, j) = \max[0, M(i, j), I_x(i, j), I_y(i, j)]$$

You find the best local alignments by searching the score matrix for the highest scoring cells, and tracing back from each. Thus local alignment is most useful for sequence sets which have conserved synteny as maximal similarity.

- **Repeat-finding alignment** is used when you expect subsequences of one string to be found multiple times in another string, eg repeats or TFBSs. This is a variant of local alignment where instead of starting over if the score becomes negative, you start with a repeat.
- **End-free repeat** is another variant which combines the overlap and repeat methods in order to find a set of repeats from X within a sequence Y where the repeats may overlap at the ends without penalty.
- **Tandem repeat finding** is used for example by the repeat masker algorithm. This variant is based on the overlap repeat variant.

See Durbin's book for all the recursion rules.

1.2 Multiple alignment

Need a concrete objective function measuring the distance between a set of objects. This is typically unclear, so most multiple alignment algorithms operate on pairs of sequences ($\binom{N}{2}$ such pairs).

1.2.1 Consensus objective function

The consensus objective function returns a consensus alignment error, given a set of sequences: $\epsilon_c = \sum_{i=1}^q d(i)$, q is number of columns of longest string, $d(i)$ is number of mismatches in column i .

1.2.2 Profile alignment

Profile: a set of positions with associated relative weights. Intuitively capture the notion of a class, type, or family. A consensus sequence is a realization of a particular profile.

Alignment: Align a sequence against a profile by averaging the scoring matrix over the profile weights for each position. The alignment score is the sum of these averages.

Sequence to profile and profile to profile alignments are thus possible.

1.2.3 Multiple alignment methods

Sum-of-pairs alignment: Find the alignment s.t. the sum of pairwise alignment scores is optimized. This is just a basic extension of pairwise alignment and can use either edit distance or consensus objective functions. Complexity: $O(N^k)$, too hard for $N > 5$. Next step, only align a subset of all pairs.

Tree alignment: SP alignment can be seen as a fully-connected graph, but only need to look at a tree (acyclic) in the graph, which decouples dependencies between alignments, permitting independent pairwise alignment. Vertices are sequences. A multiple alignment is consistent with a given alignment tree if sum of pairs is optimal for adjacent vertices. Progressive alignment algorithm (heuristic) (given a set of sequences and a tree):

1. Align 2 vertices X and Y on an edge, creating multiple alignment $X'Y'$
2. Align some Z to Y' (Z adjacent to Y'), creating $Z'Y''$
3. Unless Y'' has more gaps than Y' , add Z' to $X'Y' = X'Y''$. Else, add gap characters to the end of every string in $X'Y'$, creating $X''Y''Z'$
4. Repeat 2 & 3 until no unaligned sequences

Time complexity: $O(NL^2)$, N is the number of strings, L is the length of a string, with each pairwise alignment taking $O(L^2)$.

Clustering-type tree alignment: Steiner trees and evolutionary trees.

Steiner tree alignment: A Steiner tree is a star tree with central string X , which need not be an input string (unlike tree alignment). Alignment of Steiner tree involves determining which string should be in the center, is equivalent to the Optimal Consensus Alignment problem, and is NP hard.

Evolutionary tree alignment: All non-terminal vertices have unknown sequences. The alignment problem is to determine the internal strings such that the given alignment tree is optimal. This is an extension of a Steiner tree and also NP hard.

Determining the alignment tree: Heuristic solutions work better than approximate ones. Estimate the tree in the following manner:

1. Compute all pairwise alignments, ranking by distance
2. use rankings to construct a minimal spanning tree (all vertices) for tree alignment or Steiner tree (all vertices and can add new ones) for evolutionary tree. Can then use consensus OF to determine hidden strings.

Minimum spanning tree alignment:

1. compute all $\binom{N}{2}$ pairwise alignments and select the optimal pair.
2. Merge an unaligned string, which is closest to one of the aligned strings. Repeat.

Cluster alignment:

1. compute all $\binom{N}{2}$ pairwise alignments and cluster using distances.
2. Create a tree where internal vertices are either consensus or profiles.

1.3 Classification of real-life (heuristic) implementations

MLAGAN: progressive global alignment using SP metric, fixed guide tree and local homologous anchors.

1. multi-anchor sequences using CHAOS algorithm
2. progressive alignment using given tree in order of decreasing similarity
3. iterative refinement using LAGAN (remove a sequence x , realign other sequences using anchors, then align x to the multiple alignment).

CLUSTALW: progressive alignment constructing evolutionary tree using neighbor joining algorithm, followed by pairwise alignment in order of decreasing similarity

FASTA: local alignment using exact length k-mers. Extend k-mer diagonal runs in alignment matrix, then connect the runs. Also locally align (inexact) best diagonal run.

BLAST: local alignment between query and a database, using high scoring pairs (similar to local anchors) bounded by a cutoff, which are extended in both directions. The idea is that true alignments are most likely to contain short stretches of these high scoring segments. Random walk theory is used to assess probability of a HSP. Variants of blast are available to align almost anything.

MultiPIPMaker: local alignment using a reference sequence and a set of secondary sequences to be aligned.

1. pairwise alignments using blastz. Prune overlapping local alignments so that each position in the reference aligns to at most one position in a secondary sequence.
2. Serialize remaining local subsequences, distinguishing between end-gaps (not penalized) and internal gaps (typical gap open/gap extend penalty). This allows end-gaps to float freely within the sequence's row in the alignment.

1.4 Choosing an alignment method

- Manual sequence editing of unalignable regions
- Search over parameter space for optimal penalty scheme
- Concatenate subsequences aligned using different algorithms

2 Stochastic processes

Stochastic processes simply describe the accumulation of stochastic (random) events over time. A family of random variables is called a stochastic process if it can be viewed as a sequence of random variables, or if the family can be indexed by a real valued parameter. eg, $S(t)$ is a random value at index t , where $S(t)$ is the state of the process at t .

2.1 Categories and Markov models

The index (time) and state variable can be either discrete or continuous. The probability of a state transition can either depend on the current index and state, not depend on index but depend on state (common), or be independent of both state and index. See the section on Finite State Automata for the relation between Markov chains and Regular grammars.

		State	
		Discrete	Continuous
Time	Discrete	Discrete Markov chain	Discrete Markov process
	Continuous	Continuous time Markov chain	Brownian motion

Modeling with Random Variables:

How? Two ways: Either by arguing “looks like” or by some stochastic process

- **Binomial and multinomial:** Counts of things falling into k categories (e.g., A, C, G, T) drawing n times with fixed probability of drawing from each category.
- **Poisson:** Counts of rare events over some interval (e.g., number of mutations at a gene over time, number of defects in a picture of microarray experiment)
- **Exponential:** Waiting time for an event (e.g., waiting time until the next mutation)
- **Normal:** Many different quantitative traits (e.g., expression level of a gene, body weight, enzyme activity)
- **Beta, Dirichlet:** Prior distribution of the parameters of binomial and multinomial distributions

n -step transition probabilities are calculated using

$$P_{ij}^{(n)} = \sum_k P_{ik}^{(n-1)} P_{kj}$$

Markov assumptions are used to model phenomena for which either a fully independent or a fully complex process is unnecessary or inappropriate. eg for a DNA sequence of n positions, we could create the following models:

Full complexity model: $P(s_1, s_2, \dots, s_n)$

Independent model (profile): $P(s_1, s_2, \dots, s_n) = \prod_{i=1}^n P(s_i)$

First-order Markov model: $P(s_1, s_2, \dots, s_n) = P(s_n | s_{n-1}) P(s_{n-1} | s_{n-2}) \cdots P(s_1)$

There is a lot of information on Markov models, as well as many definitions that are left undefined here, including equilibrium/stationary distributions (finite, irreducible, aperiodic), waiting time, state space, the formal definition of Markov model, etc. See also notes on phylogeny.

2.2 Hidden Markov models (HMM)

Ideally one would like a framework that better captures dependencies in the data by being more flexible than the profile model above, where independent marginals are used (which is just a special case of HMMs). Thus an HMM is a stochastic process model used to describe sequence families; the states at each index may not be observable, and so there is a notion of a “hidden” state, and some observable state (or set of states) that exist as a function of (are emitted from) the hidden state, at each index, according to some probability distribution. More formally a HMM contains the following elements:

Hidden state set: $S = \{s_1, s_2, \dots, s_n\}$

Observable state set: $O = \{o_1, o_2, \dots, o_k\}$

State transition matrix: $\{a_{ij}\}$

Emission probability distribution, indexed by $s \in S$: $P(O|S)$

Notice three problems arise:

1. Estimation from data: How to construct a model and choose appropriate parameters?
2. Scoring: How to determine whether a given sequence belongs to this model?

3. Generation (Viterbi): Given a sequence, what realization of this model best explains the sequence?

Three recursive solutions for Scoring and Generation (Likelihood, Viterbi, forward-backward):

Likelihood: The probability of your observed sequence, given the hidden sequence/path Π and the parameter set Θ , used for the Scoring problem.

Transition probs: $a_{kl} = P(\pi_i = l | \pi_{i-1} = k)$

Emission probs: $e_k(b) = P(x_i = b | \pi_i = k)$

$$\begin{aligned} P(X|\Pi, \Theta) &= \frac{P(X, \Pi, \Theta)}{P(\Pi, \Theta)} \\ &\sim P(X, \Pi, \Theta) \\ &= a_{0\pi_1} \prod_{i=1}^n e_{\pi_i}(x_i) a_{\pi_i, \pi_{i+1}} \end{aligned}$$

Viterbi, most probable hidden state sequence: used for the Generation problem.

The path with highest probability is $\Pi^* = \underset{\Theta}{\operatorname{argmax}} P(\Pi|X, \Theta) = \underset{\Theta}{\operatorname{argmax}} P(X, \Pi, \Theta)$.

The recursive expression is $v_l(i+1) = e_l(x_{i+1}) \max_k (v_k(i) a_{kl})$. See Durbin's *BSA* or J. Kim's notes for the dynamic programming algorithm, although it is almost identical to the algorithm presented at the beginning of this document.

Marginal probability of hidden state: $P(\pi_i = k|X, \Theta)$, used in the calculation of forward-backward.

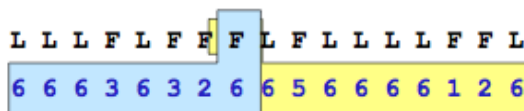
Forward-backward: The problem with Viterbi is that it only returns the single most probable path, which is not necessarily the most realistic biologically. However computing all paths gets exponentially hairy. We can get an exact expression for $P(X)$, however, using a generalization of Viterbi, by simply replacing the max by a sum: $f_l(i+1) = e_l(x_{i+1}) \sum_k (f_k(i) a_{kl})$.

In order to determine any suboptimal path probability, we need a posterior probability $P(\Pi|X, \Theta)$.

We calculate this using the relation $P(\pi_i = k|X, \Theta) = \frac{f_k(i) b_k(i)}{P(X, \Theta)}$.

The backward recursive expression is $b_k(i) = P(x_{i+1}, \dots, x_n | \pi_i = k)$.

Thus the basic idea is to partition the joint $P(X, \Pi, \Theta)$ into three expressions, a forward one $P(x_1, \dots, x_i, \pi_i = k)$, a backward one $P(x_{i+1}, \dots, x_n | \pi_i = k)$, and the joint $P(X, \Theta)$.



Partitioning of a sequence into forward and backward pieces.

2.2.1 HMM vs FSA

A finite state automaton can be generalized into a hidden markov model framework, and so are very similar approaches. However using a HMM is generally a better thing to do for a few reasons:

- HMMs explicitly incorporate probabilistic models for searching, and so you can use optimal statistics like ML. A consequence of this (and a distinction from FSA optimality) is that the optimal score is computed using the full probability of a pair of sequences $P(x, y | M)$ rather than the single most probable (Viterbi) path $P(x, y | \Pi^*)$.

2.2.2 Parameter estimation

The above calculations all assumed that the parameters used were known, however most of the time they will need to be estimated from the data. The set of parameters includes the transition and emission probabilities. Parameter estimation is a statistical procedure where you want to calculate an unbiased estimate of some parameter in an optimal way from a sampling distribution induced from your data set. Estimation procedures break down into two categories, whether or not you have labeled sequences:

- Labeled sequences (transitions are known): Just count transitions and emissions from data.
- No prior knowledge (this often involves the EM algorithm):
 1. Maximum Likelihood: $\Theta^{ML} = \underset{\Theta}{\operatorname{argmax}} P(X|\Theta) \sim \underset{\Theta}{\operatorname{argmax}} P(\Theta|X)$ ML is consistent (will return the correct parameter value given infinite data), but can yield poor estimates with little data.
 2. Maximum *a-posteriori* (MAP): $\Theta^{MAP} = \underset{\Theta}{\operatorname{argmax}} P(X|\Theta)P(\Theta)$ You can incorporate prior knowledge by choosing the prior distribution of Θ .
 3. Least Squares
 4. Method of moments

When data is sparse enough that you might not have positive frequencies for all transitions, you can add small-valued pseudocounts to each of the outcomes of an individual random variable. This is done in a more formal fashion through a Bayesian prior distribution.

2.2.3 Profile HMM

These are HMMs suited for modeling multiple alignments, are often used to compare a sequence of some sequence (DNA or amino acid sequence) to a model of a particular family or class of sequences (eg globin protein). Need lots of data to properly estimate parameters (many parameters).

2.2.4 Pair HMM

As mentioned above, sequence alignment problems are generalized in the framework of HMMs. See Durbin's book for details.