What is information?: We all have an intuitive idea about what information is. If we ask a kid what their favorite food is and they say 'chocolate' or 'ice cream', it's not very informative. We expected an answer like that. If they say 'broccoli and cheese' we would probably remember that weird kid. Information is a degree of surprise. The more surprising the **message**, the informative the message. The mathematical definition of information assumes there is an information **source** that emits messages. The information of any particular message is given in equation 1. Information is almost always calculated in base 2 and therefore given the unit **bits** (binary digits). Sometimes you may see **nats**, which corresponds to the log base e. For our purposes, we will usually use base 2 for log, and bits will be the units. As an example, let's say that the answer to an average kid's favorite food is 'chocolate' 50% of the time. The information of 'chocolate' is simply the -log(0.5) or 1.0 bit. Similarly, if the probability of 'ice cream' is 0.25, this is 2.0 bits. An easy way to think about this is as powers of two: 0.5 is 2^{-1} and 0.25 is 2^{-2}. A message occurring ~1/1000 times has 10 bits of information (2^{10} = 1024).

Equation 1

$$I(m) = -\log_2 P(m)$$

I: information
m: message
P(m): probability of message

Most programming languages have a log function that returns values in log base e. To convert to base 2, simply divide by log(2).

What does this have to do with genomics?: In order to determine if two sequences are related to each other, we need (a) some way to align sequences (b) some way to determine if the alignments are significant. It is common to use information theory to determine the significance.

Information content: Generally, we are more interested in the information content of a source rather than the information of any particular message. A rich source of information provides you with a lot of surprise *on average*. Information content is simply the average information per message (equation 2). Information content is also called **entropy** or Shannon's Entropy because it was invented by Claude Shannon. An information source can be thought of as a frequency distribution (histogram). Some histograms are more *predictable* than others. For example, consider two coins, one is fair, the other a trick coin. The fair coin comes up heads 50% of the time. The trick coin is heads 90% of the time. Which one is more predictable? As an information source, which one has higher information content? What about a fair vs loaded die? What about DNA? What about DNA with highly biased composition? Try working some examples.

Equation 2

$$H = -\sum P_i \log_2 P_i$$

H: information content
Pi: probability of message i

Relative entropy: Let's say you have nucleotide frequencies from several different genomes and you want to know which ones are the most similar to each other. How might you compare them? If you consider the sequences to be information sources and nucleotides to be messages, then you can use relative entropy to measure the similarity. This is also called **Kullback-Leibler distance** (equation 3). Strictly speaking, D(P||Q) is not always D(Q||P) but it's usually close. Note that if P or Q contains any zero probability values, you may get numerical errors (divide by zero or log zero).

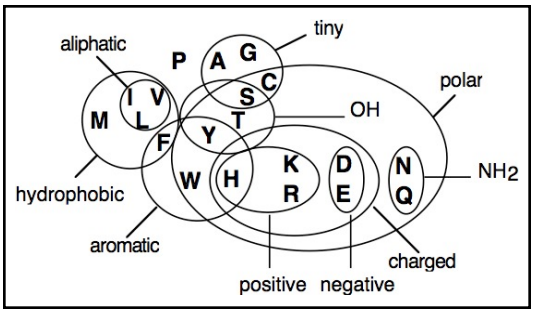
Equation 3

$$D(P \,\|\, Q) = \sum P_i \log_2 \left(\frac{P_i}{Q_i} \right)$$

D: distance
P: some frequency distribution
Q: other frequency distribution
Pi: probability of message i in P
Qi: probability of message i in Q

Codon bias: In the genetic code, some triplets code for the same amino acid, but not all codons are used with the same frequency. In different genomes the biased codons may not be the same. Codon bias probably exists because of translational efficiency. Not all tRNAs are expressed at the same level. As a result, highly expressed genes are optimized to use abundant tRNAs so they don't have to "wait" for rare tRNAs. Given that the codon bias of an organism is a kind of signature, if you found a gene with very different codon usage, you might expect

horizontal gene transfer. You could use K-L distance to find outliers in a genome. In such an experiment, the source is the codon usage, and each message is a triplet.



Amino acid similarity: One of the most fundamental concepts in biology is that similarity is an indicator of common evolutionary history. Many phylogenetic trees are based on proteins. Before we can compare proteins, we need to be able to compare amino acids. How similar are any two amino acids? One could look at size, shape, charge, hydrophobicity, functional groups, etc. You can imagine a lot of different ways. The image at the right summarizes some of these ideas. Another way to determine similarity is by asking *how often can one amino acid substitute for another in a protein?* How would you design such an experiment? Replace an amino acid with another and perform some kind of assay for protein function? That would be great but it's a lot of work. Luckily, these experiments have already been performed billions and billions of times... in nature (evolution).

Margaret Dayhoff: The 'mother of bioinformatics' aligned orthologous proteins by hand and published the multiple alignments in the Atlas of Protein Sequence and Structure. She and her colleagues produced multiple volumes of the atlas. Using known phylogenetic relationships, she was able to observe the rate at which one amino acid changes to another, which is called the **substitution frequency**. These changes are not symmetrical. That is, changing from G -> V ≠ V -> G. This is the truth, but we generally ignore this and average them. Today, there are too many proteins for a print publication. Databases like SwissProt, GenBank, and TrEMBL take place of the Atlas. Similarly, aligning all the sequences by hand would not be possible. There are several computer programs for creating multiple alignments (ClustalW, Dialign, T-Coffee).

```
HM40_CAEEL/188-247      RRKRRNFSKTSTEILNEYFLANIN..HPYPSEEVKQALAMQC.....NISVAQVSNWFGNKRIRYKK
PBX1_HUMAN/234-293      RRKRRNFNKQATEILNEYFYSHLS..NPYPSEEAKEELAKKC.....GITVSQVSNWFGNKRIRYKK
CUT_DROME/1746-1802     KKQRVLFSEEQKEALRLAFA...L..DPYPNVGTIEFLANEL.....GLATRTITNWFHNHRMRLKQ
CUTL2_MOUSE/1114-1170   KKPRVVLAPAEKEALRKAYQ...L..EPYPSQQTIELLSFQL.....NLKTNTVINWFHNYRSRMRR
CUTL1_MOUSE/1240-1296   KKPRVVLAPEEKEALKRAYQ...Q..KPYPSPKTIEELATQL.....NLKTSTVINWFHNYRSRIRR
Q22810_CAEEL/212-268    KKTKSPFTEHEIAVMMALFE...I..NKSPNHEEVQKLAVQL.....NLGYRSVANFFMNKRAKERK
Q9FNM1_ARATH/51-113     PKPEWKPNQHQAQILEELFI...G.GTVNPSLTSIKQITIKLQSYGEEVDDADVYKWFHNRKYSRKP
WOX9_ARATH/52-113       PKPRWNPKPEQIRILEAIFN...S.GMVNPPREEIRRIRAQLQE.YGQVGDANVFYWFQNRKSRSKH
WUS_PETHY/44-106        NSTRWTPTTDQIRILKDLYY...NNGVRSPTAEQIQRISAKLRQ.YGKIEGKNVFYWFQNHKARERQ
WOX6_ARATH/58-119       ATLRWNPTPEQITTLEELYR...S.GTRTPTTEQIQQIASKLRK.YGRIEGKNVFYWFQNHKARERL
WOX1_ARATH/73-134       VSSRWNPTPDQLRVLEELYR...Q.GTRTPSADHIQQITAQLRR.YGKIEGKNVFYWFQNHKARERQ
WOX2_ARATH/11-72        SSSRWNPTKDQITLLENLYK...E.GIRTPSADQIQQITGRLRA.YGHIEGKNVFYWFQNHKARQRQ
Q8LR86_ORYSA/41-102     ANARWTPTKEQIAVLEGLYR...Q.GLRTPTAEQIQQITARLRE.HGHIEGKNVFYWFQNHKARQRQ
Q9LIX7_ORYSA/24-85      STTRWCPTPEQLMMLEEMYR...G.GLRTPNAAQIQQITAHLST.YGRIEGKNVFYWFQNHKARDRQ
WOX4_ARATH/87-148       GGTRWNPTQEQIGILEMLYK...G.GMRTPNAQQIEHITLQLGK.YGKIEGKNVFYWFQNHKARERQ
WOX5_ARATH/21-82        KCGRWNPTVEQLKILTDLFR...A.GLRTPTTDQIQKISTELSF.YGKIESKNVFYWFQNHKARERQ
WOX5_ORYSA/11-72        KCGRWNPTAEQVKVLTELFR...A.GLRTPSTEQIQRISTHLSA.FGKVESKNVFYWFQNHKARERH
```

The **score for pairing amino acids** is shown in Equation 4. The score, *S*, for any two amino acids *i* and *j* is the log of the observed substitution frequency (Q_{ij}) divided by the expected substitution frequency. The observed frequency comes from counting occurrences in multiple alignments. The expected frequency is simply the chance that any two amino acids would be selected at random, so this is the product of the probabilities of the individual amino acid frequencies P_i and P_j.

Equation 4

$$S_{ij} = \log\left(\frac{Q_{ij}}{P_i P_j}\right)$$

Amino acid score examples

Given: $P_M = 0.02$, $P_L = 0.1$, $P_E = 0.04$

$Q_{ML} = 0.004$, $Q_{ME} = 0.001$, $Q_{LE} = 0.002$

Calculate:

S_{ML}, S_{LE}, S_{ME}

S_{ML} = log(0.004 / (0.02)(0.1)) = 1.0 bit

S_{LE} = log(0.002 / (0.04)(0.1)) = -1.0 bit

S_{ME} = log(0.001 / (0.02)(0.04)) = 0.32 bits

Scoring matrices: A **scoring matrix** (at right) is simply a table of all pairwise scores. The matrix produced by Dayhoff is called the PAM matrix (a rearrangement of acceptable point mutations). If you look at the scores in a matrix, you will note that they are all integers. What happened to values like 0.32 bits? They were scaled and rounded off. For example, one might scale 0.32 by a factor of 2 and then round off 0.64 to +1. Why? Historically, computers were slow and had little memory, so people used integers. There is no reason to do this now (floating point calculations are actually faster than integer today), but the practice of using integers for scoring matrices continues. Once the scores in a matrix are scaled and rounded off, the units are no longer bits.

BLOSUM62 Scoring Matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1

Expected score: An important property of a matrix is its expected score (equation 5). To calculate this, one sums up the score contribution of each pairing (the contribution depends on the score and the expected frequencies of the individual amino acids). In general, the expected score of a matrix is negative.

Equation 5

$$Exp = \sum_i \sum_j P_i P_j S_{ij}$$

Relative entropy: The most important property of a scoring matrix is its relative entropy (equation 6). This is the bits per *aligned* pair of amino acids. To gain some intuition for this, imagine if the observed pairing (Q_{ij}) is equal to expected (P_iP_j). In this case, H = 0. That is, the scoring system reflects the random expectation. This is not so different from K-L distance if you compare to identical histograms. The distance is zero. H is maximum when what is observed is very different from what is expected. When does this happen? Continuing from the previous example where $P_M = 0.04$ and $P_L = 0.1$, the expectation is 0.004. If M is rarely observed to align with L, then Q_{ML} will be different from P_MP_L. If you create a scoring matrix from proteins that are all very similar to each other, there will be few substitutions, and Q_{ij} will be very different from P_iP_j. In biological terms, a scoring matrix from highly conserved orthologous proteins will result in a matrix with high H whereas a matrix derived from less similar proteins will have low H. If the alignments are random sequences with no real relationship, H will be zero. Choosing the correct matrix is important. If you are looking for distant similarities, you will not find them with a matrix with high H. On the other hand, if you are looking for very short sequences, H must be high to be significant (more on this later).

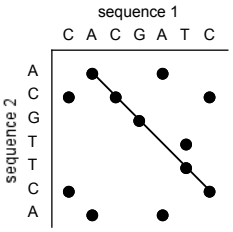
Equation 6

$$H = \sum_i \sum_j Q_{ij} \log\left(\frac{Q_{ij}}{P_i P_j}\right)$$

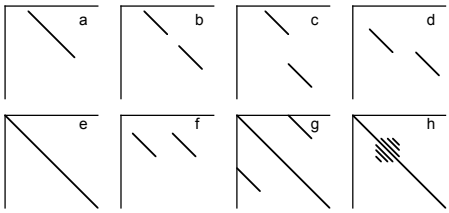
BLOSUM matrices: Henikoff & Henikoff created their scoring matrices automatically. They did not restrict themselves to proteins with known phylogenetic relationships. To calculate the various Q_{ij} values, they assumed all pairings were possible. For any column in a multiple alignment, the counts of different amino acids is N_iN_j and the counts for the same amino acid is N choose 2. N! / 2! (N - 2)!

Imperfect spelling: Have you ever wondered how a spelling checker works? How does it know if the word is misspelled, and how does it suggest correct spellings? This area of computer science is called inexact (approximate, fuzzy) **string matching**. The bioinformatics equivalent is called sequence alignment. In bioinformatics, we often treat nucleotides and proteins as strings

of letters. Even though we know that biological sequences are 3-dimensional entities with physical and chemical properties, it's much easier to process them as 1-dimensional strings. Sequence alignment is used for a variety of bioinformatics tasks. Sometimes we take have it for granted that we can assemble a genome and identify repeats. There are also many other tasks that require sequence comparisons such as determining the function of newly discovered proteins, gene finding, constructing phylogenetic trees, and designing oligos. So how do we *know* if two sequences are similar? There are two fundamental concepts: (1) creating alignments (2) determining if alignments are significant. We will first discuss how to create alignments and then consider their significance.



Dot plots: A simple way to look at the relationship between two sequences is a dot plot (or dot matrix). This is a 2D matrix with a sequence along each axis. Each point in the matrix corresponds to a specific letter in each sequence. Regions of similarity appear as diagonals in the matrix. Rather than draw dots, it easier to draw lines showing just the similar regions. In the 8 graphs shown: (a) an alignment showing a regional similarity between two sequences (b) a section in the middle does not align as well (c, d) the similar regions are separated by a gap (e) a sequence aligned to itself (f) sequence 1 has a duplication (g) a sequence with a repeat aligned to itself (h) a sequence with an SSR aligned to itself. Note that in (c, d) there is either an insertion in one sequence or a deletion in the other. Gaps are therefore often called **indels**.



Pairwise alignment: There are two "flavors" of pairwise alignment: **global** and **local**. In global alignment, the goal is to align every letter of the two sequences. Consider aligning the letters in these two sequences: (1) ACTTTGA (2) TTT. One possible alignment between these is shown as "align 1". Every letter in each sequence is either aligned to another letter or a gap (-) symbol. If the sequences are identical, it is typical to use a | character between the sequences to indicate this. Another common convention is to use the letter. Whenever sequences have unequal lengths, there will be gaps. The gaps can occur anywhere. For example, an alignment between (1) and another sequence (3) ACTGA is shown as "align 2". In local alignment, only the best region is kept. "Align 3" shows two possible local alignments of sequences (1) and (3). Both are equally good. There are also a large number of really poor alignments one could make.

Align 1
```
ACTTTGA
  |||
--TTT--
```

Align 2
```
ACTTTGA
|| | ||
AC-T-GA
```

Align 3
```
ACT     TGA
||| or |||
ACT     TGA
```

Alignment scoring: In order to compare alignments to each other, we can give them a **score**. A simple scoring scheme is to give every matching letter a score of +1 and every mismatch or gap a score of -1. Under such a scheme, the scores for alignments 1-3 are: -1, 3, and 3.

Needleman-Wunsch algorithm: To find the best global alignment one uses the N-W algorithm (or some variant of it). The number of possible alignments between two sequences is huge. You can put gaps in either sequence anywhere you like (but not across from each other). A naive alignment algorithm would enumerate all possible gaps and then choose the alignment with the best score. Even with short sequences this quickly becomes unwieldy and in biological sequences, which can be huge in the case of chromosomes, the number of alignments becomes astronomical. N-W uses dynamic programming (DP) to efficiently find a single highest scoring alignment. There may be more than one alignment with the maximum score, but the algorithm usually only returns one of these. To begin the N-W algorithm, the sequences are entered into a matrix (like a dot plot) with an extra 1st column and row. There are 3 steps to the algorithm: (1) initialization (2) fill (3) trace back. In the initialization, the first row and column are set to gap scores. In the fill, a recursive operation is used to update the maximum score of every

cell. In the trace back, the alignment is recovered by following the maximum alignment from the bottom right of the matrix through the top left.

		A	A	C	G	A	T	G
	0	-1	-2	-3	-4	-5	-6	-7
A	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	2	1	0	-1	-2	-3
G	-3	-1	1	1	2	1	0	-1
A	-4	-2	0	0	1	3	2	1
T	-5	-3	-1	-1	0	2	4	3

Let's take a close look at the fill. In order to fill a cell, you must have 3 neighboring cells located above, to the left, and diagonally above and left. At the beginning, there is only one cell that can be filled. This is the one that aligns the first A and A in the example. To fill this cell, you must determine the maximum score of 3 possible directions (diagonal, up, and left).

Diagonal score = score of diagonal cell + match or mismatch score (either +1 or -1)
Left score = score of left cell + gap score (-1)
Up score = score of up cell + gap score (-1)

When you move horizontally or vertically, you do not consider wether the nucleotides match or not because this operation introduces a gap character. The power of DP is that we do this same operation of looking at 3 possible alignments at every position in the matrix. But we are not enumerating all possible alignments, we are always extending the previous maximum alignment.

Smith-Waterman algorithm: To find the maximum scoring local alignment, you can use the exact same procedure as N-W except that (a) any score below 0 is given the score of 0. At the end, the trace back is performed from the highest score in the matrix rather than the last cell of the matrix.

		A	A	C	G	A	T	G
	0	0	0	0	0	0	0	0
A	0	1	1	0	0	1	0	0
A	0	0	2	1	0	1	0	0
G	0	0	1	1	2	1	0	1
A	0	1	1	0	1	3	2	1
T	0	0	0	0	0	2	4	3

Scoring system: Different match, mismatch, and gap scores will result in different alignments. Try the same sequences with a gap score of -2 and you will get a slightly different alignment.

Computational considerations: The N-W and S-W algorithms as described are not used for aligning long sequences. One reason is that the amount of memory to hold the DP matrix becomes excessive. Each cell in the matrix must hold a score and a directional pointer. This might be 5 bytes of RAM per cell. In order to align two BACs of 100 kb each, you would need about 50 GB of RAM (1e5 x 1e5 x 5). What if you wanted to align some genomes? No computer on the planet has enough RAM. Another reason not to use N-W and S-W is that most of the the space in a DP matrix has a low score. Why align everything rather than just the best parts? Sequence alignment is one of the oldest areas of bioinformatics research, but it is still very active. There are a lot of clever programs that perform alignments very quickly without using much memory. At the root of all these programs is some variant of the S-W algorithm.

Aligning Proteins: Previously we aligned sequences using a +1 match -1 mismatch -1 gap scoring scheme. In practice, this match/mismatch scoring scheme is only used for aligning nucleotides. For aligning proteins, we use scoring matrices like BLOSUM62 to take into account that alignment often preserves chemical properties.

Sequence similarity: To determine protein similarity we simply align two proteins and sum up the amino acid scores. In principle, we could determine similarity scores from local or global alignments. In practice, we use local alignment only. One reason for this is that there is no established procedure for determining global alignment significance.

Alignment scores: What does an alignment score mean? Is a score of 30 *good*? Does 30 mean the proteins are homologous or functionally related? What if the scores in the matrix were scaled by 10 vs 5? Is a score of 100 necessarily better than 50?

Significance: In typical frequentist statistics, one accepts or rejects an hypothesis based on some random model. For local alignments, we use the same idea. Given an alignment score, we would like to know *how often such a score would be expected to occur at random.* If the score is easily attained at random, then it is probably not very significant.

Karlin-Altschul statistics: Local alignment statistics were formalized by Karlin & Altschul using information theoretic methods. Given certain assumptions (see box) the K-A equation (equation 7) tells you how often such a score (or higher) is expected at random. For some intuition in this, imagine comparing two books to see if they have similar sentences. If the books are very short, you don't expect may similar sentences. Conversely, if the books were gigantic, you would expect to find many more similar sentences. The product MN is called the search space, and the number of expected alignments varies linearly with the size of the space. Now imagine that you have a threshold score for what you accept as similar sentences. If you ask for a higher score, you will find fewer sentences. The K-A equation shows that this is an inverse exponential relationship. In other words, a small change increase in score can lead to a large reduction in the number of alignments expected at random. The fact that λ is in the exponent indicates that E is also highly dependent on its value. λ is effectively the inverse of the scaling factor used to create the matrix (but not exactly due to rounding). In other words, λ turns the matrix score into a log-odds score. Now we can begin to answer the questions we previously posed. Is a score of 30 good? It depends on the search space. In a large search space, 30 may be expected at random, but it might be highly significant in a small search space. Is a score of 100 better than 50? If the only difference is the scaling factor, then the significance is the same because λ will normalize them to the same bit score.

Equation 7

$$E = kMNe^{-\lambda S}$$

E: number of alignments
k: a constant
M: size of sequence 1
N: size of sequence 2
e: 2.7182818...
λ: scaling factor
S: score of alignment

K-A issues: Let's take a look at the K-A assumptions. #1 and #2 are true of any scoring matrix derived from multiple alignments. But we can also make up an arbitrary scoring scheme such as our original +1/-1 match/mismatch scheme. Is this legal? What would happen if the scheme was +2/0? What about -1/-2? What about +10/-1? When might +1/-1 be illegal? #3 is only a problem when sequences are very short. To deal with this problem, people consider the search space to be smaller in each dimension by log(kMN)/H, which is the length of the expected random alignment. #4 states that letters are independent and identically distributed. In other words, the probability of finding a sequence such as AAA is simply the product of finding A cubed. Does this make sense? Not really considering that genomes and proteins contain a lot of repeats. #5 disallows gaps. But we know S-W alignments can contain gaps. We will return to the gap problem in a bit.

Karlin-Altschul Assumptions
1.A positive score must be possible
2.Expected score of matrix must be negative
3.Sequences are infinitely long
4.Letters are independent and identically distributed
5.Alignments do not contain gaps

Lambda revisited: In order to compute E, we need λ for our scoring scheme. We might know this value ahead of time if we created our own scoring matrix, but if someone else created it, or we used a system like +1/-1, we need to be able to derive λ somehow. λ cannot be solved for algebraically, but we can estimate its value to arbitrary precision.

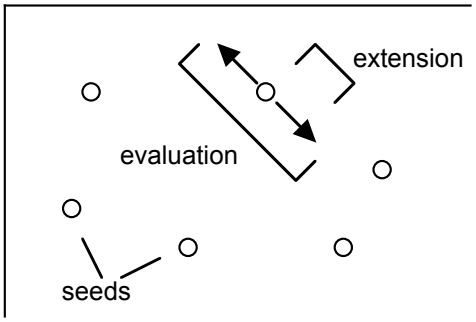
It turns out that our old +1/-1 scoring system implies a pairing frequency of about 75% given that the 4 nucleotides have 25% frequency. If we had started with nucleotide alignments containing

about 75% identity, and the marginal nucleotide frequencies were all 25%, we would have ended up with a +1/-1 scoring system. +1/-1 does not imply 75% identity in proteins however.

$S_{ij} = \log\left(\frac{Q_{ij}}{P_i P_j}\right)$	The usual equation for the score of any amino acid pair.
$\lambda S_{ij} = \log\left(\frac{Q_{ij}}{P_i P_j}\right)$	λ is the inverse of the scaling factor used when the matrix was scaled and rounded off. When scores are in bits, $\lambda = 1$.
$e^{\lambda S_{ij}} = \frac{Q_{ij}}{P_i P_j}$	Exponentiate each side of the equation.
$Q_{ij} = P_i P_j e^{\lambda S_{ij}}$	This is the most important part. It shows that an observed pairing frequency is *implied* given the marginal compositions and a scoring scheme.
$\sum\sum Q_{ij} = 1$	By definition all observed pairing frequencies sum to 1.0
$\sum\sum = P_i P_j e^{\lambda S_{ij}}$	We can solve for λ by making refined guesses at its value. If our guess is too high, the sum will be > 1. If it is too low, the sum will be < 1.

Gaps revisited: So what do we do about gaps? Gaps make it *easier* to align two sequences. Therefore, gaps effectively reduce H. To account for this in the K-A equation, we can simply decrease λ, and this will decrease the bit score of the alignment and therefore increase the E value. How much we reduce λ depends on the specific match, mismatch, and gap penalties. It is not possible to compute these adjustments algebraically, so they are computed via simulation (e.g. do billions of random alignments with a variety of scoring systems to compare how gapped and ungapped alignments are related).

BLAST: One of the most famous and popular bioinformatics applications is BLAST (Basic Local Alignment Search Tool). This combines sequence alignment and statistical evaluation in a single, efficient program. BLAST is similar to S-W in principle: both are local alignment algorithms. But BLAST is much faster because it does not explore the entire search space. There are 3 steps to the BLAST algorithm: (1) seeding (2) extension (3) evaluation. In the seeding phase, regions containing identical (or highly similar) strings are identified. These *points* in the *space* are expected to contain the good local alignments. In the extension phase, each seed undergoes a S-W-like alignment, but the extension stops if the alignment quality degrades too much. In the evaluation phase, the alignment is subjected to the K-A equation to determine how often the alignment is expected by chance. If the E value is less than some user-defined threshold, then the alignment is reported.



Program	Database	Query	Example
BLASTN	DNA	DNA	Align mRNA to genome
BLASTP	AA	AA	Search for proteins related to ____
BLASTX	AA	DNA	Find coding exons in a BAC
TBLASTN	DNA	AA	Search for transcripts similar to ____
TBLASTX	DNA	DNA	Find orthologous coding exons

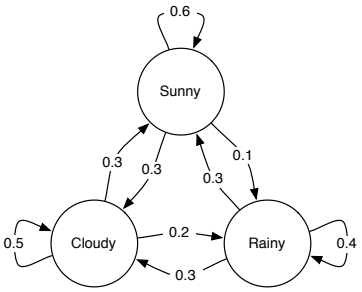
From extrinsic to intrinsic models: Sequence alignment assumes all letters are independent of one another. This is useful if you are looking for conservation in general, but what if you are interested in a particular type of sequence, such as a promoter, or a protein domain? For this kind of question, we can need more specific models.

Composition: A simple yet useful attribute of a genome is it's base composition. For example, A. thaliana is AT-rich and D. radiodurans is GC-rich. The composition is generally not uniform, however. For example, A. thaliana exons are more GC-rich than introns. When we describe a sequence by its composition, we are implicitly creating a sequence model where every nucleotide is independent of every other nucleotide. We would therefore conclude that the probability of seeing AAAAAA is simply the probability of A to the 6th power. However, this is not the case in real sequences. Low-complexity sequences occur much more frequently than expected by chance. While composition is a useful attribute, it ignores the *context* of each nucleotide.

Base	A. thaliana			D. radiodurans
	Genome	Exon	Intron	Genome
A	32.00	29.85	26.73	16.54
C	18.02	20.14	15.46	33.51
G	18.01	20.16	17.16	33.45
T	31.97	29.84	40.64	16.49

Context: The context of a letter is defined by the letters closest to it. Before we consider the context of biological sequences, let's think about a more familiar case: language. In English, Q is almost always preceded by a vowel (or nothing) and followed by U. A simple compositional description of English would assume that some words contain QQ, but we know this to be false. Context matters. Context is also important in biological sequences. For example, in a trans-membrane domain, all of the amino acids are hydrophobic

Markov models: In sequence analysis, we consider the context of a letter to be the preceding letter only. We know that there are letters on either side of any particular letter, but by considering the context from one side only, it makes our lives much easier because we can treat sequences as the products of Markov models. A Markov model has a fixed number of **states** and **transition probabilities** for moving between states. During each "time step", the model moves randomly from one state to another. Think of a Markov model as a machine that randomly generates a set of states. As an example, let's consider the weather as such a machine. Suppose that each day (or hour, or other time point) can be Sunny, Cloudy, or Rainy. If we do not take context into account, we would expect that any weather can follow any weather. But we know from personal experience that it usually gets cloudy before raining, and sunny or rainy days tend to follow each other. Taking context into account means that we want the weather tomorrow to depend on the weather today. We can draft these concepts quite simply in a Markov model. Look at the example figure. Imagine starting in a state and "rolling dice" to change from one state to the next to generate weather patterns. Markov models often have beginning and ending states. This might not make sense in terms of the weather, but imagine music being generated from a Markov model. The song should end.



Sequences as Markov models: Now let's get back to biology. The A. thaliana genome is approximately 32% A. Under an independent model, we would expect that seeing AA is simply (0.32)(0.32) = ~10%. What we actually observe is 36%. Very different. To make a Markov model of the A. thaliana genome, we can simply build a table showing the conditional probabilities for each letter. Now if we "roll dice", the resulting sequence will look a little more like the A. thaliana genome.

	Preceding Symbol			
Symbol	A	C	G	T
A	36.18	35.24	35.58	23.98
C	16.36	18.81	16.65	20.02
G	18.57	12.99	18.74	19.86
T	28.89	32.97	29.03	36.14

Previously we noted that introns and exons do not have the same compositions. As Markov models they are even more different. Context is very important, and this becomes more

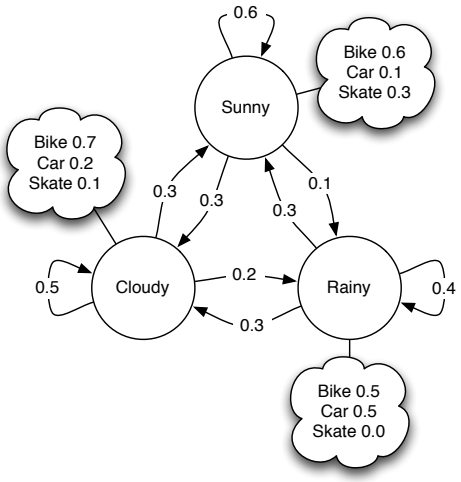
apparent with greater context. The **order** of a Markov model is the number of letters of context. Simple base composition is 0th order. The weather model was 1st order. We can take even more letters into context if we like. For example, in a 2nd order model, the probability of the next state depends on the previous 2 states.

Training: Assigning the various probabilities is called **training**. Generally, we train the model by making observations. For example, we can count up how often we see a T given that the previous base is a C. Consider what would happen if we tried to train a 15th order model. Each 15-mer of context occurs on average approximate 4^-15 times, which is about 1 in 1 billion. For a genome the size of A. thaliana, each context is seen less than one time. As a result, most of the observations are zeroes, which leads to a useless model.

Intron Mediated Enhancement (IME): Now let's look at a biological example of where these kinds of analyses can be useful. Most people consider introns to be "junk DNA". It turns out they do sometimes serve a useful role. This is most often seen when people make trans-genes and find that they do not express very well. Placing an intron inside the coding sequence often improves expression. In many organisms, people put introns into their constructs out of habit, but nobody really knows why they help. Recently, the Rose and Korf research groups (of UC Davis) made a large advancement in our understanding of IME using Markov modls.
http://www.plantcell.org/cgi/content/abstract/tpc.107.057190v1

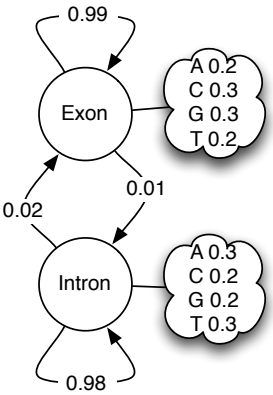
Hidden Markov models: In an HMM, the Markov model is hidden behind emissions.

Emissions: A Markov model has states and transitions. In a hidden Markov model (HMM) we add one more feature: emissions. An HMM is similar to a Markov model except that each stay in a state emits a symbol. In an HMM, we cannot observe the state directly. Instead, we observe something that happens in that state (the symbol).



Weather machine revisited: What if you were interested in the weather, but were not able to actually observe it. Perhaps all you could observe is how I arrive to work: bike, car, or skate. I generally like to bike, but on rainy days I might take a car, and on sunny days I might skate. By making a lot of observations, you could create an HMM as drawn.

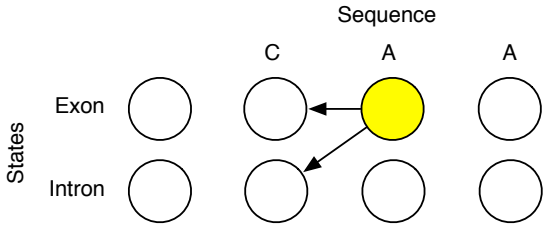
HMMs are generative models, so think of them as machines that generate sequences. In this case, the sequence generated will be Bike, Car, or Skate. The weather HMM might generate a sequence such as BBBCBBSBSSBCBB. The point of an HMM is not to generate sequences, but to make inferences about the underlying Markov model given some observations. To give a little intuition in this, consider what you would conclude if you saw SSSSS for the week. Not rainy, for sure, and probably more likely to be sunny than cloudy. But what is the most likely sequence of states, and how likely is any given prediction? We will see that in just a bit.



Toy gene finder: Let's consider a biological example. The sequences we observe in biology are DNA, RNA, and protein. What we want to know is what kind of functional category they belong to. For gene finding, we might want to know where the exons are. We can build a gene finder by making an HMM that emits gene-like sequences. Let's begin with just two states, exon and intron. Let's say that on average exons are 100 bp

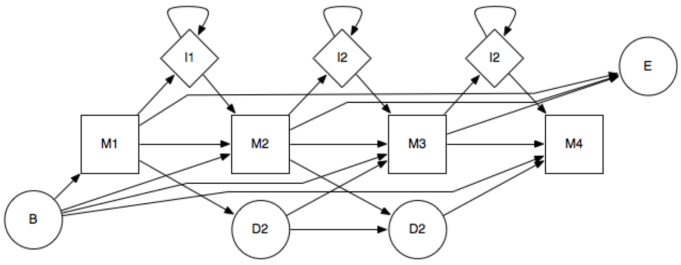
long and 60% GC. Introns are 50 bp on average and 40% GC. Given these parameters, we can build a gene HMM as shown.

Decoding: The point of HMMs is not to generate sequences, but to decode them. That is, given a sequence such as CAATATATAGCAGTGGACCCGCATATATAAAATA, the underlined portion is obviously more GC-rich than the ends and was therefore probably emitted by an exon state. To determine the most likely path, we use the **Viterbi** algorithm. It is very similar to the N-W and S-W algorithms for sequence alignment. It uses dynamic programming to find the most likely state path given the observed sequence. Like S-W and N-W, the point is to extend the optimal path by one cell. The best path at the shaded cell is found by finding the maximum probability from all states leading to the shaded cell. There are 3 components: (1) the probability in the previous cell (2) the transition probability from a preceding cell to the shaded cell (3) the emission probability in the shaded cell. Just like in alignment, we find the maximum value, record this in the cell, and record a trace back pointer to the previous cell. The DP matrix is initialized with probabilities for Begin and End states (often not shown) and the trace back begins from the cell with the greatest probability in the last column. By tracing back through the DP matrix, one finds the maximum state path for any observed sequence. There are other decoding algorithms, such as the forward-backward algorithm, which tells you the likelihood of each state at each position.



Prosite patterns and PWMs: HMMs are generalizations of Prosite patterns and PWMs. For example, the Protein Kinase C pattern [ST]-X-[RK] can be written as a 3 state HMM with emission probabilities 50% S or T, anything, 50% R or K. To introduce variable length regions, we simply put in some extra states or self-loops. A PWM is simply an HMM where each position is a state that emits A, C, G, or T, and transition probabilities are all 100%.

Profile HMMs: One of the most common and useful applications for HMMs is to describe protein domains and families. Here, a generic structure is used for all proteins. So rather than devise an HMM for each protein, one only needs to change the the transitions and emission probabilities. In the diagram, the M states correspond to columns of a multiple alignment. The emission probabilities here reflect the column. The I and D states are for insertion and deletions. The insertions generally follow the average amino acid composition. A more complete model also includes N- and C-terminal extensions. There are two common software packages for protein profile HMMs: HMMER and SAM.

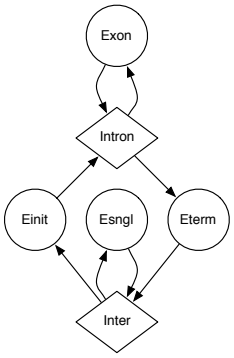


Pfam: The Pfam database contains a large collection of multiple alignments and profile HMMs. This is one of the most important bioinformatics databases.

Generalized HMMs: In a traditional HMM, each state emits a single symbol and then transitions to another state. One consequence of this is that the length of time one stays in any particular state is geometrically distributed. For example, given a 50% chance to return to the same state, the chance of staying in that state for 1 time period (0.5) is greater than 2 time periods (0.5 x 0.5). From a modeling perspective, this means that even if we create a model to have an average stay of 100 bp (i.e. a 99% chance of returning to the same state), the maximum probability is always at 1 bp. The lengths of exons, introns, and other biological sequence features often have some minimum and value and a peaky shape. To model this property we use generalized HMMs (GHMMs). In a GHMM, each state emits a randomly generated

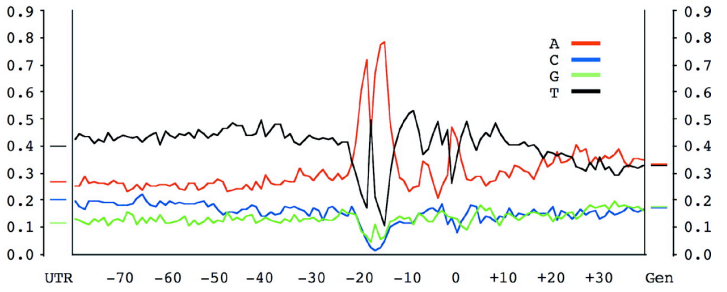
sequence whose length follows any distribution. For this reason, GHMMs are also called explicit duration HMMs.

GHMM for eukaryotic genes: At the right is a simple GHMM for eukaryotic genes. The Einit state is initial exons which contain an ATG and a splice donor site. The Eterm contains a splice acceptor and a stop codon. For multi-exon genes, there are intron and exon states. Each exon has a splice acceptor and donor site. The Esngl state corresponds to genes without introns. The Inter state is for intergenic sequence between genes. We could add more states to the model, such as 5'UTR, 3'UTR, promoter, poly-A site etc. If you look at GHMMs described in the scientific literature (try this one), you might notice that there are many more intron and exon states. A splice site may interrupt a codon in any of 3 positions. To prevent frame shifts and fused stop codons, it is necessary to add extra states exon and intron states that capture this information.



A GHMM for 3' end formation: The most common place to find GHMMs is gene structure prediction, but they can be used to model other features. One example is 3' end formation. Recall that the AATAAA motif lies ~15 bp upstream of the poly-A tail. Rather than using just the AATAAA motif to predict poly-A sites, one can use all the information available including the cleavage site (found by aligning mRNAs to a genome) and regions surrounding the cleavage site and AATAAA motif.

a



b