ICQB

Introduction to Computational & Quantitative Biology (G4120)
Spring 2017

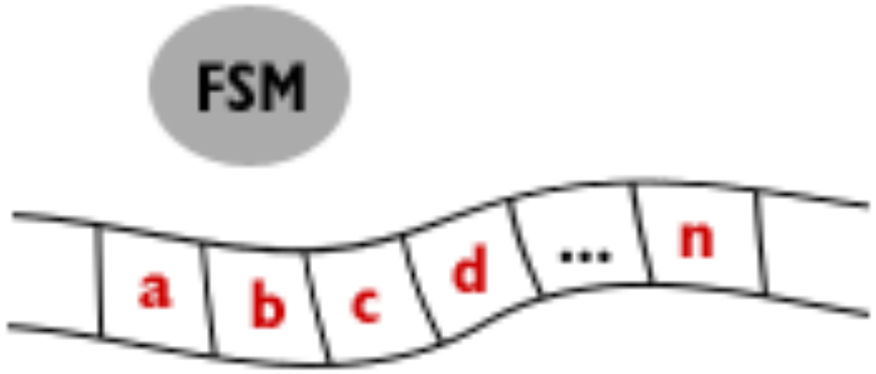
Oliver Jovanovic, Ph.D.
Columbia University
Department of Microbiology & Immunology

Lecture 2: Introduction to Computing
February 6, 2017

Binary Computing and DNA

Modern computers are digital machines, which means their basic function involves using discrete symbols from a finite set.

In 1936, Alan Turing proved that a finite state machine (FSM) moving up or down a tape of symbols, reading or writing one symbol at a time, could solve any computable problem, and serve as a universal machine.



Universal Turing Machine

The most basic level of information in nearly all current computers represents only one of two possibilities: **0 (*off*) or 1 (*on*).** A signal that can carry one of two possible messages (**0** or **1**) is called a binary signal, or a ***bit,*** so these computers are binary machines.

The Digital Language of Computers

Binary Units

0 or 1 = 1 bit
8 bits = 1 byte
1,024 bits = 1 kilobit
1,024 bytes = 1 kilobyte (K)
1,024 kilobytes = 1 megabyte (M)
1,024 megabytes = 1 gigabyte (G)
1,024 gigabytes = 1 terabyte (T)

1 bit =	0 or 1	= 2 possibilities
2 bits =	0 or 1 0 or 1	= 2 x 2 = 4 possibilities
3 bits =	0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 = 8 possibilities
4 bits =	0 or 1 0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 x 2 = 16 possibilities
5 bits =	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 x 2 x 2 = 32 possibilities
6 bits =	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 x 2 x 2 x 2 = 64 possibilities
7 bits =	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 x 2 x 2 x 2 x 2 = 128 possibilities
8 bits =	0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1 0 or 1	= 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 256 possibilities

DNA has only four possibilities *(so can be represented by 2 bits)*

G = 00
C = 11
A = 01
T = 10

Complementation *(with intelligent choice of representation)*

G C C A = 00 11 11 01
C G G T = 11 00 00 10

ASCII Coding of DNA

American Standard Code for Information Interchange (ASCII)

- For practical purposes, DNA and RNA is generally represented in ASCII code, using the upper or lower case letters A, C, G, and T or A, C, G and U.
- Each ASCII character occupies one byte, and thus has 256 possibilities, including all upper and lower case letters of the English alphabet, the ten Arabic numerals, punctuation, and special characters, such as @.
- Thus, a kilobase of DNA (1,000 base pairs) occupies just under a kilobyte (1 K = 1,024 bytes) of storage in ASCII. An entire human genome, roughly 3 billion base pairs (3 gigabases), occupies just under 3 gigabytes of storage in ASCII.

Transcription

- Transcription is computationally trivial. One need only substitute a U for a T if dealing with a sense strand, or complement, then transcribe if dealing with the antisense strand.

Translation

- Translation is also computationally trivial. A computer can refer to a species appropriate translation table to translate DNA or RNA into the appropriate protein sequence.

AUA	I	Isoleucine
AUC	I	Isoleucine
AUG	M	Methionine start
AUU	I	Isoleucine
etc.		

Alternate Representation

- Can readily convert an ASCII representation of DNA into other forms, such as graphics, or even music.

Information Content

Uncertainty

Uncertainty can be thought of as the number of yes/no questions required to identify the state something is in. It can be measured in bits.

- A coin toss, with only 2 possibilities, can be identified with a single question (i.e., "Is it heads?")
- A nucleotide, with 4 possibilities, can be identified with two questions (i.e. "Is it a purine? Is it adenine?")

Maximum Uncertainty

Maximum Entropy = $\log_2(n)$ where n is the number of possible states

Coin	$\log_2(2) = 1$ bit
DNA	$\log_2(4) = 2$ bits
Protein	$\log_2(20) = 4.32$ bits

Compression algorithms offer one approach to testing the randomicity of a DNA sequence. A very random DNA sequence will require close to 2 bits per nucleotide to represent it, even when compressed. A sequence of DNA that has repeating patterns, or is otherwise highly structured, should be capable of being represented by less than 2 bits per nucleotide.

Algorithms in Computational Biology

Algorithm

- An algorithm is simply a series of steps used to solve a problem. One of a computer's great strengths is its ability to rapidly and accurately repeat recursive steps in an algorithm.

Consensus

- Early algorithms for searching sequence data depended on consensus sequences. Thus, to find a prokaryotic promoter, one would try to find something that matched a consensus -10 sequence (TATAAT), not too far downstream of a consensus -35 sequence (TTGACA).
- It rapidly became clear that biologically significant sequences rarely perfectly matched a consensus, and more sophisticated approaches were adopted, including the use of matrices, Markov chains and hidden Markov models.

Matrices

- Matrices take into account the distribution of every possible nucleotide (or amino acid) at a position in a set of known sequences. Searching with a matrix is therefore more sensitive than searching with a consensus, and can find biological features that a strict consensus approach would miss.

Markov chains and hidden Markov models (HMMs)

- Markov chains and hidden Markov models are probabilistic models of sequences, and have proven useful in database searching, gene finding and multiple sequence alignment.
- A first-order Markov chain is a finite state automaton (a restricted Turing machine which only moves left to right) with probabilities for each transition to a new state (symbol) based on its current state. Higher order Markov chains take into account one or more previous states.
- A hidden Markov model is a Markov chain in which only the output can be observed (its current state is hidden).

Consensus vs. Matrix

***E. coli* Promoter Consensus**

-35 Region **-10 Region**

TTGACA.....TATAAT

***E. coli* Promoter Matrix**

-35 Region

									T	T	G	A	C	A	
A	11	8	8	7	8	7	3	5	5	0	1	0	14	5	9	5
C	3	4	2	4	4	3	5	2	8	1	1	2	3	11	2	5
G	3	2	4	2	4	5	5	5	5	2	1	17	1	2	3	3
T	4	7	7	8	5	6	8	9	3	17	18	2	4	3	7	9

Spacer Region

Length	9	10	11	12	13	14	15
	1	6	14	6	1	1	1

-10 Region

						T	A	T	A	A	T			
A	4	5	3	4	4	0	20	5	12	11	0	7	4	6
C	5	4	5	4	5	2	0	3	3	4	1	2	7	6
G	2	5	5	8	7	2	0	3	3	3	0	6	5	6
T	10	6	8	5	6	17	1	9	3	4	20	6	5	4

Matrix Analysis Example

Position	Score	Predicted promoter sequence (-35 < gap> -10)	Name
2313	52.94	GTTAATTGCTTTTCGA <10> TTAGCTAAACTTTC	
3075	55.29	CGACATTGCTTGACCC <11> GCGTGTTCAATTCG	*korE*
3772	55.29	GCGTCATGCTTGAAAA <11> TGGCGTGCAATCAG	
5552	51.17	AAAAGGATCTTCACCT <10> AAATTAAAAATGAA	
5585	53.52	AAATGAAGTTTTAAAT <15> ATGAGTAAACTTGG	
5695	52.35	CATAGTTGCCTGACTC <12> ATAACTACGATACG	
6133	54.11	ACTGCATAATTCTCTT <11> ATCCGTAAGATGCT	
6478	56.47	ACGGAAATGTTGAATA <11> CCTTTTTCAATATT	
6511	53.52	TCAATATTATTGAAGC <12> TATTGTCTCATGAG	
6532	52.35	TCAGGGTTATTGTCTC <11> CATATTTGAATGTA	
6618	57.05	AAGTGCCACCTGACGT <10> ATTATTATCATGAC	
7292	59.41	ACGATTTAATGGACAC <11> CGTTTTACTATGTC	*tnpA*
8080	51.17	ATGAAGCAATTGAACG <11> TGAACGATTTTGGC	
8248	50.58	CCTTTGTCCTTGCATG <11> CGTTGGATGATGCA	
8482	58.23	AAAAGCTGGCTGAAAT <11> TCCGTGAAATTGCC	
11698	51.76	ATTATGCCCTAGCCTG <10> TTAGCTAAACTATG	
13937	60.58	TTGTCATGCTTGACAC <12> AAACATAATATGTC	*tetA*
17967	53.52	ACCGCTATATCGAAAA <10> CTTGTTAGAATTGC	
17999	57.64	AGAATTGCCATGACGT <11> ACGGGTAAGATTAC	*trbA*
18729	53.52	ATTAGCTGTTTGTCTT <14> TTCGGTATATCGTT	*trbB*
26625	50.58	ACCAGGCGTTTGACTA <9> AGGAGTAACTTATG	
35054	43.52	CTCGCGCTGTAGCCTC <11> TGTGCTAATGTGGT	*parD*
38324	64.70	ATCGTGGCGTTGACAA <11> CTGGCTACACTATG	*aphA*
40006	54.70	TCGTAGTTCTTGCCGA <11> TTCTCAAAGATGCC	
47326	52.35	ATCAGTTGCTTGATGC <11> TTGCTGACGTTGCG	
48938	54.70	CAAACGGTTTTGGCTT <12> TTTCGTCCAATGCG	
51306	57.64	GAAAAAGGATGGATAT <9> ATCGCTATAATGAC	*traK*
59051	54.70	TGTTTTTTCTTGGCGT <11> TTCCGGACGATGTA	
2375c	66.47	CTAAAGGTGTTGACGA <12> TTAGCTAAACTTCT	*klaA*
3711c	52.94	ATTCTTGTTTTGAGGC <11> CCAGGTCAATTACC	
3745c	67.05	TAAAATTGCTTGACAA <12> TGCCCTATTCTTGT	*kleC*
3777c	52.94	GACGCCTCGCTGAATC <11> TTAGCTAAAATTGC	
4400c	65.88	TAAATTTCCTTGACTA <12> TGCCCTAATATAGC	*kleA*
5562c	53.52	AGATCCTTTTTGATAA <14> TCCCTTAACGTGAG	
5619c	51.76	CATATATACTTTAGAT <12> TCATTTTTAATTTA	
5647c	54.70	CATTGGTAACTGTCAG <11> TCATATATACTTTA	

SeqMatrix *E. coli* promoter output:

```
DNA Location: 3,075
Spacer Length: 11
Similarity Score: 55.29

CGACATTGCTTGACCC <11> GCGTGTTCAATTCG
    (TTGACA................TATAAT)
```

Stochastic Modeling

Stochastic Model

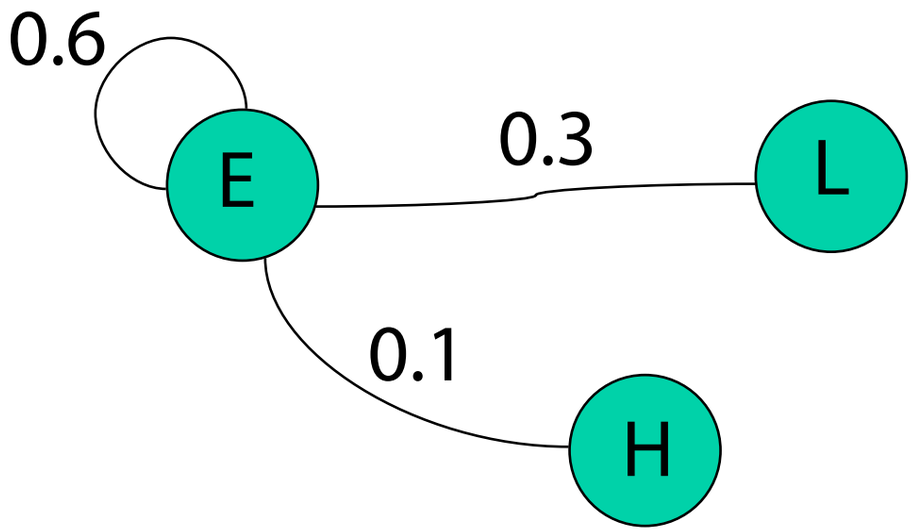
A model involving chance or probability. Markov models are a particular form of stochastic model.

Current Residue	Next Residue A	C	G	T
A	40%	15%	15%	30%
C	25%	25%	25%	25%
G	20%	25%	30%	25%
T	35%	20%	20%	25%

Markov Modeling

Markov State

A Markov state emits a symbol each time you visit it. It connects to other states (and possibly itself), with transition probabilities attached. The sum of the transition probabilities is 1.

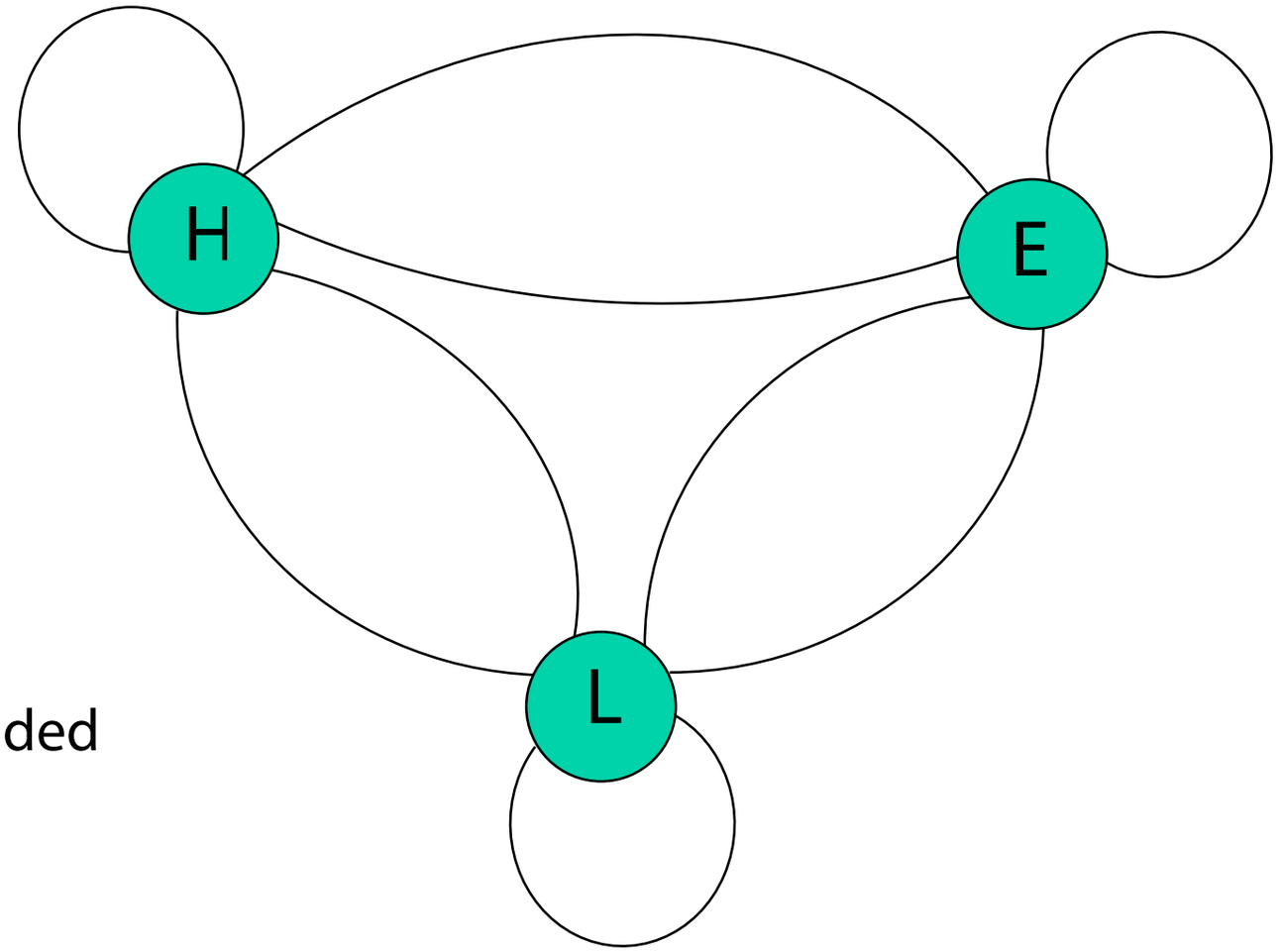


E = Extended
H = Helix
L = Loop

Markov Chains

Markov Chain

A Markov chain is an interlinked chain, or network, of states connected by transition probabilities.

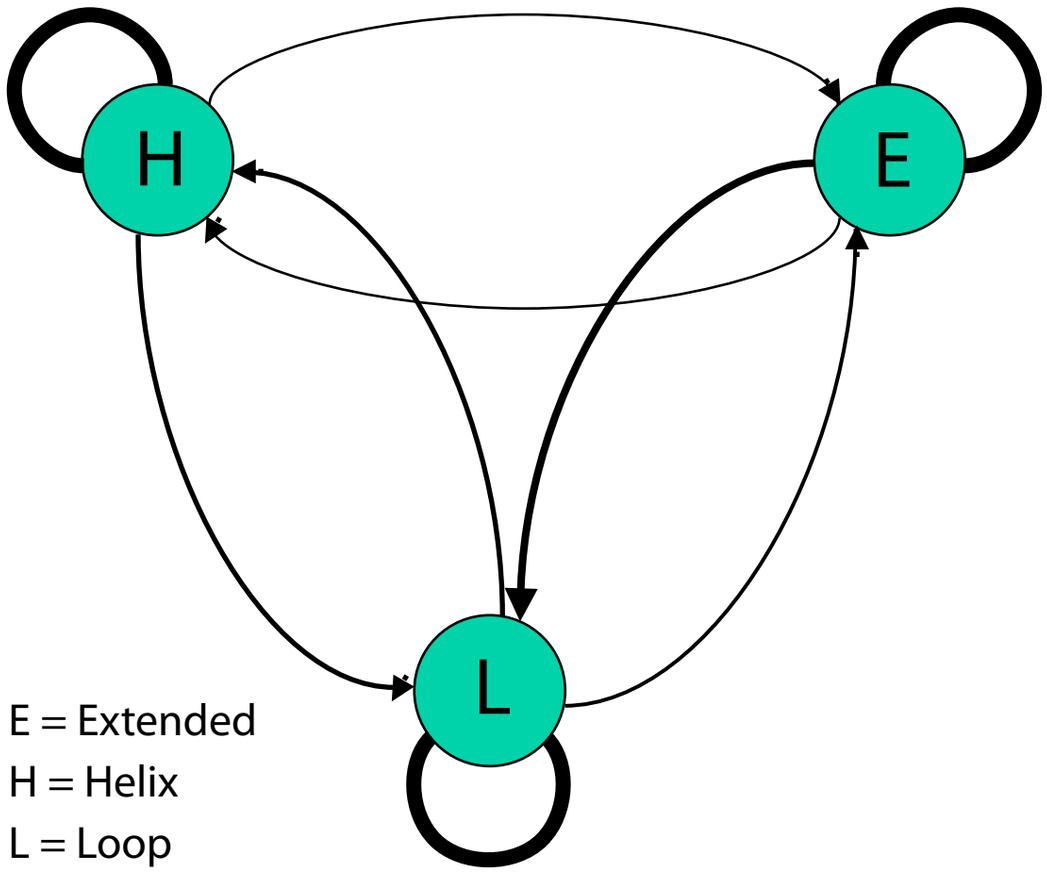


E = Extended
H = Helix
L = Loop

Markov Transition Matrices

Transition Matrix

A transition matrix for a first order Markov chain, the simplest kind. The sum of the transition probabilities from each state is 1.



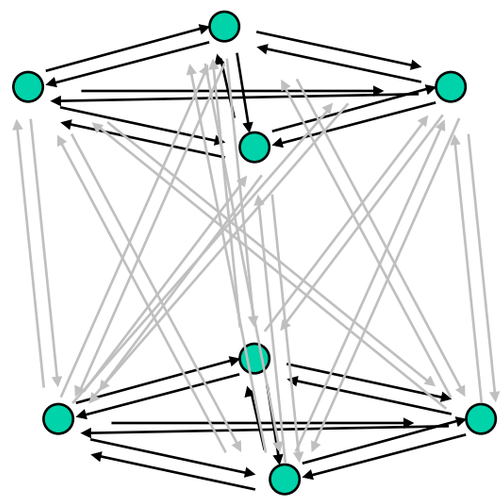
	H	E	L
H	**0.93**	0.01	0.06
E	0.01	**0.80**	**0.19**
L	0.04	0.06	**0.90**

Hidden Markov Models

Hidden Markov Model (HMM)

A hidden Markov model consists of two Markov chains connected such that a one to one correspondence between the state and the emitted symbol no longer exists.

Model 1 →



← **Transitions between models**

Model 2 →

GeneMark

GeneMark and GeneMark.hmm

Mark Borodovsky, Georgia Institute of Technology
http://exon.gatech.edu/GeneMark/

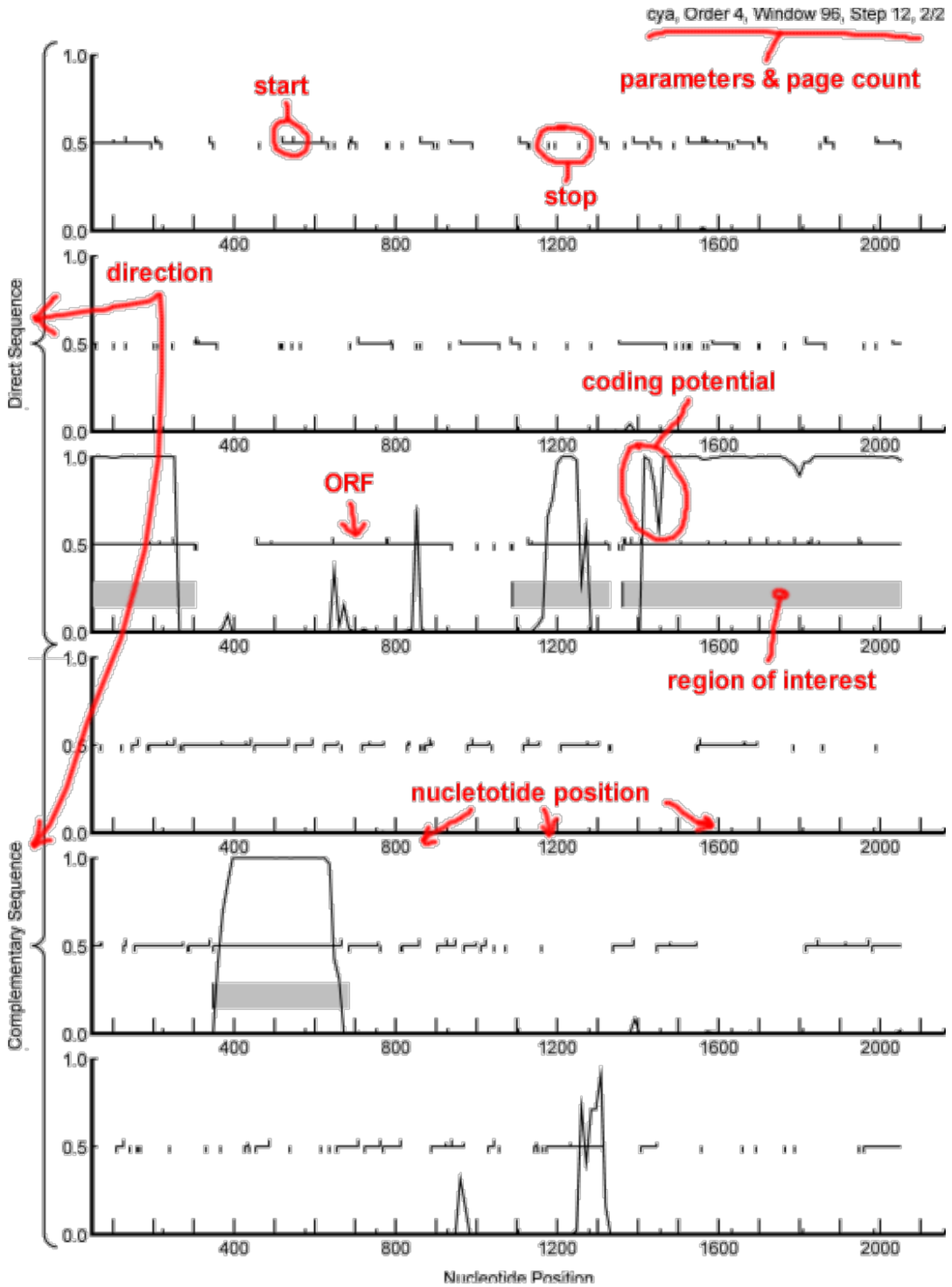
GeneMark

GeneMark evaluates the protein-coding potential of a DNA sequence (within a sliding window) by using Markov models of coding and non-coding regions for various prokaryotic species. This approach is sensitive to local variations of coding potential, and the GeneMark graph shows details of the coding potential distribution along a sequence. It has been used since 1995 to provide automatic gene annotation for the *H. influenza, M. jannaschii, B. subtilis* and *E. coli* genomes.

GeneMark.hmm

GeneMark.hmm predicts genes and intergenic regions in a sequence as a whole using hidden Markov models with a hidden state network reflecting the "grammar" of gene organization. It identifies the most likely parse of the whole sequence into protein coding genes (with possible introns) and intergenic regions. It is currently used as a microbial genome annotation tool by the NCBI.

GeneMark Example



Source
http://bioweb.pasteur.fr/docs/genemark/images/cyay.gif

Search Algorithms in Bioinformatics

Global Alignment Search

- **Needleman-Wunsch** algorithm, Needleman & Wunsch, 1970
- Finds the best complete alignment of two sequences that maximizes the number of matches and minimizes the number of gaps.

Local Alignment Search

- **Smith-Waterman** algorithm, Smith & Waterman, 1981
- Makes an optimal alignment of the best segment of similarity between two sequences.
- Often better for comparing sequences of different lengths, or when looking at a particular region of interest.

Heuristic Approximations to Smith-Waterman

- **FASTA**, Pearson, 1988
- **BLAST**, Altschul, 1990
- **Gapped BLAST and PSI-BLAST**, Altschul, 1997
- **BLAT**, Kent, 2002
- **DELTA-BLAST**, Boratyn, 2012

Global Alignment (Needleman-Wunsch)

Gap, from the GCG Wisconsin Package, uses the algorithm of Needleman and Wunsch to find the alignment of two complete sequences that maximizes the number of matches and minimizes the number of gaps.

```
GAP RK2_ssb x Ecoli_ssb       January 29, 2003 00:07


           .         .         .         .         .
  1 ..MSHNQFQFIGNLTRDTEVRHGNSNKPQAIFDIAVNEEWRNDA.GDKQE 47
        |.   :||| .| |||:  .    |   :| .| ||. | |: .|
  1 ASRGVNKVILVGNLGQDPEVRYMPNGGAVANITLATSESWRDKATGEMKE 50
           .         .         .         .         .
 48 RTDFFRIKCFGSQAEAHGKYLGKGSLVFVQGKIRNTKY.EKDGQTVYGTD 96
   .|:. |:  ||  ||   .|| ||| |:::|.:|  |: :. ||  | |:
 51 QTEWHRVVLFGKLAEVASEYLRKGSQVYIEGQLRTRKWTDQSGQDRYTTE 100
           .         .         .         .         .
 97 FIAD...KVDYLDTKAPGGSNQE.......................... 116
     : .    .  |  :  ||.
101 VVVNVGGTMQMLGGRQGGGAPAGGNIGGGQPQGGWGQPQQPQGGNQFSGG 150
           .         .
     ..........................

151 AQSRPQQSAPAAPSNEPPMDFDDDIPF 177

Matrix:  blosum62
Gap Penalties:  default
Length:  177
Percent Similarity:  45.690
Percent Identity:  32.759
```

Local Alignment (Smith-Waterman)

BestFit, from the GCG Wisconsin Package, makes an optimal alignment of the best segment of similarity between two sequences. Optimal alignments are found by inserting gaps to maximize the number of matches using the local homology algorithm of Smith and Waterman.

```
BESTFIT RK2_ssb x Ecoli_ssb       January 29, 2003 00:08


                .         .         .         .         .
 4 NQFQFIGNLTRDTEVRHGNSNKPQAIFDIAVNEEWRNDA.GDKQERTDFF 52
   |.   :||| .| |||:  .    |   :| .| ||. | |: .|.|:.
 6 NKVILVGNLGQDPEVRYMPNGGAVANITLATSESWRDKATGEMKEQTEWH 55
                .         .         .         .
53 RIKCFGSQAEAHGKYLGKGSLVFVQGKIRNTKY.EKDGQTVYGTDFIAD 100
   |:  ||  ||   .|| ||| |:::|.:|  |: :. ||  | |: : .
56 RVVLFGKLAEVASEYLRKGSQVYIEGQLRTRKWTDQSGQDRYTTEVVVN 104


Matrix:  blosum62
Gap Penalties:  default
Length:  99
Percent Similarity:  50.515
Percent Identity:  36.082
```

Global (Needleman-Wunch) vs. Local (Smith-Waterman) Alignment

```
  1 ..MSHNQFQFIGNLTRDTEVRHGNSNKPQAIFDIAVNEEWRNDA.GDKQE  47
        |.   :||| .| |||:  .    |   :| .| ||. | |: .|
  1 ASRGVNKVILVGNLGQDPEVRYMPNGGAVANITLATSESWRDKATGEMKE  50
              .         .         .         .         .
 48 RTDFFRIKCFGSQAEAHGKYLGKGSLVFVQGKIRNTKY.EKDGQTVYGTD  96
    .|:. |:  ||  ||   .|| ||| |:::|.:|  |: :. ||  | |:
 51 QTEWHRVVLFGKLAEVASEYLRKGSQVYIEGQLRTRKWTDQSGQDRYTTE 100
              .         .         .         .         .
 97 FIAD...KVDYLDTKAPGGSNQE........................... 116
     : .    .  |  :  ||.
101 VVVNVGGTMQMLGGRQGGGAPAGGNIGGGQPQGGWGQPQQPQGGNQFSGG 150
              .         .
    ...........................

151 AQSRPQQSAPAAPSNEPPMDFDDDIPF 177
```

Dynamic Programming and Optimal Alignment

Dynamic Programming

Solves a problem by breaking the problem into smaller subproblems, which are separately solved, then sequentially reassembled to solve the entire problem. The solution to each subproblem is stored in a table along with a score, and the final answer is arrived at by choosing the sequence of solutions that yields the highest score.

Dynamic programming was invented in 1950 by Richard Bellman at Princeton University, and works well when many solutions are possible but an optimal solution must be found.

Dynamic Programming in Sequence Alignment

This approach was first applied to solving biological sequence alignment problems by Saul Needleman and Christian Wunsch in 1970. It can be used to optimally solve either a global alignment problem (Needleman-Wunsch) or a local alignment problem (Smith-Waterman).

Dynamic Programming Steps

How Dynamic Programming Compares Sequences

1. Create matrix and fill with best scores

There are only three possible choices at each position of the matrix: (a) match the residues present; (b) insert a gap in the top sequence; or (c) insert a gap in the side sequence. The exact score depends on the substitution, gap creation and gap insertion values chosen. The best score of each choice is selected, and a pointer to the preceding position used to arrive at the score is stored with the score.

2. Find highest score

For global alignment (Needleman-Wunsch), the highest score in the final row and final column is used. For local alignment (Smith-Waterman), the highest score anywhere in the matrix is used.

3. Trace pointers back to start and generate alignment

The sequence alignment is created in reverse order, by tracing the pointer back from the highest score to the previous highest score, until either the very start (global) or when it reaches a starting score of 0 (local).

Dynamic Programming Illustrated

		G	C	T	G	G	A	A	G	G	C	A	T
	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	0	0	5	5	0	0	5	5	0	0	0
C	0	0	10	3	0	1	1	0	0	1	10	3	0
A	0	0	3	6	0	0	0	6	.	0	3	15	8
G	0	5	0	0	11	5	0	2	11	5	0	8	11
A	0	0	1	0	4	7	10	5	4	7	1	5	4
G	0	5	0	0	5	9	3	6	10	9	3	0	1
C	0	0	10	3	0	2	5	0	3	6	14	7	0
A	0	0	3	6	0	0	7	10	3	0	7	19	12
C	0	0	5	0	2	0	0	3	6	0	5	12	15
T	0	0	0	10	3	0	0	0	0	2	0	5	17

DYNAMIC PROGRAMMING STEPS

1. Create matrix and fill with best scores (keeping pointers)

2. Find highest score (global in final row and column; local anywhere)

3. Trace pointers back to start and generate sequence alignment

Scores are from a local dynamic programming example in Gibas & Jambeck

```
G C T G G A A G G C A . T
| |   |     |   | | |   |
G C A G . . A . G C A C T
```

GLOBAL

Needleman-Wunsch (GCG Gap)

Starts at final row and column in lower right (17), retraces path

```
  G A A G . G C A
  |   | |   | | |
  G C A G A G C A
```

LOCAL

Smith-Waterman (GCG BestFit)

Starts at highest score in matrix (19), retraces path

Heuristic Sequence Comparison

Heuristic Algorithms
Solve a problem by using rules of thumb to reach a solution. The solution is not guaranteed to be an optimal solution, but is generally arrived at far faster than using an optimal solution approach such as dynamic programming.

Heuristic Sequence Comparison Algorithms
In sequence comparison, commonly used heuristic approaches include the BLAT algorithm, developed by Jim Kent in 2002, the BLAST algorithm, developed by Stephen Altschul in 1990, and the FASTA algorithm, developed by William Pearson in 1988. The best known of these is the BLAST algorithm.

BLAST Heuristic Algorithm

How BLAST Works in Nucleotide Sequence Comparison

1. Make a list of 11 letter words (default DNA word length (W) = 11) in the DNA sequence. In a query sequence of length L, the maximum number of words will be L - W + 1 (L - 10 for the default word length of 11).
2. Search with each word for exact matches to words in a pre-indexed database of sequences.
3. For each match found, keep extending the alignment in each direction along the sequence, allowing small gaps, until the score drops below a dropoff value (X), to create high scoring segment pairs (HSPs).
4. Select HSPs that score above the HSP cutoff score (S).
5. Determine the statistical significance of each selected HSP score.
6. Use the Smith-Waterman algorithm to generate a local sequence alignment for each selected HSP.

Nucleotide BLAST Illustrated

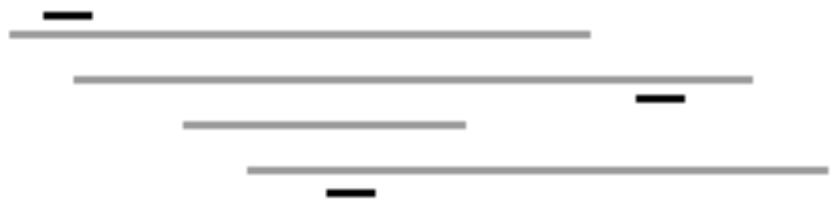
Query sequence of length **L**

Generate a maximum of **L - W + 1** words of length **W**

Match

Find all words that exactly match a database sequence

Database sequences



For each word match, extend in both directions until the score drops below a dropoff value, **X**

Extend

Find High Scoring Pairs, extended alignments that score higher than a cutoff value, **S**

HSPs

Nucleotide BLAST

blastn

Compares a nucleotide query sequence against a nucleotide sequence database. It is best used to find closely related DNA sequences (such as those with over 80% identity) to analyze non-coding DNA or highly conserved DNA regions. It is not well suited to doing other kinds of comparisons.

megablast and discontiguous megablast

This uses a variant of the BLAST algorithm called Mega BLAST which is optimized for quickly comparing nearly identical nucleotide sequences (over 90% identity). It is fastest with larger word sizes (16 and up). The discontiguous version of Mega BLAST is optimized to quickly compare more divergent nucleotide sequences (80% and below identity) and should give better results than Mega BLAST or BLAST for sequence comparisons involving distantly related organisms.

blastx

Compares the six-frame translation of a nucleotide query sequence against a protein sequence database. It is best used to find potential translation products of an unknown nucleotide sequence. It works well if you are unsure of the quality of your sequence data, as it can identify a coding region, despite sequencing induced errors such as frameshifts.

tblastx

Compares the six-frame translation of a nucleotide query sequence against the six-frame translation of a nucleotide sequence database. It is useful for discovering new proteins and ESTs, but is considerably more computationally intensive.

Search for short input sequences

By default, blastn will automatically adjust variables for short input sequences (no filter, word size 7, expect 1000), and this default is best used when working with short query nucleotide sequences or looking for short, nearly exact matches.

BLAST Parameters

Search Set

You will generally be running searches against the nr (nonredundant) database. You may need to choose another database, or limit a search by Organism or Entrez Query. Excluding Uncultured/environmental samples may also be useful.

Algorithm parameters

Filter

Unless you are working with a query sequence that contains repetitive residues or has a biased composition, it is best to turn filtering off, particularly when working with shorter query sequences. Masking of segments in the query sequence is indicated by X's, resulting in "**XXXXX...**" marked regions.

The **Low complexity regions** filter uses the DUST (blastn) or SEG (blastp) programs to filter low complexity regions from sequences.

The **Species-specifics repeats** filter in blastn masks repeat sequences such as LINE's, SINE's, and retroviral repeats.

The **Mask lower case letters** feature allows you to manually select which residues to mask by making them lower case, e.g. transmembrane or coiled-coil regions.

The **Mask for lookup table only** feature uses selected filters only for the initial match, but not the subsequent extension.

Expect threshold (E Value)

The default expectation value threshold is **10**, which means that you can expect up to 10 matches by chance alone. You can decrease the value to get only more significant results, or increase it to **1000 or more** when searching with short queries, which are more likely to be found by chance in a given database.

Word size

Decreasing the default word size (e.g. to **7** from **11** for blastn)may result in a more sensitive search and find shorter regions of homology, but will increase the search time. Decreasing the word size is particularly helpful with shorter query sequences. Increasing the word size will find longer regions of homology and decrease the search time.

BLAST Alignment Example

Download GenBank Graphics Sort by: E value

Broad host range vector vector pCVD047, complete sequence

Sequence ID: gb|KM017898.1| Length: 7140 Number of Matches: 3

Range 1: 6820 to 6879 GenBank Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Identities	Gaps	Strand
109 bits(120)	2e-21	60/60(100%)	0/60(0%)	Plus/Minus

```
Query  1     ACTAACTGTCACGAACCCCTGCAATAACTGTCACGCCCCCCTGCAATAACTGTCACGAAC  60
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  6879  ACTAACTGTCACGAACCCCTGCAATAACTGTCACGCCCCCCTGCAATAACTGTCACGAAC  6820
```

Range 2: 6801 to 6855 GenBank Graphics ▼ Next Match ▲ Previous Match ▲ First Match

Score	Expect	Identities	Gaps	Strand
82.4 bits(90)	3e-13	51/55(93%)	0/55(0%)	Plus/Minus

```
Query  3     TAACTGTCACGAACCCCTGCAATAACTGTCACGCCCCCCTGCAATAACTGTCACG  57
             |||||||||||  ||||||||||||||||||||  ||||||||||||||||||||
Sbjct  6855  TAACTGTCACGCCCCCCTGCAATAACTGTCACGAACCCCTGCAATAACTGTCACG  6801
```

Range 3: 6796 to 6833 GenBank Graphics ▼ Next Match ▲ Previous Match ▲ First Match

Score	Expect	Identities	Gaps	Strand
69.8 bits(76)	2e-09	38/38(100%)	0/38(0%)	Plus/Minus

```
Query  3     TAACTGTCACGAACCCCTGCAATAACTGTCACGCCCCC  40
             ||||||||||||||||||||||||||||||||||||||
Sbjct  6833  TAACTGTCACGAACCCCTGCAATAACTGTCACGCCCCC  6796
```

BLAST Results

Formatting options
You can remove the graphical overview. You can also limit your results by Organism, a particular Entrez query, an expectation value range, or a precent identity range.

Score (bits)
A statistical measure of the significance of the alignment, measured in bits. The higher the score, the more likely for the alignment to be significant. ***A score of* 50 bits or higher *is likely to be significant*.**

Expect value
The expectation or E value. An estimate of the number of times the match may have occurred by chance. The lower the value, the more likely for the alignment to be significant. ***An Expect value of* 0.0001 (1e-4) or below *is likely to be significant*.**

Identities
The fraction of identical residues in the final alignment. The higher the identity, the more likely for the alignment to be significant. **A 70% or greater identity for nucleotide sequences of longer regions of alignment (100 nucleotides and up) *is likely to be significant*, a 30% or greater identity for protein sequences along their entire length *is likely to be significant*.**

Length
The length of the final alignment. The longer the length, the more likely for the alignment to be significant. Very short alignments (e.g. 10 nucleotides) may have a very high percent identity or very low E value, yet not be significant. ***Longer alignments (i.e. 100 nucleotides and above) with high scores, high identities or low E values are likely to be significant*.**

*Remember that BLAST hits are not transitive, unless the alignments overlap, since BLAST alignment is fundamentally a local alignment. Thus, if query **A** results in significant hits to **B** and **C**, **B** and **C** are not necessarily similar to each other (**A** might contain domain **1** and **2**, **B** might contain only domain **1**, and **C** might contain only domain **2**). It is thus wise to carefully check the length and extent of any alignment.*

Protein BLAST

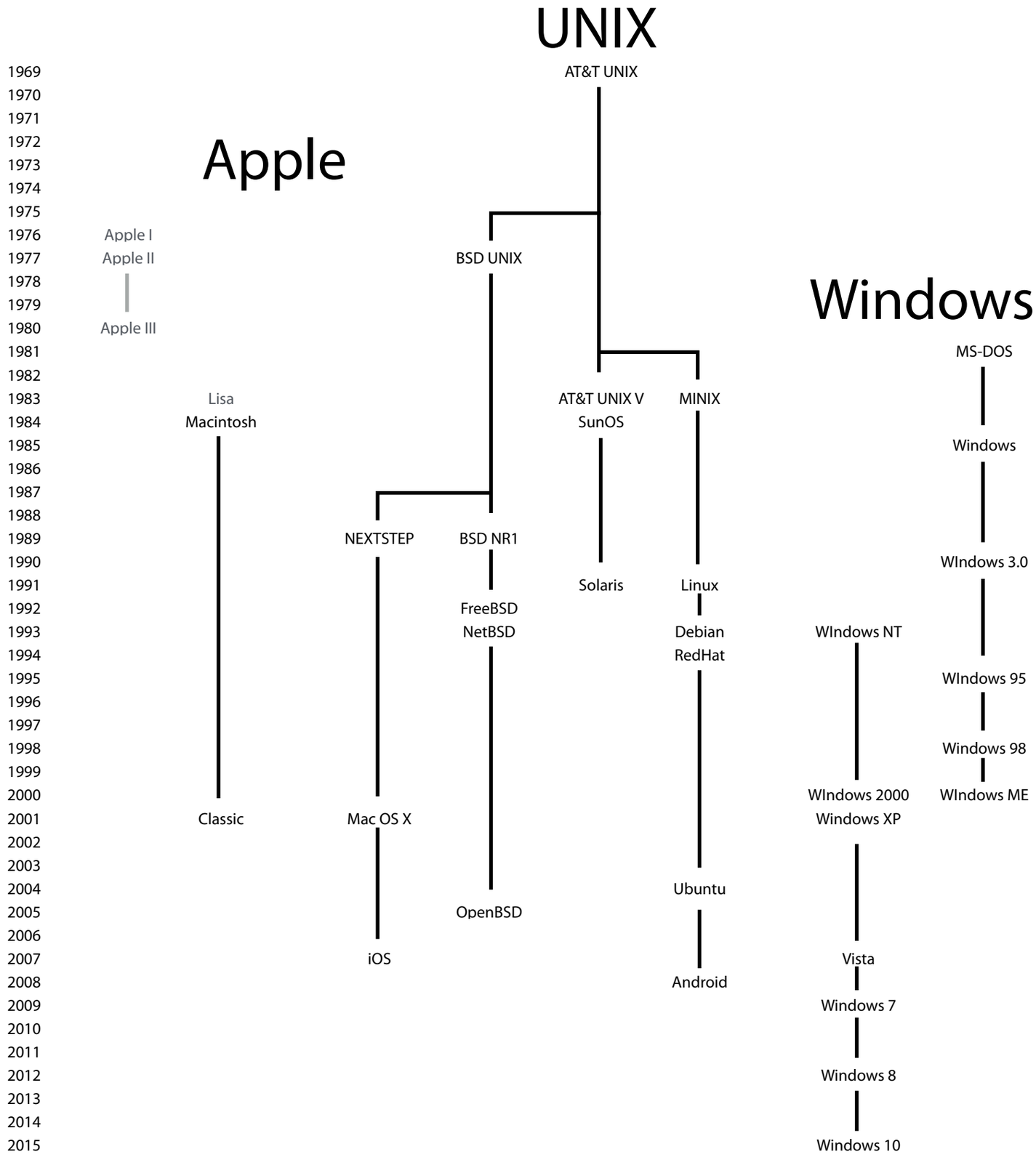
BLAST Protein Sequence Comparison

- A variety of BLAST programs are featured by NCBI for protein-protein comparison, including blastp (protein vs. protein), PSI-BLAST (position specific iterated), PHI-BLAST (pattern hit initiated), DELTA-BLAST (incorporates a Conserved Domain Database search) and tblastn (protein vs. translated database).
- The default word size for protein BLAST searches is 3, this can be changed to 2 for more stringent, but slower searches.
- The choice of Dayhoff substitution matrix can be important. The default matrix for NCBI BLAST protein comparison is BLOSUM62, which is optimized for long query sequences (over 85 aa) and known close homologies. When searching with short query sequences or distant homologies, be sure to try other matrices.

Protein BLAST Results Rules of Thumb

- Proteins that are more than 30% identical throughout their entire lengths are likely homologous.
- Proteins that are 20 to 30% identical throughout their entire lengths may or may not be homologous (the "gray zone").
- Proteins that are less than 20% identical throughout their entire lengths are not likely homologous.
- Matches that are more than 50% identical in a 20 to 40 amino acid region occur frequently by chance.

Evolution of Modern Operating Systems



Lecture 2:
Introduction to Computing
February 6, 2017

The Era of Modern Computing

1964	Mouse & Graphical User Interface	Douglas Engelbart, Xerox PARC
1969	ARPAnet	UCLA, Stanford, UC Santa Barbara & University of Utah
1969	UNIX	Ken Thompson & Dennis Ritchie, Bell Laboratories
1973	C	Dennis Ritchie & Brian Kernighan, Bell Laboratories
1973	Ethernet	Robert Metcalfe, Harvard University/Xerox PARC
1973	FTP	Alex McKenzie, BBN
1974	TCP	Vint Cerf & Robert Kahn
1975	Microsoft Corporation	Bill Gates & Paul Allen
1976	Apple Computer	Steve Wozniak & Steve Jobs
1978	Usenet	Tom Truscott, Jim Ellis & Steve Bellovin
1981	IBM PC	IBM Corporation
1982	TCP/IP	ARPA
1984	DNS	Jon Postel
1984	Macintosh	Apple Computer
1985	Windows	Microsoft Corporation
1986	NeXT Computer	Steve Jobs
1989	HTML & HTTP/BSD Unix NR1	Tim Breners-Lee, CERN/University of California, Berkeley
1991	Linux	Linus Torvalds
1993	Mosaic	Marc Andreessen
2001	OS X	Apple Computer
2004	Google	Larry Page & Sergey Brin
2007	iOS	Apple Computer
2008	Cloud Computing	Amazon Web Services

Open Source

The Open Source Initiative (OSI) certifies Open Source licenses. To be OSI certified, the software must be distributed under a license that guarantees the right to read, redistribute, modify, and use the software freely. A variety of Open Source licenses exist, from the more permissive BSD style license (allows commercial resale) to the stricter GNU General Public License (GPL) license (any software created with the code must remain free).

Open Source Definition for Open Source Initiative Certification

1. Free Redistribution (may not be restricted)
2. Source Code (unobfuscated, included or readily available)
3. Derived Works (modifications must be allowed to be distributed under the same terms)
4. Integrity of The Author's Source Code (can be protected)
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of License (license applies to all to whom the program is redistributed)
8. License Must Not Be Specific to a Product (including a particular software distribution)
9. License Must Not Restrict Other Software (distributed along with it)
10. License Must Be Technology-Neutral

Examples of Open Source Operating Systems and Software

BSD NR1 Unix (BSD), Linux (GPL), Darwin (Apple), Ubuntu (GPL), Android (Apache) and Apache (Apache), MySQL (GPL/commercial), Firefox (Mozilla), GIMP (GPL), Staden (BSD), EMBOSS (GPL), Python (Python), Biopython (Biopython), R(GPL), Bioconductor (Artistic)

Unix and Computational Biology

- Historically, Unix has been used as a free academic and research operating system (BSD, FreeBSD, etc.), and many forms of Unix and programs that run on Unix are Open Source, available for free with source code.
- Unix is stable, efficient and easy to program, with many powerful scripting, automation and programming tools built in.
- New algorithms in computational biology are generally first implemented in Unix.
- All the bioinformatics tools needed to do complex analysis are available for free on Unix (BLAST, FASTA, CLUSTAL, PHYLIP, PHRED, PHRAP, CONSED, EMBOSS, HISAT, Lighter, Bowtie, etc.).
- Unix operating systems generally use a command line environment known as a shell to interact with them. A shell interprets and executes the commands you give it, and can also run text files called shell scripts,which allow for commands to be strung together, manipulated, and automated. The default shell for OS X and cygwin is **bash**.
- Some Unix operating systems, particularly OS X, Linux and Ubuntu, have good GUIs as well (notably, Linux Mint/Cinnamon and elementaryOS/Pantheon).

Terminal and the Unix Command Line

Terminal

Terminal, located in /Applications/Utilities, is the application which gives an OS X user command line shell access to the underlying Unix operating system. *One can drag a folder or application to the Terminal window to get its pathname,* which is often required when issuing Unix commands.

ls (list)

Lists the current directory's contents. Adding the **-a** option (**ls -a**) lists all contents, including what is normally invisible (file or directory names starting with a period, e.g. **.bash_profile**, are normally invisible). Adding the **-l** option (**ls -l**) lists long information about files: type, permissions, links, owner, group, size, modification date & time and name. The wild card character (*) is often useful in arguments for this command, e.g. **ls *.doc** will list all Word files with that extension in a directory.

cd (change directory)

By itself, **cd** takes you to your home directory. Using an argument of two periods, i.e. **cd ..**, moves you to the directory directly above the current directory, while **cd /** moves you to the root directory. If you get lost, type **pwd** to print your working directory, that is, list your current directory as a pathname.

exit

Type **exit** to logout of a Terminal session.

Unix Commands

Commands
A command in Unix consists of a **program** that is executed by typing its name. That program then generates output, by default to the Terminal window, based on the options and arguments it was provided with. Options follow the command, and arguments follow options, all separated by single spaces,
e.g. **`command -option(s) arguments(s)`**

Program
A command line program is executed by typing its name on the command line and pressing return, e.g. typing **`ls`** and pressing return will list files in the current directory. Hundreds of command programs are built-in.

Options
Options modify the behavior of a program. Traditionally, options are represented by a hyphen followed by a single letter. Adding the **`-l`** option to **`ls`**, i.e. **`ls -l`** results in long file listings. Multiple options can be used together by typing more than one letter after the hyphen.

Arguments
Arguments are the input the program acts upon. Typically, they are names of files or directories, but can be nearly anything, including the output of other programs. Unix wild card characters can be useful here, such as `*`, which stands for any group of characters, so **`ls *.txt`** would list only files with names ending in **`.txt`**.

Output
The result of a program is its output, which in OS X by default goes to the Terminal window, and can be copied and pasted. However, a program's output can be sent to a file, or even to another program.

Regular Expressions and grep

grep (Globally search for Regular Expression and Print)

Grep is a powerful Unix text searching utility that is available at both the command line in OS X and in certain applications such as TextWrangler. Grep can search the input for lines containing a match to a given pattern, using regular expressions if necessary, then output the lines that matched. Grep can thus greatly automate searching for information in text files. Similar functionality is provided at the Windows command line by Findstr.

The Unix **grep** utility can be invoked from the Terminal in OS X using the following syntax:

```
grep -[options] 'pattern' filename(s)
```

Useful grep Options

`-c` print count of matching lines, rather than the matching lines themselves
`-i` ignore case distinctions in pattern and file(s)
`-l` print filenames containing matching lines, but not the matching lines

grep Results

grep normally prints a list of every line within the file(s) searched containing a match.

In Terminal, typing `grep 'RNA' sars.txt` will find all lines containing RNA within the **sars.txt** file. To automatically output those lines to a file called **sarsrna.txt**, one would type `grep 'RNA' sars.txt > sarsrna.txt` (in Unix, **>** redirects output).

`grep '>' sequences.fasta` will return the name of every sequence in the **sequences.fasta** file (in a fasta file, the sequence name is on a line beginning with >).

`grep -c '>' sequences.fasta` will only return the number of sequences in the file.