Practical Bioinformatics

Mark Voorhies

5/22/2015

PAM (Dayhoff) and BLOSUM matrices

- PAM1 matrix originally calculated from manual alignments of highly conserved sequences (myoglobin, cytochrome C, etc.)

PAM (Dayhoff) and BLOSUM matrices

- PAM1 matrix originally calculated from manual alignments of highly conserved sequences (myoglobin, cytochrome C, etc.)
- We can think of a PAM matrix as evolving a sequence by one unit of time.

PAM (Dayhoff) and BLOSUM matrices

- PAM1 matrix originally calculated from manual alignments of highly conserved sequences (myoglobin, cytochrome C, etc.)
- We can think of a PAM matrix as evolving a sequence by one unit of time.
- If evolution is uniform over time, then PAM matrices for larger evolutionary steps can be generated by multiplying PAM1 by itself (so, higher numbered PAM matrices represent greater evolutionary distances).

PAM (Dayhoff) and BLOSUM matrices

- PAM1 matrix originally calculated from manual alignments of highly conserved sequences (myoglobin, cytochrome C, etc.)
- We can think of a PAM matrix as evolving a sequence by one unit of time.
- If evolution is uniform over time, then PAM matrices for larger evolutionary steps can be generated by multiplying PAM1 by itself (so, higher numbered PAM matrices represent greater evolutionary distances).
- The BLOSUM matrices were determined from automatically generated ungapped alignments. Higher numbered BLOSUM matrices correspond to *smaller* evolutionary distances. BLOSUM62 is the default matrix for BLAST.

Motivation for scoring matrices

Frequency of residue i:

$$p_i$$

Motivation for scoring matrices

Frequency of residue i:

$$p_i$$

Frequency of residue i aligned to residue j:

$$q_{ij}$$

Motivation for scoring matrices

Frequency of residue i:

$$p_i$$

Frequency of residue i aligned to residue j:

$$q_{ij}$$

Expected frequency if i and j are independent:

$$p_i p_j$$

Motivation for scoring matrices

Frequency of residue i:

$$p_i$$

Frequency of residue i aligned to residue j:

$$q_{ij}$$

Expected frequency if i and j are independent:

$$p_i p_j$$

Ratio of observed to expected frequency:

$$\frac{q_{ij}}{p_i p_j}$$

Motivation for scoring matrices

Frequency of residue i:

$$p_i$$

Frequency of residue i aligned to residue j:

$$q_{ij}$$

Expected frequency if i and j are independent:

$$p_i p_j$$

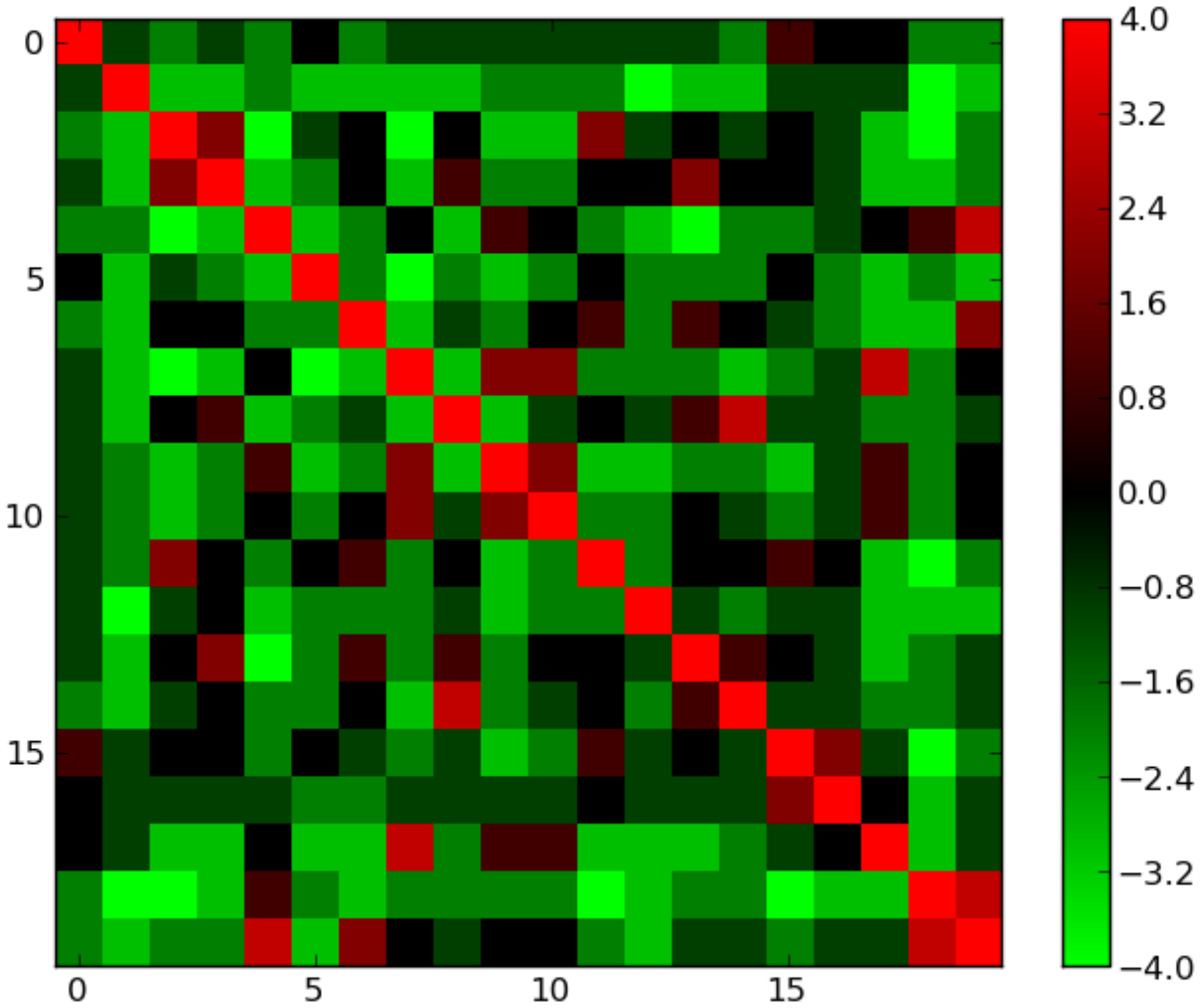
Ratio of observed to expected frequency:

$$\frac{q_{ij}}{p_i p_j}$$

Log odds (LOD) score:

$$s(i,j) = \log \frac{q_{ij}}{p_i p_j}$$

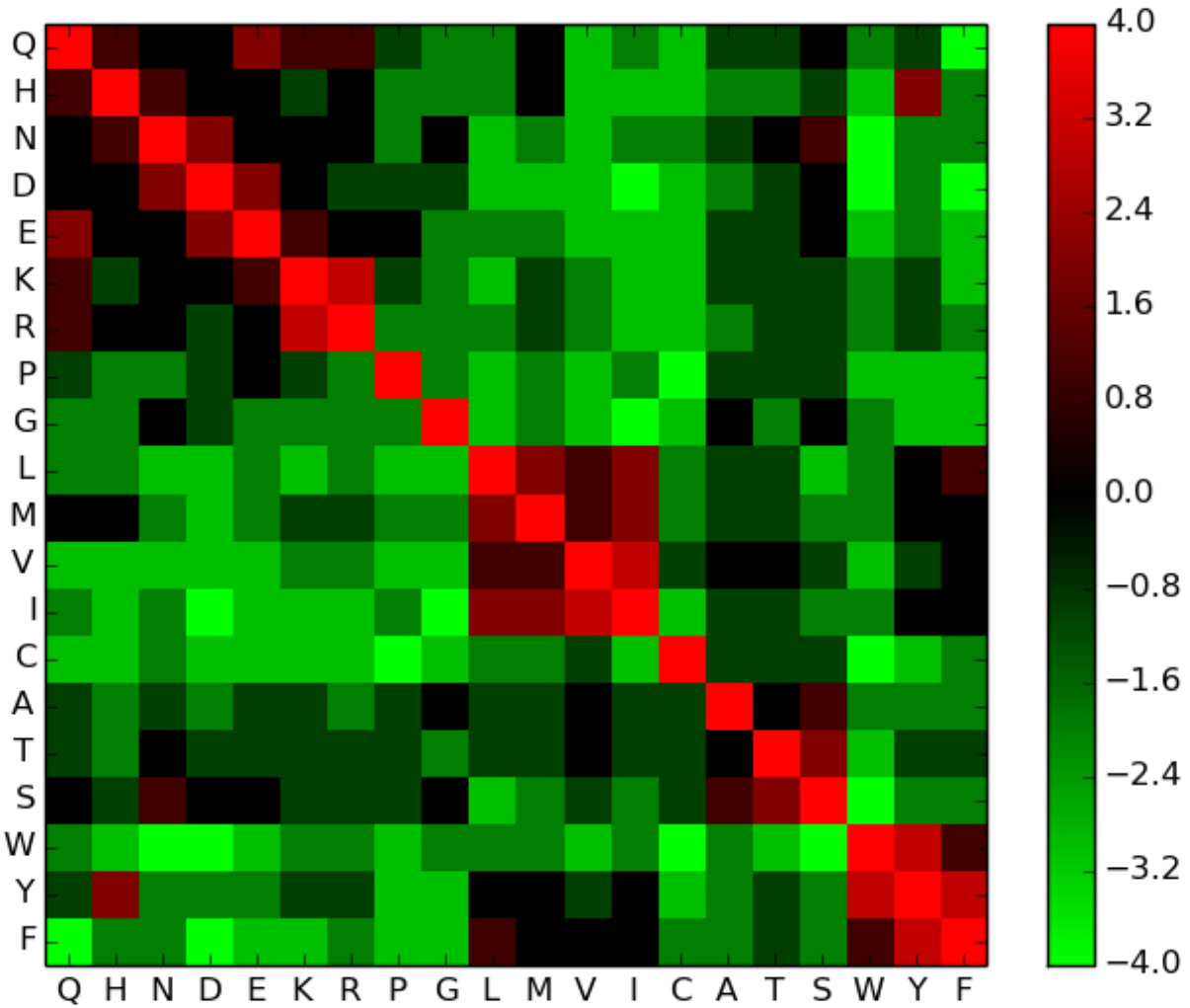
BLOSUM45 in alphabetical order



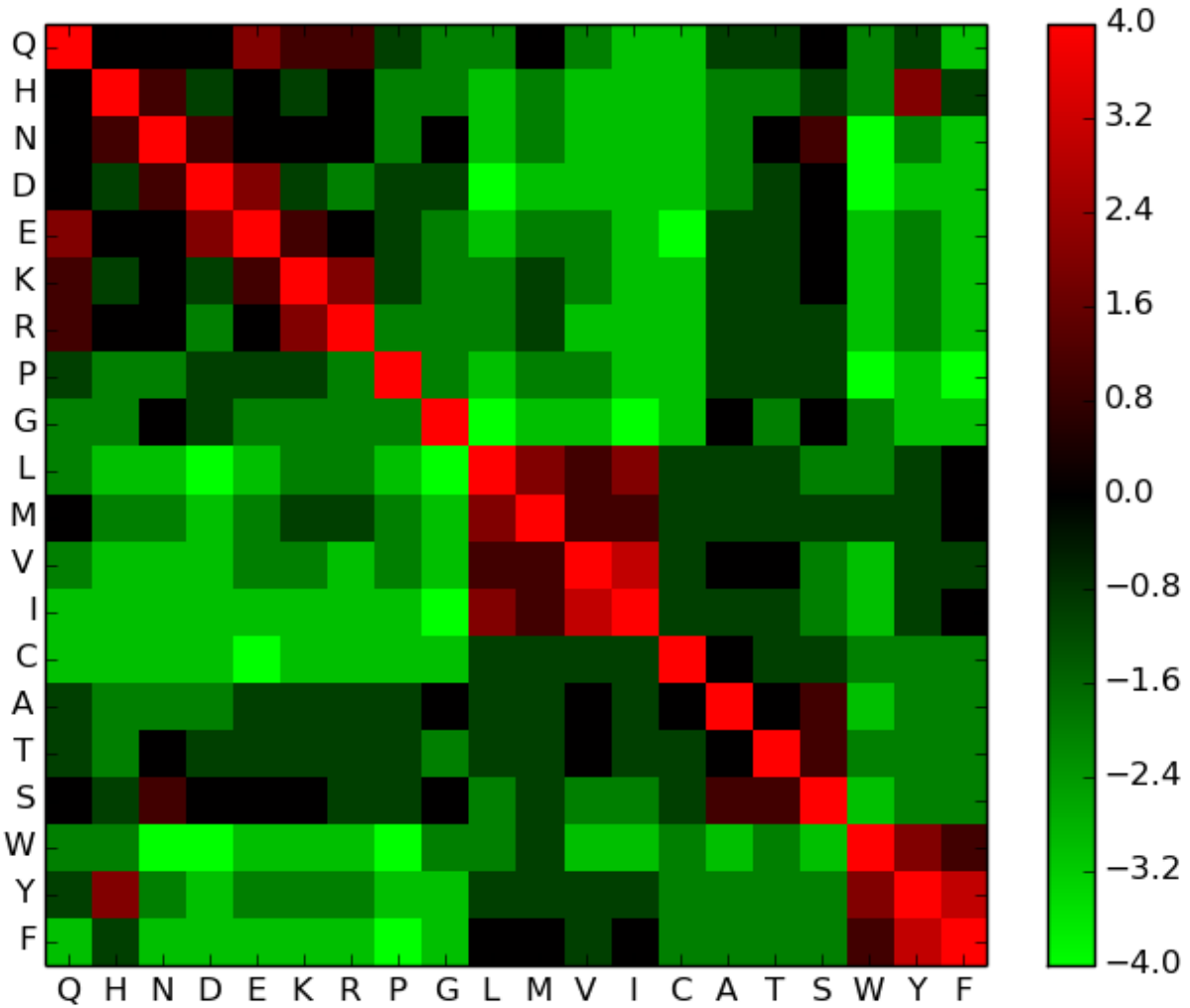
Clustering amino acids on log odds scores

```
import networkx as nx
try:
    import Pycluster
except ImportError:
    import Bio.Cluster as Pycluster

class ScoreCluster:
    def __init__(self, S, alpha_aa = "ACDEFGHIKLMNPQRSTVWY"):
        """Initialize from numpy array of scaled log odds scores."""
        (x,y) = S.shape
        assert(x == y == len(alpha_aa))

        # Interpret the largest score as a distance of zero
        D = max(S.reshape(x**2))-S
        # Maximum-linkage clustering, with a user-supplied distance matrix
        tree = Pycluster.treecluster(distancematrix = D, method = "m")

        # Use NetworkX to read out the amino-acids in clustered order
        G = nx.DiGraph()
        for (n,i) in enumerate(tree):
            for j in (i.left, i.right):
                G.add_edge(-(n+1),j)

        self.ordering = [i for i in nx.dfs_preorder(G, -len(tree)) if(i >= 0)]
        self.names = "".join(alpha_aa[i] for i in self.ordering)
        self.C = self.permute(S)

    def permute(self, S):
        """Given square matrix S in alphabetical order, return rows and columns
        of S permuted to match the clustered order."""
        return array([[S[i][j] for j in self.ordering] for i in self.ordering])
```

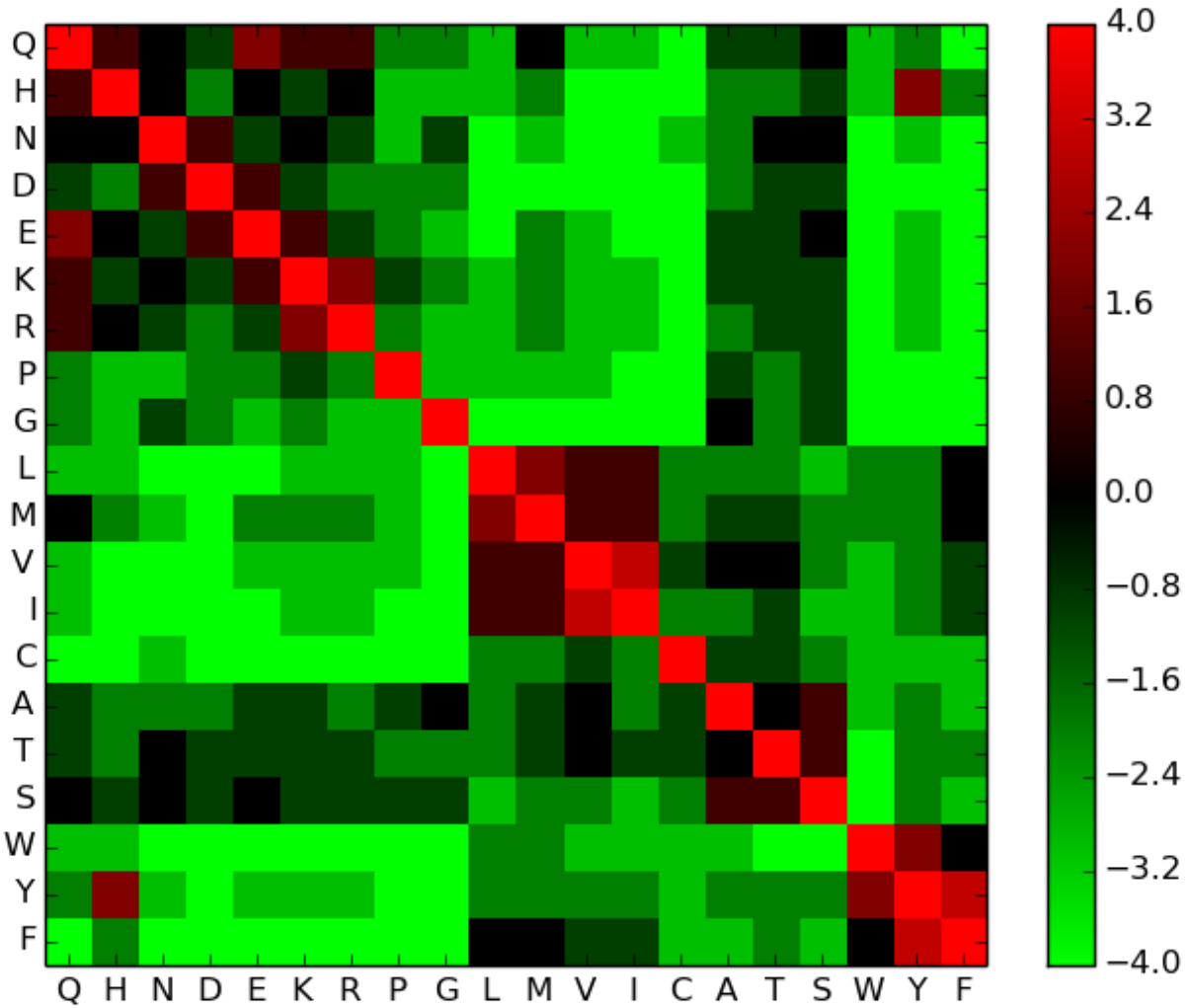
BLOSUM45 – maximum linkage clustering



BLOSUM62 with BLOSUM45 ordering



BLOSUM80 with BLOSUM45 ordering



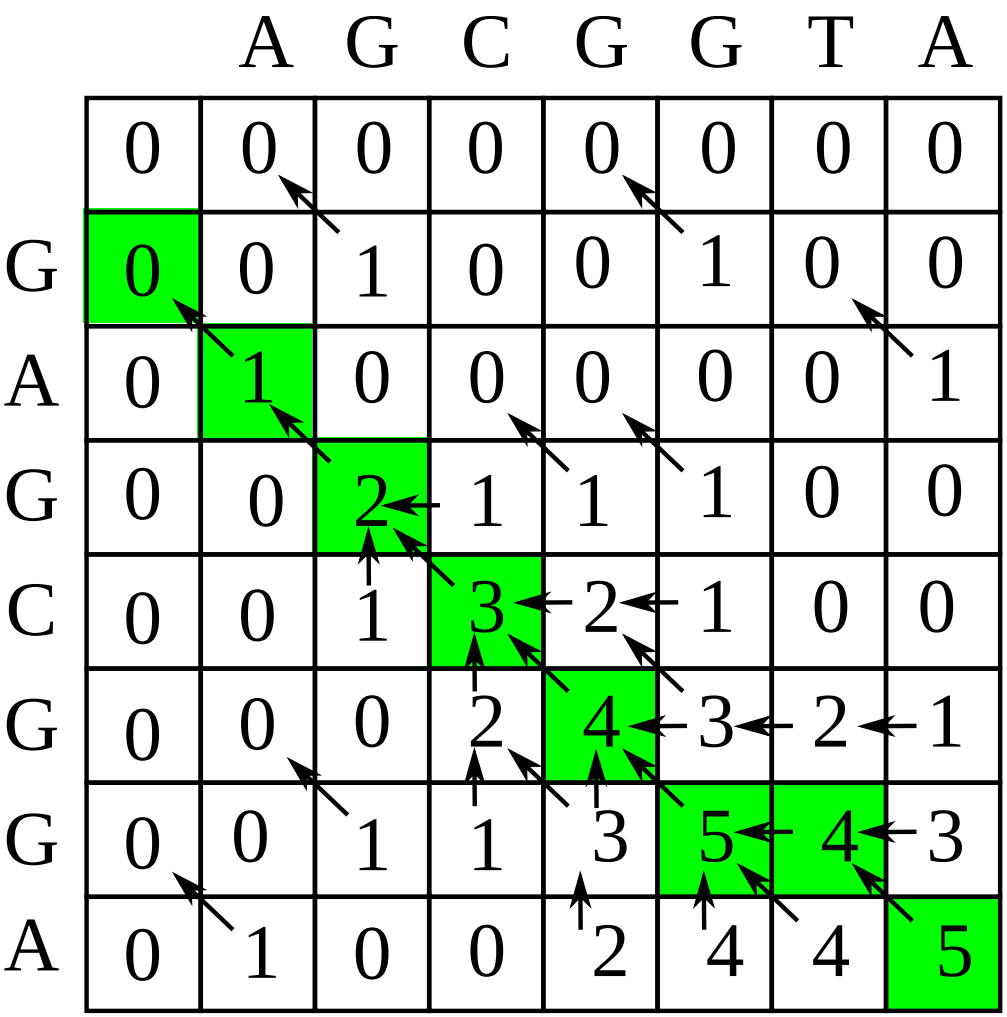
Smith-Waterman

The implementation of local alignment is the same as for global alignment, with a few changes to the rules:

- Initialize edges to 0 *(no penalty for starting in the middle of a sequence)*
- The maximum score is never less than 0, and no pointer is recorded unless the score is greater than 0 *(note that this implies negative scores for gaps and bad matches)*
- The trace-back starts from the highest score in the matrix and ends at a score of 0 *(local, rather than global, alignment)*

Because the naive implementation is essentially the same, the time and space requirements are also the same.

Smith-Waterman



Timing CLUSTALW

Timing CLUSTALW from the command line:

```
for i in 50 100 150 200 250 300 350 400 450; do
    head -n $i -q G217B_iron.fasta Pb01_iron.fasta > temp.fasta;
    time clustalw -infile=temp.fasta -type=DNA -align;
done
```

The output looks like this:

```
Sequences (1:2) Aligned. Score:  0
Guide tree file created:   [temp.dnd]

There are 1 groups
Start of Multiple Alignment

Aligning...
Group 1:                        Delayed
Alignment Score 7238

CLUSTAL-Alignment file created  [temp.aln]


real    0m3.400s
user    0m3.388s
sys     0m0.012s
```

Timing CLUSTALW

Format the timing results as CSV for your favorite curve fitting program

```
#!/usr/bin/env python
# Time-stamp: <ParseTimes.py 2011-03-29 21:10:59 Mark Voorhies>
"""Parse wall times from a log file on stdin and write them as a CSV
formatted column for Excel/OpenOffice/etc on stdout.  If command line
arguments are given, treat them as a second output column."""

from csv import writer
import re
time_re = re.compile("^real.*(?P<minutes>[\d]+)m(?P<seconds>[\d]+\.[\d]+)s",re.M)

if(__name__ == "__main__"):
    import sys
    args = sys.argv[1:]
    out = writer(sys.stdout)
    i = 0
    for t in time_re.finditer(sys.stdin.read()):
        try:
            y = args[i]
            i += 1
        except IndexError:
            y = ""
        out.writerow(
            (float(t.group("minutes"))*60+float(t.group("seconds")),y))

    del out
```

Timing CLUSTALW

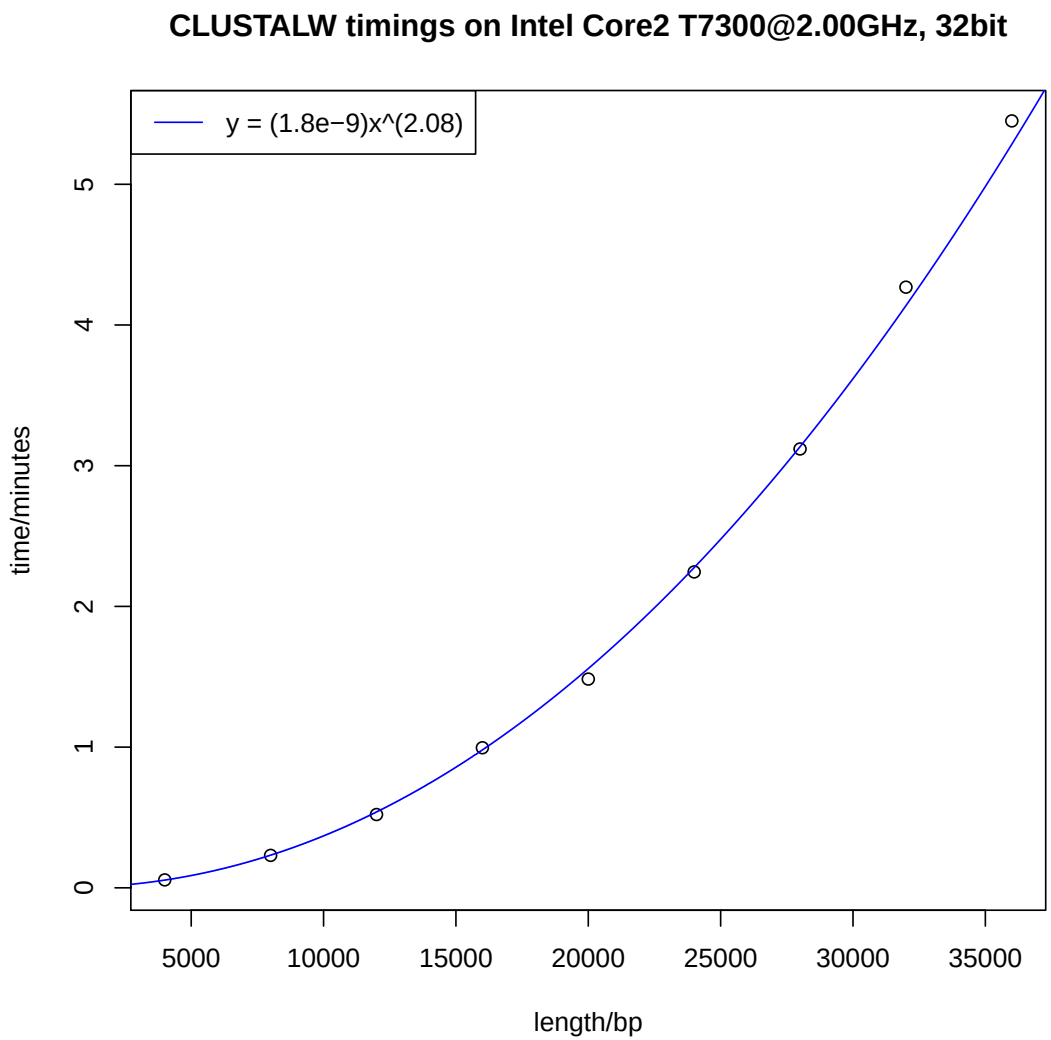
You can fit the timing results to a curve with SciPy.

$$\begin{aligned} y &= Ax^B \\ \log y &= \log Ax^B \\ &= \log A + B \log x \\ &= A' + B \log x \end{aligned}$$

Here is an R script that does the same thing:

```
data <- read.csv("timings.csv", header = FALSE, col.names = c("t","n"))
x <- log(data$n*80)
y <- log(data$t/60)
f <- lm(y ~ x)
x0 <- 0:40000
a <- exp(f$coeff[1])
b <- f$coeff[2]
pdf("ClustalwTimings.pdf")
plot(data$n*80, data$t/60, xlab = "length/bp", ylab = "time/minutes",
     main = "CLUSTALW timings on Intel Core2 T7300@2.00GHz, 32bit")
points(x0, a*x0^b, col = "blue", type = "l")
legend("topleft", c("y = (1.8e-9)x^(2.08)"), col = "blue", lty = 1)
dev.off()
```

CLUSTALW takes $O(MN)$ time



Basic Local Alignment Search Tool

Why BLAST?

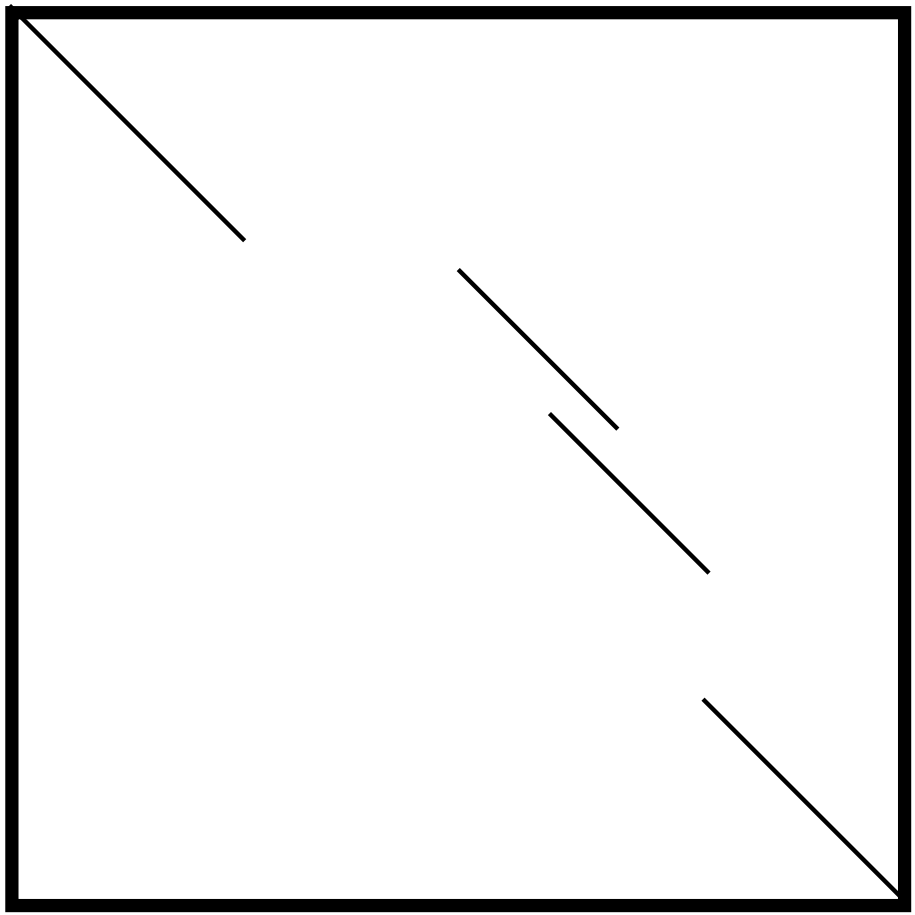
- Fast, heuristic approximation to a full Smith-Waterman local alignment
- Developed with a statistical framework to calculate expected number of false positive hits.
- Heuristics biased towards "biologically relevant" hits.

Seeding searches

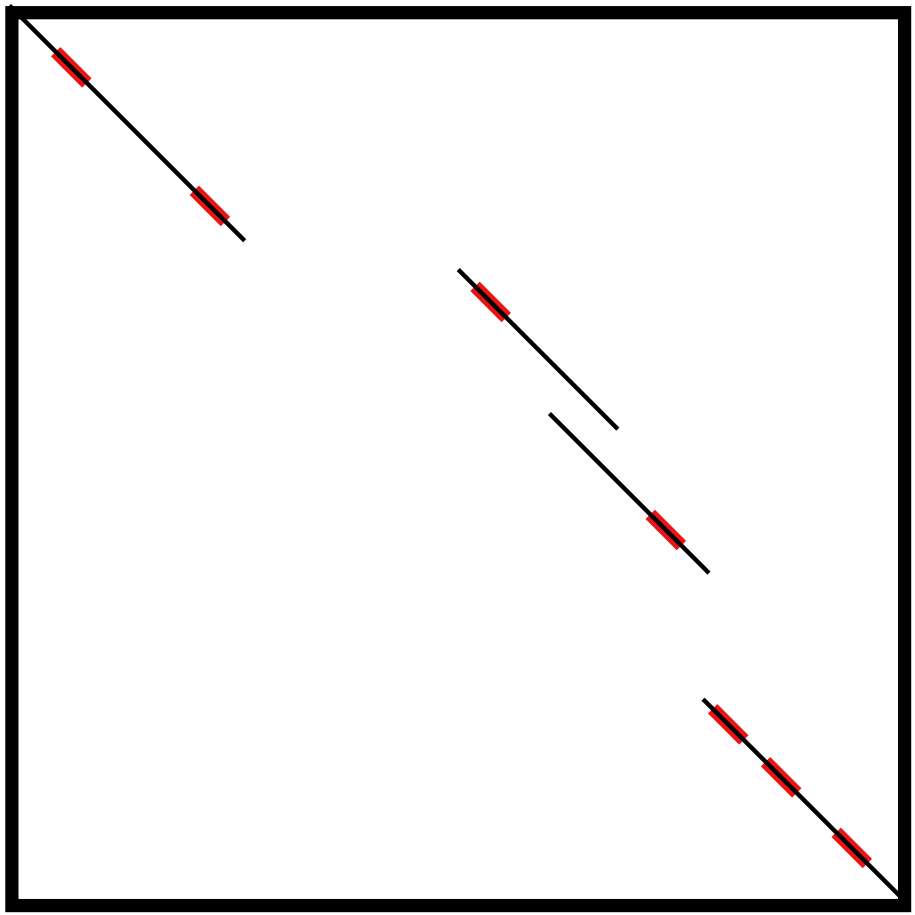
Most of the magic in a sequence-search tool lives in its indexing scheme

Program	Purpose	Indexing
BLAST	Database searching	Target indexing, 3aa or 11nt words
BLAT	mRNA mapping	Query indexing
BOWTIE(2)	RnaSeq	Enhanced suffix tree (BWT)
HOBBES	RnaSeq	Inverted index for non-heuristic search

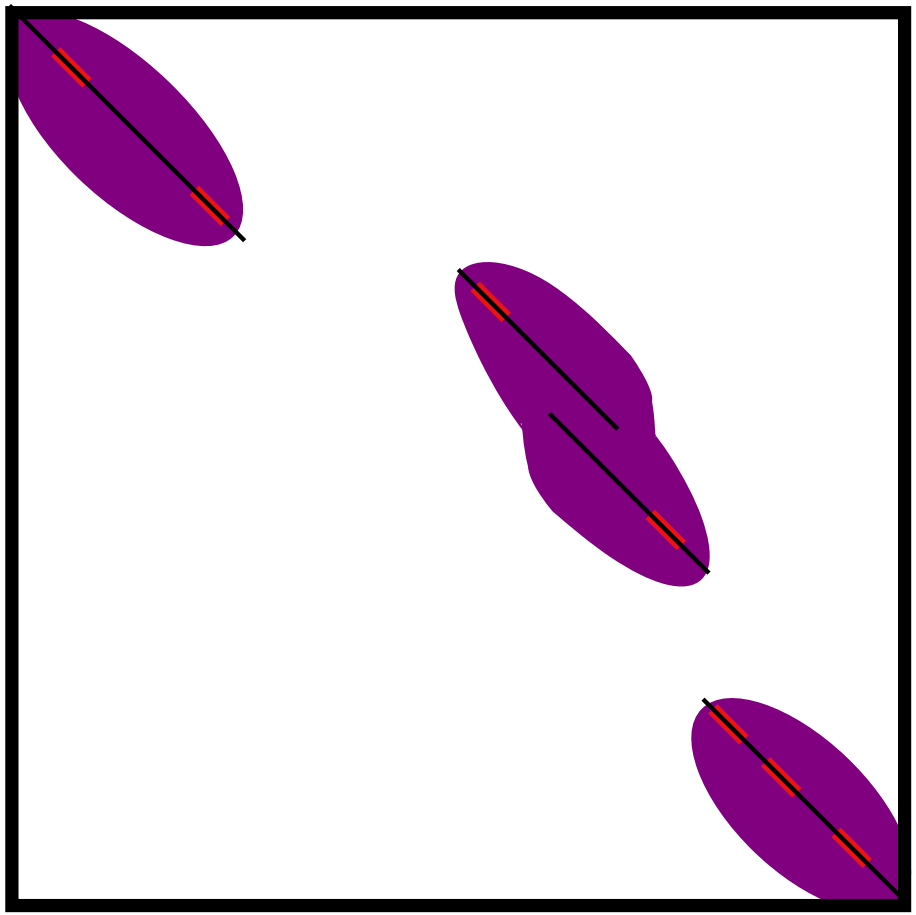
BLAST: A quick overview



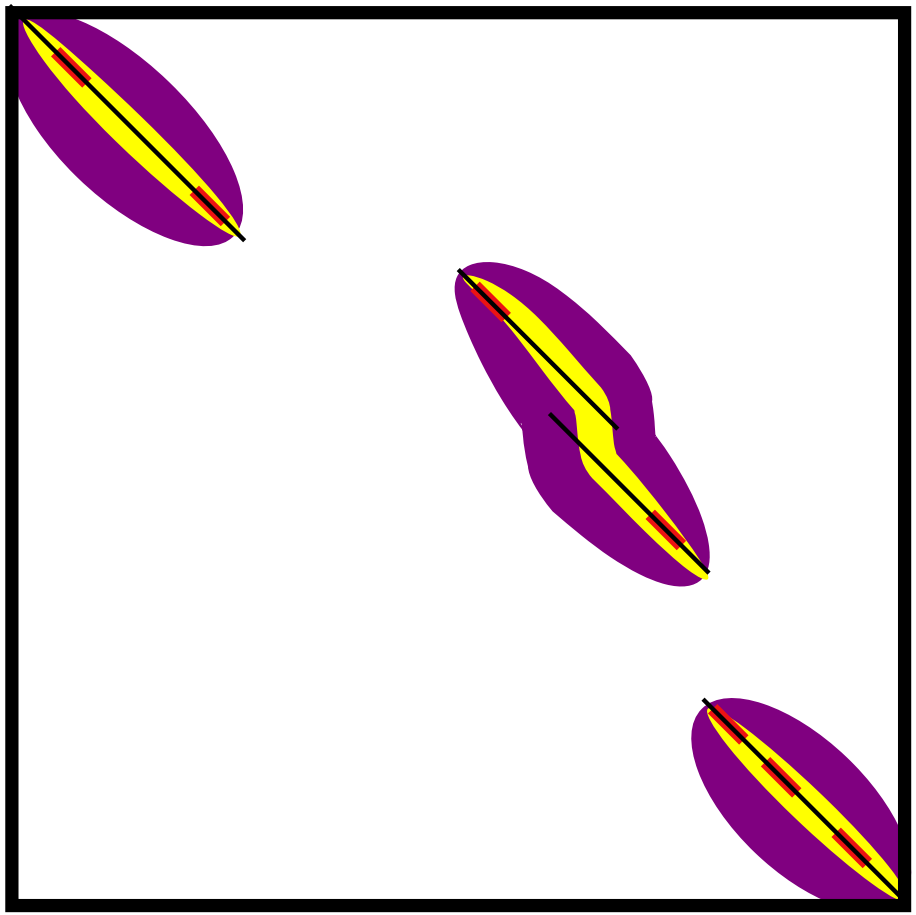
BLAST: Seed from exact word hits



BLAST: Myers and Miller local alignment around seed pairs



BLAST: High Scoring Pairs (HSPs)



Karlin-Altschul Statistics

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

- E: Expected number of "random" hits in a database of this size scoring *at least* S.
- S: HSP score
- m: Query length
- n: Database size
- k: Correction for similar, overlapping hits
- λ: normalization factor for scoring matrix

Karlin-Altschul Statistics

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

- E: Expected number of "random" hits in a database of this size scoring *at least* S.
- S: HSP score
- m: Query length
- n: Database size
- k: Correction for similar, overlapping hits
- λ: normalization factor for scoring matrix

A variant of this formula is used to generate sum probabilities for combined HSPs.

Karlin-Altschul Statistics

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

- E: Expected number of "random" hits in a database of this size scoring *at least* S.
- S: HSP score
- m: Query length
- n: Database size
- k: Correction for similar, overlapping hits
- λ: normalization factor for scoring matrix

A variant of this formula is used to generate sum probabilities for combined HSPs.

$$p = 1 - e^{-E}$$

Karlin-Altschul Statistics

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

- E: Expected number of "random" hits in a database of this size scoring *at least* S.
- S: HSP score
- m: Query length
- n: Database size
- k: Correction for similar, overlapping hits
- λ: normalization factor for scoring matrix

A variant of this formula is used to generate sum probabilities for combined HSPs.

$$p = 1 - e^{-E}$$

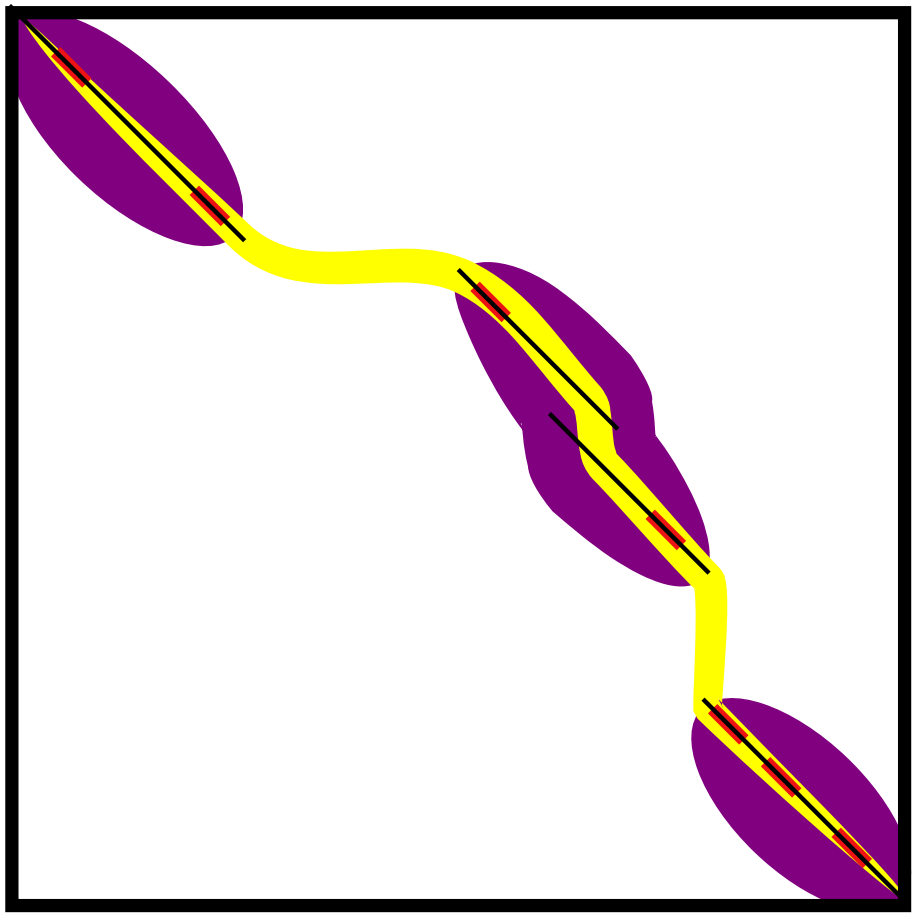
(If you care about the difference between E and p, you're already in trouble)

Karlin-Altschul Statistics

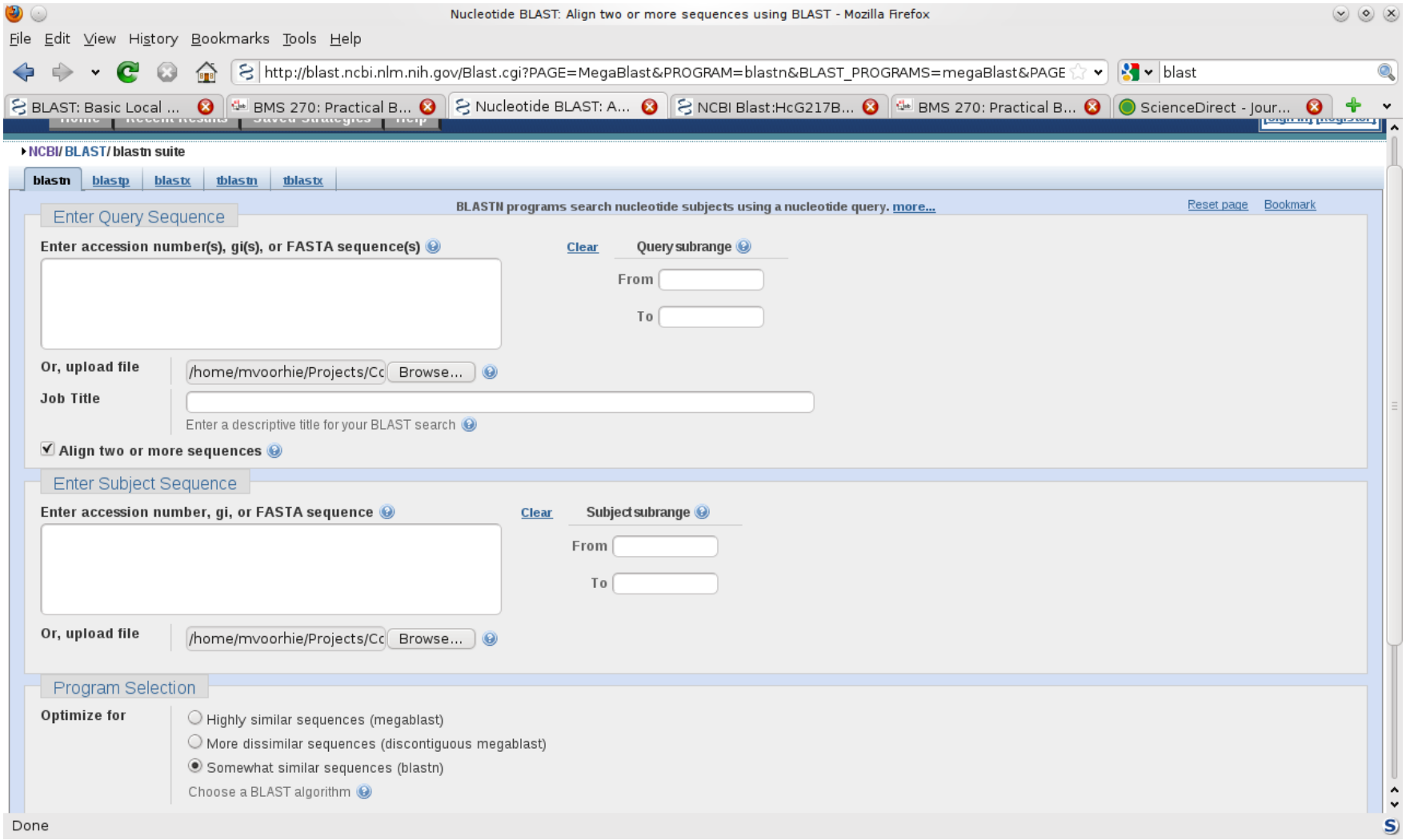
Important points:

- Extreme value distribution
- Assumption of infinite sequence length
- No rigorous framework for gap statistics (hmmer3 tries to fill this gap)

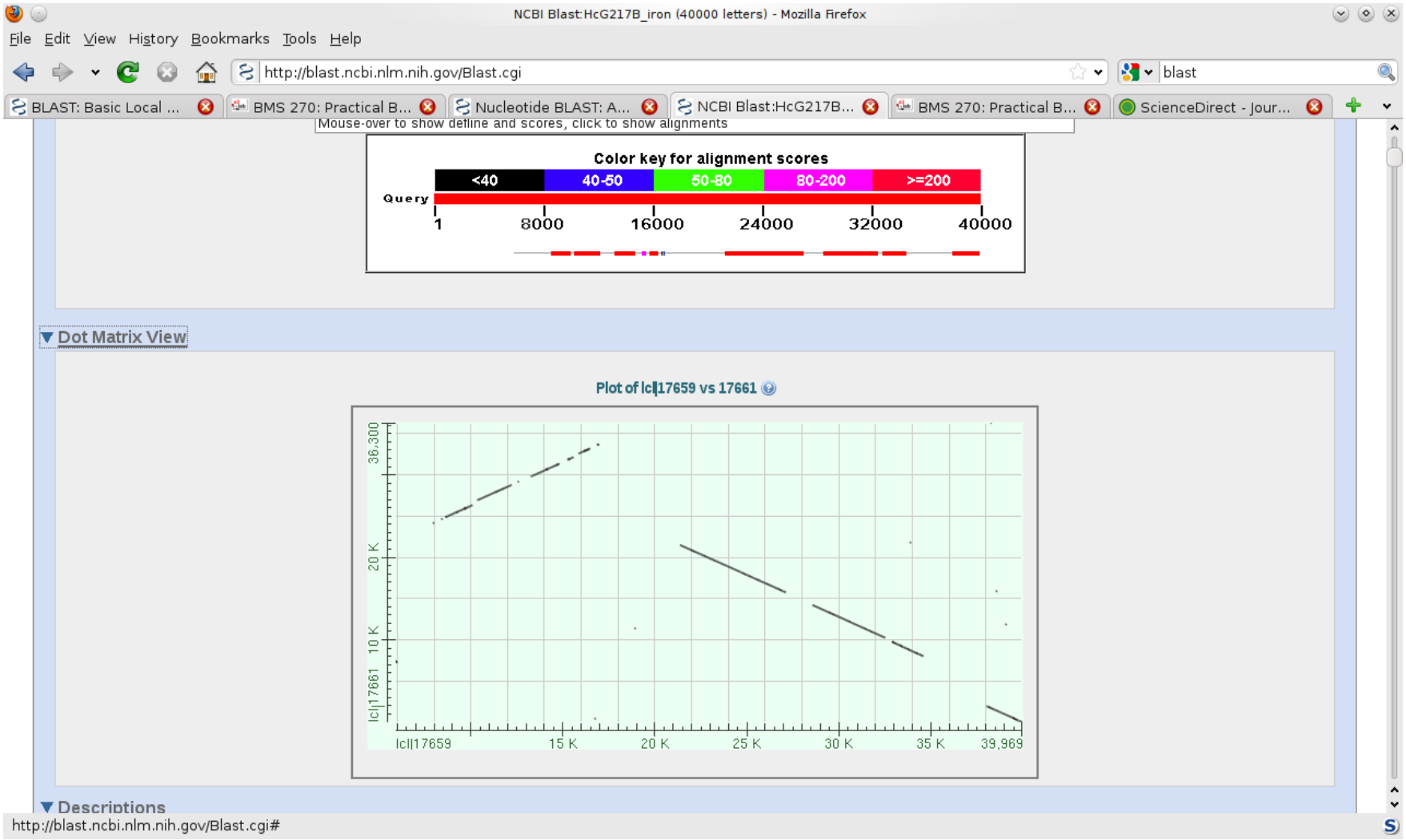
Gapped BLAST: Merge neighboring HSPs



How fast is BLAST?



How fast is BLAST?



How fast is BLAST?

```
time bl2seq -p blastn -i G217B_iron.fasta -j Pb01_iron.fasta -e 1e-6 > temp.blastn

real    0m0.342s
user    0m0.080s
sys     0m0.032s
```

The basic flavors of BLAST

Target Query	Protein	DNA
Protein	BLASTP	TBLASTN
DNA	BLASTX	BLASTN TBLASTX

Summary

- BLAST is very fast, at the expense of not guaranteeing globally optimal results

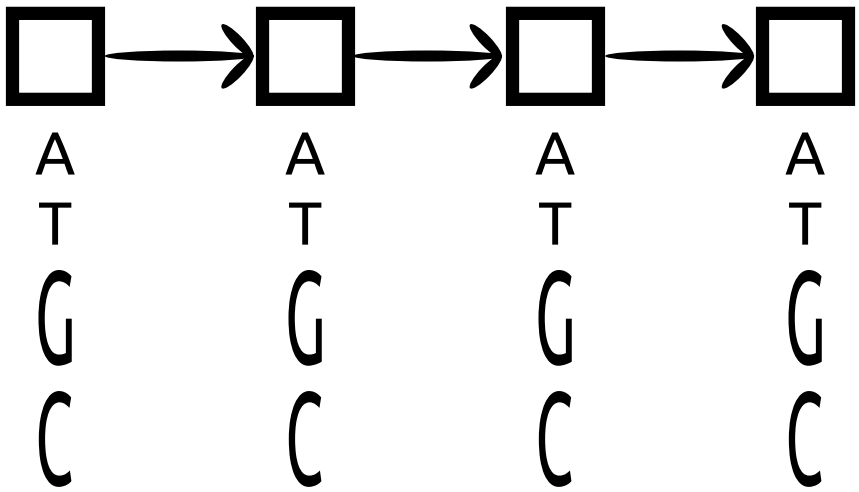
Summary

- BLAST is very fast, at the expense of not guaranteeing globally optimal results
- But the trade-offs that it makes are biased towards "biologically relevant" results

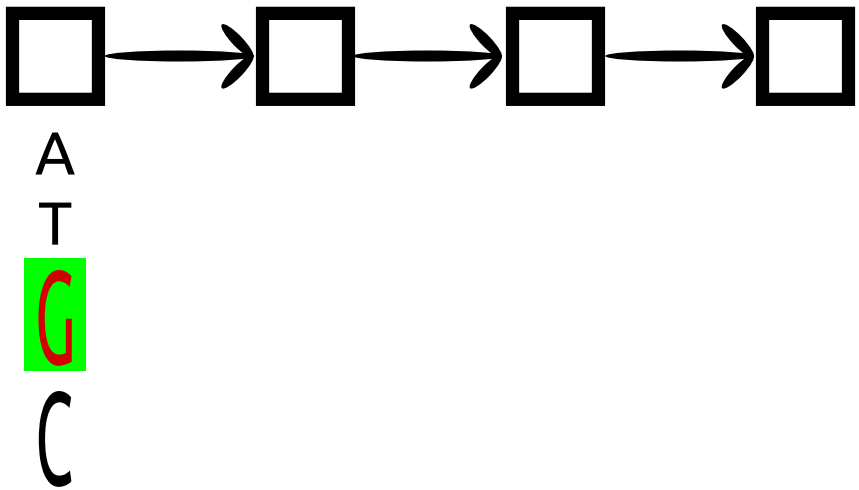
Summary

- BLAST is very fast, at the expense of not guaranteeing globally optimal results
- But the trade-offs that it makes are biased towards "biologically relevant" results
- And it provides a statistical framework for evaluating its results.

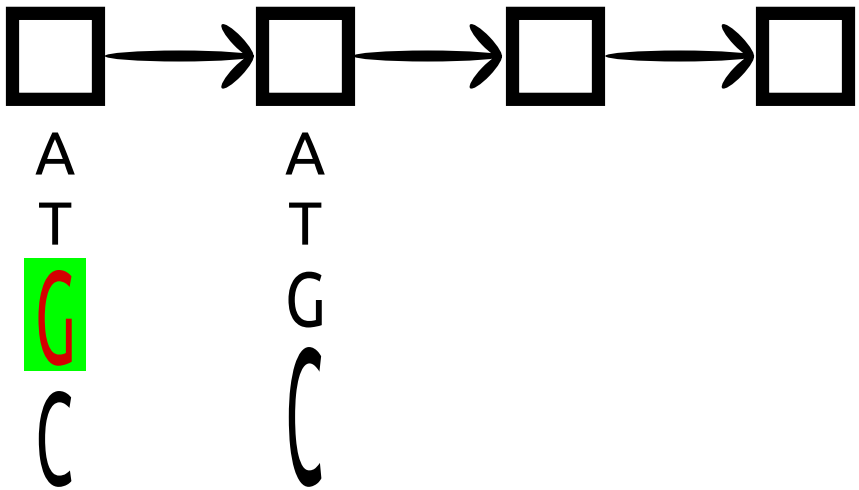
0^{th} order Markov Model



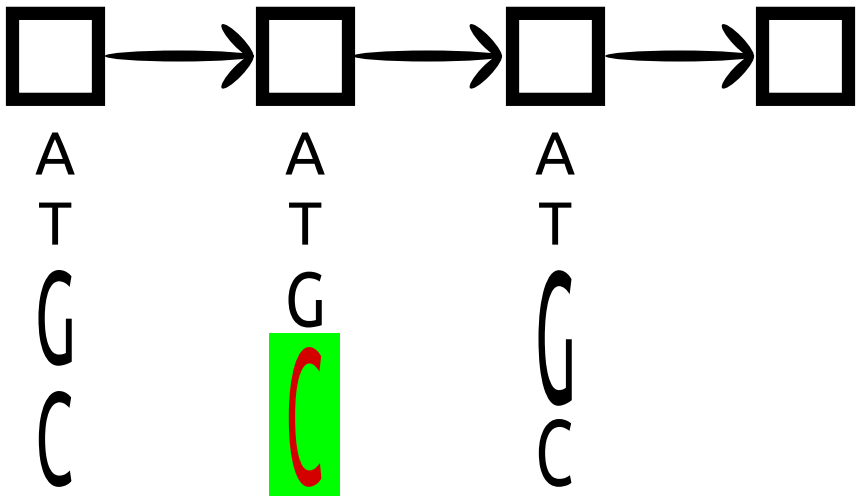
1^{st} order Markov Model



1^{st} order Markov Model



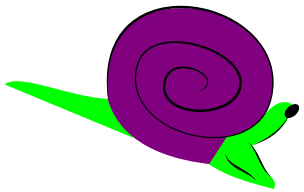
1^{st} order Markov Model

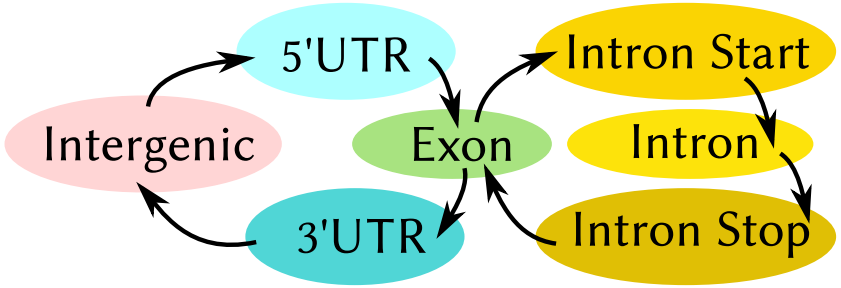


What are Markov Models good for?

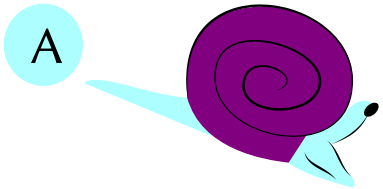
- Background sequence composition
- Spam

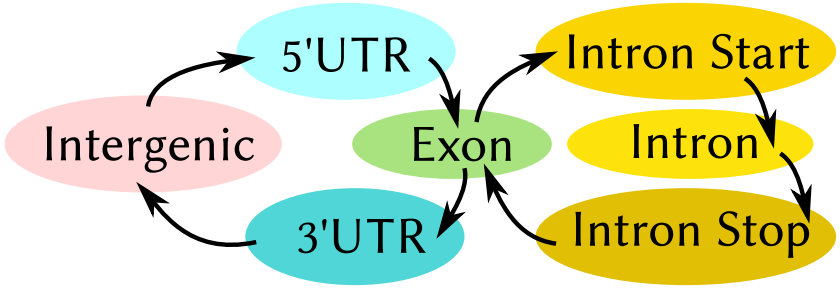
Hidden Markov Models



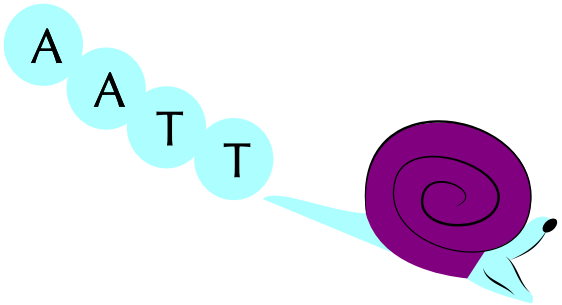


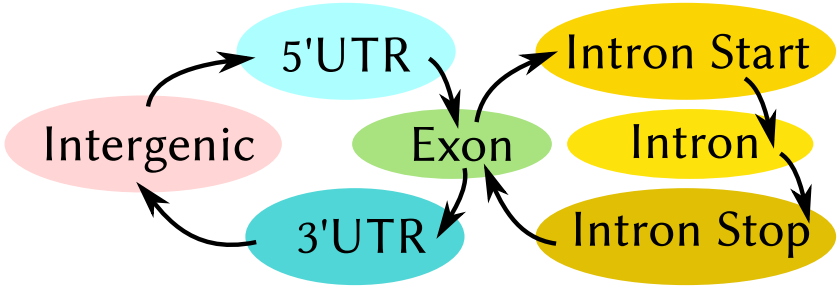
Hidden Markov Models



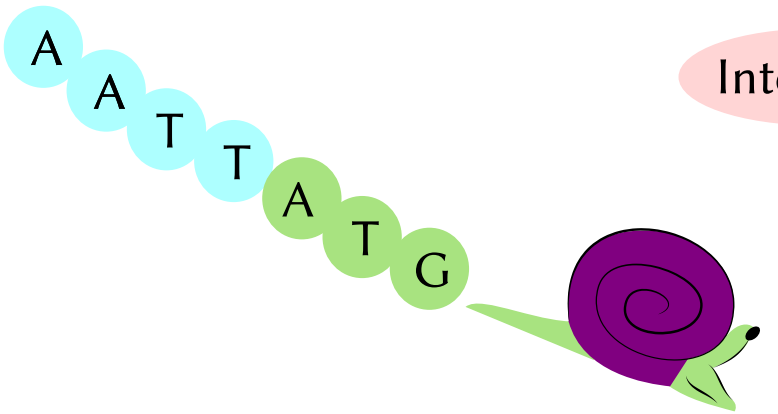


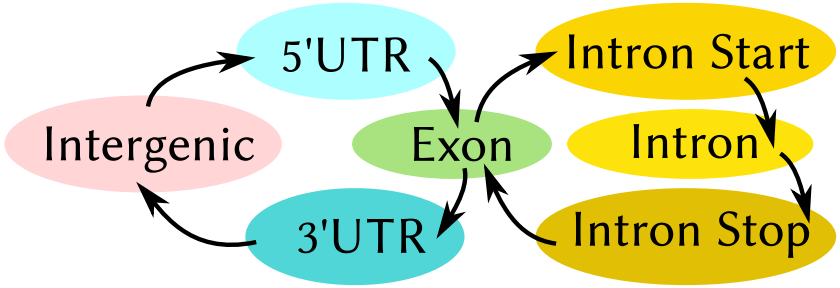
Hidden Markov Models



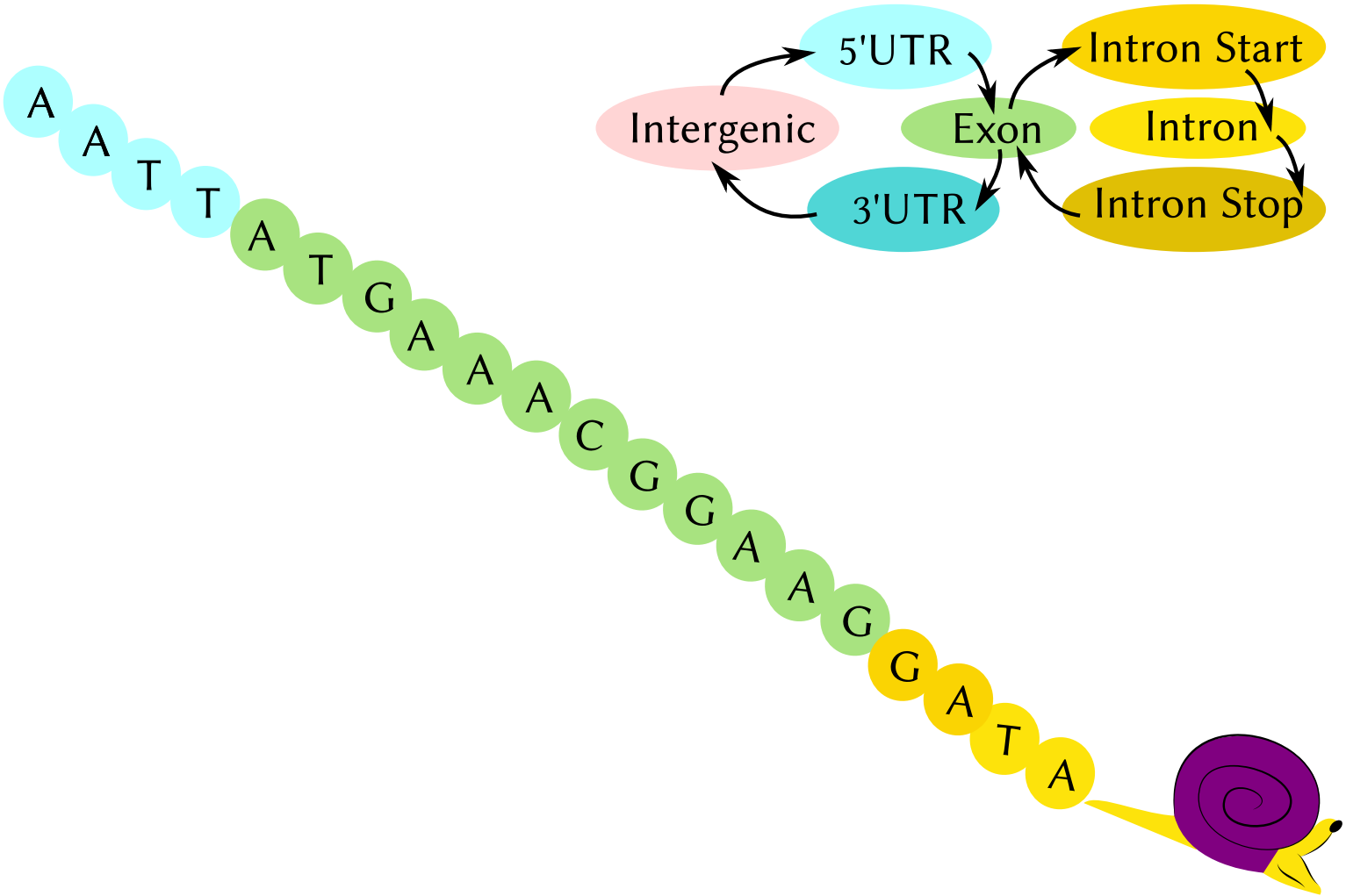


Hidden Markov Models

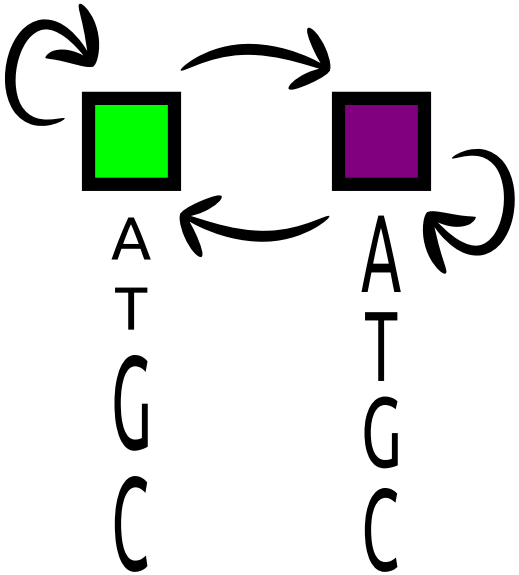




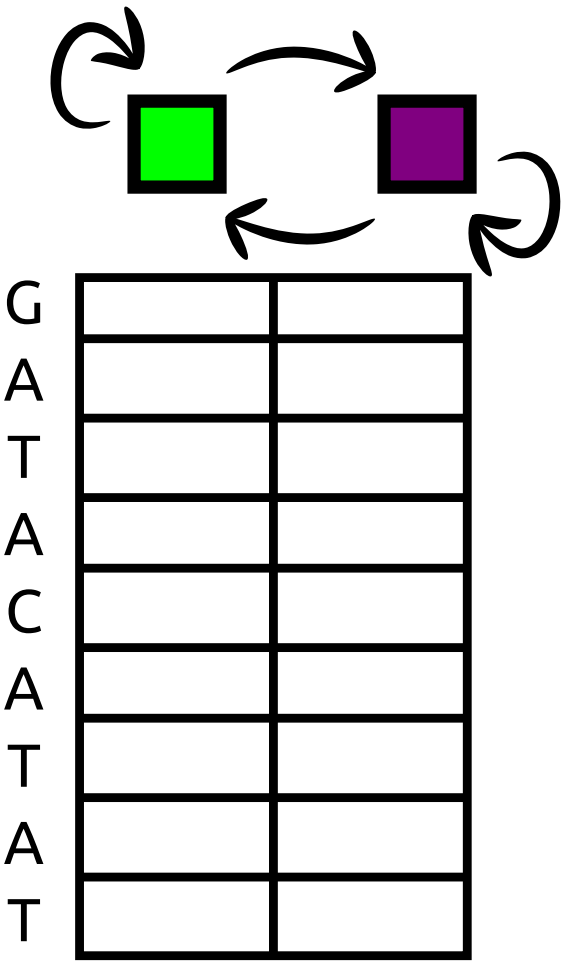
Hidden Markov Models



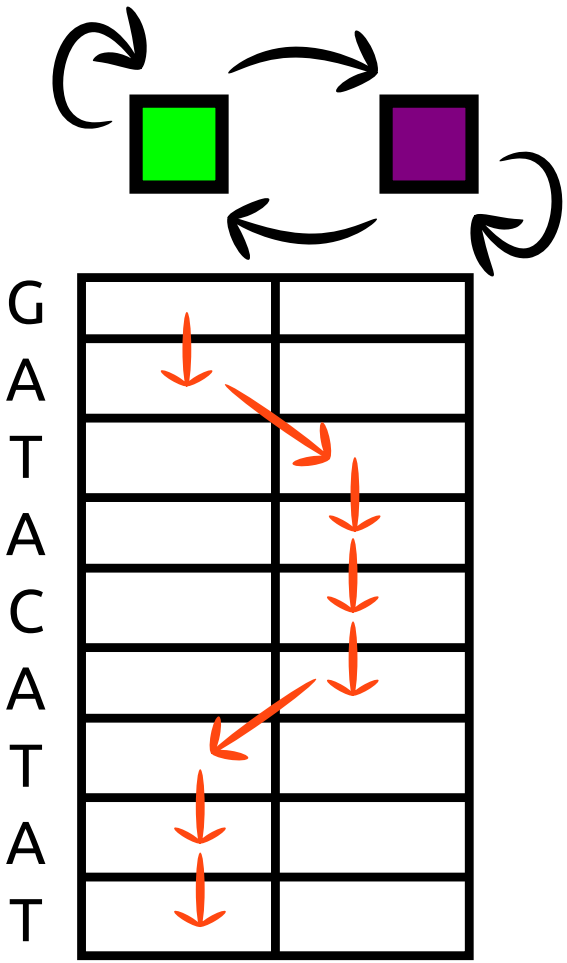
Hidden Markov Model



The Viterbi algorithm: Alignment

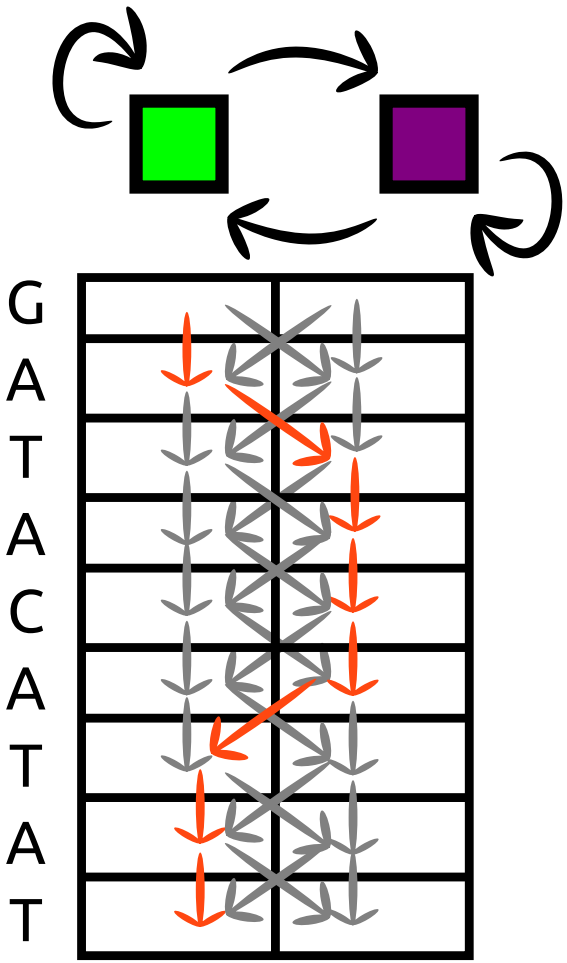


The Viterbi algorithm: Alignment



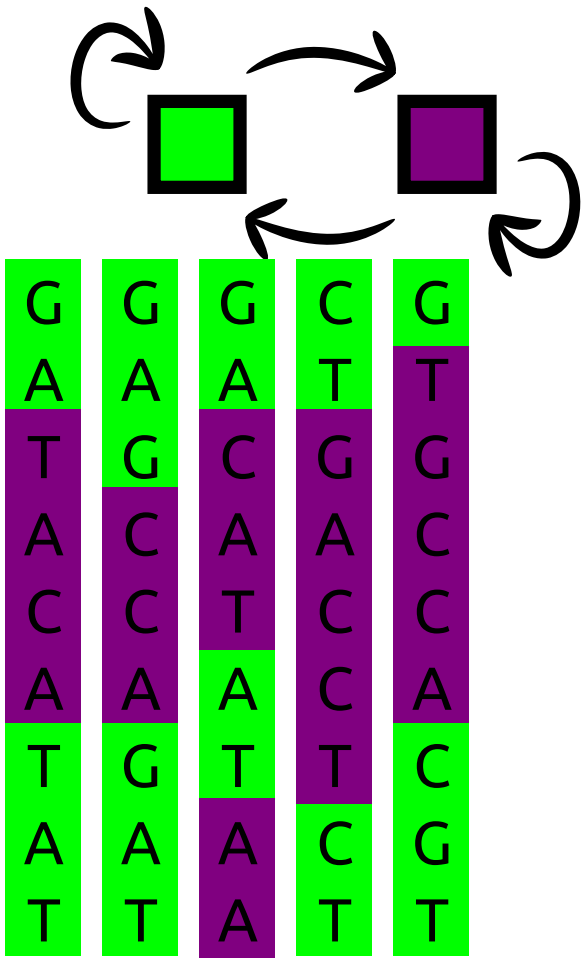
- Dynamic programming, like Smith-Waterman
- Sums *best* log probabilities of emissions and transitions (*i.e.*, multiplying independent probabilities)
- Result is most likely annotation of the target with hidden states

The Forward algorithm: Net probability



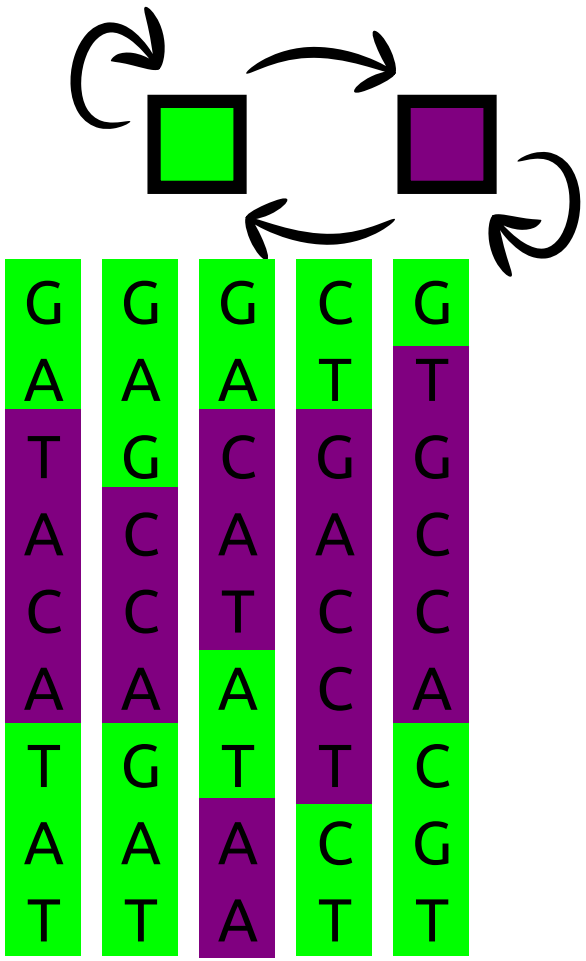
- Probability-weighted sum over all possible paths
- Simple modification of Viterbi (although *summing* probabilities means we have to be more careful about rounding error)
- Result is the probability that the observed sequence is explained by the model
- In practice, this probability is compared to that of a null model (*e.g.*, random genomic sequence)

Training an HMM



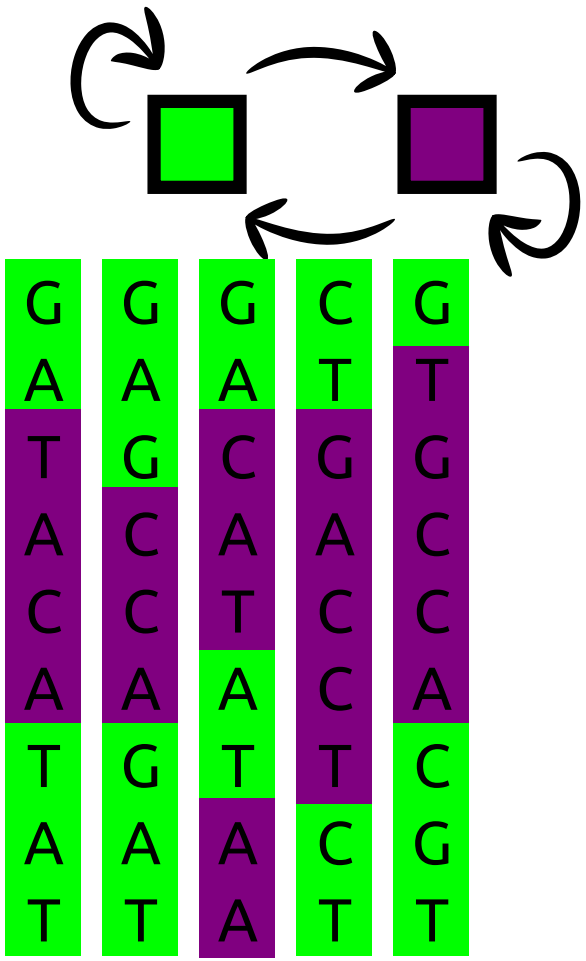
- If we have a set of sequences with known hidden states (*e.g.*, from experiment), then we can calculate the emission and transition probabilities directly

Training an HMM



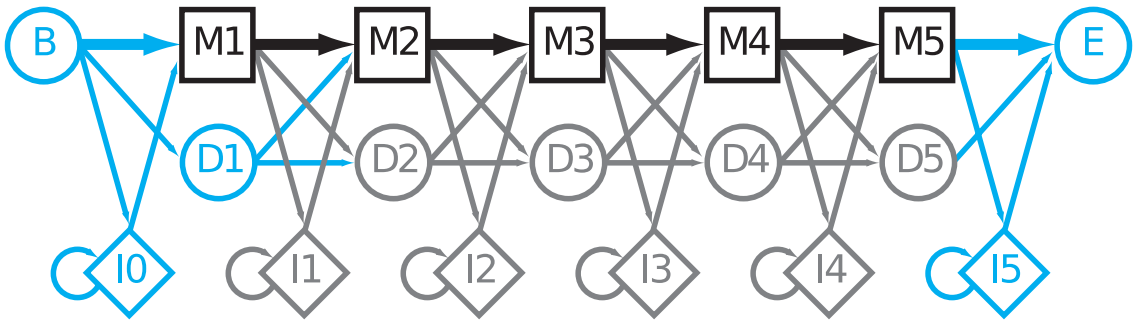
- If we have a set of sequences with known hidden states (*e.g.*, from experiment), then we can calculate the emission and transition probabilities directly
- Otherwise, they can be iteratively fit to a set of unlabeled sequences that are known to be true matches to the model

Training an HMM



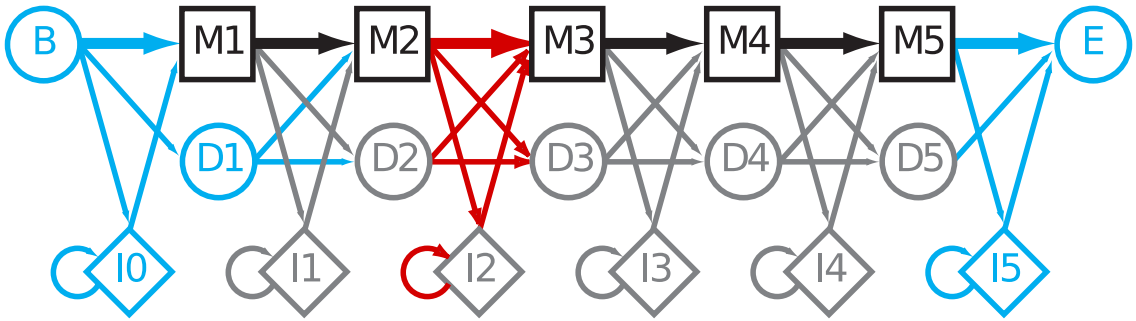
- If we have a set of sequences with known hidden states (*e.g.*, from experiment), then we can calculate the emission and transition probabilities directly
- Otherwise, they can be iteratively fit to a set of unlabeled sequences that are known to be true matches to the model
- The most common fitting procedure is the Baum-Welch algorithm, a special case of expectation maximization (EM)

Profile Alignments: Plan 7



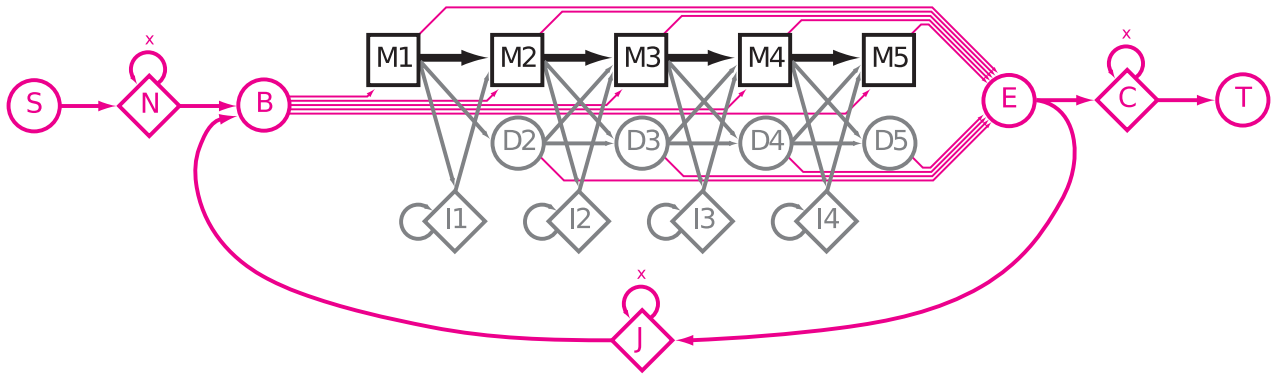
(Image from Sean Eddy, PLoS Comp. Biol. 4:e1000069)

Profile Alignments: Plan 7 (from Outer Space)



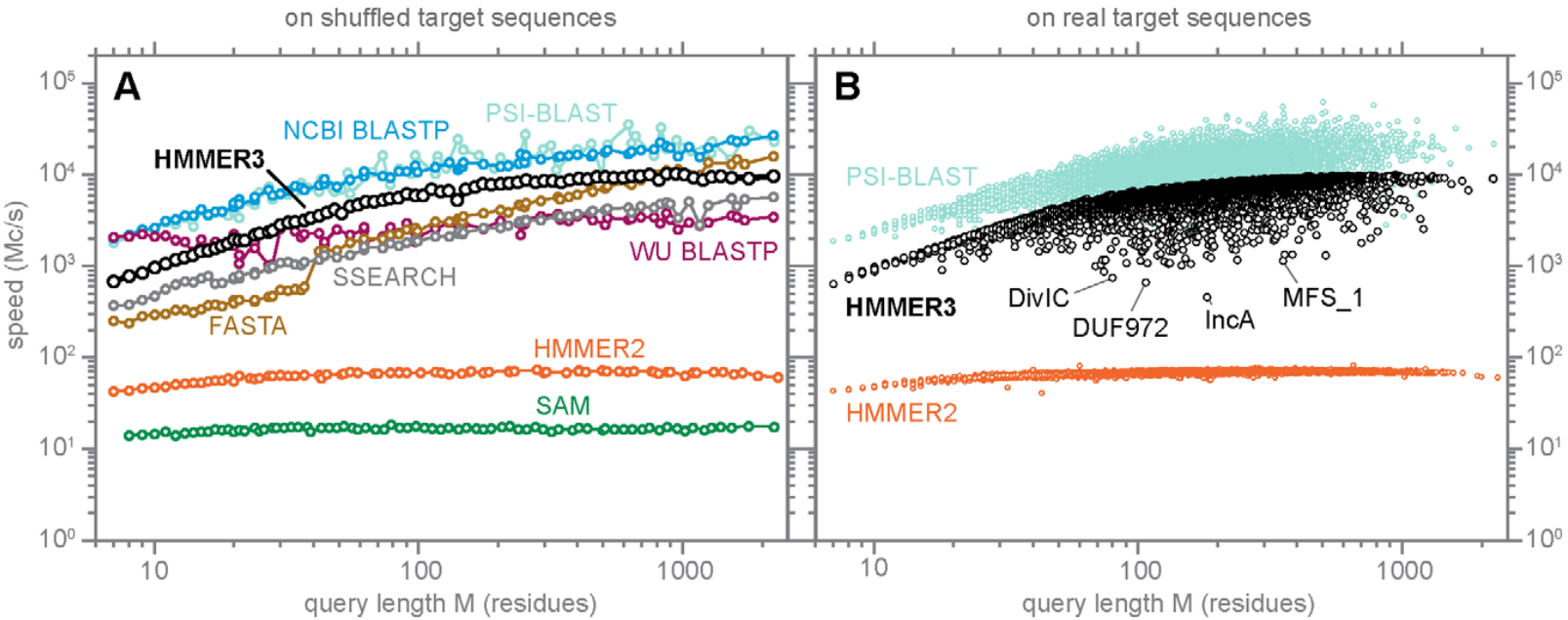
(Image from Sean Eddy, PLoS Comp. Biol. 4:e1000069)

Rigging Plan 7 for Multi-Hit Alignment



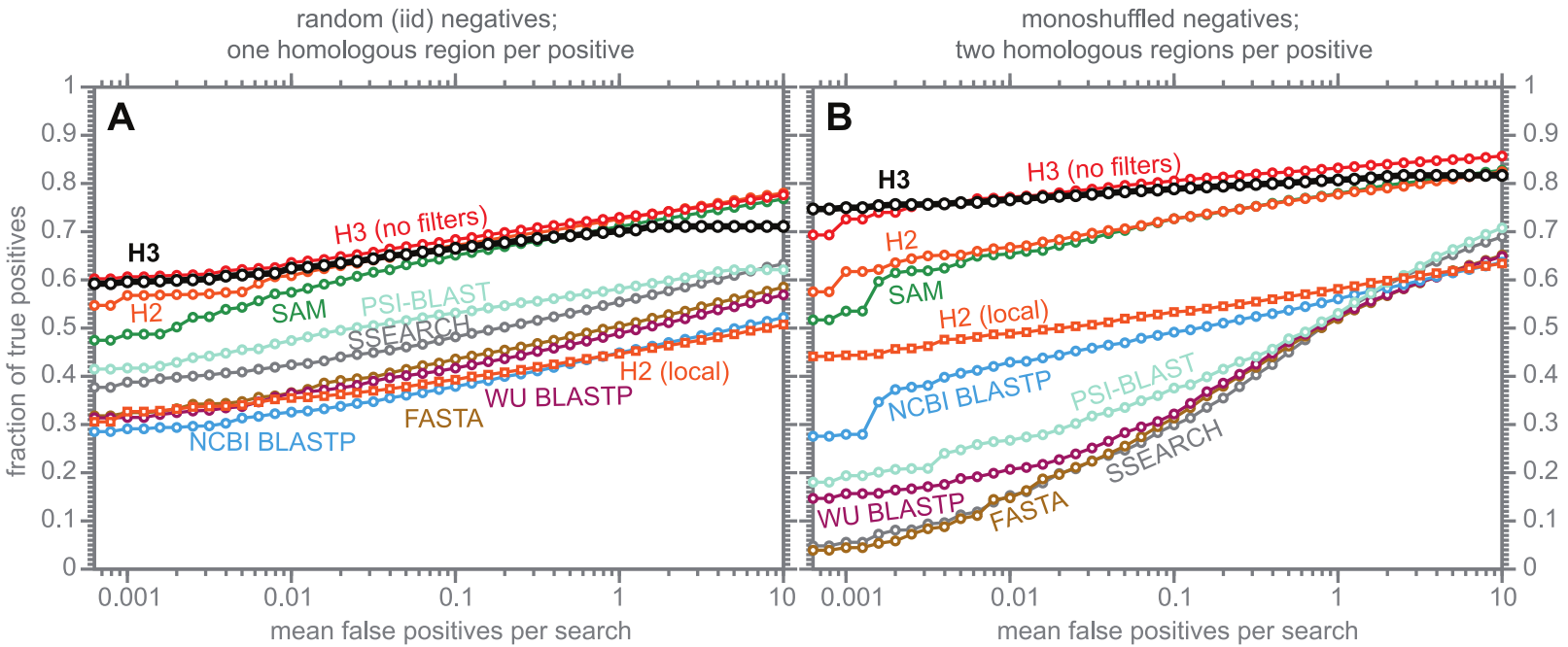
(Image from Sean Eddy, PLoS Comp. Biol. 4:e1000069)

HMMer3 speeds



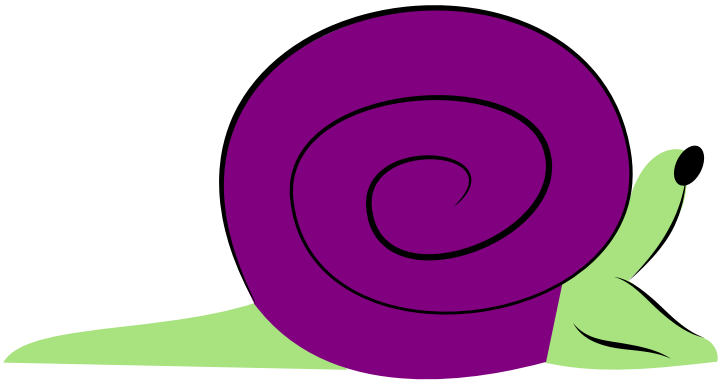
Eddy, PLoS Comp. Biol. 7:e1002195

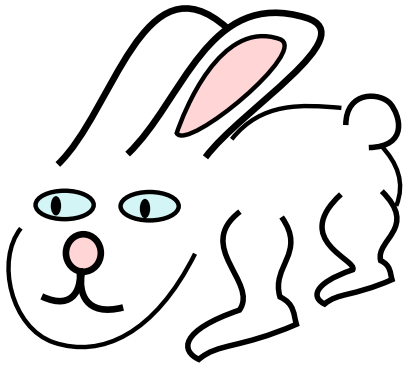
HMMer3 sensitivity and specificity



Eddy, PLoS Comp. Biol. 7:e1002195

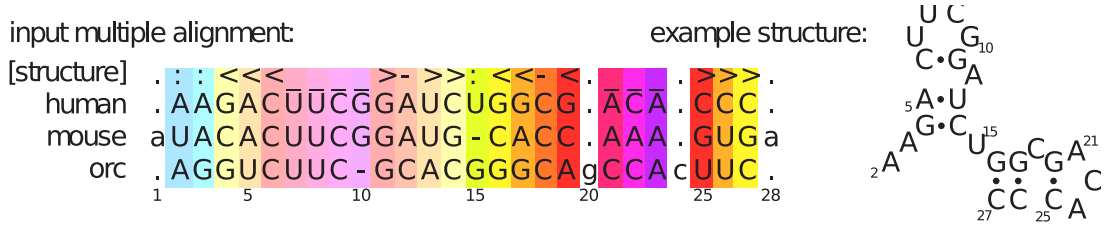
Stochastic Context Free Grammars

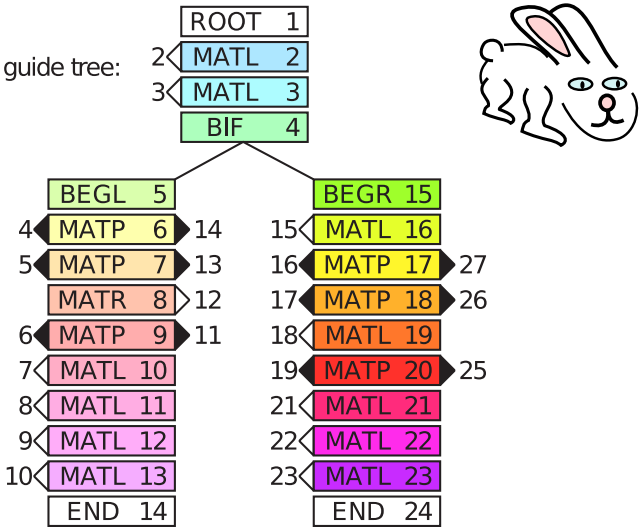




- Can emit from both sides $\rightarrow$ base pairs
- Can duplicate emitter $\rightarrow$ bifurcations

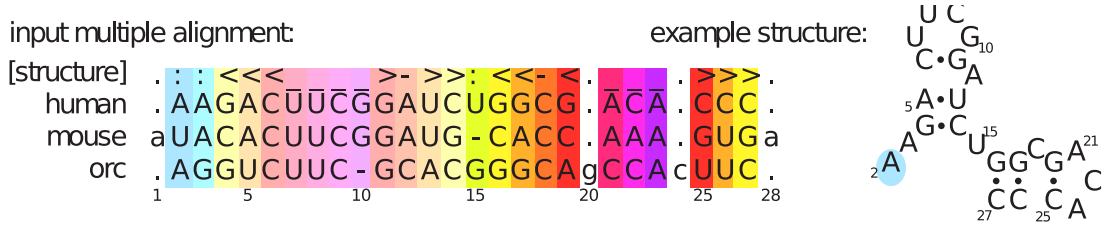
INFERNAL/Rfam

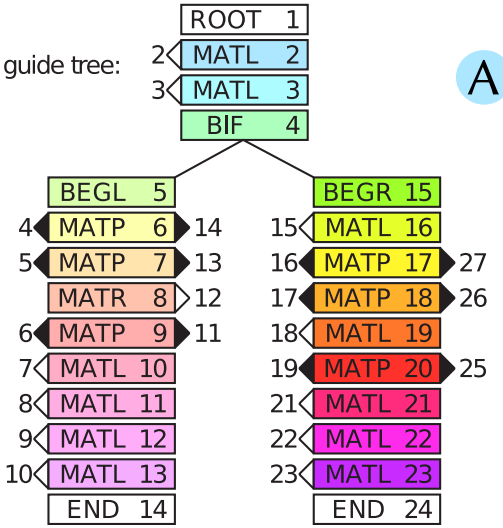


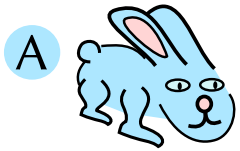


Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

INFERNAL/Rfam

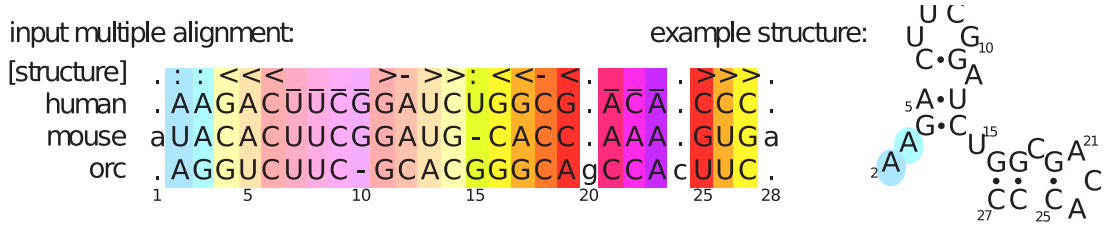


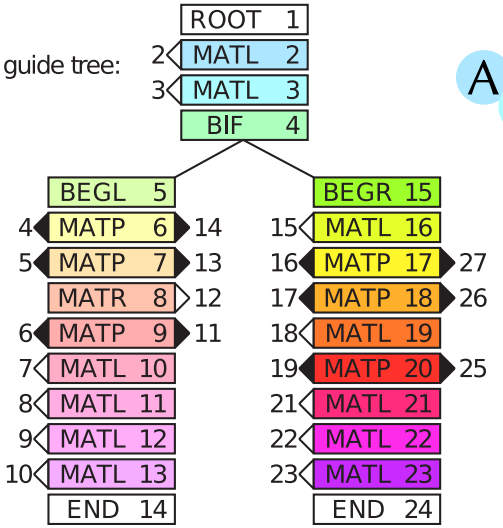




Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

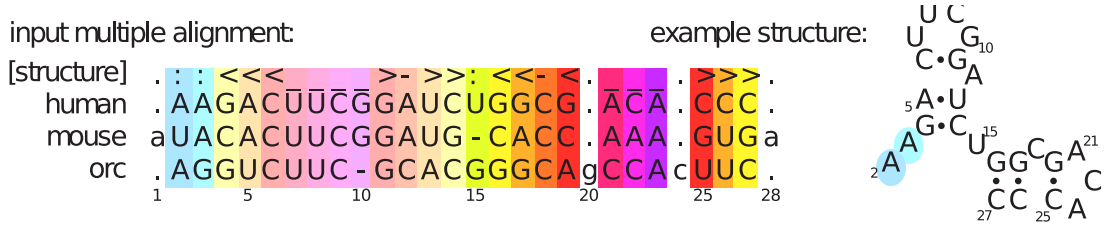
INFERNAL/Rfam

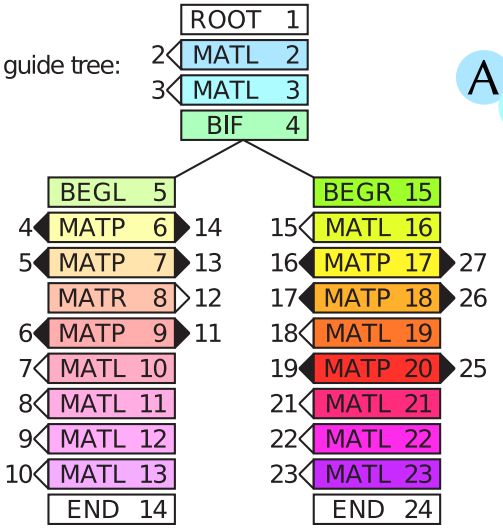


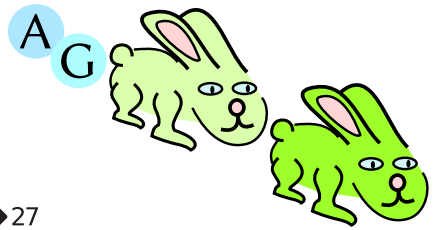


Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

INFERNAL/Rfam

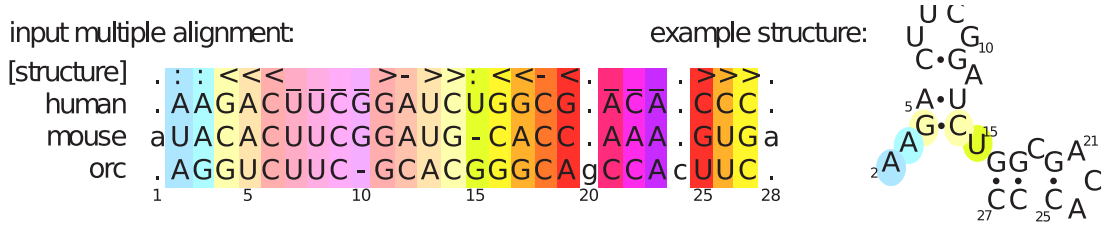


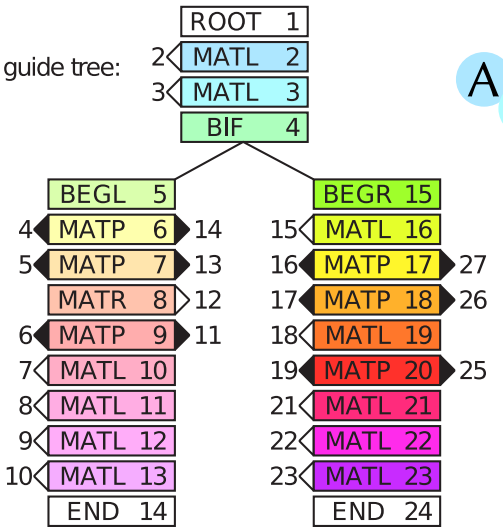


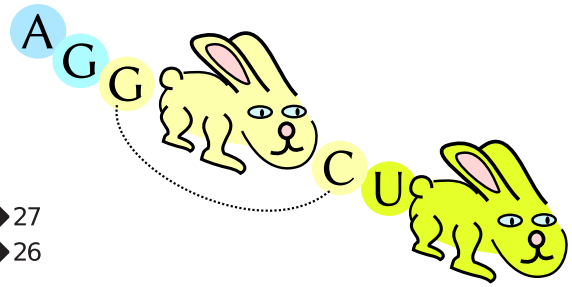


Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

INFERNAL/Rfam

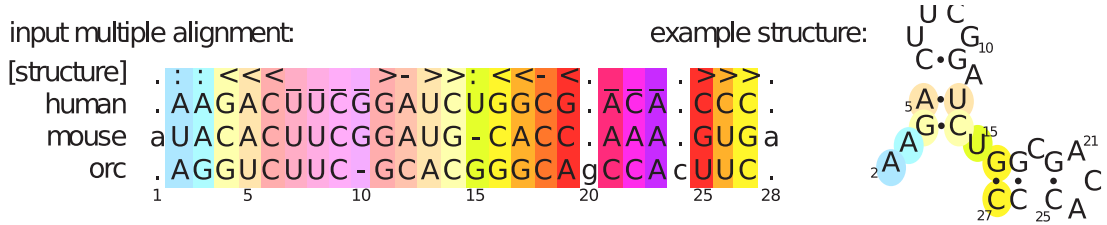


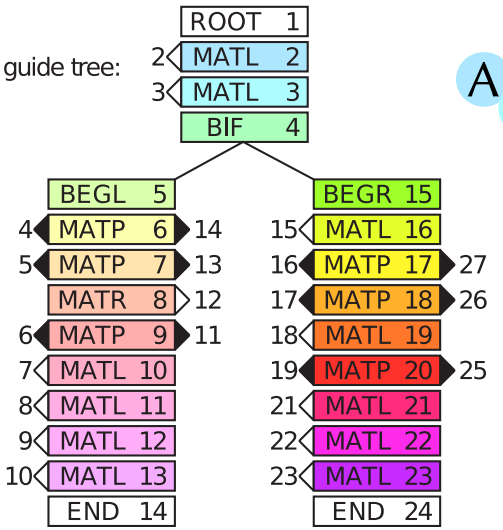


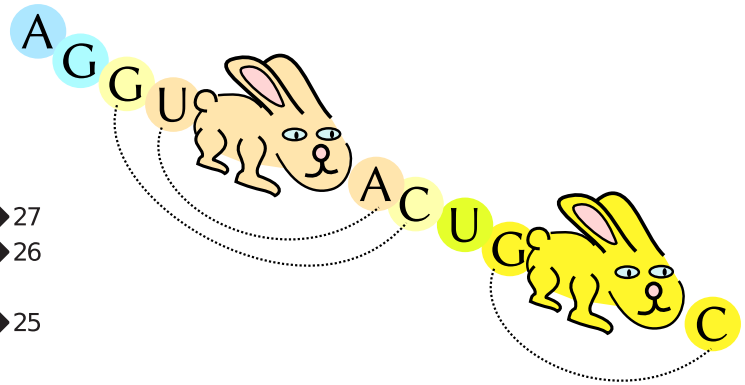


Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

INFERNAL/Rfam

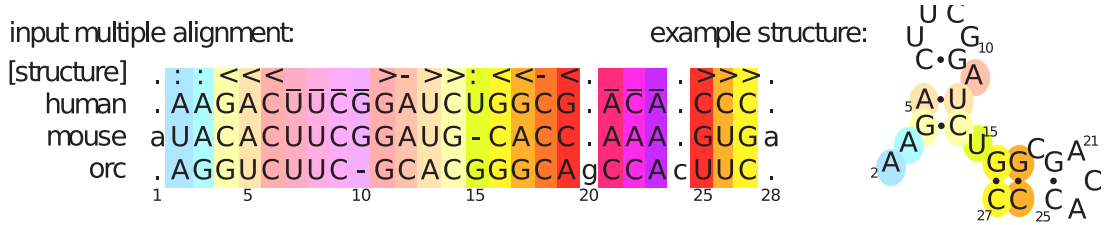


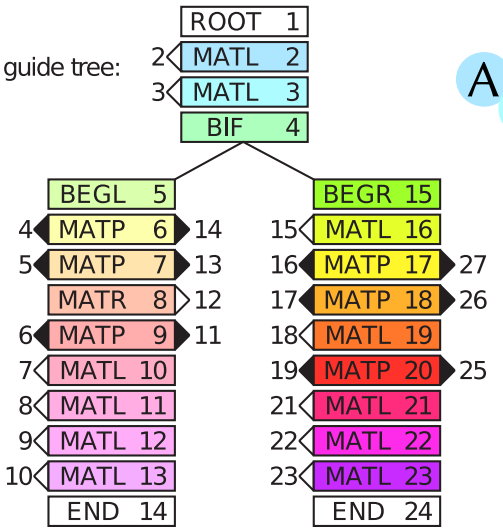


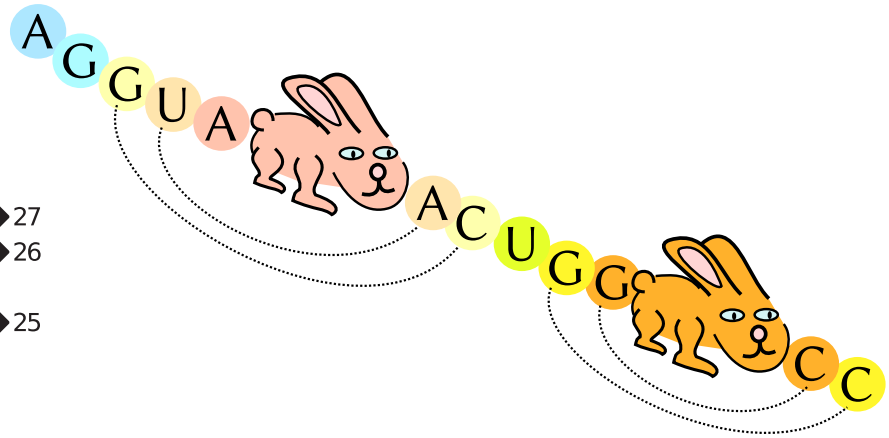


Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

INFERNAL/Rfam







Modified from the INFERNAL User Guide – Nawrocki, Kolbe, and Eddy

Homework

- Download CLUSTALX and JalView
- Keep working on your dynamic programming code.