Practical Bioinformatics

Mark Voorhies

5/4/2011

Writing stand-alone scripts

```
#!/usr/bin/env python
"""Example command line program."""
import sys
if(__name__ == "__main__"):
    print "Hello, world"
    for arg in sys.argv:
        sys.stderr.write(arg+"\n")
    for line in sys.stdin:
        sys.stdout.write(line)
```

Scripting Cluster

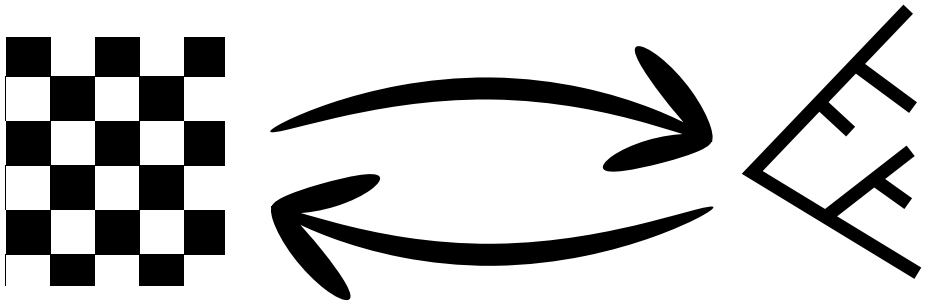
```
from subprocess import check_call
check_call(
    # Which program to run
    ("cluster",
     # Input file
     " f","supp2data.tdt",
     # Output prefix
     " u","supp2data.Uncentered.Complete",
     ##
```

Scripting Cluster

```
metrics = ("None",
           "Uncentered",
           "Pearson",
           "UncenteredAbs",
           "PearsonAbs",
           "Spearman",
           "Kendall",
           "Euclidean",
           "City")
linkage = (("Complete","m"),
           ("Single","s"),
           ("Centroid","c"),
           ("Average","a"))

# Loop over all 32 possible methods
print "Starting hierarchical clustering runs..."
from subprocess import check_call
for metric in xrange(1,len(metrics)):
    print "   ",metrics[metric],"..."
    for (linkname, link) in linkage:
        print "      ",linkname
        check_call(("cluster"," f","shuffled.txt",
                    " u",".".join(("shuffled",
                                   metrics[metric],
                                   linkname)),
                   " m",link," g",str(metric)))
```

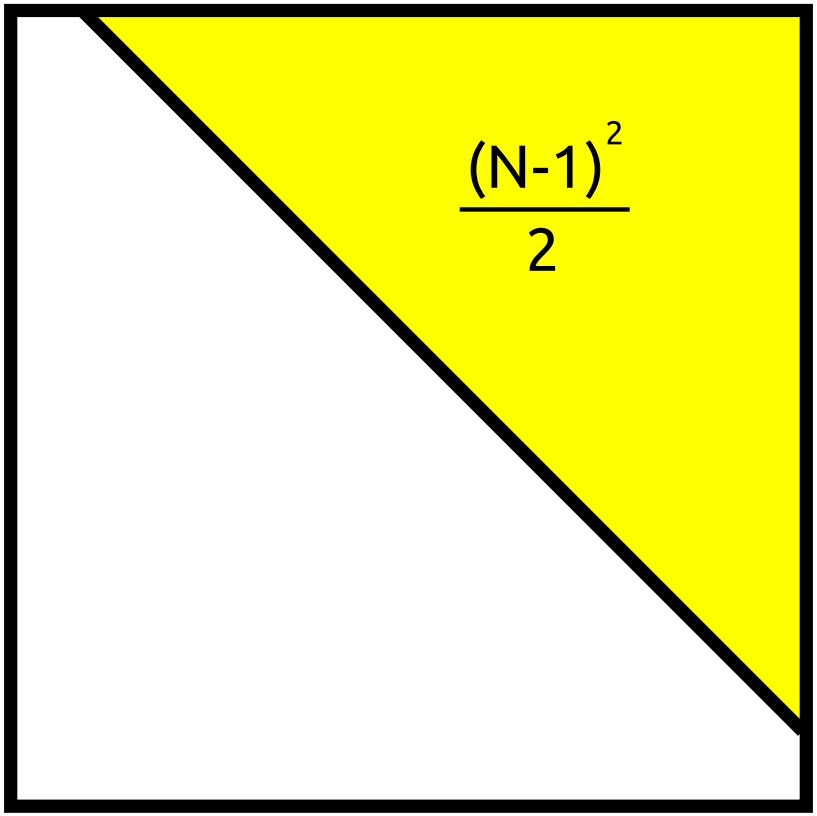
Evolution implies a self-consistent model



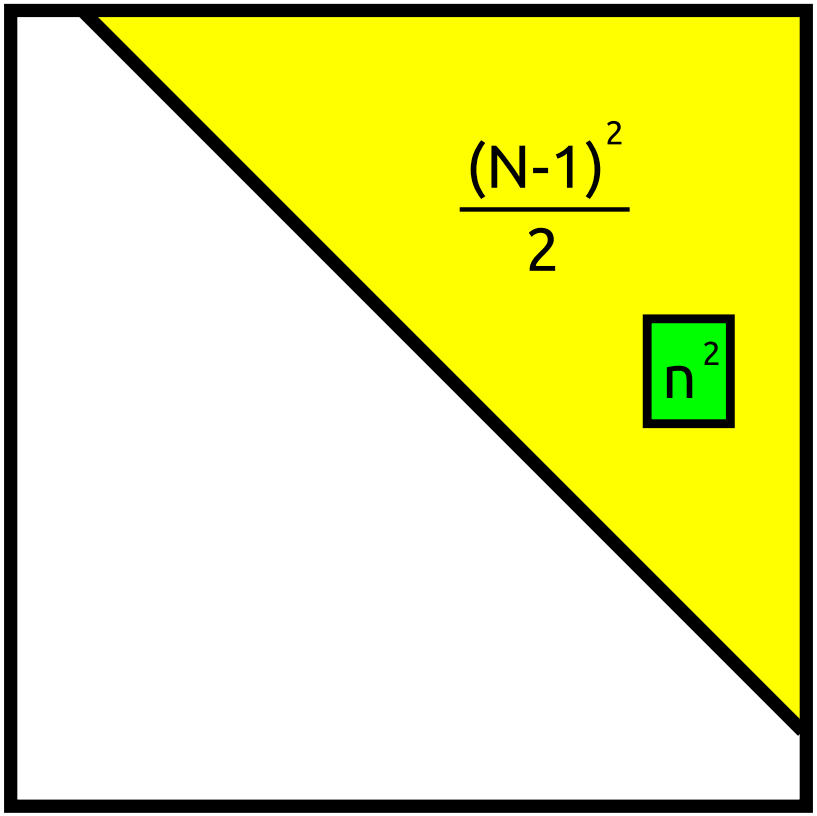
Distances
(Pairwise relationships)

Topology
(Evolutionary history)

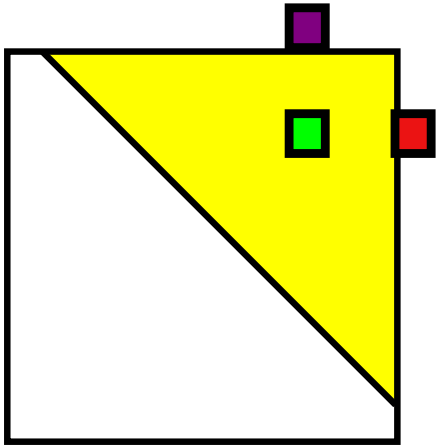
Measure all pairwise distances by dynamic programming

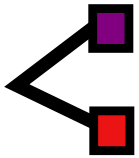


Measure all pairwise distances by dynamic programming

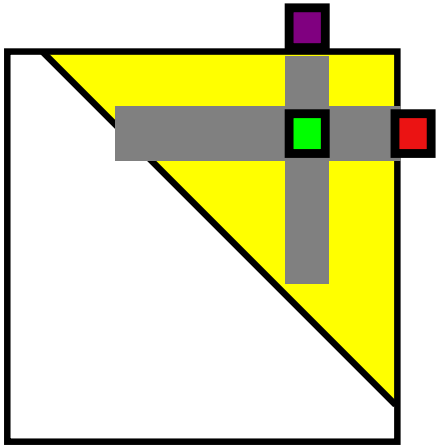


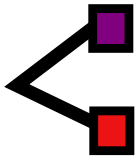
Generate a guide tree by UPGMA



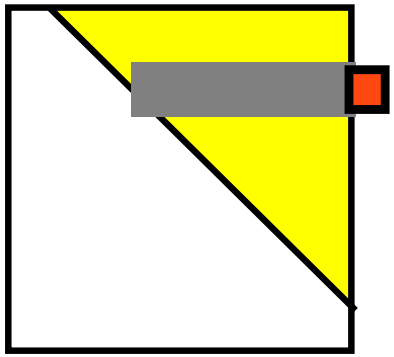


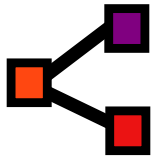
Generate a guide tree by UPGMA



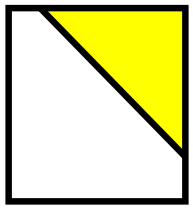


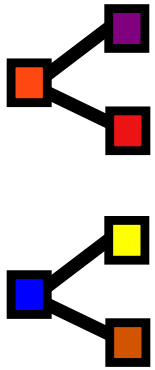
Generate a guide tree by UPGMA



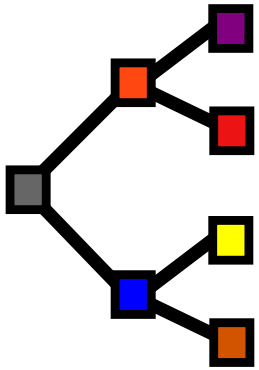


Generate a guide tree by UPGMA

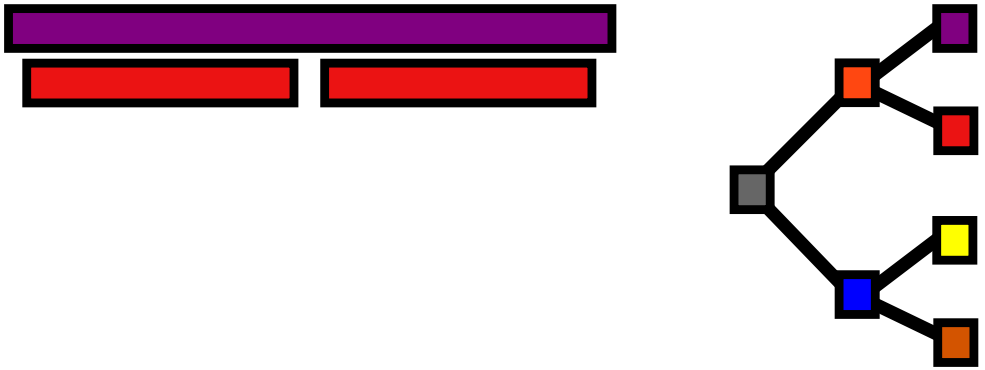




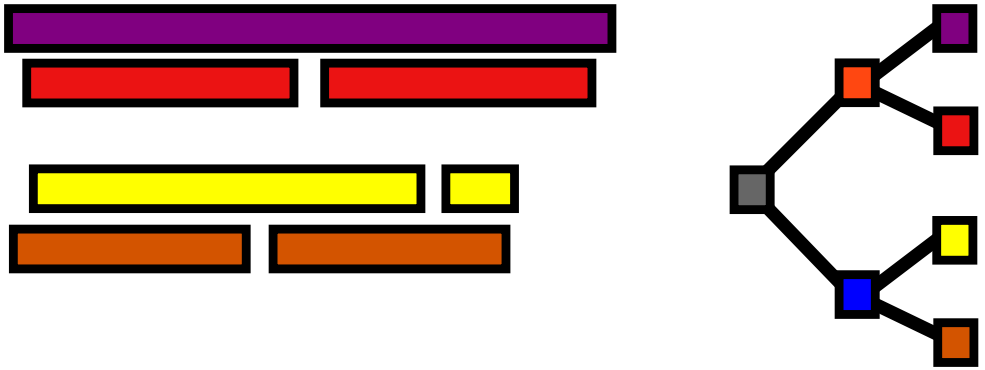
Generate a guide tree by UPGMA



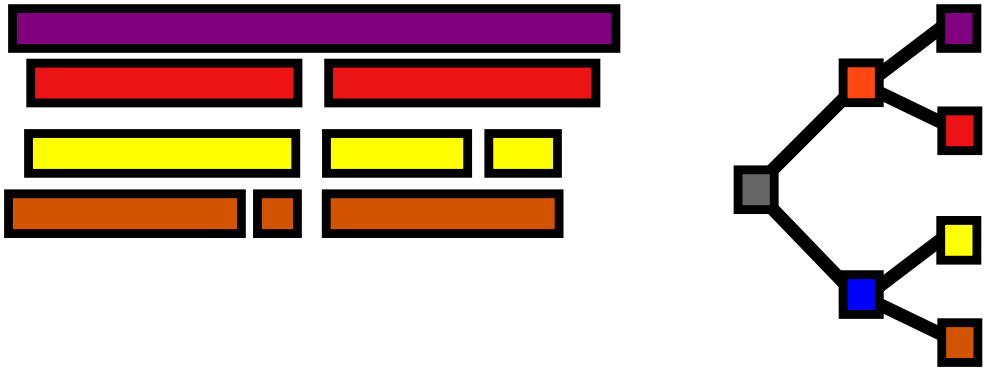
Progressive alignment following the guide tree



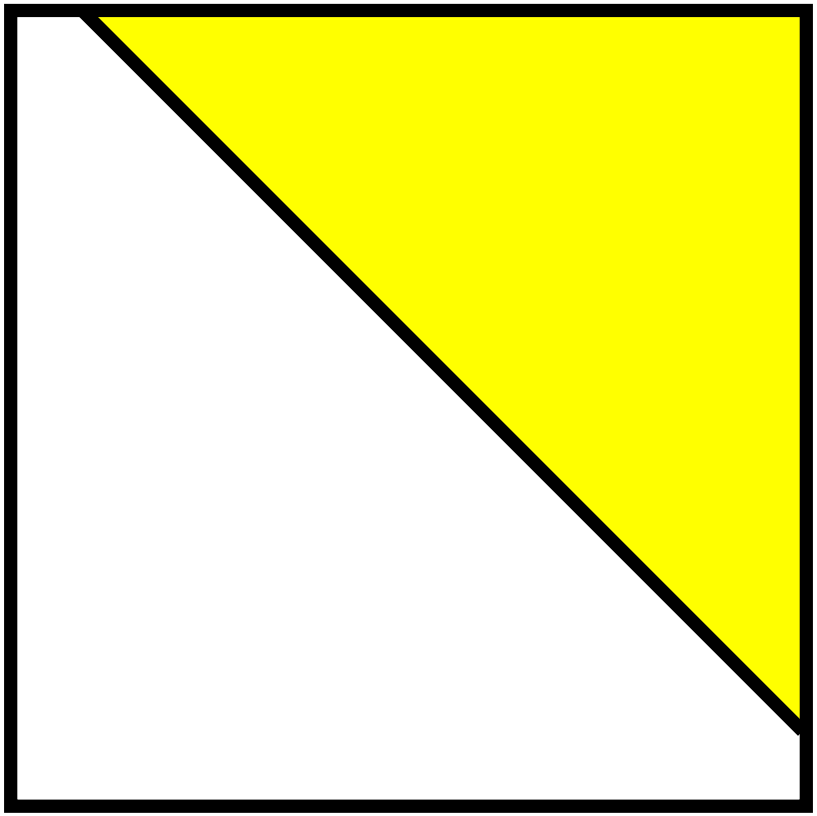
Progressive alignment following the guide tree



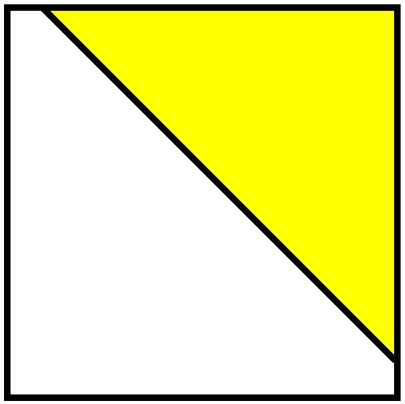
Progressive alignment following the guide tree

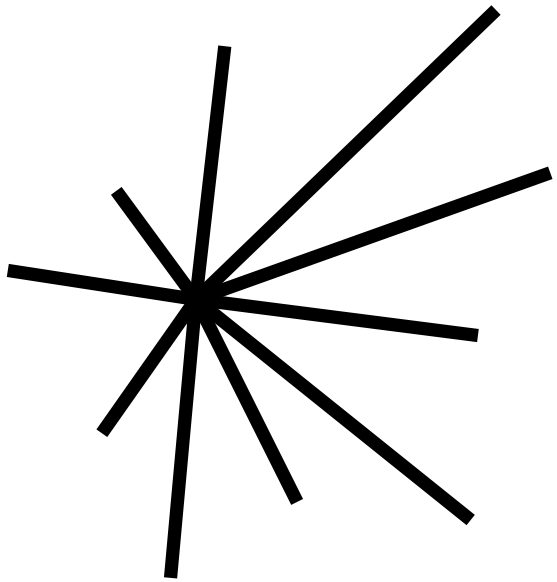


Measure distances directly from the alignment

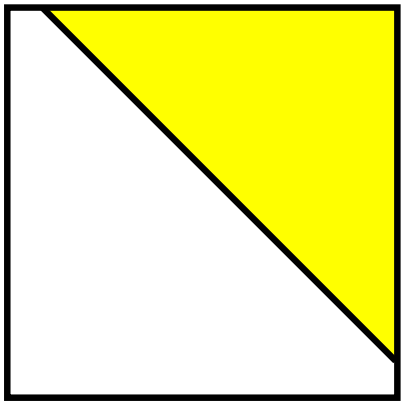


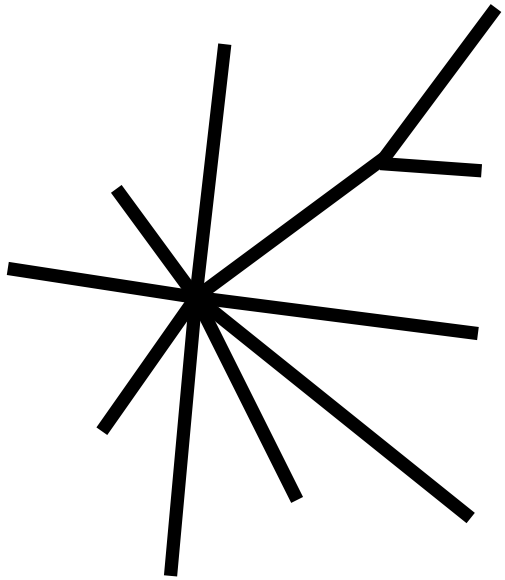
Generate neighbor-joining tree from new distances



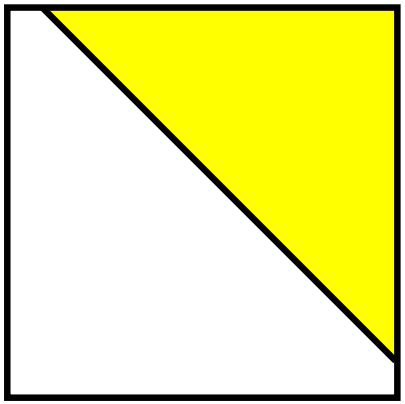


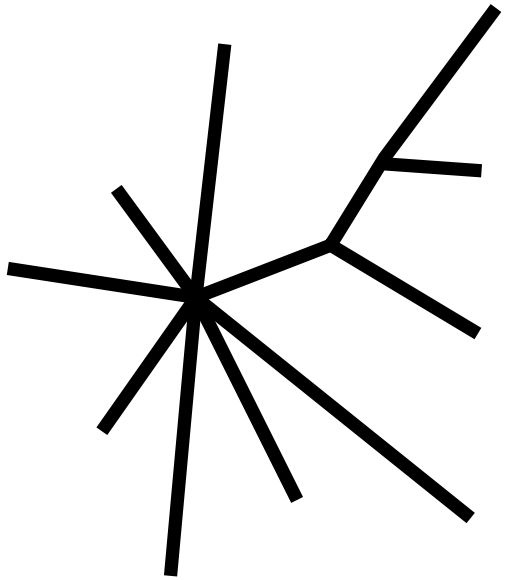
Generate neighbor-joining tree from new distances



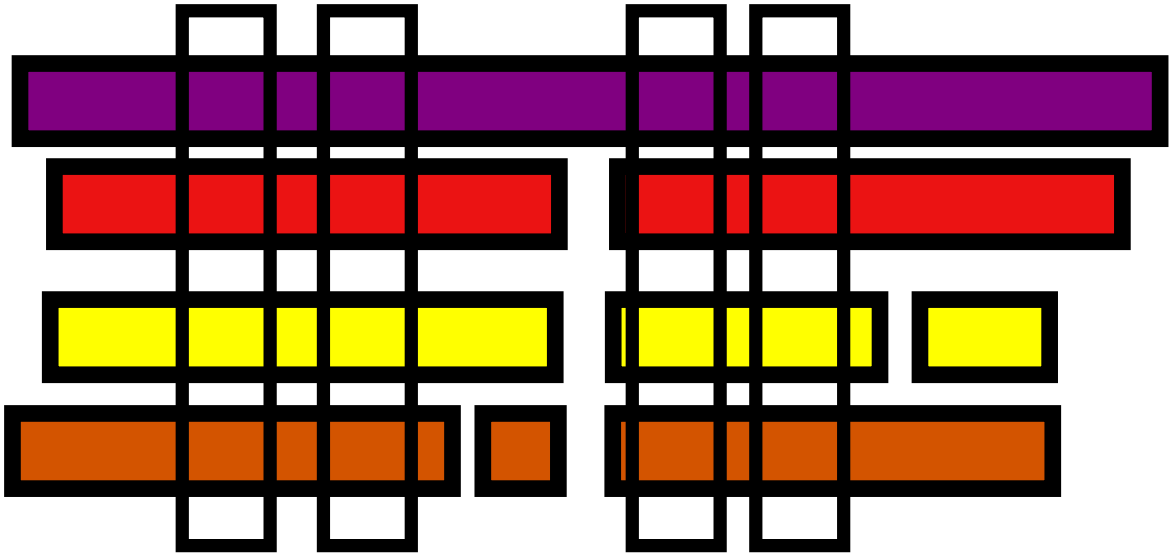


Generate neighbor-joining tree from new distances



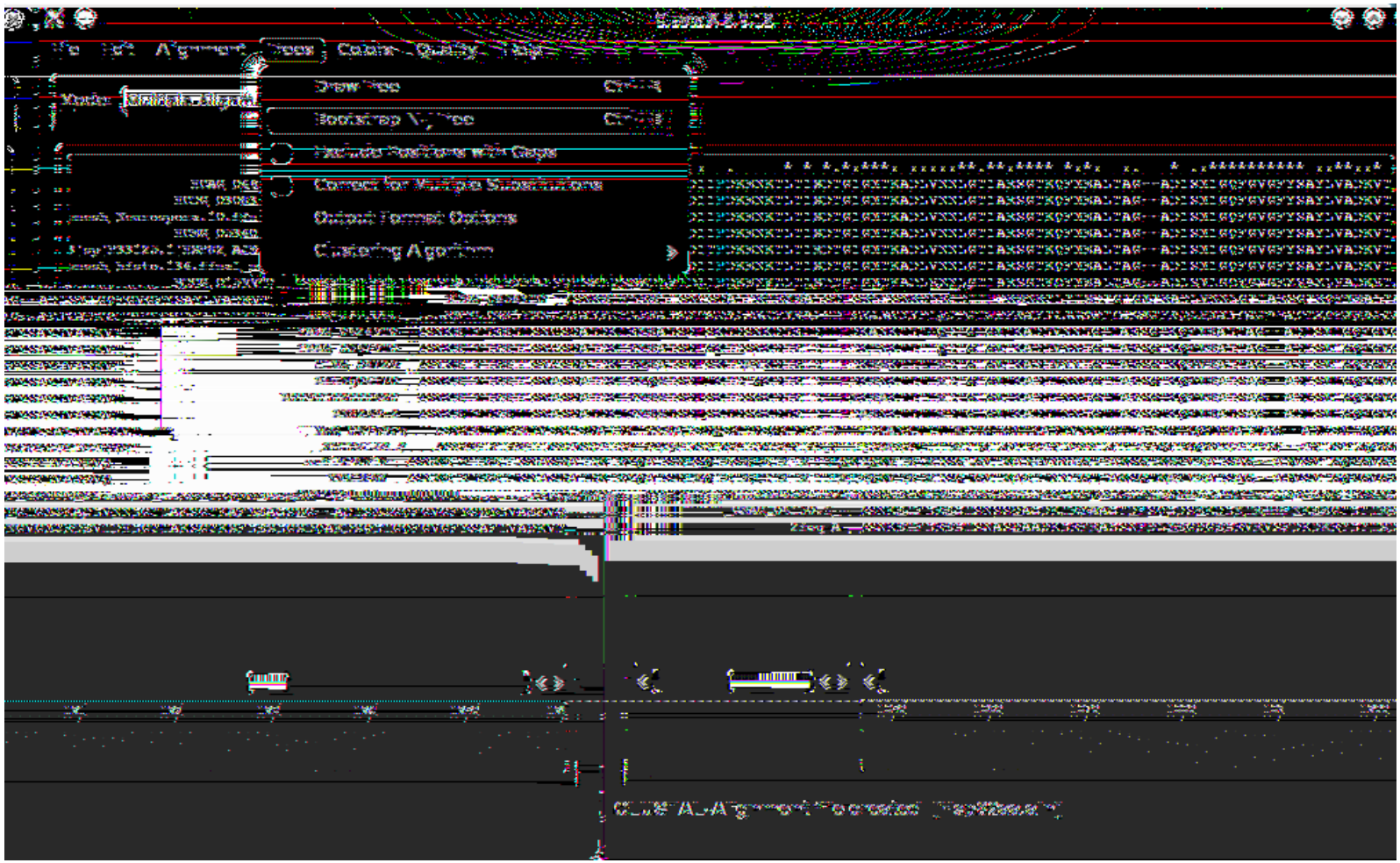


Generate bootstrap values from subsets of the alignment

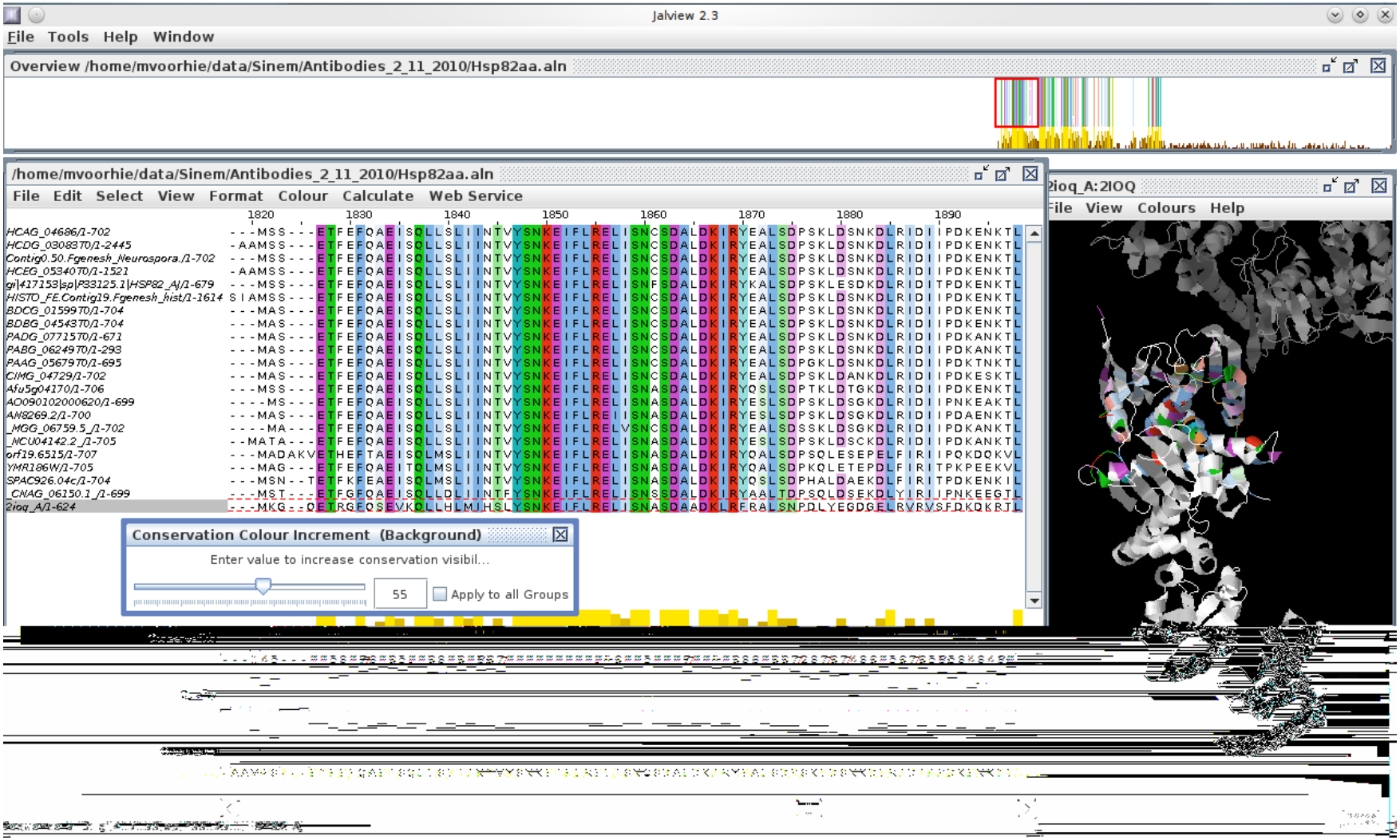


Generating a multiple alignment in CLUSTALX

Generating a neighbor joining tree in CLUSTALX



Viewing the alignment and tree in JALVIEW



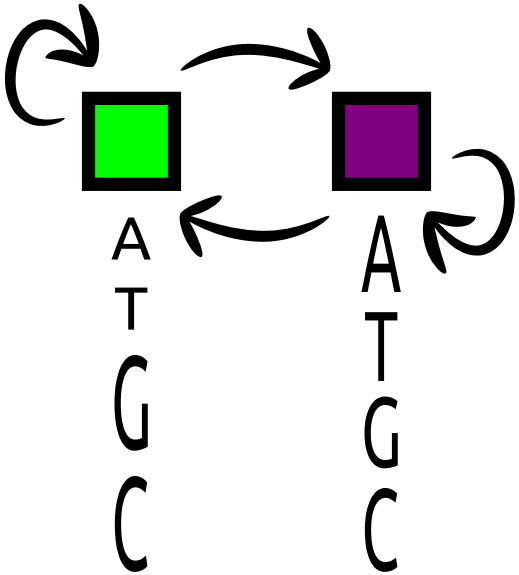
Related tools

- Multiple Alignment
  - T-Coffee
  - MUSCLE
  - COBALT
- Tree building
  - MrBayes (Bayesian MCMC)
  - PhyML (maximum likelihood)

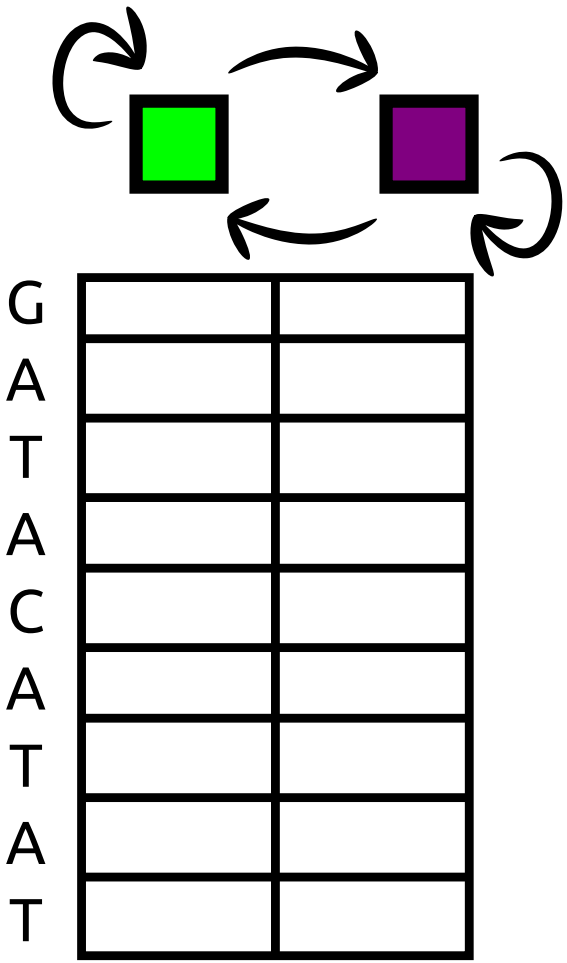
Scripting CLUSTALW

```
from subprocess import check_call
check_call(
    #  Run the clustalw program
    ("clustalw",
     # Read input from sequencefile
     # (e.g., FASTA file)
     " infile=%s" % sequencefile,
     # Specify that input is protein sequences
     " type=PROTEIN",
     # Verb 1: do full multiple alignment
     # (generates DND guide tree and ALN alignment)
     " align",
     # Verb 2: Bootstrap a neighbor joining tree
     #  with the default number of bootstraps (1000)
     " bootstrap"))
```

Hidden Markov Model



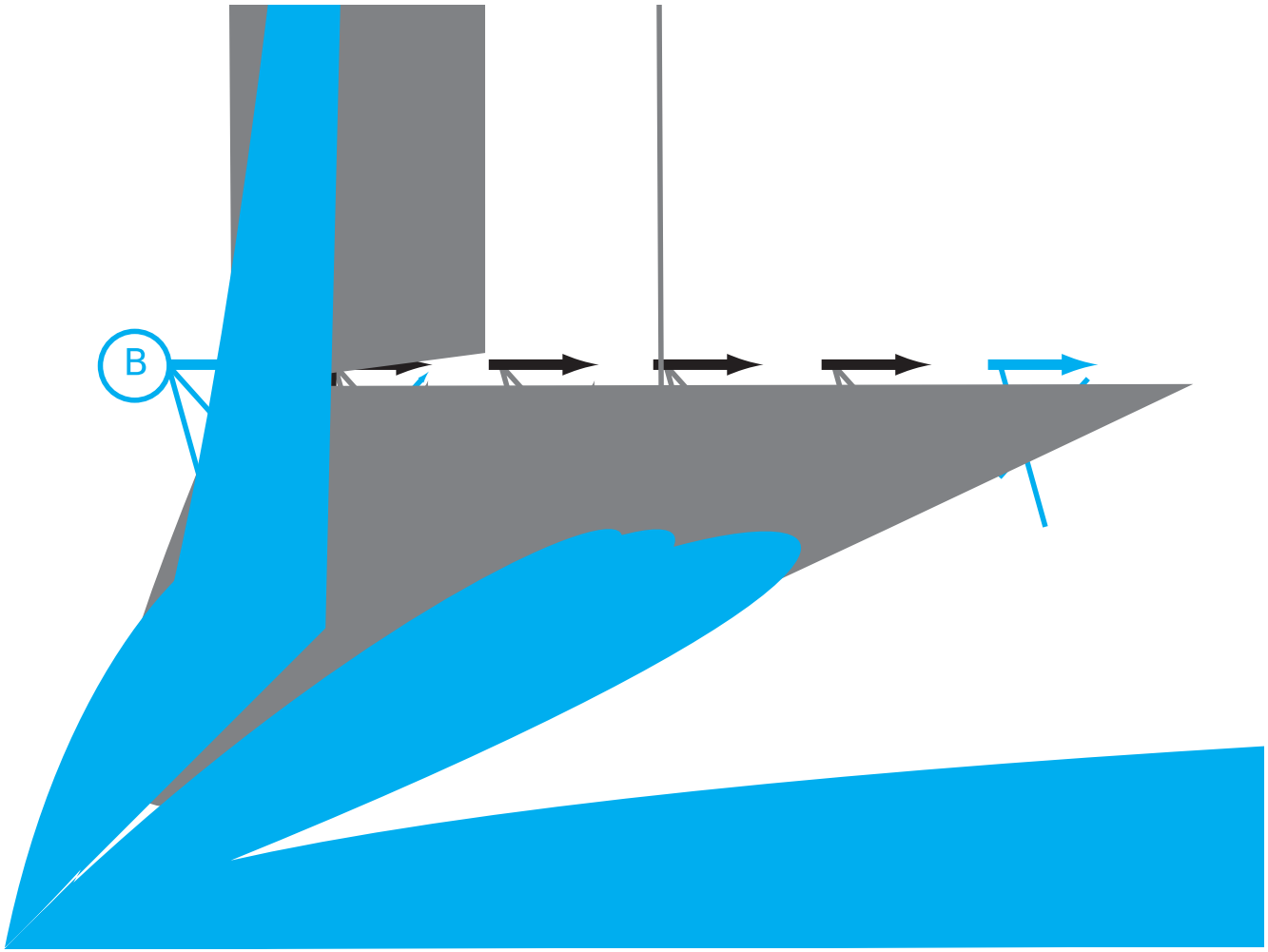
The Viterbi algorithm: Alignment



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

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)

Combining CLUSTALW with HMMer

```
# Align sequences to HMM using HMMer
check_call(("hmmalign", " o", hmmalign,
            "  informat", "fasta",
            hmm, sequences))

# convert alignment from Stockholm to CLUSTAL format
# and clip to just the aligned subsequences
from ClustalTools import MultipleAlignment
hmmaln = ".".join((jobname,"aln"))
alignment = MultipleAlignment.fromStockholm(open(hmmalign))
rf = alignment.colAnnotations["RF"]
start = rf.find("x")
assert(start >  1)
stop = rf.rfind("x")+1
motifaln = alignment[start:stop]
motifaln.writeClustal(open(hmmaln,"w"))

# Generate bootstrapped NJ tree using CLUSTALW
check_call(("clustalw",
            " infile=%s" % hmmaln,
            " type=PROTEIN",
            " bootstrap"))
```

A class for multiple alignment

```
class MultipleAlignment:
    def __init__(self, seqmatrix, seqnames = None, colAnnotations = None):
        self.seqmatrix = seqmatrix
        if(seqnames is not None):
            self.seqnames = seqnames
            assert(len(self.seqnames) == len(self.seqmatrix))
        else:
            self.seqnames = ["Seq%05d" % i for i in xrange(len(self.seqmatrix))]

        if(colAnnotations is not None):
            self.colAnnotations = colAnnotations
        else:
            self.colAnnotations = fg

        self.name2seq = dict((i,j) for (i,j) in zip(seqnames, seqmatrix))
        assert(len(self.name2seq) == len(self.seqnames))
```

Defining indexing and slicing on a MultipleAlignment

```
def __getitem__(self, i):
    """Return the sequence named "i" if i is a string
    or column vector [i] (counting from 0) if i is an integer)
    """
    if(isinstance(i, int)):
        return [row[i] for row in self.seqmatrix]
    if(isinstance(i, str)):
        return self.name2seq[i]
    raise KeyError

def __getslice__(self, i, j):
    """Return a MultipleAlignment corresponding to columns
    [i:j] (indexing from 0, j is one past the last column).
    """
    return MultipleAlignment(
        seqmatrix = [seq[i:j] for seq in self.seqmatrix],
        seqnames = self.seqnames,
        colAnnotations = dict(
        (key, seq[i:j])
        for (key,seq) in self.colAnnotations.items()))
```

From HMMer to MultipleAlignment

```
@classmethod
def fromStockholm(cls, fin, headercheck = True):
    import re
    seqnames = []
    seqs = fg
    colAnnotations = fg
    if(headercheck):
        line = fin.next()
    for line in fin:
        if(len(line.strip()) < 1):
            continue
        if(line[:2] == '//'):
            break
        if(line[0] == '#'):
            if(line[1:4] == "=GC"):
                (comment, tag, seq) = line.split()
                if(not colAnnotations.has_key(tag)):
                    colAnnotations[tag] = seq
                else:
                    colAnnotations[tag] += seq
            continue
        w = line.split()
        if(not seqs.has_key(w[0])):
            seqnames.append(w[0])
            seqs[w[0]] = re.sub('[ .]',' ',w[1])
        else:
            seqs[w[0]] += re.sub('[ .]',' ',w[1])
    n = len(seqs[seqnames[0]])
    if(len(colAnnotations) == 0):
        colAnnotations = None
    return cls(seqmatrix = [seqs[i] for i in seqnames],
               seqnames = seqnames, colAnnotations = colAnnotations)
```

From MultipleAlignment to CLUSTALW

```
def writeClustal(self, fout):
    """Write alignment in (inferred) CLUSTALX .aln format."""
    seqnames = [i[max(0,len(i) - 35):] for i in self.seqnames]
    for i in seqnames:
        assert(len(i) < 36)
    assert(len(seqnames) == len(set(seqnames)))

    fout.write('CLUSTAL X (1.83) multiple sequence alignment\n\n')
    fout.write('\n')
    step = 50
    offset = 0
    n = len(self.seqmatrix[0])
    while(offset < n):
        # spacer
        fout.write('\n')
        for (name, seq) in zip(seqnames, self.seqmatrix):
            fout.write('%-35s %s\n' % (name, seq[offset:offset+step]))
        offset += step
        # conservation line
        fout.write(' ' * 36 + '\n')
```