A Exploration of Phylogentic Inference Algorithms and Distance Methods

Content

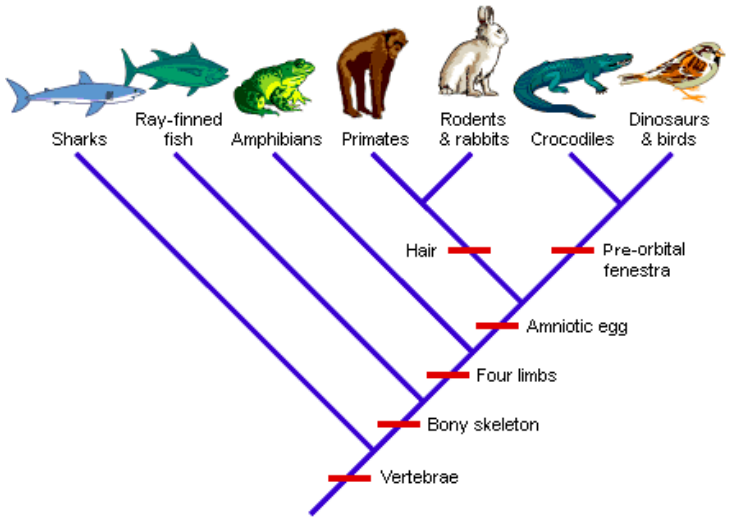
What is phylogenetics?

Introduction to the problem

Phylogentics is a study of the evolutionary relationships between living organims. We are able to do this, because we assume there is some underlying evolutionary tree. In essence, it tries to determine whether you're more closely related to one group (e.g. amphibians) or another (e.g. rodents) and by how much. As reference, one group or leaf of the tree is called a taxon. To give some history of this field, in the time period around the 1820s, phylogenetic trees were built based on observed similarities such as an organism's morphology. As you can presume, this can lead to many sources of misconceptions between two scientists. Below is an example of what a phylogenetic tree looks like.

In [1]:
```
import IPython.display
phyl = IPython.display.Image("http://medsocnet.ncsa.illinois.edu/MSSW/moodle/file
.php?file=/evolution_tree_000.PNG")
phyl
```

Out[1]:



As science and technology advanced, we encountered the revolution in the biotechnology, that is the development of the laboratory process of genome sequencing and even the discovery of the structure of DNA. DNA, *deoxyribonucleic acid*, is the genetic "blueprint" that encodes the information used to manufcture proteins and cells needed for every facet of an organism's life. Even viruses who aren't considered "living", biologically, require DNA. Within these DNA sequences are strings of the letters A, T, C, and G. These "letters" represent nucleotides that pair to one another according to their respective chemical properties. The way in which they pair gives rise to DNA's double helix structure. Because of the uniqueness and specificity of DNA, the information retrieved from it has allowed us to make better predictions on the molecular evolution of all kinds species. Moreover, having the ancestor and progeny DNA sequences makes observing evolutionary history much clearer, since DNA is passed from parent to offspring. More specifically, this is done by observing when two sequences of the same type of DNA mutate and then using mathematical models and algorithms to construct a tree. Once complex computations are done, the algorithm yields a tree according to that model's parameters. It is important to note that using DNA sequences instead of just mere human observations does not necessarily imply accurate trees. Nonetheless, we are able to extrapolate as authentic a phylogentic tree possible given those data. Deducing an accurate tree is still an open field of research.

These models create a framework in which phylogenetic distances are computed according to properties of that model. More generally, the algorithms determine how certain species are related at a certain point in the tree using distances calculated. You can think of these trees as fancy history timelines with some uncertainty. The "distances" calculated are the distances between two species within this tree. Each root or node of a tree represents a common ancestor; and each time some form of evolutionary event occurs, the tree branches and splits off into two or more leaves. These leaves represent taxa, which are a group of one or more organisms. As you can imagine, over time there can be as many or few leaves on a tree and not all the same lengths! In fact, there are multiple ways to calculate these distances that we explored. It is also important to realize that all these disances methods take on different assumptions and parameters in a way the final tree constructed does as well.

In [2]:
```
import IPython.display
phyl = IPython.display.Image('http://4.bp.blogspot.com/-u9XslQrACb8/VK71Cym3zQI/AAAAAAAAA0/DAxkKTcCKvc/s1600/markovdiag.png')
phyl
```

Out[2]:

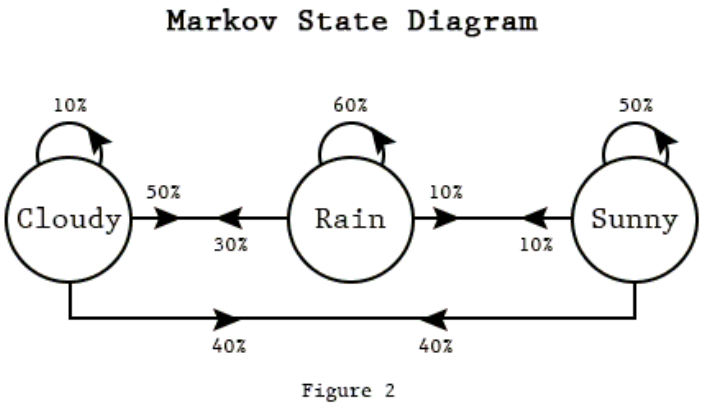


Figure 2

The diagram above is an example of the Markov Chain process. Simply put, the probabiity of the next day being "Cloudy", "Rainy", or "Sunny" is dependent on the weather condtions the day before and not on the many days before it.

Unfortunately, all of the known distance methods and inference algorithms are not explored here, because of time constraints and man-effort; nonetheless, just the complexity of biology will have stumped us like it stumps the many people who are work at the forefront of this field. This is because there are a plethora of parameters that can be used to determine the distance. Furthermore, even within those parameters, there are ample biological exceptions--possibly undiscovered ones too--and spontaneous possible events that would violate those assumptions. And so, to get anywhere with trying to understand the evolutionary history of some species, presumptions that are not too harmful or unreasonable have to be made. As aforementioned, certain distance methods have distinguishing assumptions made in order to calculate said distances. Once these distances are calculated, the issue of whether the species who are closest in distance are actually evolutionary neighbors arises, since after all it can just be coincidence. Moreover, there is another dilemma being how close one group of organisms is from another group on a separate branch. It truly is a complex field.

To reiterate, this project investigates a few of the many mathematical and computational techniques used to model evolution between different species. The algorithms compared were
UPGMA (Unweighted Pair Group Method with Arithmetic Mean) and **Neighbor-Joining**. These phylogenetic inference algorithms yielded trees according to evolution simulated by the **Jukes-Cantor model**. Despite only modeling evolution according the Jukes-Cantor model, there are other models. All, if not a majority of these distance models, fall under the umbrella of being a modificiation of the very general DNA substitution model: the **Markov chain model**. The Markov Chain models are models with the property that changes, such as the mutation of DNA, imposed on the future state of the system is dependent on only present state and not the many previous states before. Additonally, this also means the nucelotide base changes are independent of each other. Namely, the Markov model is "forgetful" and has no "memory" of the nucelotide changes that may have occurred before as influence.

Phylogenetics is not just the study of the evolutionary history of various organisms. In reality, the information given by these relationsips are used to tackle practical biological problems. We explore an application of this when we attempt to determine if a dentist infected his patients with HIV given the respective sequences. More modern and pressing problems such as understanding the role and possible progression (or reduction) of noncoding DNA within differing species'genomes over time and the modeling the development of tumor cells over time. In other words, phylogenetics can be used on a smaller scale with quite critical results. In addition to the biological applications, this project explores the refinement of the phylogenetic inference algorithms and distance methods through demonstration and discussion of the striking differences between the results acquired.

A Layman's Guide to Models for DNA Mutation

There are many ways to infer phylogenetic trees. In fact, the first way is to just look in the space of trees--the number possible trees--in a taxa and then choose what best fits the data. This is impractical because of the amount of computation for say, twenty taxa, is difficult to achieve. Another fun fact: the number of trees within this space of trees can exceed the number of atoms in the universe. Insane! Though, it is difficult for compute for that many taxa, there are methods such as **Maximum Parsimony** and **Maximum Likelihood**--methods we will not delve into--that use this concept to pick the most reasonable tree. The idea behind Maximum Parisomy derives from the principle of *Occam's Razor* which basically states that nature wants to choose the most convenient path. What this algorithm does is assign a tree a parsimony score based on the given DNA sequences and determines what is most reasonable. Maxmimum Likelihood, on the other hand, picks a the most "likely" evolutionary tree given the relationships. And so, the other method is to try and build a tree that fits the data given. This is what "distance-based methods" are. The distance-based method algorithms compared are UPGMA (Unweighted Pair Group Method with Arithmetic Mean) and Neighbor-Joining. We only compared two inference algorthims, but there are definitely more algorithms and distance models. For instance, in between the development of UPGMA and Neighbor-Joining, there was the **Fitch-Margoliash algorithm** that is actually quite close to Neighbor-Joining. Hence, we compared the two methods more striking in their differences. These phylogenetic inference algorithms yielded trees according to evolution simulated by the Jukes-Cantor model. Once we obtained the model for DNA mutation, we were then able to calculate possible phylogenetic distances, in other words, how closely related two sequences are.

To begin, evolution was simulated using a function, `mutate (A, t, seq)` , on sample aligned DNA sequences according to whichever model for DNA mutation we chose; for example, Jukes-Cantor, Kimura-2, and Kimura-3. We used the Jukes-Cantor model. This function "mutated" the aforementioned sequence according to time and the individual parameters given by each model. To get a better picture, when using the function `mutate()` to simulate evolution according to the Jukes-Cantor model, Jukes-Cantor distances between the ancestor and progeny sequences were computed. Once these distances were calculated, phylogenetic inference algorithms, such as the neighbor-joining algorithm, were implemented to produce phylogenetic trees. Each algorithm evidently deduced a different tree, and those trees were compared. In addition, different trees would be constructed when the distance method used changed.

Jukes- Cantor Model

The first and simplest distance model used was the **Jukes-Cantor** model. The Jukes-Cantor model begins by assuming each site mutates independently according to a Markov chain and all bases (nucleotides) within a DNA sequence occur with equal probability: $\frac{1}{4}$. This means the likelihood of the purine (Adenine(**A**) and Guanine(**G**) being substituted by a pyrimidine (Cytosine(**C**) and Thymine(**T**)) Lastly, it also assumes that the transition probabilities of the base substitutions are the same α; this is the rate at which evolution occues. You can probably recall the particulars of DNA and how certain nucleotides bond with each other due to their chemical and structural properties. The pitfall of the this assumption is that sort of substitution is highly unlikely--or at least occur with different probabilties--because of the steric hindrance and chemical properties between their molecular structures. Lastly, this model--and the other ones as well--adopts the *molecular clock* assumption which presumes that DNA mutation rates of observable substitutions, α, are constant. Realistically, the rates may not be constant, since it has been shown that the rates can be dependent on whether DNA is noncoding or coding and can change based upon the time and location of a particular sequence within DNA. Moreover, it is not strictly true that everything evolved at the same time; some changes happen sooner than later than others, especially if you consider possilble environmental phenomena. Mathematically, the distance is given by:

$$d_{JC}(S_0, S_n) = \frac{3}{4}\ln(1 - \frac{4}{3}p)$$

where S_0 and S_n is the ancestor and is the progeny sequences, respectively; and p is the fraction of the sites changed between S_0 and S_n. Despite making many assumptions, this model was useful in that it allows preliminary estimation calculations to be made.

Kimura-2 and Kimura-3 Models

Unlike the Jukes-Cantor model, the **Kimura** models consider more than one parameter to compute the distances between ancestor and progeny DNA sequences. In addition to mutation rates, the **Kimura-2** model incorporates different transition probabilties based on the various types of biological mutations such as the rate of transitions, p, (i.e.purine $\longleftrightarrow$ purine) and rates of transversions, q (i.e.purine $\longleftrightarrow$ pyrimidine). Transversions occur when a nucelotides change change from A $\leftrightarrow$ C, A $\leftrightarrow$ T, C $\leftrightarrow$ G, and G $\leftrightarrow$ T. Transitions occur when nucleotides change from A $\leftrightarrow$ G and C $\leftrightarrow$ T. Mathemtically, the distance given by the Kimura-2 parameter model is:

Libraries and Tools Used

In [3]:
```
import numpy as np
from scipy.io import loadmat
from numpy import linalg as LA
from ete3 import Tree
import sys
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')

import pylab

pylab.rcParams['figure.figsize'] = (4.0, 2.0)
```

In [4]:
```
hiv_data = loadmat('flhivdata.mat')

dnt    = hiv_data["dnt"][0]

ctrl_1 = hiv_data["lc1"][0]
ctrl_5 = hiv_data["lc5"][0]

ptb    = hiv_data["ptb"][0]
ptc        = hiv_data["ptc"][0]
ptd    = hiv_data["ptd"][0]

min_len = min(len(dnt),len(ctrl_1),len(ctrl_5),len(ptb),len(ptc),len(ptd))

def chop(seq):
    ans    = min_len*["o"]
    tokens = list(seq)
    for i in xrange(min_len):
        ans[i] = tokens[i]
    return ''.join(ans)

dnt    = chop(dnt)
ctrl_1 = chop(ctrl_1)
ctrl_5 = chop(ctrl_5)

ptb    = chop(ptb)
ptc    = chop(ptc)
ptd    = chop(ptd)
```

Implementations of the proposed models

Distance Functions

Jukes-Cantor Distance

In [5]:
```
"""
Constructs a Jukes Cantor transition Matrix with a specified alpha level a
Args:
     a: alpha level for the Jukes Cantor Matrix
Returns:
     Transition Matrix corresponding to the Jukes-Cantor Algorithm
"""

def JC_matrix(a):

    """
    >>> np.trace(JC_matrix(.25))
    3.0
    """

    b = a/3
    M = np.array([[1-a, b, b, b],
                  [b, 1-a, b, b],
                  [b, b, 1-a, b],
                  [b, b, b, 1-a]])
    return M

"""
Computes proportion of differing letters from two strings of the same size
Args:
    s1: string 1
    s2: string 2
Returns:
    Throws error if the strings are not of the same length
    Else, returns proportion (in between 0 and 1) of differing letters
"""

def prop_diff(s1,s2):
    if len(s1) != len(s2):
        raise ValueError("Cannot compute compare DNA sequences of differing lengt
h")
    diffs = 0
    i     = 0
    while i < len(s1):
        if s1[i] != s2[i]:
            diffs += 1
        i += 1
    return float(diffs)/float(len(s1))

"""
Computes the JC distance between two sequences.
Args:
    s1: string 1
    s2: string 2
Returns:
    Throws error if the strings are not of the same length
    Else, computes JC distance
"""

def JC_distance(s1,s2):
    diffs = prop_diff(s1,s2)
    return 1 - (np.log(1 - 4/3*diffs))

"""
Returns JC Matrix give sequences
"""

def JC_matrix_maker(seqs):
  M = np.zeros((len(seqs),len(seqs)))
  for i in xrange(len(seqs) - 1):
    s1 = seqs[i]
```

Mutate as a test for the accuraccy of Jukes Cantor

We used mutate() to simulate evolution according to the Jukes Cantor model on a suitable starting sequence. The Jukes Cantor distance was then computed between the final sequence and the initial sequence and subsequently compared to the actual distance. By doing this, we were able to answer the question of what effects does varying the parameters (Jukes-Cantor parameter and time) have.

In [6]:
```
"""
    Given a character 'A' 'G' 'C' or 'T' returns number corresponding
    to the index in whcih we need to index in our vector
    Args:
        nuc: character 'A' 'G' 'C' or 'T'
    Returns:
        number 0,1,2,3 indicatin what index to set to 1
"""

def DNA_to_position(nuc):
    if nuc == 'A':
        return 0
    if nuc == 'G':
        return 1
    if nuc == 'C':
        return 2
    if nuc == 'T':
        return 3

"""
    Given a number 0,1,2,3 returns character 'A' 'G' 'C' or 'T'
    corresponding to the nucleic acid represented in the model
    Args:
        number 0,1,2,3
    Returns:
        character 'A' 'G' 'C' or 'T' orresponding to the
        nucleic acid represented in the model
"""

def position_to_DNA(pos):
    if pos == 0:
        return 'A'
    if pos == 1:
        return 'G'
    if pos == 2:
        return 'C'
    if pos == 3:
        return 'T'

"""
    Mutates the DNA string seq, int t times according to the
    Jukes-Cantor model specified by the alpha value, a
    Args:
        a:   alpha level for the Jukes Cantor Matrix
        t:   number of time steps we plan to simulate
        seq: Sequence of DNA represented as a string
    Returns:
        simulated descendant sequence after t time steps
        consitently represented as a string
"""

def mutate(a, t, seq):
    M = JC_matrix(a)
    M = LA.matrix_power(M, t)
    tokens = list(seq)
    for i in xrange(len(tokens)):
        p = np.zeros(4)
        p[DNA_to_position(tokens[i])] = 1
        p = np.dot(M,p)

        rand= np.random.rand()
        if rand < p[0]:
            val = 'A'
        elif rand < p[0]+p[1]:
            val = 'G'
        elif rand < p[0]+p[1]+p[2]:
```

We would like a function with which to judge our mutate function: judge_jukes

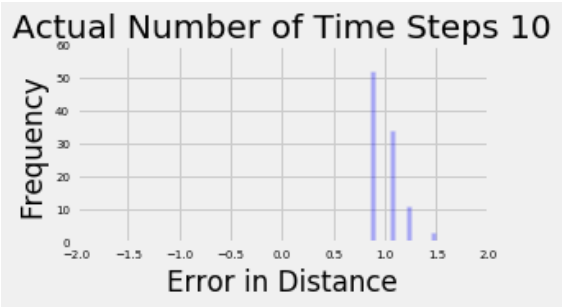
In [7]:
```
"""
Judges the quality of Jukes Cantor ov
    anc       : ancestor sequence
    diff      : actual difference between descendant and ancestor
    parameter : parameter of uniform distribution (i.e. on average how many will
survive)
"""
def judge_jukes(alpha,reps,ancs):
    ans  = np.zeros(100) # this should give significant enough dat
    for i in xrange(100): # this should give significant enough data
        desc   = mutate(alpha,reps,ancs)
        ans[i] = JC_distance(desc,ancs) -(reps*alpha)
    return ans
```

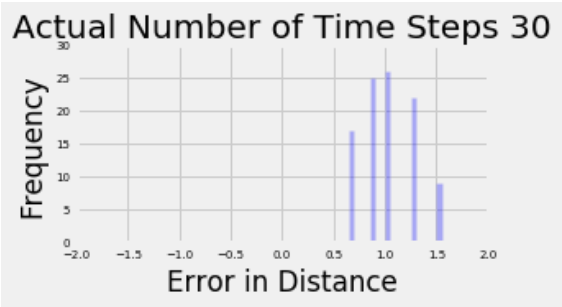
Small alpha, varying time

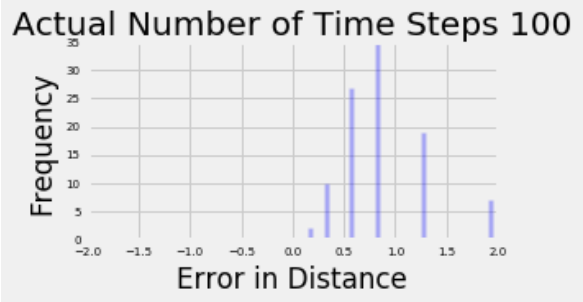
In [8]:
```
def judge_time(time):
    ans = judge_jukes(.01,time,"GATTACA")
    # ans = plots.hist(ans, bins = np.arange(-2,2,0.1), alpha =.3, color ="blue")
    stats = [np.subtract(*np.percentile(ans, [75, 25])),np.median(ans)]
    print(stats)
    return ans
```

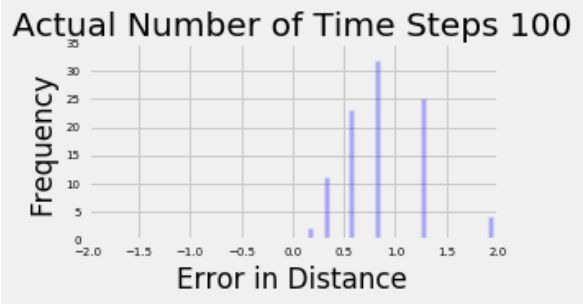
In [9]:
```
plots.rc('xtick', labelsize=7)
plots.rc('ytick', labelsize=7)
times = [1,10,30,100,100,100,100]
for i in times:
    judge_time(i)
    ans   = judge_time(i)
    title = "Actual Number of Time Steps " + str(i)
    x_lab = "Error in Distance"
    y_lab = "Frequency"
    plots.hist(ans, bins = np.arange(-2,2,0.05), alpha =.3, color ="blue")
    plots.title(title)
    plots.xlabel(x_lab)
    plots.ylabel(y_lab)
    plots.figure()
```

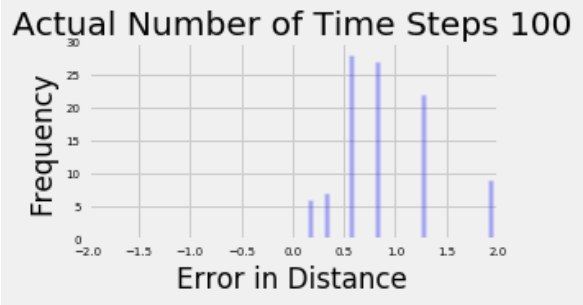
```
[0.0, 0.98999999999999999]
[0.0, 0.98999999999999999]
[0.15415067982725816, 1.0541506798272582]
[0.15415067982725816, 0.90000000000000002]
[0.18232155679395468, 1.0364722366212129]
[0.40546510810816438, 1.0364722366212129]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
[0.6931471805599454, 0.84729786038720345]
```

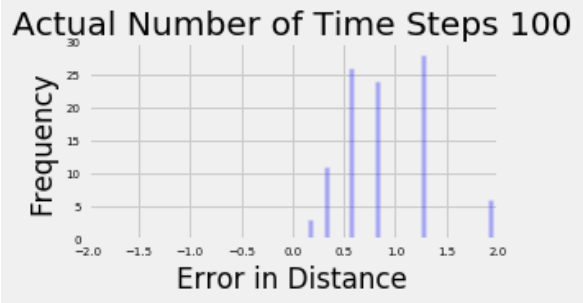













```
<matplotlib.figure.Figure at 0x113b3fad0>
```

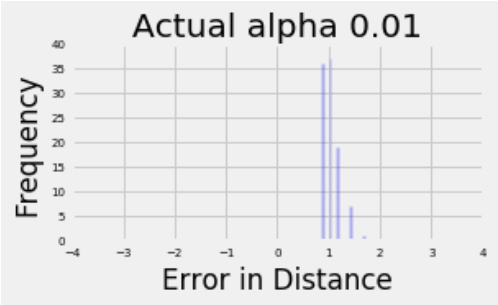
From above we see that for a reasonable mutation rate of $.01$, and for time steps progressively larger, we see the α t is consistently over-estimated by the Jukes-Cantor Distance. Furthermore se see that all of the histograms with 100 time-steos are very similar in shape, center, and spread

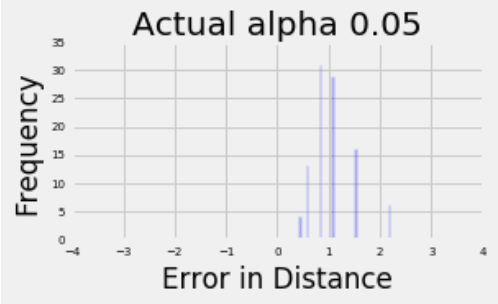
Varying alpha, consistent time-steps = 15

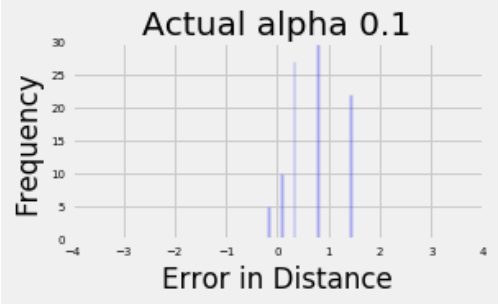
In [10]:
```
def judge_alpha(alpha):
    ans = judge_jukes(alpha,15,"GATTACA")
    # ans = plots.hist(ans, bins = np.arange(-2,2,0.1), alpha =.3, color ="blue")
    stats = [np.subtract(*np.percentile(ans, [75, 25])),np.median(ans)]
    print(stats)
    return ans
```

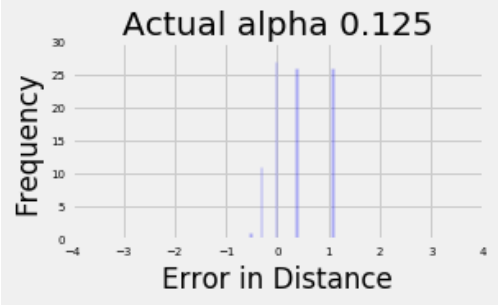
In [11]:
```
plots.rc('xtick', labelsize=7)
plots.rc('ytick', labelsize=7)
alphas = [.01,.05,.1,.125,.15,.2,.25,.3]
for i in alphas:
    ans   = judge_alpha(i)
    title = "Actual alpha " + str(i)
    x_lab = "Error in Distance"
    y_lab = "Frequency"
    plots.hist(ans, bins = np.arange(-4,4,0.05), alpha =.3, color ="blue")
    plots.title(title)
    plots.xlabel(x_lab)
    plots.ylabel(y_lab)
    plots.figure()
```

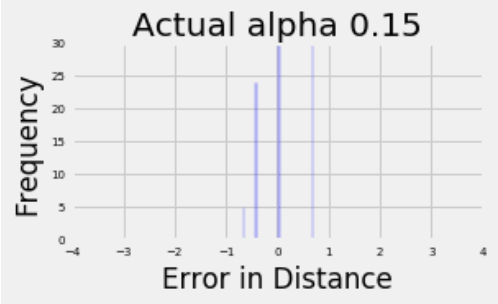
```
[0.33647223662121306, 1.0041506798272584]
[0.28768207245178079, 1.0972978603872034]
[1.0986122886681096, 0.75276296849536806]
[1.0986122886681096, 0.37776296849536806]
[1.0986122886681096, 0.0027629684953680567]
[1.0986122886681096, -0.74723703150463194]
[0.7945134575869861, -1.4972370315046319]
[0.69314718055994495, -2.2472370315046319]
```

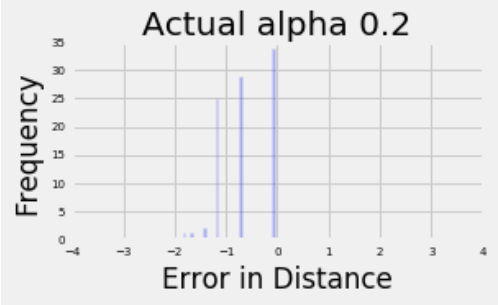


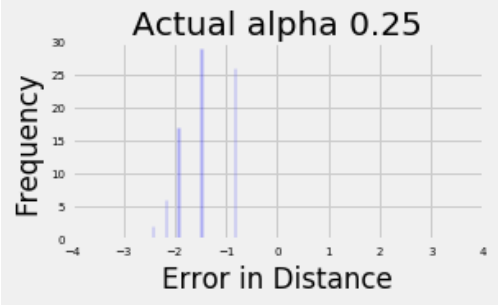


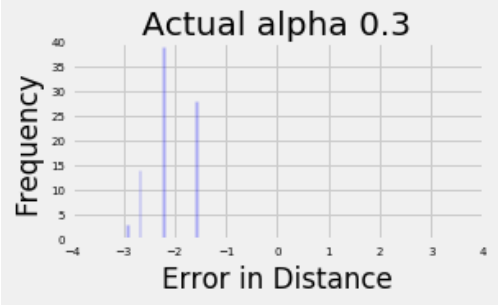













```
<matplotlib.figure.Figure at 0x114b02950>
```

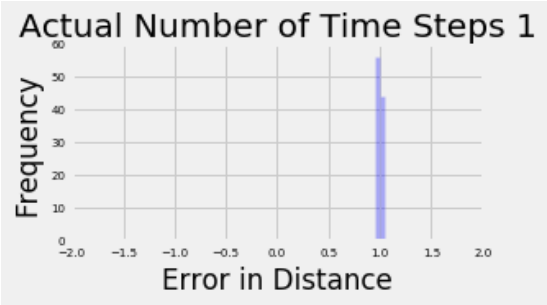
We see that for a constant time step, larger alpha implies a correspondingly larger underestimate.

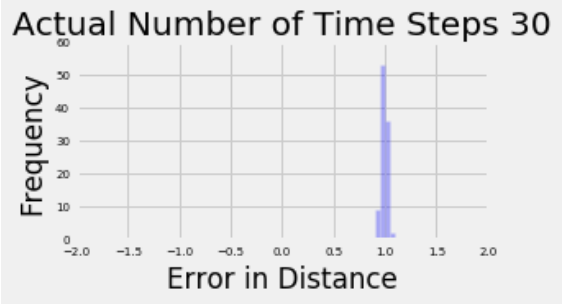
After experimeting with the mathematics of the Jukes-Cantor Distance we hypothesized that perhaps the sequence GATTACA was too short causing these extreme and varying biases in the distance estimate. Building off of this idea we decided to try and run a similar analysis on a longer sequence.

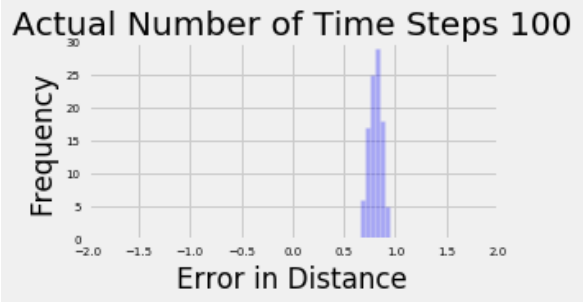
In [12]:
```
def judge_time_dentist(time):
    ans = judge_jukes(.01,time,dnt)
    # ans = plots.hist(ans, bins = np.arange(-2,2,0.1), alpha =.3, color ="blue")
    stats = [np.subtract(*np.percentile(ans, [75, 25])),np.median(ans)]
    print(stats)
    return ans
```

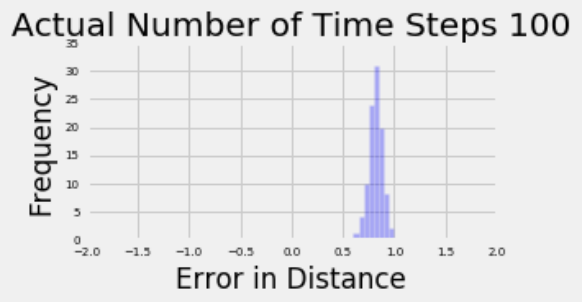
In [13]:
```
plots.rc('xtick', labelsize=7)
plots.rc('ytick', labelsize=7)
times = [1,10,30,100,100,100,100]
for i in times:
    judge_time_dentist(i)
    ans   = judge_time_dentist(i)
    title = "Actual Number of Time Steps " + str(i)
    x_lab = "Error in Distance"
    y_lab = "Frequency"
    plots.hist(ans, bins = np.arange(-2,2,0.05), alpha =.3, color ="blue")
    plots.title(title)
    plots.xlabel(x_lab)
    plots.ylabel(y_lab)
    plots.figure()
```

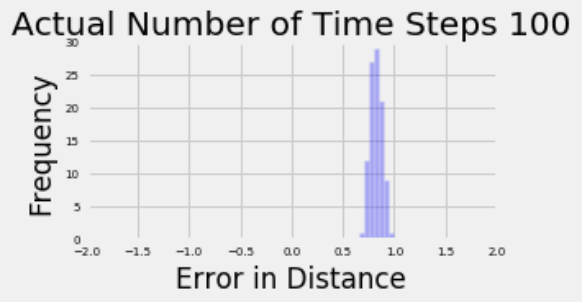
```
[0.0062112000926408673, 0.99927363678532921]
[0.0093313274288844283, 0.99927363678532921]
[0.022034839732640421, 0.99684982598991778]
[0.027120306219193702, 0.99684982598991778]
[0.041526559521244844, 0.97849364639737457]
[0.036965194233471665, 0.99282559758111999]
[0.0989490819946377, 0.81401188275373704]
[0.095963988261743793, 0.80021856062140095]
[0.10389959501815182, 0.81401188275373704]
[0.090654368268130892, 0.82098055206983034]
[0.075637414205619979, 0.8070914399091631]
[0.097298910986799658, 0.81401188275373704]
[0.082805903684232796, 0.8070914399091631]
[0.082805903684232796, 0.8070914399091631]
```

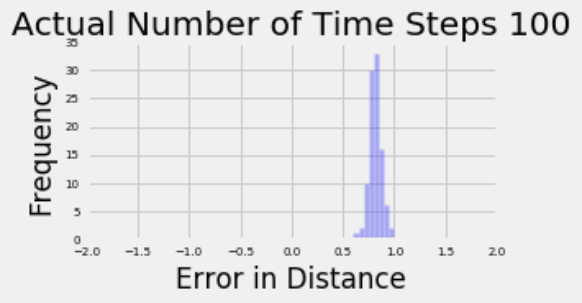












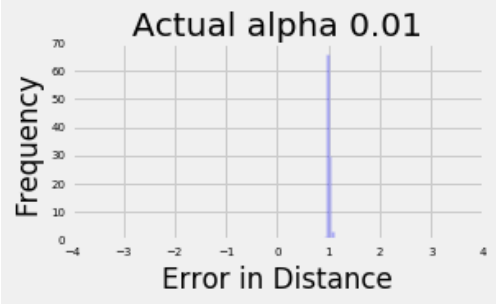

```
<matplotlib.figure.Figure at 0x115f775d0>
```

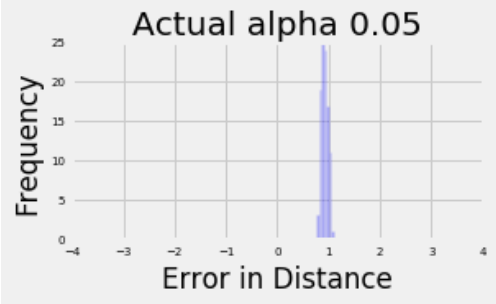
We see that the data has much less variance as evident by the smaller IQR as well as the visual evidence in the histograms. Unfortunately we still have an overstimating bias for alpha = 0.01, the next natural question is to see whether or not the biases match what we saw with GATTACA with varying alpha.

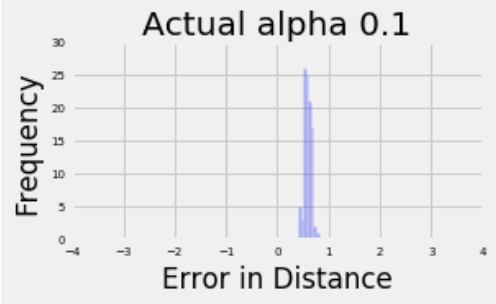
In [14]:
```
def judge_alpha_dentist(alpha):
    ans = judge_jukes(alpha,15,dnt)
    # ans = plots.hist(ans, bins = np.arange(-2,2,0.1), alpha =.3, color ="blue")
    stats = [np.subtract(*np.percentile(ans, [75, 25])),np.median(ans)]
    print(stats)
    return ans
```

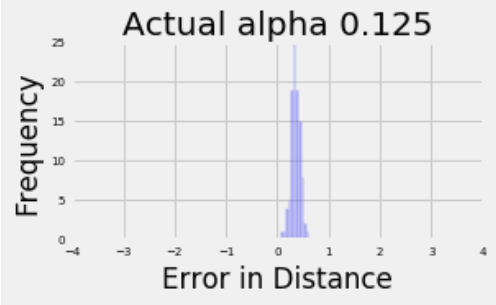
In [15]:
```
plots.rc('xtick', labelsize=7)
plots.rc('ytick', labelsize=7)
alphas = [.01,.05,.1,.125,.15,.2,.25,.3]
for i in alphas:
    ans   = judge_alpha_dentist(i)
    title = "Actual alpha " + str(i)
    x_lab = "Error in Distance"
    y_lab = "Frequency"
    plots.hist(ans, bins = np.arange(-4,4,0.05), alpha =.3, color ="blue")
    plots.title(title)
    plots.xlabel(x_lab)
    plots.ylabel(y_lab)
    plots.figure()
```

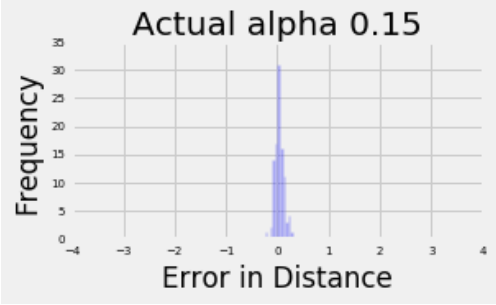
```
[0.025743724215814057, 0.99547051299599187]
[0.1072455303535973, 0.90986120292647854]
[0.093364140959764308, 0.58334481653732118]
[0.118104171660526, 0.33929559734412784]
[0.11384346007704838, 0.034015511999472281]
[0.099317440143467772, -0.63501542546686052]
[0.12210269680090002, -1.3969916165135761]
[0.13974938831746631, -2.1044127123434224]
```

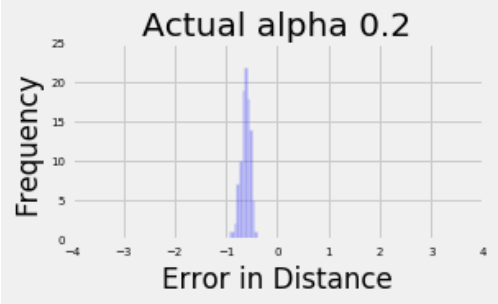


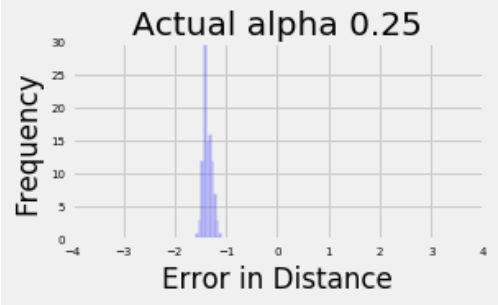


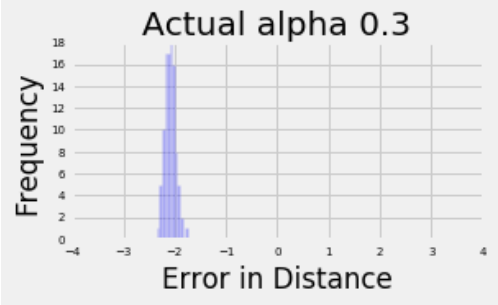













```
<matplotlib.figure.Figure at 0x114b999d0>
```

We still observe the same errors and biases on the larger dentist sequence. We wondered why this is and did an analysis of the derivation of the Jukes-Cantor estimate, we found there to be two places in the derivation where the bias could occur, first when we did a taylor expansion of ln, the other where we sample the proportion changed in a sequence and take that to be the probability of a base pair having changed. We know that from the taylor expansion that our esitmate should be biased in only one direction, however because we both over and underestimate with different parameters we conclude that the sampling bias of the sequence is a significant source of error and must at least sometimes cause the estimate to bias in the opposite direction.

UPGMA and Neighbor-Joining Algorithm

An interesting thing to note is that we first implemented Neighbor Joining and UPGMA seperately, but then noticed that we could refactor the code to be much more concise. In doing so we created neighbor_based_method and used first class functions in Python as arguments to implement the exact parts of the algorithm which differ, essentially using a composition of more basic functions for the actual implmentation of Neighbor Joining and UPGMA. We found it really cool how after doing this the nearly 1 to 1 correspondence of the functional code and the algorithms themselves highlights the precise differences between Neighbor Joining and UPGMA, mostly that UPGMA is like Neighbor Joining but does not necissarily join true neighbors (we knew this from our proofs in class, but it was really interesting to see how apparent this distinction is by implementing non-redundant code).

In [16]:
```
"""
Computes a list whose ith entry is the distance from taxa i to all of the other t
axa in the distance matrix
Args:
    M: Upper triangular distance matrix
Returns:
    "sums" which is a list whose ith entry is the distance from taxa i to all of
the other taxa in the distance matrix
"""
def sums_others(M):
    size = len(M)
    sums = np.zeros(size)
    for i in xrange(size):
        s = 0
        for other in xrange(size):
            s += M[min(other, i), max(other, i)]
        sums[i] = s
    return sums

"""
Computes a tuple of "coordinates" whose entries i and j correspond to the numbers
 of the two taxa that are to be joined
Args:
    M: Criterion matrix could be Q matrix or just a distance matrix
Returns:
    "coordinates" whose entries i and j correspond to the numbers of the two taxa
 that are to be joined
"""
def closest_neighbors(M):
    size = len(M)
    min = sys.maxint
    coordinates = (0,0)
    for i in xrange(size-1):
        for j in xrange(i + 1, size):
            if M[i,j] <= min:
                min = M[i,j]
                coordinates = (i,j)
    return coordinates

"""
Makes Q matrix that decides what will be joined
Args:
    M: Upper triangular matrix
Returns:
    Q matrix
"""
def make_Q_matrix(M):
    sums = sums_others(M)
    N  = M.shape[0]        # number of taxa
    Q  = np.zeros(M.shape) # matrix to be returned
    for i in xrange(N):
        for j in xrange(N):
            if i < j:
                Q[i][j] = (N-2)*M[i][j] - sums[i] - sums[j]
    return Q

"""
Computes the subsequent Distance Entry of the UPGMA algorithm
Args:
    M:     the transition Matrix for the Jukes Cantor Algorithm
    taxa1: taxa that was combined
    taxa2: taxa that was combined
    j:     taxa that we want to find distance to new node
Returns:
    new distance using from j to cherry
"""
```

In [17]:
```
import IPython.display
art  = IPython.display.Image("http://www.southampton.ac.uk/~re1u06/teaching/upgma/upgma15.PNG")
art
```

Out[17]:

	A	B	C	D	E	F	G
A							
B	19.00						
C	27.00	31.00					
D	8.00	18.00	26.00				
E	33.00	36.00	41.00	31.00			
F	18.00	**1.00**	32.00	17.00	35.00		
G	13.00	13.00	29.00	14.00	28.00	12.00	

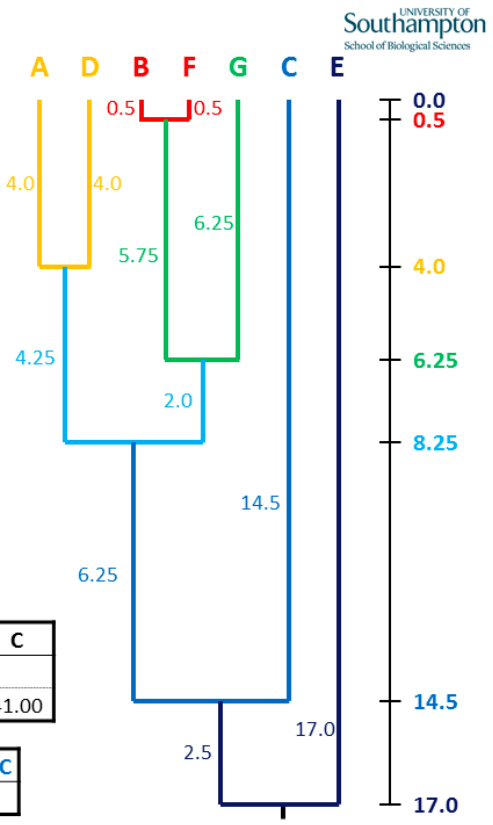
	A	BF	C	D	E
BF	18.50				
C	27.00	31.50			
D	8.00	17.50	26.00		
E	33.00	35.50	41.00	31.00	
G	13.00	12.50	29.00	14.00	28.00

	AD	BF	C	E
BF	18.00			
C	26.50	31.50		
E	32.00	35.50	41.00	
G	13.50	**12.50**	29.00	28.00

	AD	BFG	C
BFG	**16.50**		
C	26.50	30.67	
E	32.00	33.00	41.00

	ADBFG	C
C	**29.00**	
E	32.60	41.00

	ADBFGC
E	**34.00**

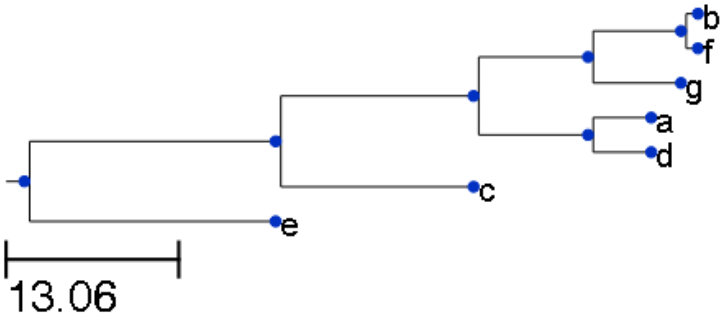


In [18]:
```
M = np.array([[0, 19.0,  27.0,  8.0, 33.0, 18.0 ,13.0],
              [0,    0,  31.0, 18.0, 36.0,  1.0, 13.0],
              [0,    0,   0,   26.0, 41.0, 32.0, 29.0],
              [0,    0,   0,      0, 31.0, 17.0, 14.0],
              [0,    0,   0,      0,    0, 35.0, 28.0],
              [0,    0,   0,      0,    0,    0, 12.0],
              [0,    0,   0,      0,    0,    0,   0 ]
               ])

names = ['a', 'b', 'c', 'd','e','f','g']


ans = neighbor_based_method(M,names, closest_neighbors, UPGMA_new_dist, split_dist)
print(ans)
```

```
          /-b
       /-|
    /-|   \-f
   |  |
 /-|   \-g
|  |
|  |   /-a
 /-|  \-|
|  |     \-d
--|  |
  |   \-c
  |
   \-e
```

In [19]:
```
art  = IPython.display.Image("http://i.imgur.com/GmsLjVW.png")
art
```

Out[19]:



Above we see that the proportions are correct, but the scaling factor is off slightly from the internet example

Simulation-based assessmnet of our models

Up until this point the only tests we have been running, have tested whether or not we gave the correct implementation of the Phylogenetic Algorithms. A natural question to ask oursevles at this point is: whether or not there is some way to simulate phylogenetic trees in order to better test the validity of our models.

Below we have created a mutate() function to simulate evolution according to Jukes Cantor on various sequencences. Compute the Jukes Cantor distance between the final sequence and the initial sequence, and compare to the actual distance. How does varying the parameters (Jukes-Cantor parameter and time) effect this?

Mutate Function and Evolution Simulator

In [20]:
```
"""
Simulates Evolution of a DNA sequence with
    a                : alpha level (mutation rate)
    sim_time         : total time simulating
    timestep         : timestep will dictate distance between father and son node
    seq              : original input seqeuence
    selectionFn      : Decides who survives from population
    paramater        : list contating parameters to be fed into the selection fn
    expansion_factor : multiplication factor that determines number of seqs in th
e population. pop_size_n = 2*pop_size_n-1 - those killed by selectionFn
"""



    t = Tree(name= seq)
    def tree_helper(curr_time, t):
        if curr_time >= sim_time - timestep:
            return
        pop = [t.name]*expansion_factor
        pop = map(lambda x: mutate(a, timestep, x), pop) # Mutates everything in
the population
        pop = selectionFn(pop,paramater) # selects the survivors
        for taxa in pop:
            new_t = t.add_child(name = taxa)
            new_t.dist = timestep
            tree_helper(curr_time+timestep, new_t)
    tree_helper(0, t)
    return t

"""
Kills on average "proportion" of the population. (Uniform Distribution)
    pop       : population of sequnces that Uniform Killing decides whether or no
t to kill
    parameter : parameter of uniform distribution (i.e. on average how many will
survive)
"""
def uniform_killing(pop, proportion):
    # filter keeps seqeunces less than proportion
    return filter(lambda y: np.random.rand() < proportion, pop)
```

Data Analysis Using Evolution Simulator

Here we benchmark our Neighbor Joining algorithm against the simulated trees using our evolution simulator. This is done by running the simulator and then using the Neighbor Joining algorithm to infer the tree from only the leaves, we use the dentist sequence to have a realistic sequence. We can then visually compare the true tree to the inferred tree.

Note that the trees produced by our mutation method will have intermediate nodes in a line such as - /- /- /- , in the reconstructed trees these will be a single tree edge. This occurs when only a single taxa survives a round of natural selection in the simulation. In comparing the trees we will consider such collections of edges as a single edge as that is what they represent topologically in terms of a phylogentic tree.

In [85]:
```
t = evolution_simulator(alpha, 50, 10, dnt, uniform_killing, .7, 2)
# full sequence is unreadable so move name to sequence, also record names and seq
uences for inference
names = []
leaves = []
counter = 0
for leaf in t.iter_leaves():
        leaf.sequence = leaf.name
        leaf.name = counter
        names.append(counter)
        leaves.append(leaf.sequence)
        counter += 1
print(t)
```

```
             /-0
       /- /-|
   /-|      \-1
  |  |
--|   \- /- /-2
  |
   \- /-3
```

In [86]:
```
M = JC_matrix_maker(leaves)
print(neighbor_joining(M, names))
```

```
      /-2
   /-|
  |   \-3
--|
  |   /-0
   \-|
      \-1
```

Now we try a longer time of simulation for a larger tree

In [95]:
```
t = evolution_simulator(alpha, 70, 10, dnt, uniform_killing, .7, 2)
# full sequence is unreadable so move name to sequence, also record names and seq
uences for inference
names = []
leaves = []
counter = 0
for leaf in t.iter_leaves():
        leaf.sequence = leaf.name
        leaf.name = counter
        names.append(counter)
        leaves.append(leaf.sequence)
        counter += 1
print(t)
```

```
                    /-0
               /- /-|
              |     \-1
          /-|
         |   |     /-2
       /-|   \- /-|
      |  |        \-3
      |  |
-- /-|   \- /- /- /-4
      |
      |             /-5
       \- /- /- /-|
                    \-6
```

In [96]:
```
M = JC_matrix_maker(leaves)
print(neighbor_joining(M, names))
```

```
          /-5
       /-|
    /-|   \-6
   |  |
   |   \-0
   |
--|          /-2
   |       /-|
   |    /-|   \-3
   |   |  |
    \-|   \-4
      |
       \-1
```

In both cases we can see that while the topology of the tree is changed, the neighbor relationships have not chnaged.

Data Analysis Using NJ and UPGMA

To begin to understand the mechanics behind the UPGMA and Neighbor-Joining algorithms, we used a small input sequence (a small string) to see what kind of tree we would be able to recover; however, we were unsuccessful, because the string was much too small and not within the parameters of the model. Because of this, we escalated to a larger data set, the data set aforementioned: the HIV dataset. This dataset originated from the curious case of a dentist infecting six of his patients with HIV.

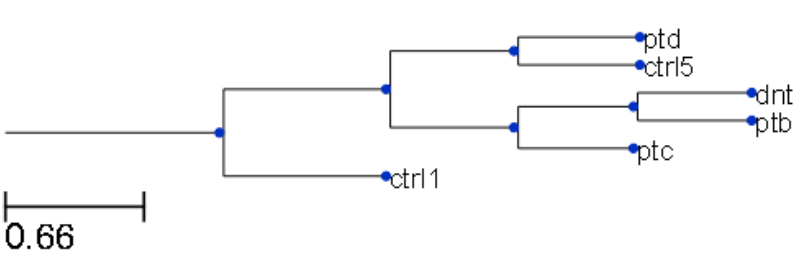
In [21]:
```
names = ["dnt", "ptb", "ptc", "ptd", "ctrl1", "ctrl5"]
seqs  = [dnt, ptb, ptc, ptd, ctrl_1, ctrl_5]
M     = JC_matrix_maker(seqs)
tree  = UPGMA(M, names)
print(tree)
```

```
          /-ptd
       /-|
      |   \-ctrl5
   /-|
  |  |       /-dnt
  |  |   /-|
--|   \-|   \-ptb
  |     |
  |      \-ptc
  |
   \-ctrl1
```

Using our metric tree browser we see:

In [22]:
```
art  = IPython.display.Image("http://i.imgur.com/IUtFvPJ.png")
art
```

Out[22]:



Data Analysis Using NJ and UPGMA

Analysis: From the visualization of the UPGMA metric tree, it appears as if the dentist infected patients B and C but there is not enough evidence from the above tree that the dentist infected patient D.

With a reasonable data set, we were able to witness the power of the Neighbor-Joining algorithm. We implemented this algorithm and as a result, were able to build a tree. Surprisingly, from the data gathered, we were able to reaffirm the same conclusions made by the researchers who were working on this question. That is, the dentist did indeed infect Patients B and C with HIV.

In [23]:
```
names = ["dnt", "ptb", "ptc", "ptd", "ctrl1", "ctrl5"]
seqs  = [dnt, ptb, ptc, ptd, ctrl_1, ctrl_5]
M     = JC_matrix_maker(seqs)
tree  = neighbor_joining(M, names)
print(tree)
```

```
            /-dnt
         /-|
      /-|   \-ptc
     |  |
   /-|   \-ptb
  |  |
--|   \-ptd
  |
  |   /-ctrl1
   \-|
      \-ctrl5
```

In [ ]:
```
M = JC_matrix_maker(leaves)
print(neighbor_joining(M, names))
```

This tree seems fairly close, the topology is different, the neighbors are correct

Now we try a longer time of simulation for a larger tree

In [ ]:
```
t = evolution_simulator(alpha, 70, 10, dnt, uniform_killing, .7, 2)
# full sequence is unreadable so move name to sequence, also record names and seq
uences for inference
names = []
leaves = []
counter = 0
for leaf in t.iter_leaves():
        leaf.sequence = leaf.name
        leaf.name = counter
        names.append(counter)
        leaves.append(leaf.sequence)
        counter += 1
print(t)
```

In [ ]:
```
M = JC_matrix_maker(leaves)
print(neighbor_joining(M, names))
```

Mathematical Explorations

Markov Chains

Definition. *Given a probability measure P on ω, a discrete **Markov process** (or Chain) is a collection of $\mathcal{T}$- valued random events $\{X_t\}_{t\in\mathbb{N}}$ such that*

$$P(X_m = y|X_{m-1} = y_{m-1}, X_{m-2} = y_{m-2}, \ldots, X_1 = y_1) = P(X_m = y|X_{m-1} = y_{m-1})$$

for all $y_1, \ldots, y_m \in \mathcal{T}$. Usually, $\mathcal{T} = \mathbb{R}$ in which case $\{X_t\}_{t\in\mathbb{N}}$ are called random variables.

Essentially, Markov process is a stochastic process in which the future states of the system is contingent only on the present state. So one could, in theory predict the future of the of the system based entirely on its present state. Random walks (on a lattice) is a classic example of a Markov process, where given a starting point in the lattice and a probability distribution, the location at each step jumps to another site based the distribution.

Markov processes are of interest to us because they provide with a nice enough model for evolution.

Computing Probabilities

We've claimed that given information on the 'initial' state of the system, we can compute the its future states. This is done via what are called transition matrix of the Markov Chain,

Definition. *The Transition Matrix of a Markov chain is a $n \times n$ square matrix M such that the element on the i^{th} row and j^{th} column $m_{ij} = P(X_m = i|\ X_{m-1} = j)$. A Regular Markov matrix is a special type of transition matrix such that for some $t \in \mathbb{N}$ and $1 \leq i, j \leq n$, $\sum_{i=1}^{n} m_{ij} = 1$ for all j and all entries of M^t are positive.*

As is the norm wherever matrices are involved, the probability vector of the succeeding state is obtained by left multiplying the Markov matrix with the probability vector.

Theorem 1. *The probability vector $\mathbf{p}_i$ of probabilities at i^{th} time step is $M\mathbf{p}_{i-1}$. Consequently, for any $t \in \mathbb{N}$, $p_t = M^t\mathbf{p}_0$*

Proof. Done in class. □

The following (rather gratifying) result about Regular Markov matrix makes our life a lot easier:

Theorem 2. *A Regular Markov matrix M always has 1 as an eigenvalue, i.e. there exists a vector $\mathbf{p}_{eq} \in \mathbb{R}^n$ such that $M\mathbf{p}_{eq} = 1 \cdot \mathbf{p}_{eq} = \mathbf{p}_{eq}$. Furthermore, all its other eigenvalues are less than 1.*

This guarantees the existence of a equilibrium state of any Markov matrix and although it appears to be slightly out of the scope of this paper, for any (regular) Markov process, the probability vectors eventually converge to the $\mathbf{p}_{\text{eq}}$, the equilibrium probability distribution.

Phylogenetic Distances

Given two sequences $\sigma^1 = \sigma_1^1\sigma_2^1\ldots\sigma_n^1$ and $\sigma^2 = \sigma_1^2\sigma_2^2\ldots\sigma_n^2$ where each $\sigma^{1,2}i \in \{A, T, C, G\}$ the phylogenetic distance between the two sequences is the expected number of substitutions per site during the time that has elapsed. Different models of evolution give us different distance functions. The distance function for the Jukes Cantor model, as discussed in class is

$$d_{JC}(\sigma^1, \sigma^2) = 1 - \frac{3}{4}\ln\left(1 - \frac{4}{3}p\right)$$

where p is the proportion of nucleotides that has mutated to another nucleotide. As expected, the Kimura 2-parameter model gives us a vastly different distance function. We shall derive it in the following subsection.

Kimura-2 Distance

To recall, the Markov matrix of the Kimura 2 parameter model is

$$M_K = \begin{bmatrix} \cdot & \beta & \beta & \alpha \\ \beta & \cdot & \alpha & \alpha \\ \beta & \alpha & \cdot & \alpha \\ \alpha & \beta & \beta & \cdot \end{bmatrix}$$

Let P denote the proportion of transitions(A $\rightleftharpoons$ G) and Q the proportion of transversions(A $\rightleftharpoons$ T and A $\rightleftharpoons$ C) within the sequence then by changing basis and exponentiating (similar to the Jukes Cantor model), we get that

$$P = \frac{1}{4}\left(1 - 2\exp(-4(\alpha+\beta)t + \exp(-8\beta t)\right) \tag{1}$$

$$Q = \frac{1}{2}\left(1 - \exp(-8\beta t)\right) \tag{2}$$

From the the matrix M_K, we see that the rate of substitution[Nei00, p. 37] is $2\beta + \alpha$. It follows, therefore that $d = (2\beta + \alpha)t = 2t\beta + t\alpha$. Notice that adding (1) and (2) we get,

$$\begin{aligned} 4P + 2Q &= 1 - 2\exp(-4(\alpha+\beta)t \\ &= 1 - 2\exp(-4dt + 4\beta t) \end{aligned}$$

Thus it follows that $4\beta - 4dt = \ln\left(\frac{1}{2} - 2P - Q\right)$. But notice, from (2),

$$4\beta t = -\frac{1}{2}\ln\left(1 - 2Q\right)$$

Combining these two equalities together and solving for d, we get the distance:

$$d_K(\sigma^1, \sigma^2) = -\frac{1}{2}\ln(1 - 2P - Q) - \frac{1}{4}\ln(1 - 2Q) \tag{3}$$

Conclusion

The application of phylogenetic inference algorithms to this HIV data set demonstrates the versatility of the composed trees. In particular, when we implemented the UPGMA algorithm on the HIV dataset, there were hints that the dentist did affect the same patients, but the distances computed did not shine through concretely enough to conclude that the dentist did infect Patient B and C. From this, it is evident that the Neighbor-Joining algorithm is much more powerful than UPGMA. The reasons for this are because it takes in many more parameters, does not assume the molecular clock is at work, and computes branching of leaves with a more complex method. Though we were unable to use the Kimura distances, it will only further show how much more accurate a tree can be constructed. The trees given by these phylogenetic inference algorithms demonstrate the wide variety of problems it can be used to address and the intricacies of developing an efficient and reasonable algorithm that is able to comply with biology's unpredictability.

References

[Nei00] Nei, M. & Kumar, S. "Molecular Evolution and Phylogenetics.", Oxford University Press, New York. (2000) Print.

Alex Pearson
Josh Nixon
Bidit Acharya
Tracy Lou

Math 127 Spring 2016