Chapter 7

SUPPLEMENTARY

7.1 Instrumentation

Software

- R version 2.15.0 (2012-03-30)
    - APE version 2.15.0
    - phangorn version 2.15.0
- MEGA 5.05 (build number: 5110426)
- PyCogent version 1.5.1

Hardware

- Notebook
    - ACER Aspire 4750G
    - Intel® Core™ i5-2410M
    - Windows® 7 Home Premium
    - 8GB RAM

7.2 Appendix

Appendix A

Frequency of Node Appearance of Protein Coding Sequence

Sequence	*MEGA*			*R*		
	A	**B**	**C**	**A**	**B**	**C**
NADH dehydrogenase subunit 1	1	1	1	1	1	0
NADH dehydrogenase subunit 2	1	1	0	1	1	0
NADH dehydrogenase subunit 3	1	1	1	1	1	1
NADH dehydrogenase subunit 4	1	1	1	1	1	1
NADH dehydrogenase subunit 4L	0	0	0	0	0	1
NADH dehydrogenase subunit 5	1	1	1	1	1	0
NADH dehydrogenase subunit 6	1	0	0	1	1	0
Cytochrome b	1	1	1	1	1	1
Cytochrome c oxidase subunit I	1	1	1	0	0	0
Cytochrome c oxidase subunit II	1	1	1	1	1	0
Cytochrome c oxidase subunit III	1	1	0	1	1	0
ATP synthase F0 subunit 6	1	1	0	1	1	0
ATP synthase F0 subunit 8	0	0	0	0	0	0

Apendix B

Frequency of Node Appearance of Ribosomal RNASequence

Sequence	*MEGA*			*R*		
	A	**B**	**C**	**A**	**B**	**C**
12S ribosomal RNA	0	0	1	0	0	1
16S ribosomal RNA	0	0	1	0	0	1

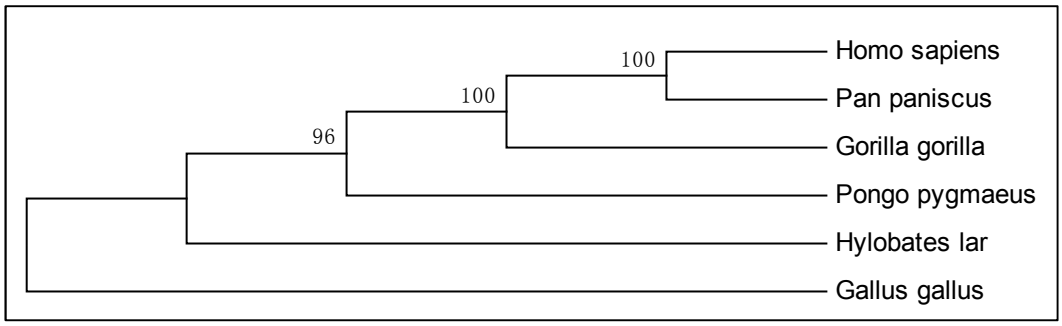
Appendix C

Frequency of Node Appearance of Transfer RNA Sequence

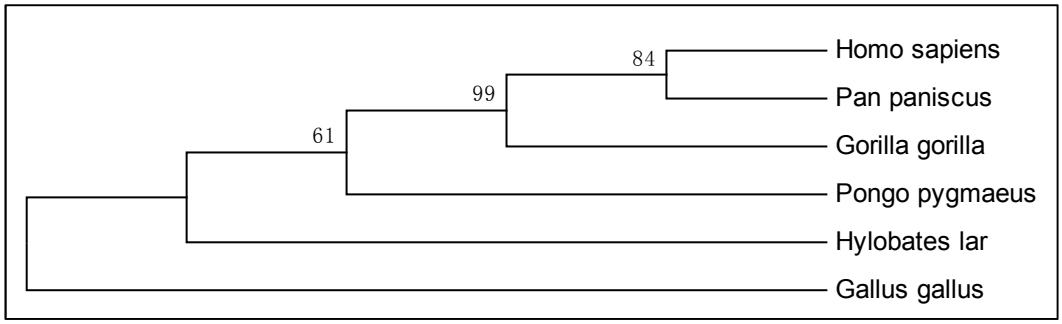
Sequence	*MEGA*			*R*		
	A	**B**	**C**	**A**	**B**	**C**
Phenylalanine	1	1	1	1	1	1
Valine	0	0	1	0	0	1
Leucine (codon – “UUR”)	0	0	0	1	0	0
Isoleucine	1	0	0	1	1	0
Glutamine	1	1	1	1	1	1
Methionine	0	0	0	0	0	0
Tryptophan	1	0	0	1	0	0
Alanine	1	0	0	0	0	0
Asparagine	1	1	0	1	1	0
Cysteine	0	0	0	0	0	0
Tyrosine	1	1	1	1	1	1
Serine (codon – “UCN”)	0	0	0	0	0	0
Aspartic Acid	0	0	0	0	0	0
Lysine	0	0	0	0	0	0
Glycine	1	0	0	1	0	0
Arginine	0	0	1	0	0	1
Histidine	1	1	1	1	1	1
Serine (codon – “AGY”)	0	0	0	0	0	0
Leucine (codon – “CUN”)	0	0	0	0	0	0
Glutamic acid	1	0	0	1	0	0
Threonine	0	0	0	0	0	0
Proline	1	1	0	1	1	0

Appendix D

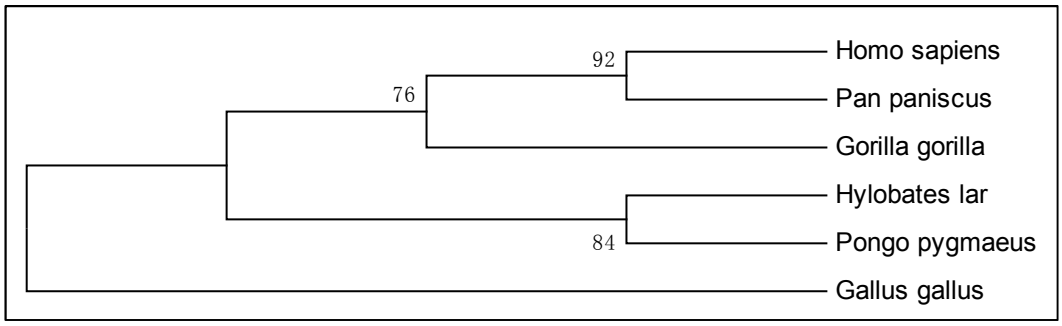
Topology of phylogenetics tress of MEGA5for all genes including mitochondrial genome



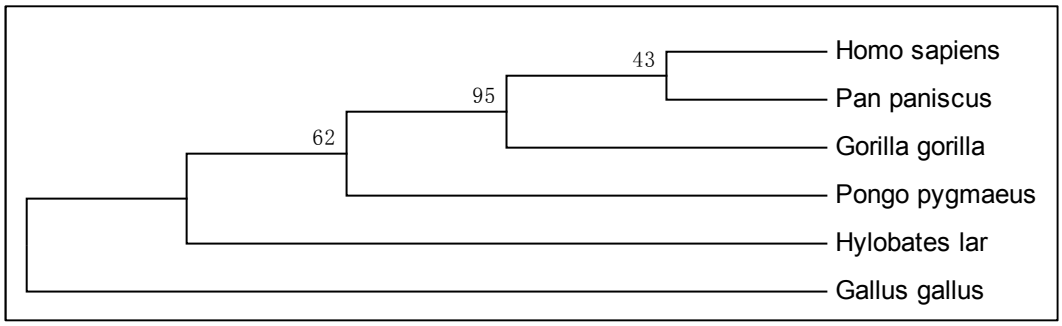
Full mitochondrial genome



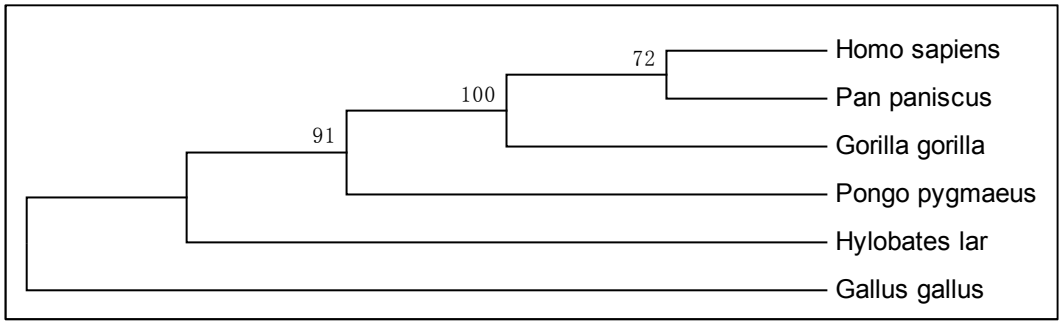
NADH dehydrogenase subunit 1



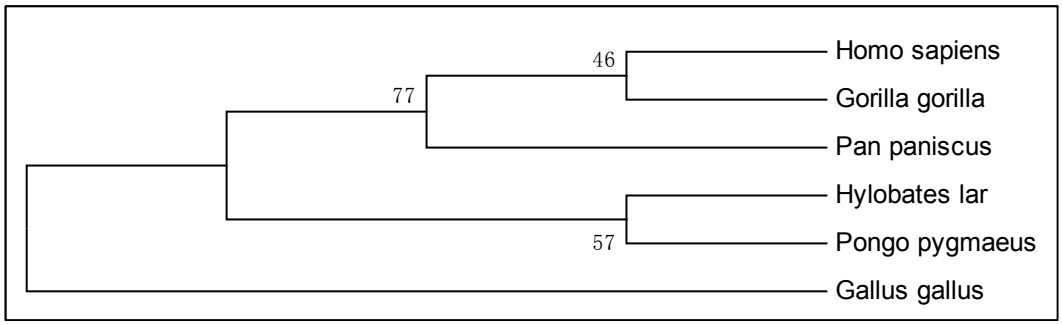
NADH dehydrogenase subunit 2



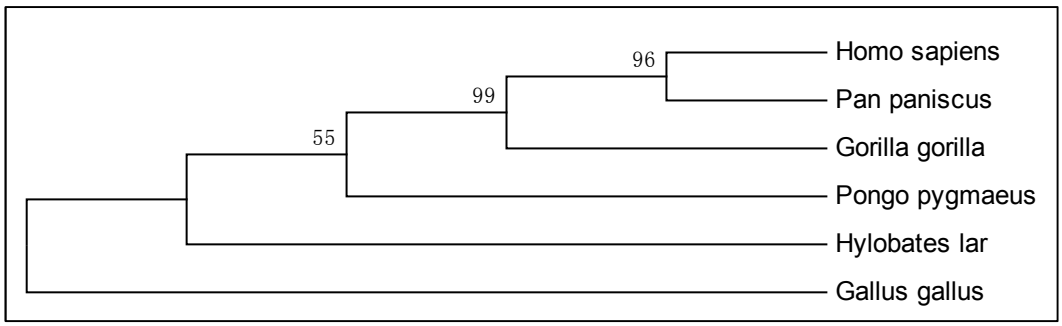
NADH dehydrogenase subunit 3



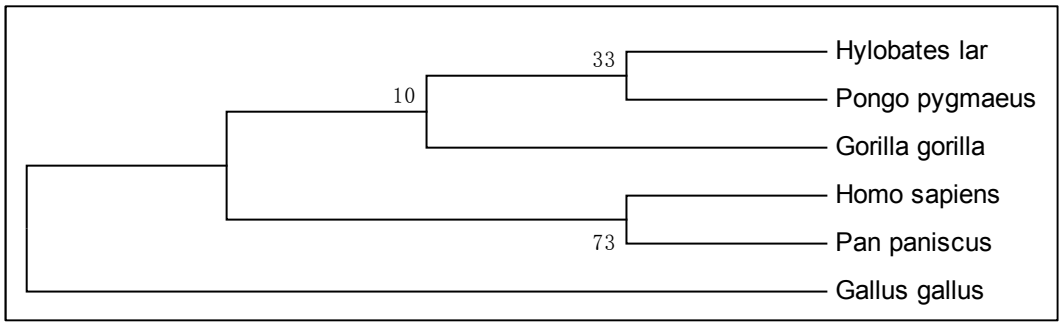
NADH dehydrogenase subunit 4



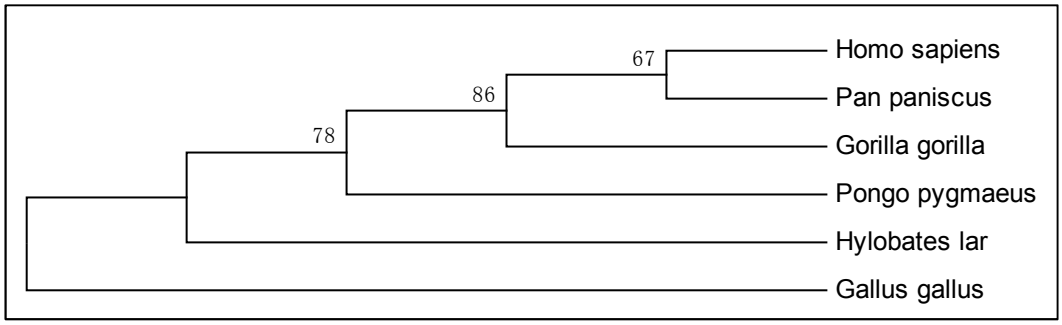
NADH dehydrogenase subunit 4L



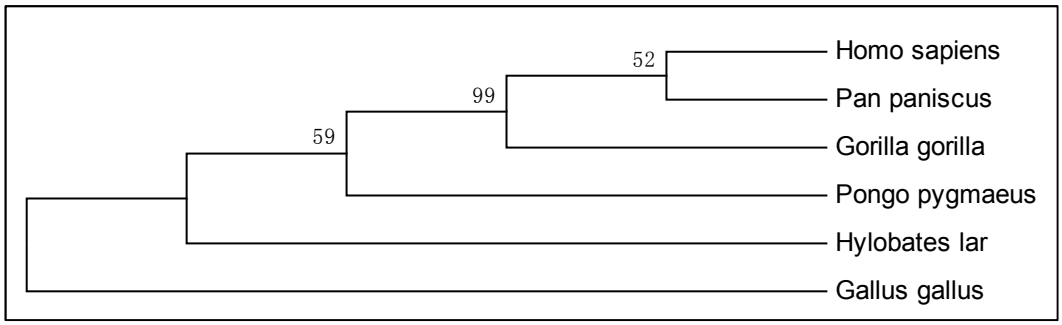
NADH dehydrogenase subunit 5



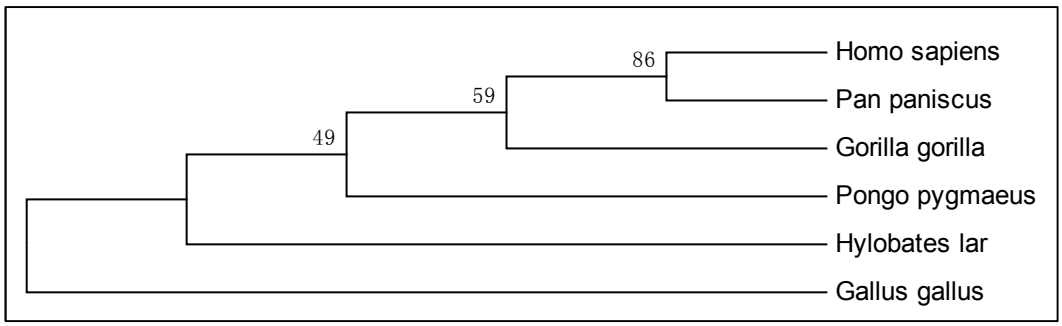
NADH dehydrogenase subunit 6



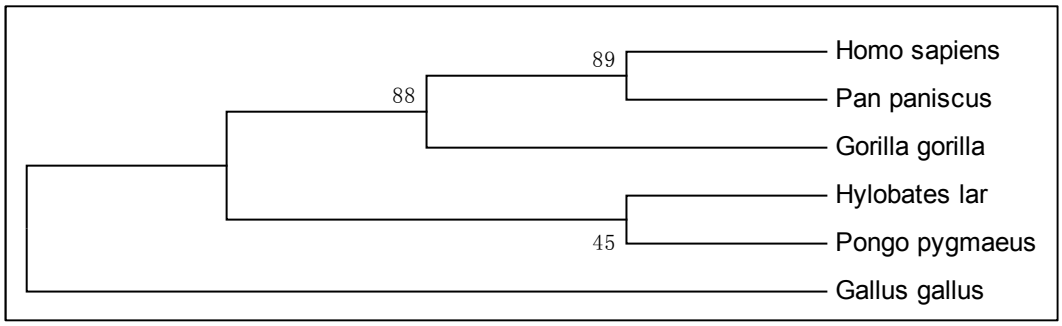
Cytochrome b



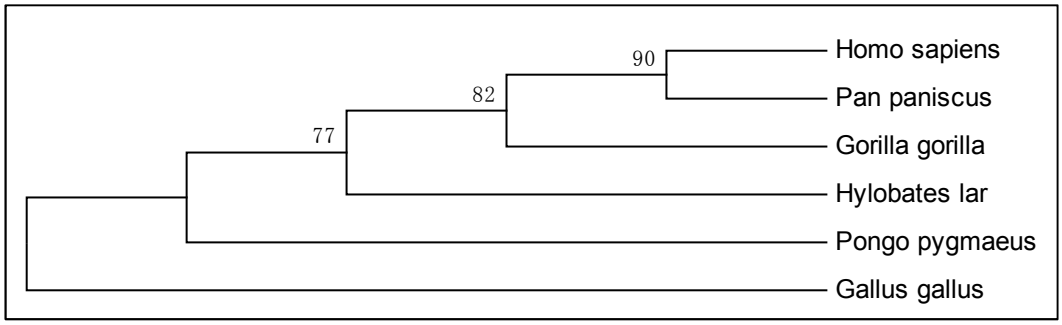
Cytochrome c oxidase subunit I



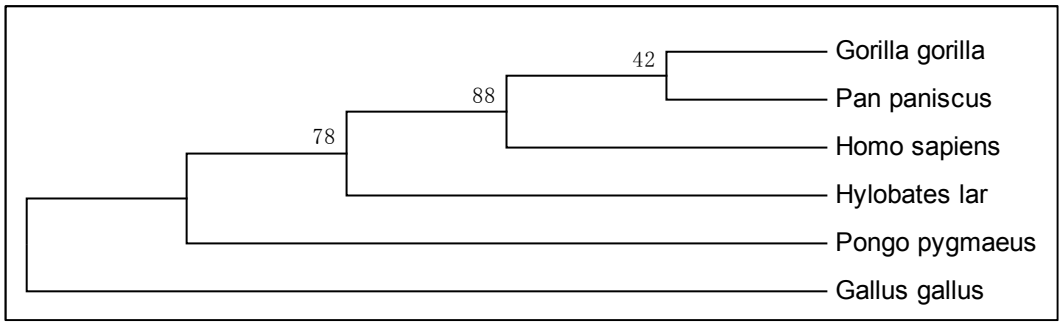
Cytochrome c oxidase subunit II



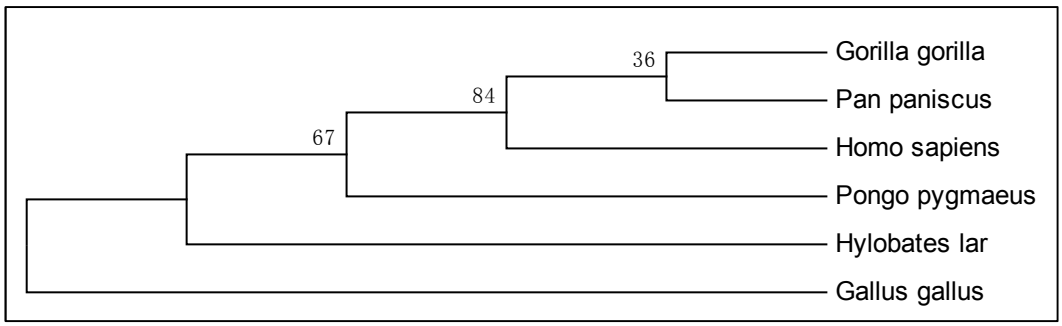
Cytochrome c oxidase subunit III



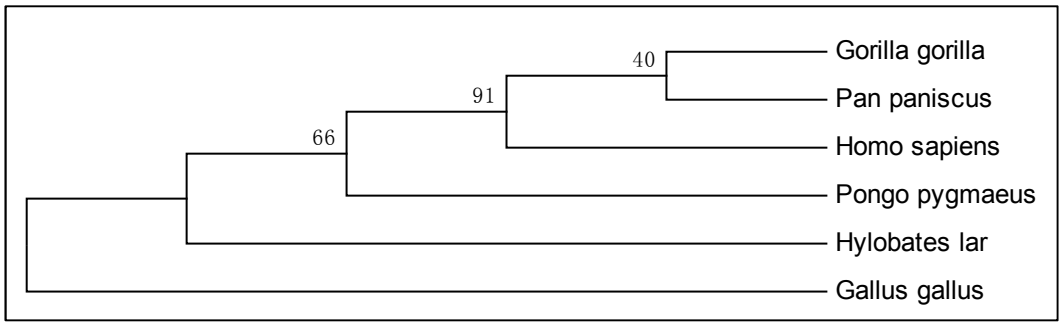
ATP synthase F0 subunit 6



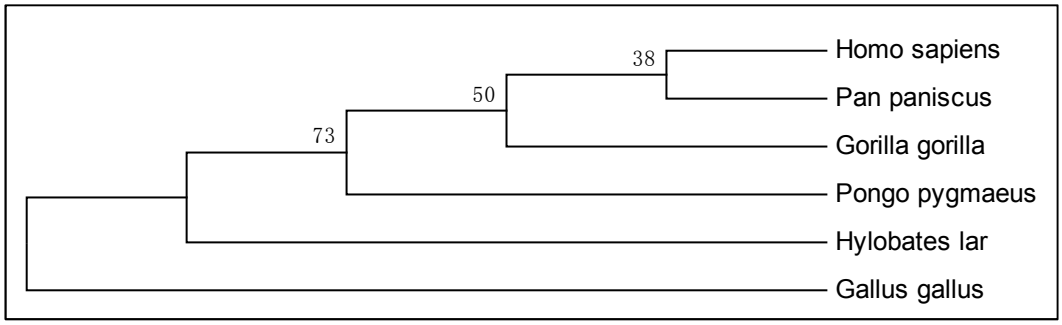
ATP synthase F0 subunit 8



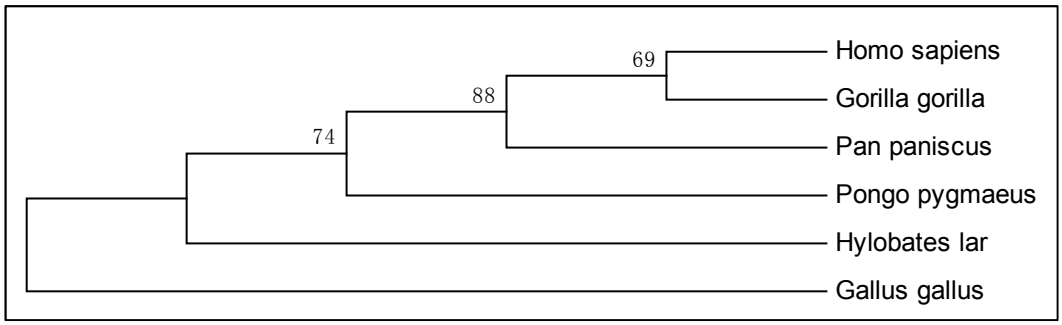
12S rRNA



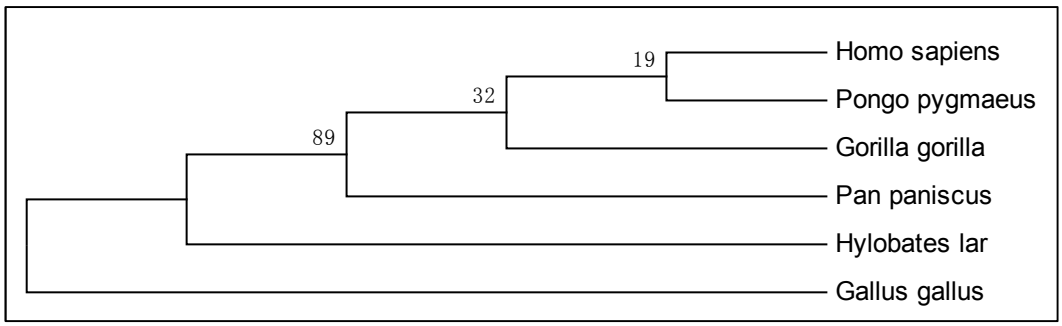
16S rRNA



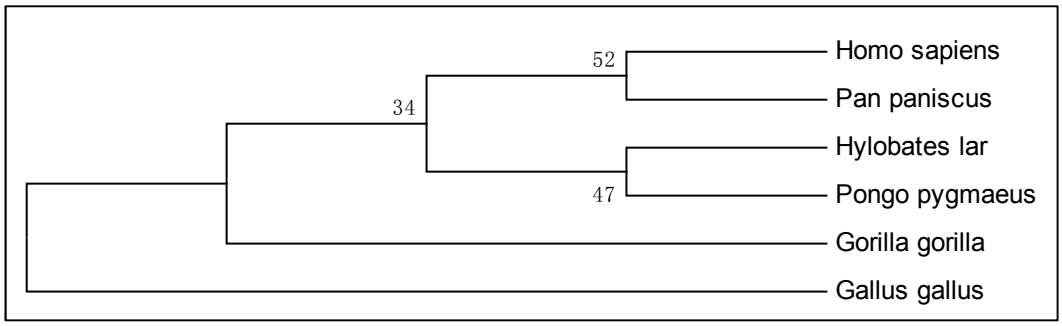
tRNA – Phe



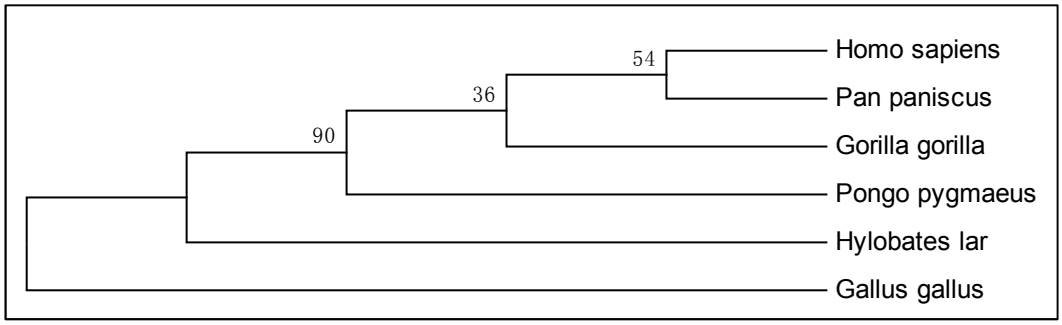
tRNA – Val



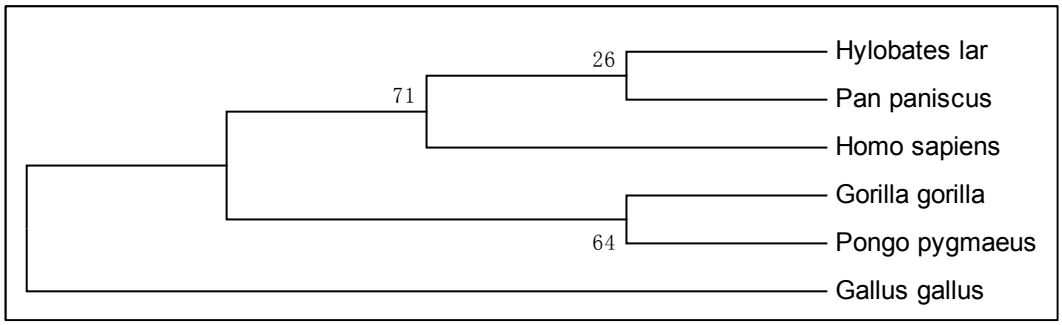
tRNA – Leu (codon = “UUR”)



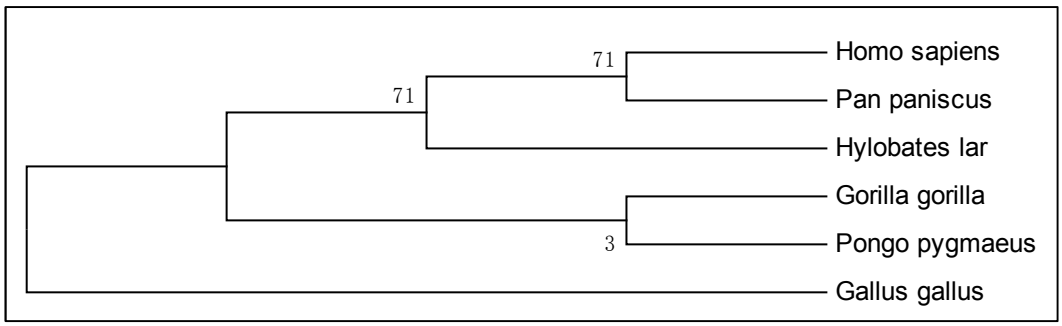
tRNA – Ile



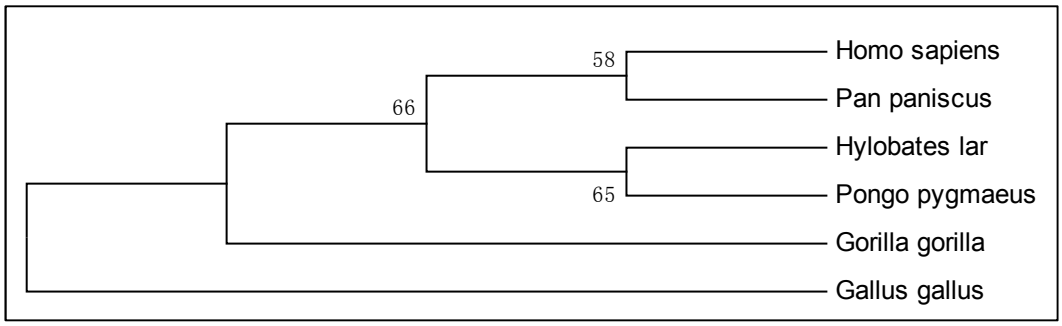
tRNA – Gln



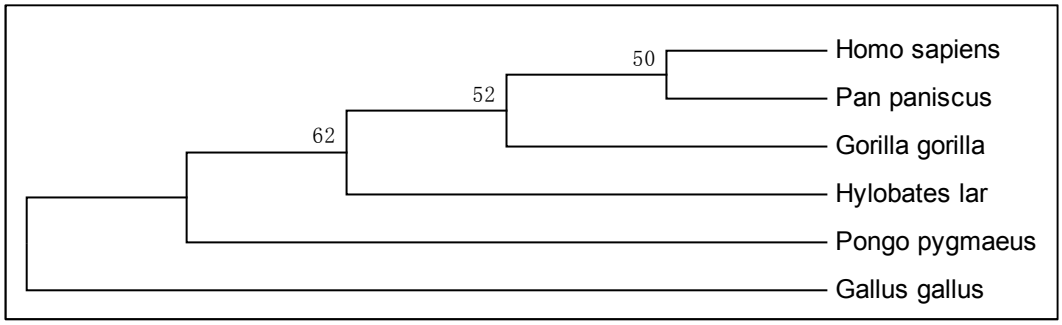
tRNA – Met



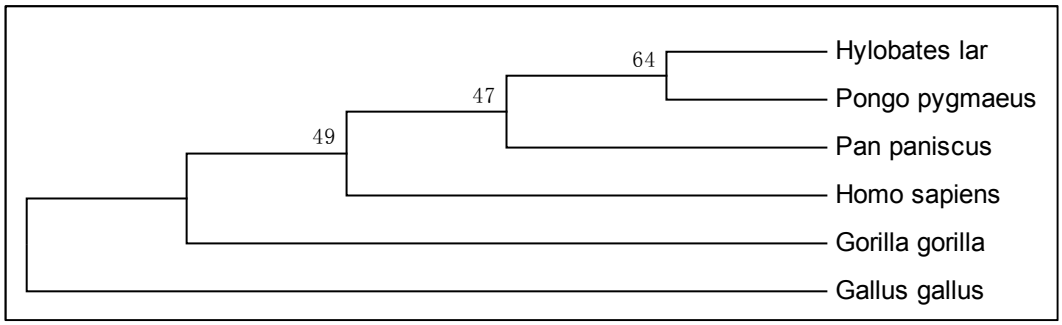
tRNA – Trp



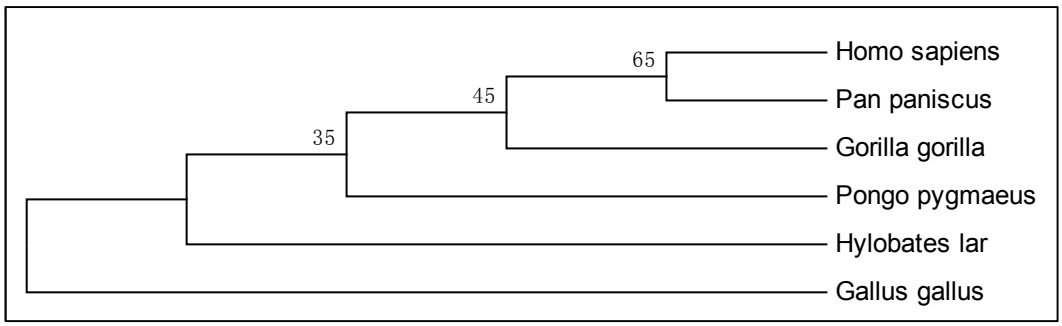
tRNA – Ala



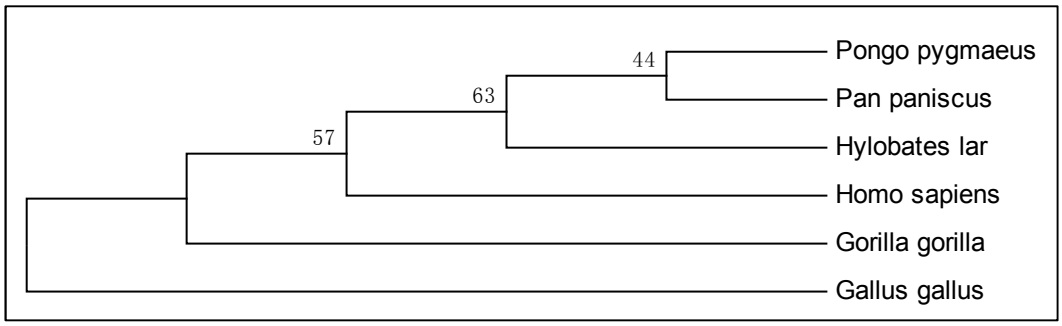
tRNA – Asn



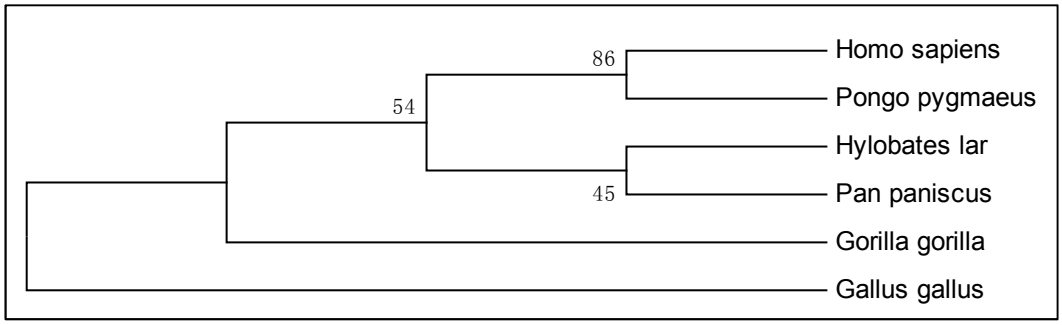
tRNA – Cys



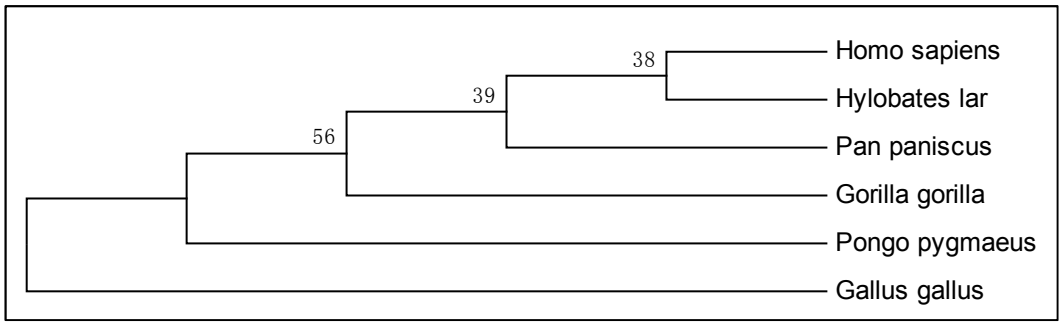
tRNA – Tyr



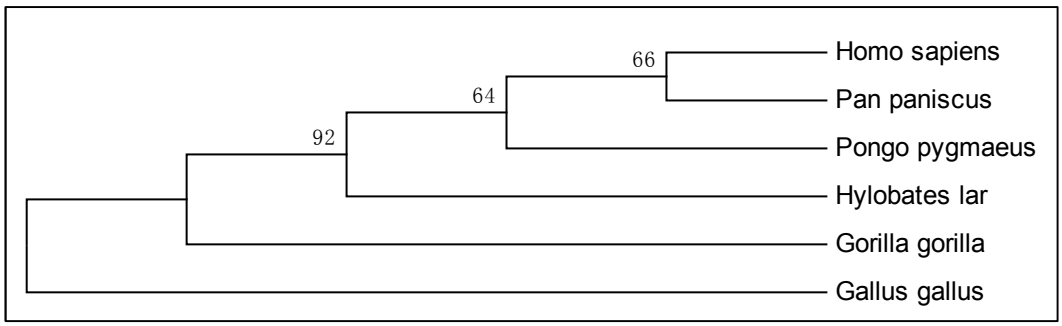
tRNA – Ser (codon – “UCN”)



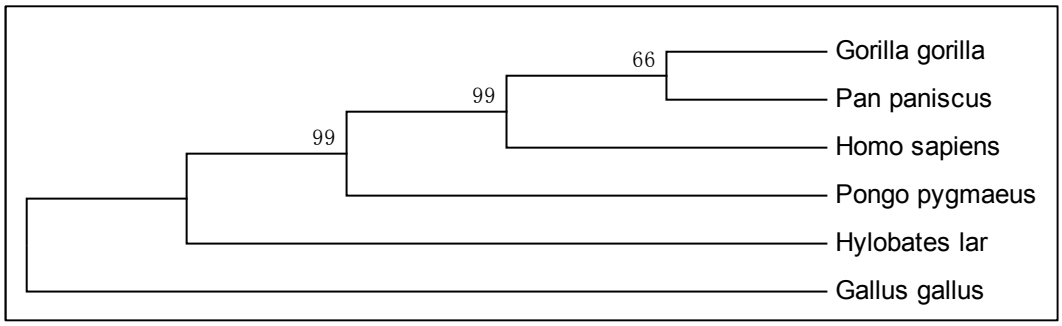
tRNA – Asp



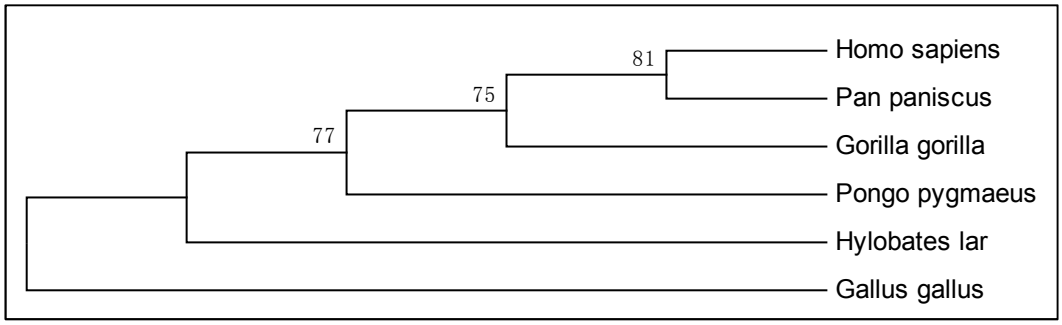
tRNA – Lys



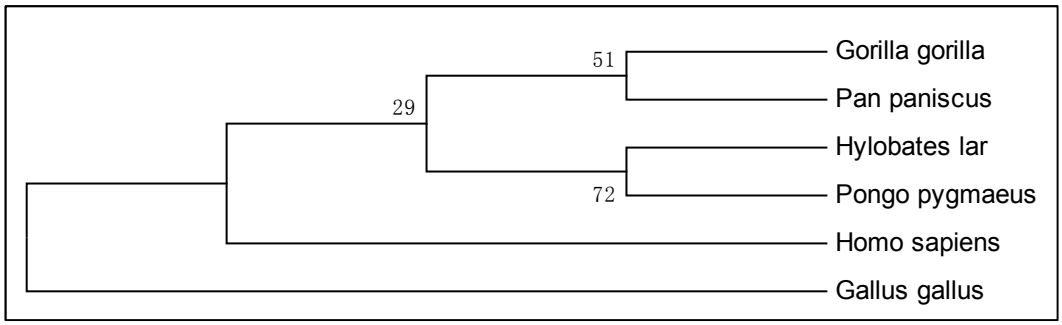
tRNA – Gly



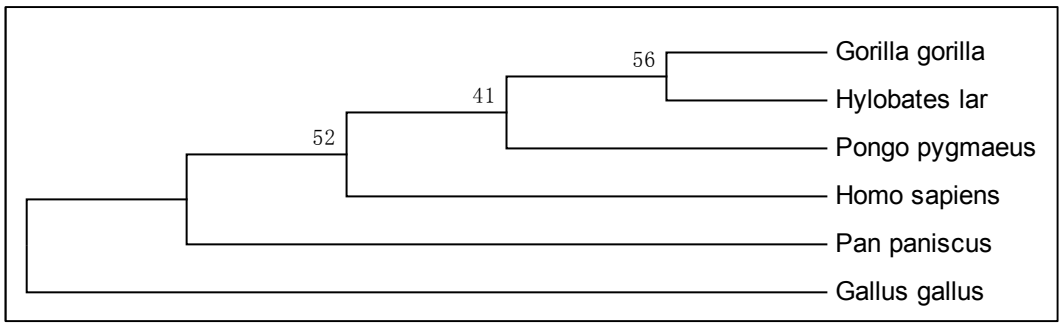
tRNA – Arg



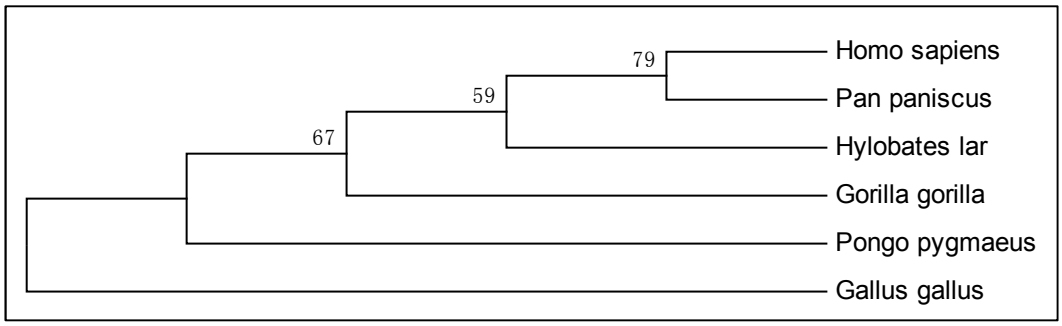
tRNA – His



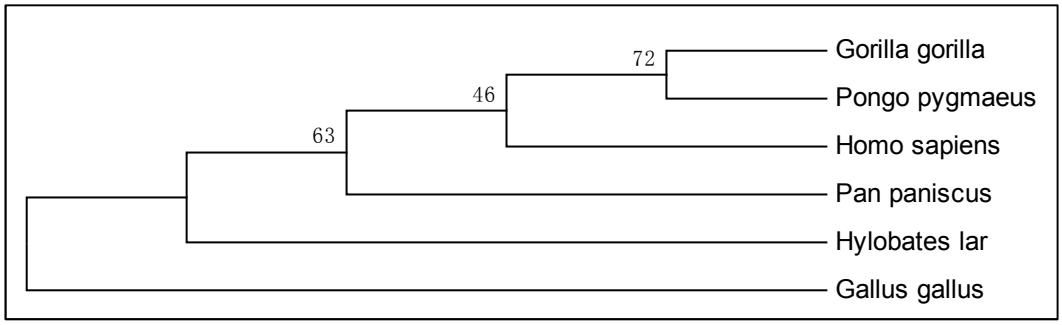
tRNA – Ser (codon – “AGY”)



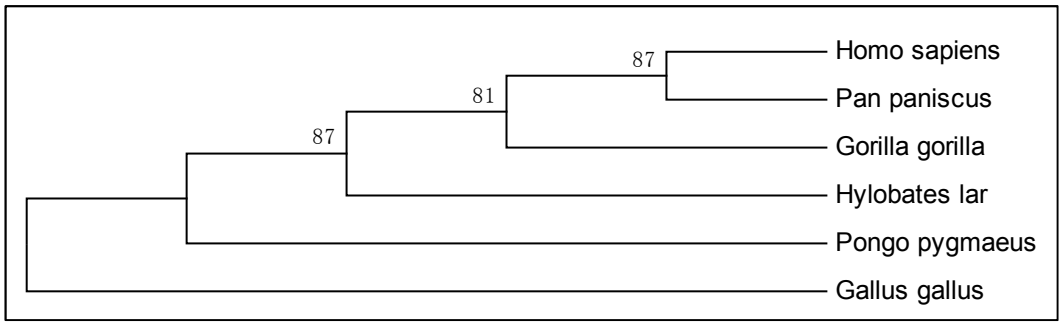
tRNA – Leu (codon – “CUN”)



tRNA – Glu



tRNA – Thr

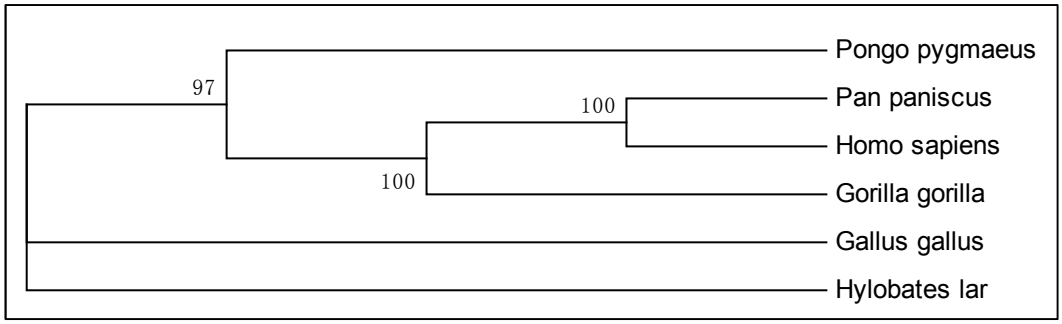


tRNA – Pro

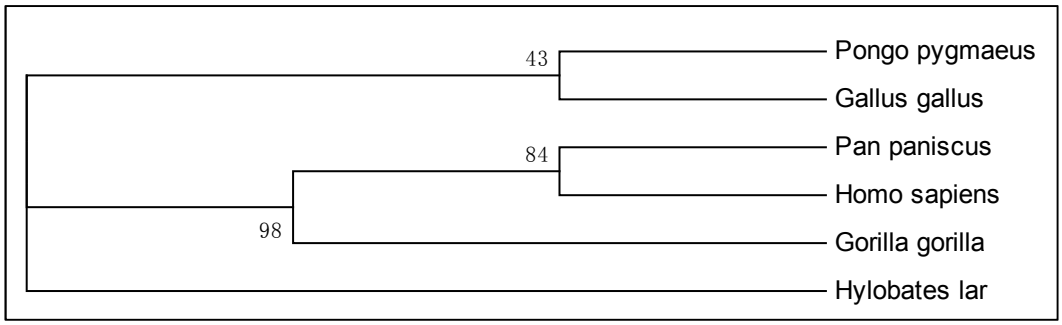
Note: All the trees are only showing topology. Distance is not included as the topology will be affected by it as there are some trees with clustered nodes due to the very large distance with the outgroup – *Gallus gallus*

Appendix E

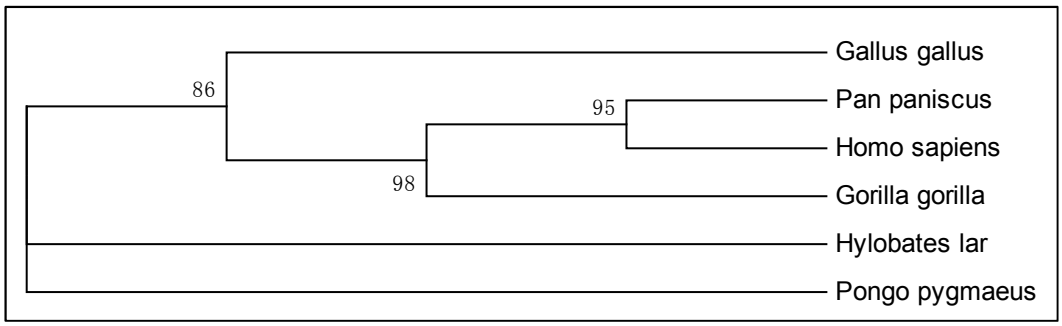
Topology of phylogenetics tress of MEGA5for all genes including mitochondrial genome



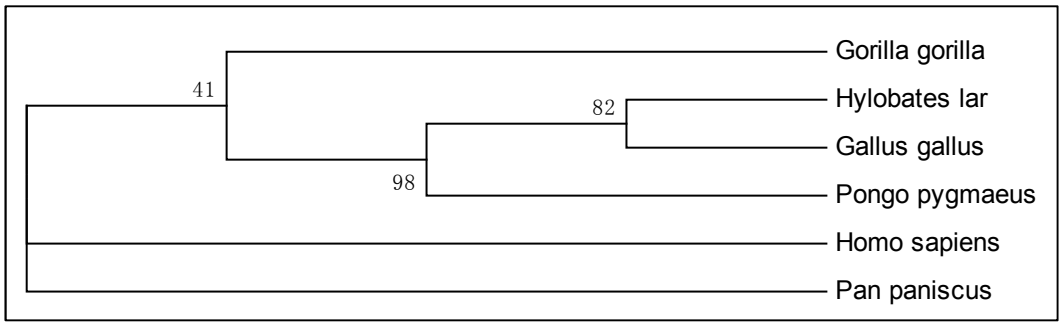
Full mitochondrial genome



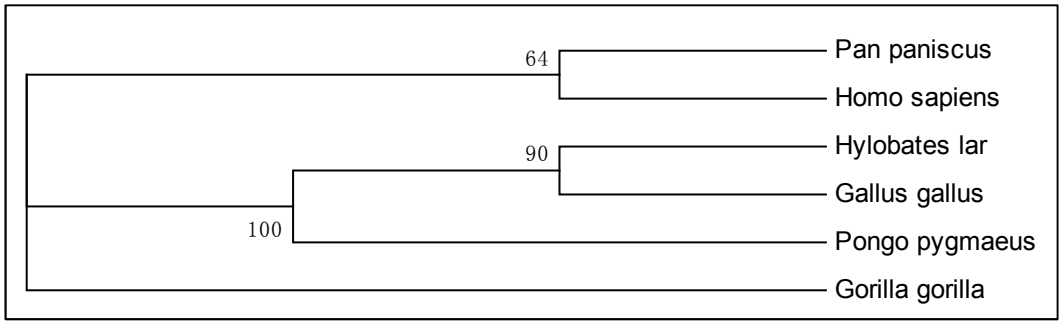
NADH dehydrogenase subunit 1



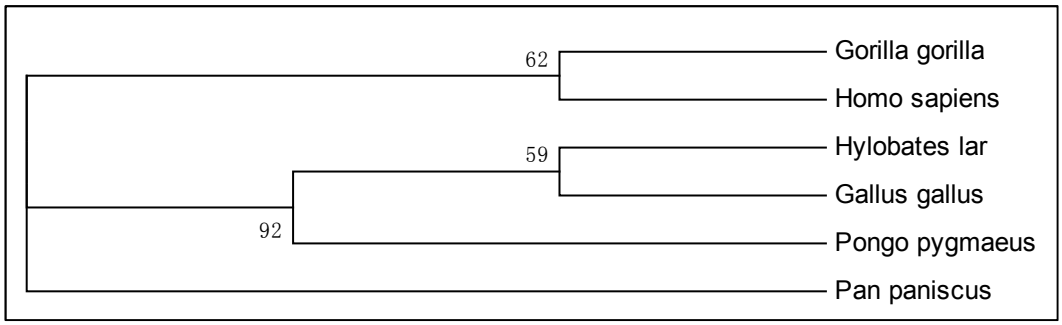
NADH dehydrogenase subunit 2



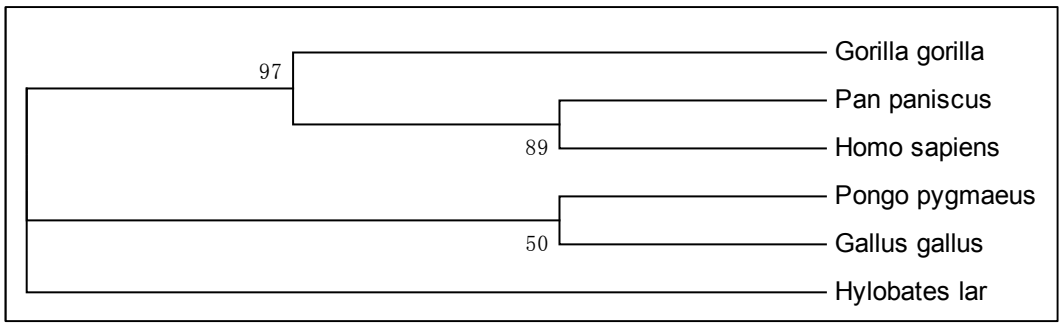
NADH dehydrogenase subunit 3



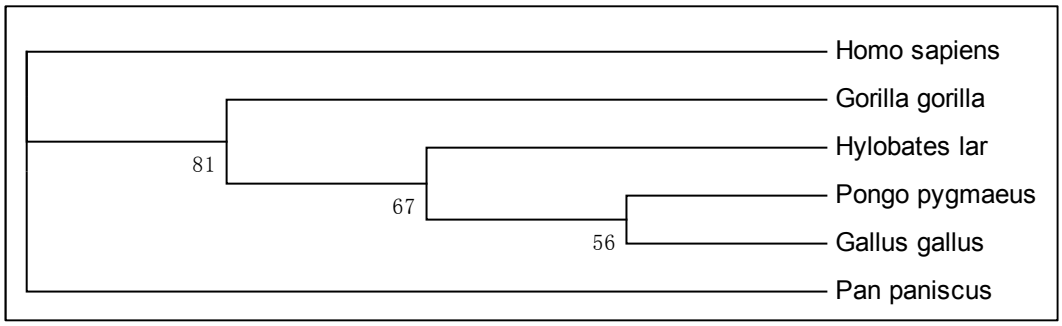
NADH dehydrogenase subunit 4



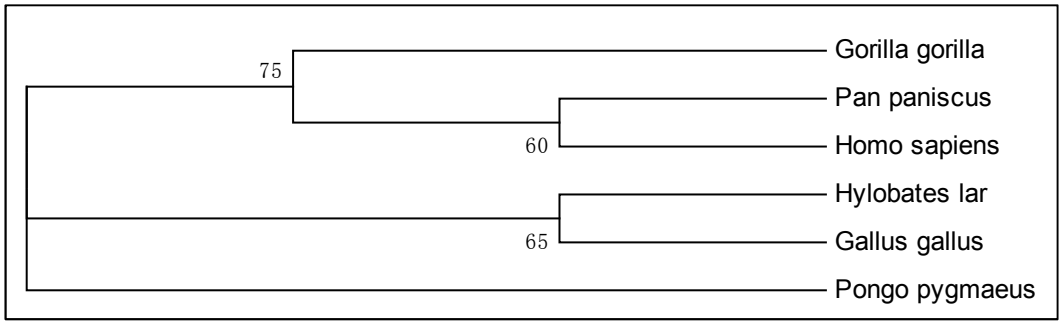
NADH dehydrogenase subunit 4L



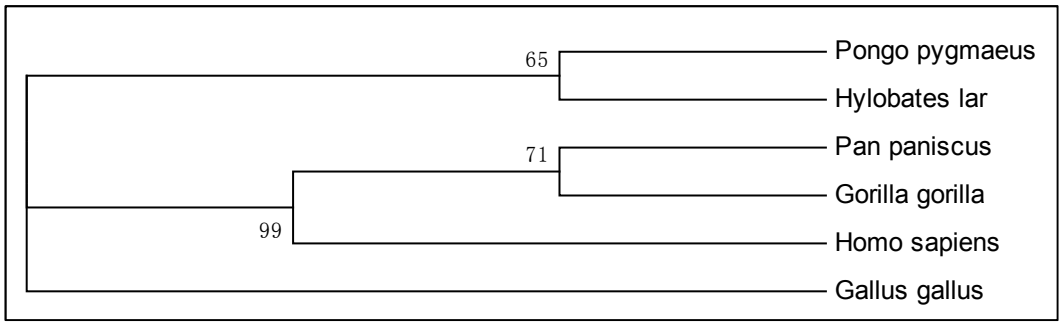
NADH dehydrogenase subunit 5



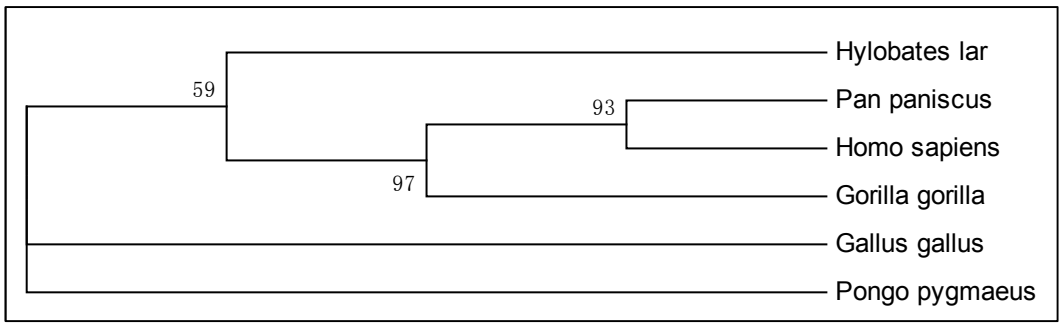
NADH dehydrogenase subunit 6



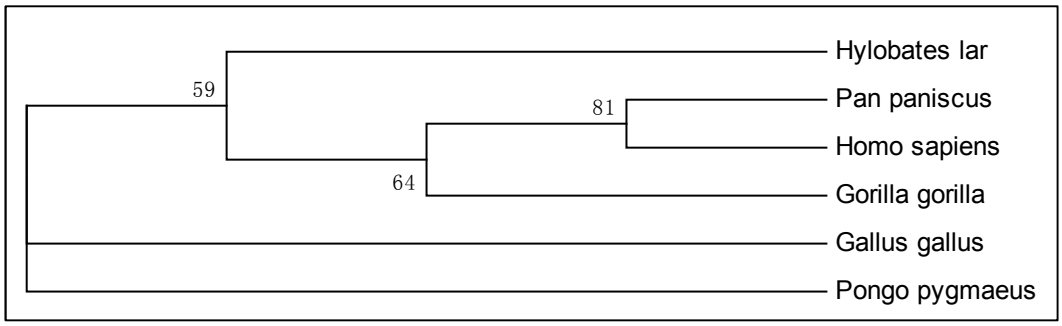
Cytochrome b



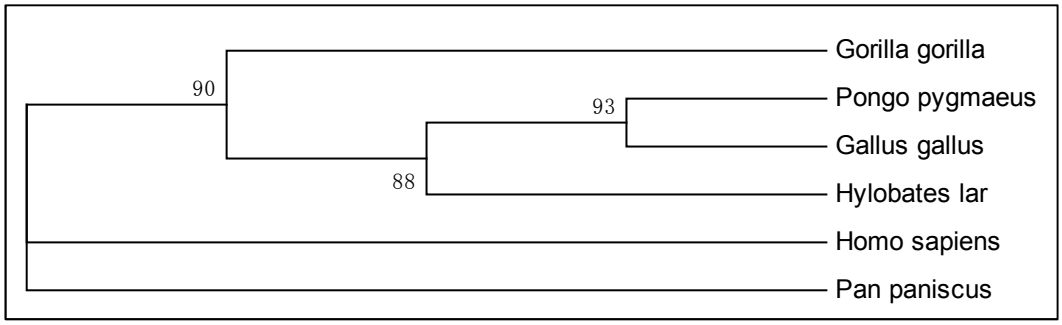
Cytochrome c oxidase subunit I



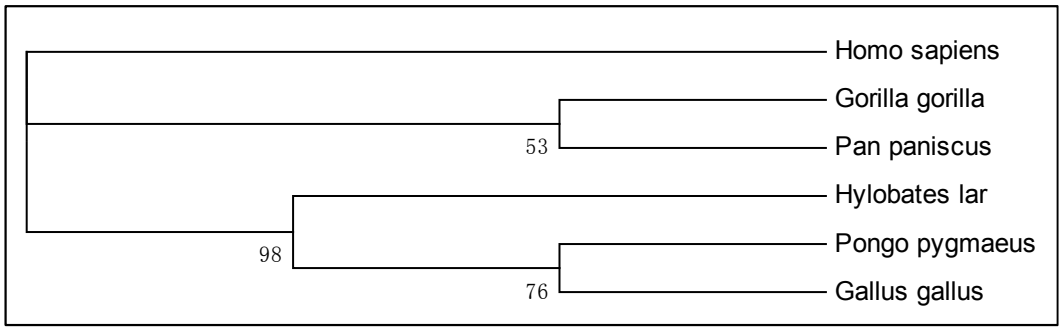
Cytochrome c oxidase subunit II



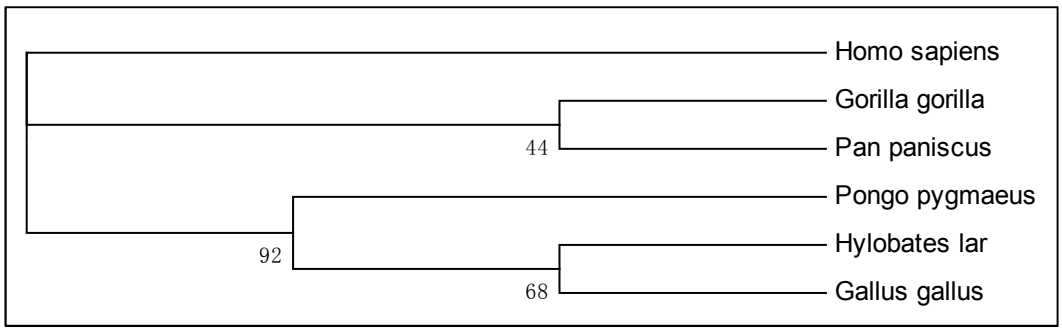
Cytochrome c oxidase subunit III



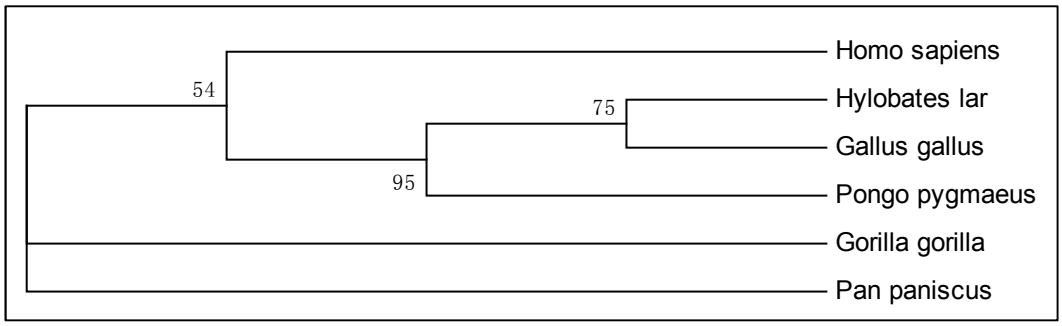
ATP synthase F0 subunit 6



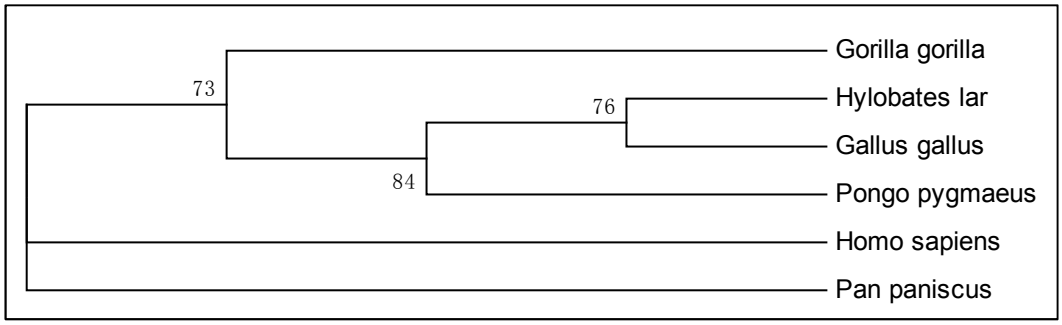
ATP synthase F0 subunit 8



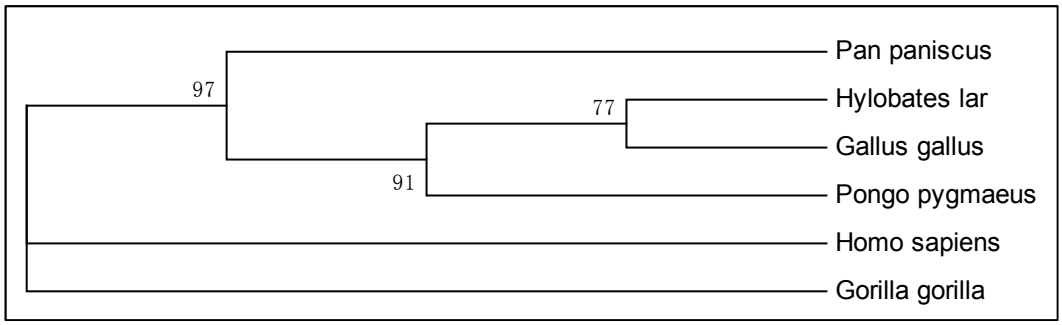
12S ribosomal RNA



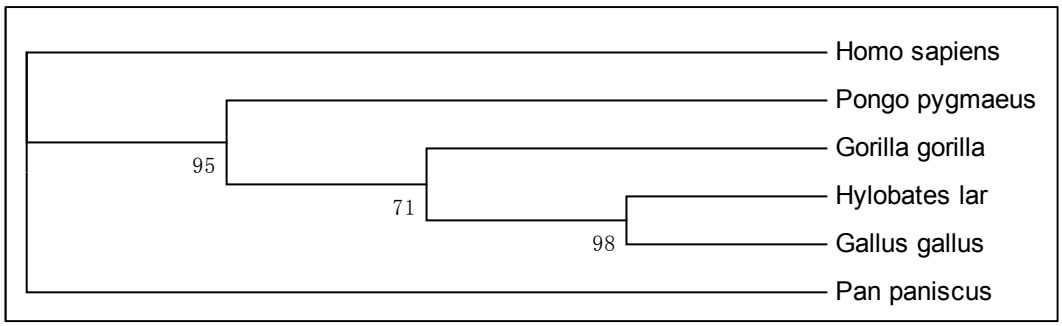
16S ribosomal RNA



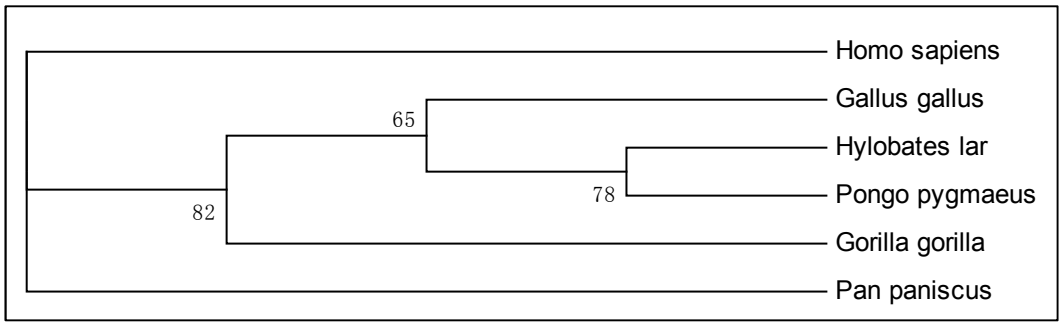
tRNA – Phe



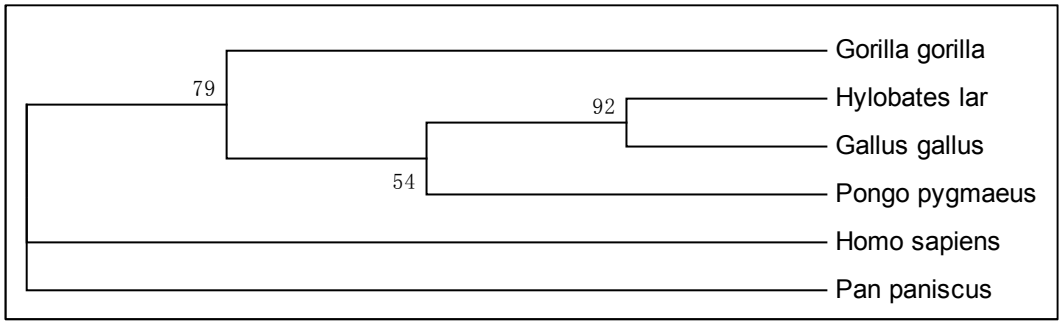
tRNA – Val



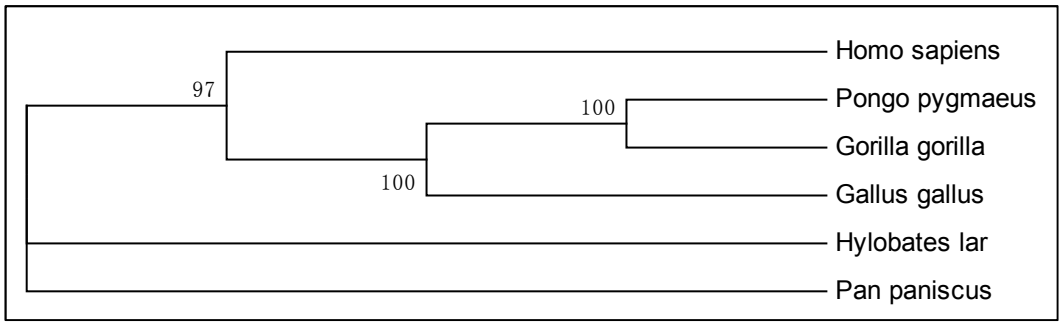
tRNA – Leu (codon – “UUR”)



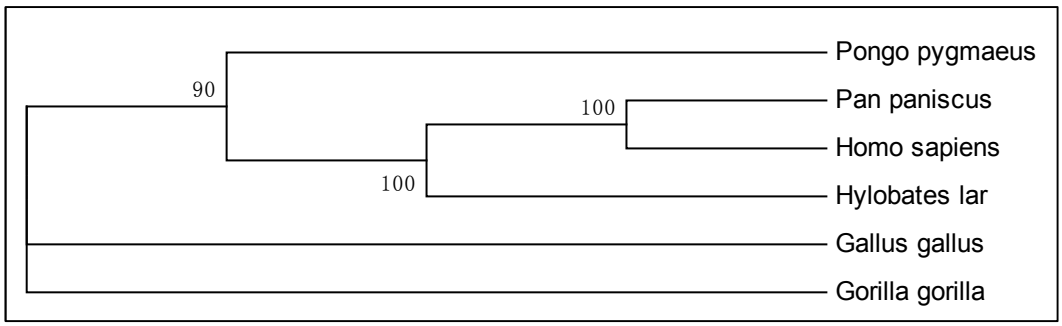
tRNA – Ile



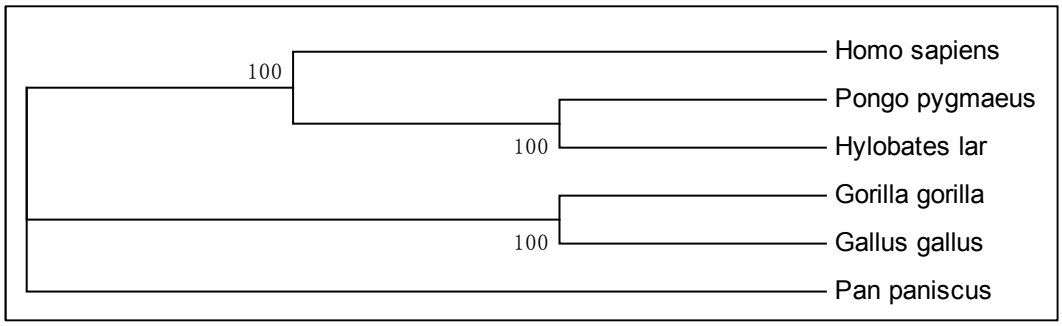
tRNA – Gln



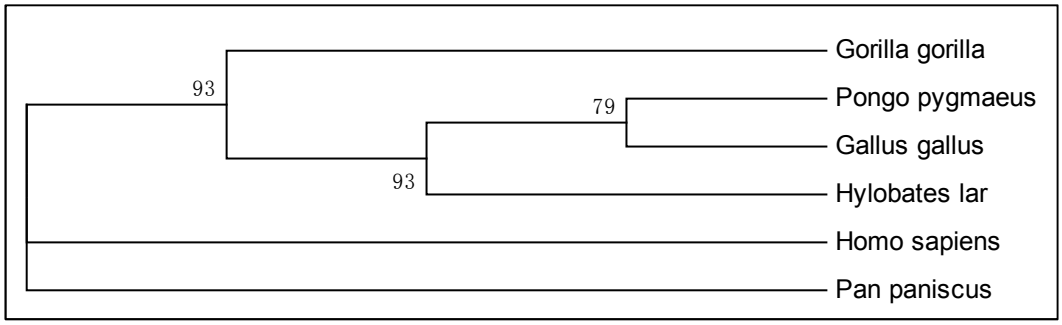
tRNA – Met



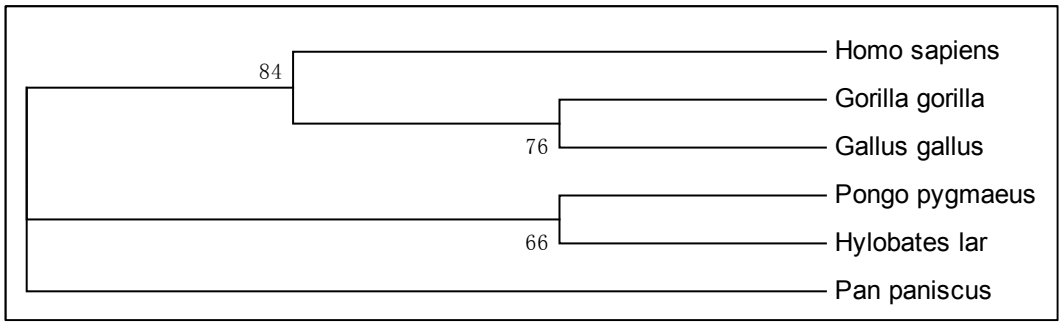
tRNA – Trp



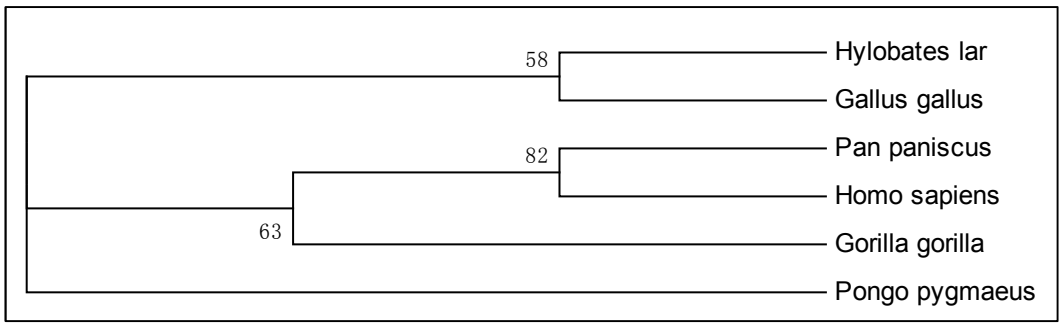
tRNA – Ala



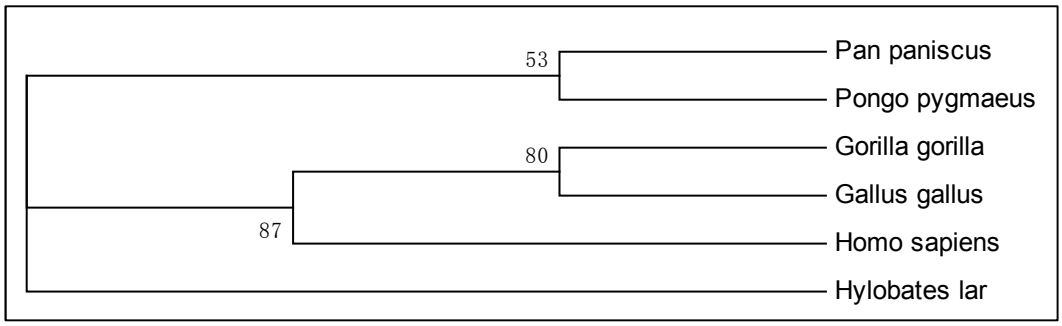
tRNA – Asn



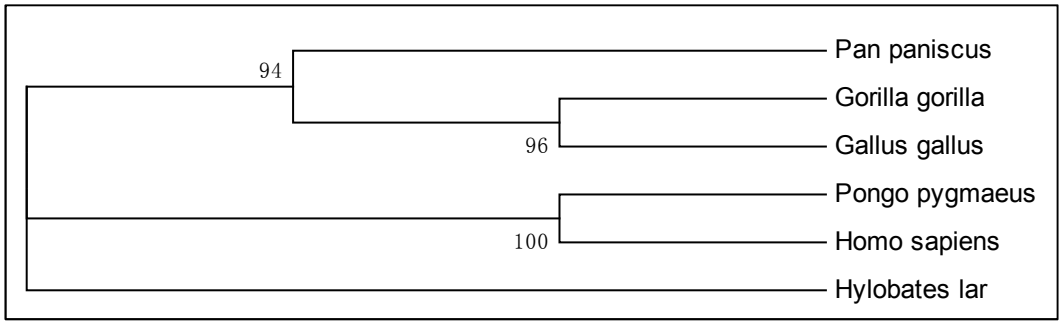
tRNA – Cys



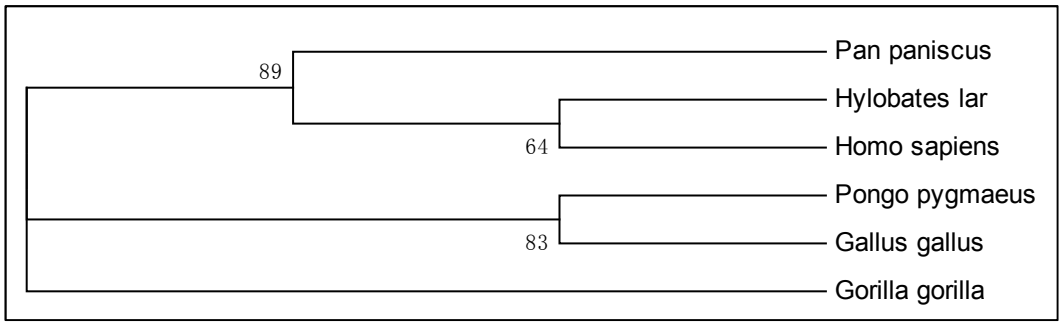
tRNA – Tyr



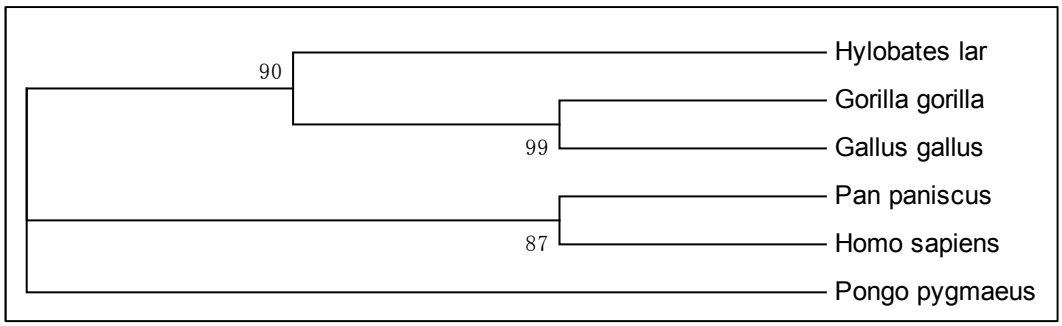
tRNA – Ser (codon – “UCN”)



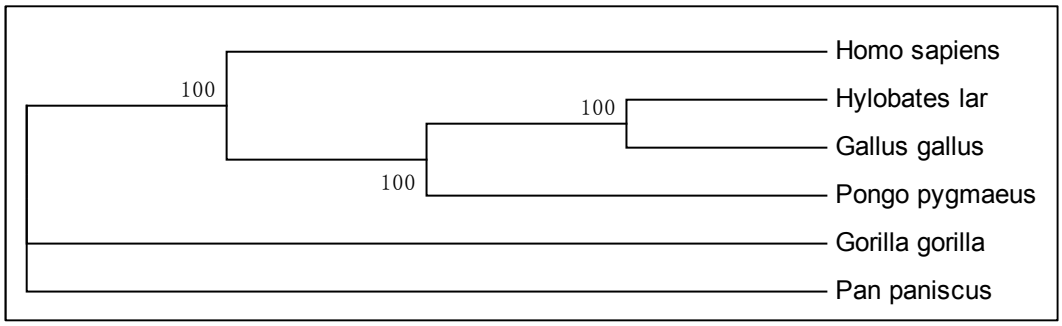
tRNA - Asp



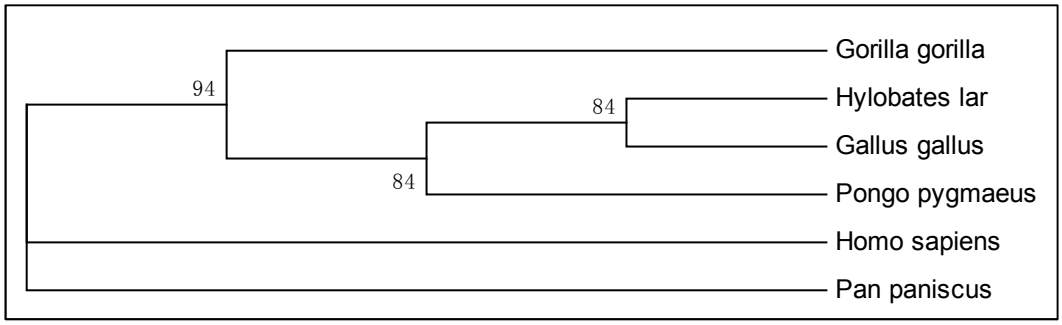
tRNA – Lys



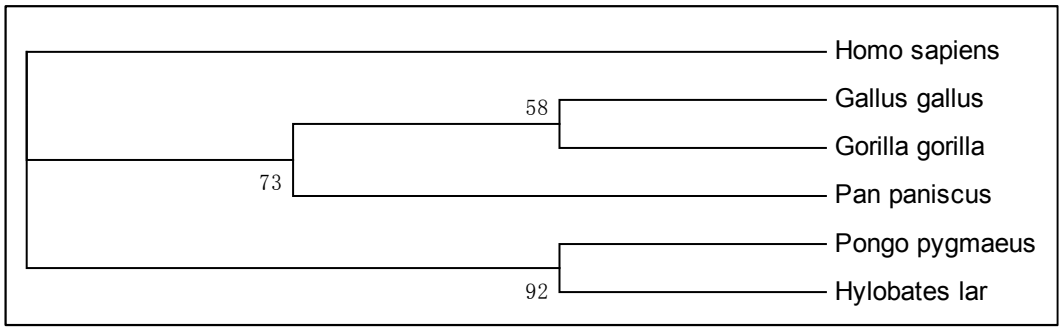
tRNA – Gly



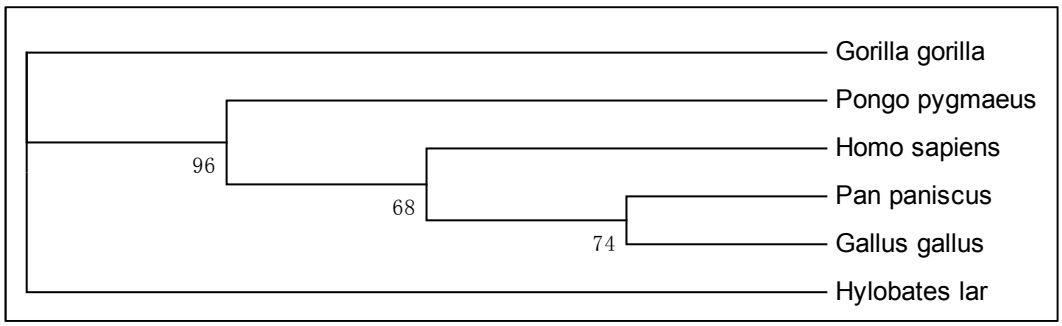
tRNA – Arg



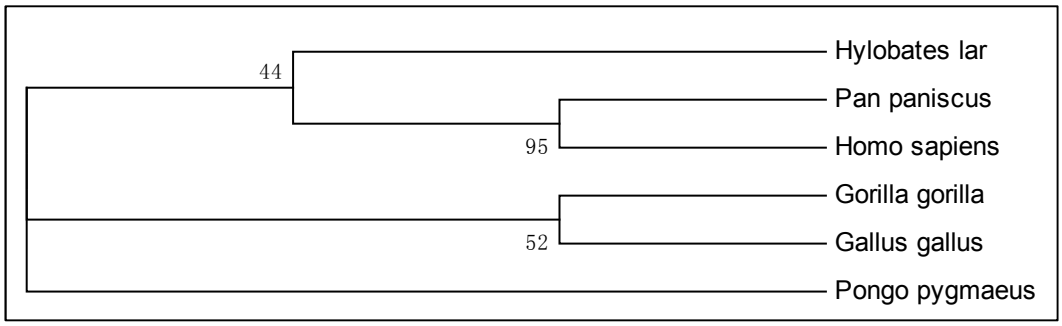
tRNA – His



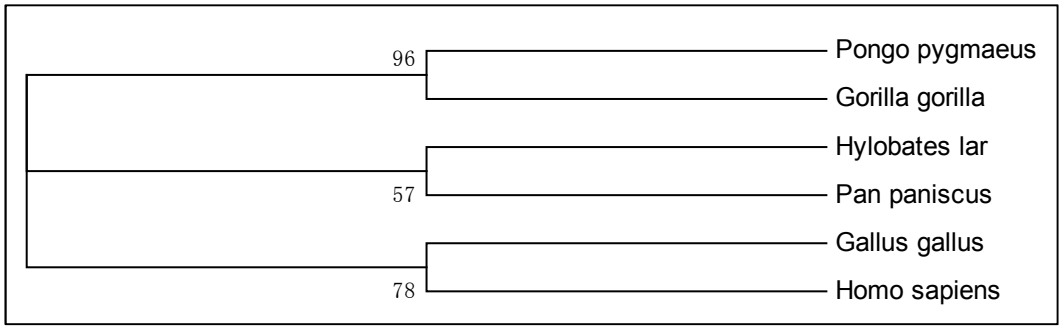
tRNA – Ser (codon – “AGY”)



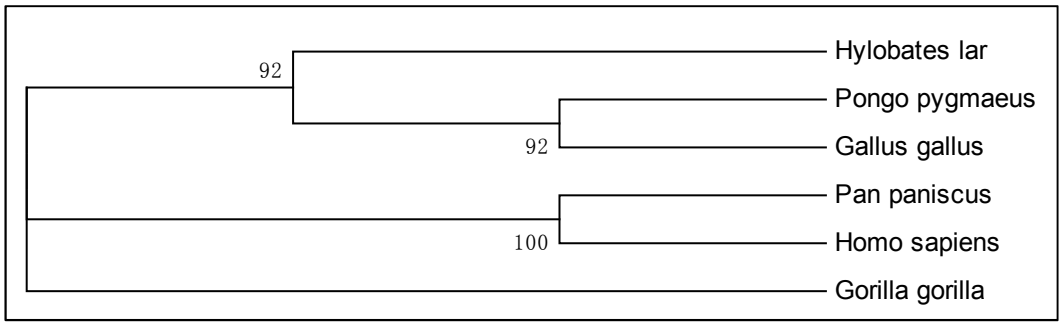
tRNA – Leu (codon – “CUN”)



tRNA – Glu



tRNA – Thr



tRNA – Pro

Note: All the trees are only showing topology. Distance is not included as the topology will be affected by it as there are some trees with clustered nodes due to the very large distance with the outgroup – *Gallus gallus*

Appendix F

PyCogent codes

```
#!/usr/local/bin/python


#started by T.F.Khang
#Institute of Biological Sciences
#University of Malaya,50603 Kuala Lumpur, MALAYSIA
#Email: tfkhang[at]um[dot]edu[dot]my
#August 2011
#################################################################
#written by W.H.So
#Email: so[dot]wei[dot]huo[at]um[dot]edu[dot]my
#     : so[dot]wei[dot]huo[at]gmail[dot]com
#for SHGS6280
#February 2012
#----------------------------------------------------------------
#<START> LIBRARY
#library for retrieval from ncbi
from cogent.db.ncbi import EFetch
from cogent.db.ncbi import EUtils
#library for parse sequence from genbank
from cogent.parse.genbank import RichGenbankParser
#library for alignment
from cogent import LoadSeqs
from cogent.app.muscle import align_unaligned_seqs as malign
from cogent.core.moltype import DNA
from cogent.core.moltype import PROTEIN
#library for elasped time
import time
#library for file manipulation
import re
# <END>  LIBRARY
#----------------------------------------------------------------
```

```
#-----------------------------------------------------------------------

#<START> GLOBAL VARIABLE

global totalTime #program start time

global path #initial path to read/write file

global taxaNumber #number of taxa

global seqNumber # number of sequences

global detailArrayAll #locus information

global fasta #all whole mitochodrion sequences

# <END>  GLOBAL VARIABLE

#-----------------------------------------------------------------------

#-----------------------------------------------------------------------

#<START> RETRIEVE SEGMENTS FROM SEQUENCE

#retrieve type from genbank format

def retrieveTypeInGenebank():

      gi                                                              =
['5835820','5835163','5835135','5835149','5834843','251831106']

      e = EUtils(db='nucleotide',rettype='gb')


      for i in range(0,len(gi)):

            record = e[gi[i]].readlines()

            parser = RichGenbankParser(record)

            accession, seq = [record for record in parser][0]


            writeDetailToFile(seq,i)


#parse/find and return type

def parseAndReturnType(f):

      return f['type'] == 'tRNA' or f['type'] == 'rRNA' or (f['type']
== 'CDS' and 'gene' in f)
```

```
#write details into file
def writeDetailToFile(seq,i):
	location = []
	features = [info for info in seq.Info.features if
parseAndReturnType(info)]
	f = open(path + '/locus/loc' + str(i) + '.txt','a')

	for feature in features:
		name = str(feature['product']).split('\'')[1]
		if 'l-rRNA' in name or 's-rRNA' in name:
			name = str(feature['note']).split('\'')[1]
		loc = str(feature['location']).replace('complement(','')
		loc = loc.replace(')','')
		loc = loc.replace('<','')
		location.append(name + '|' + loc.replace('..','#'))
	f.write('\n'.join(location))
	f.close()
# <END>  RETRIEVE SEGMENTS FROM SEQUENCE
#--------------------------------------------------------------------
#--------------------------------------------------------------------
#<START> SEGMENTS EXTRACTION
#populate global variable on locus
def populateDetailArrayAll():
	global detailArrayAll
	readFastaFromWholeMitoFile()
	for i in range(0,taxaNumber):
		detailArrayAll.append(readSegmentFromFile(i))

#read from the locus file
def readSegmentFromFile(i):
	detailArray = []

	f = open(path + '/locus/loc' + str(i) + '.txt','r')
```

```
	for line in f:
		tmp = line.replace('\n','').split('|') #name
		tmp[1] = tmp[1].split('#') #location
		tmp[1][0] = str(int(tmp[1][0])-1)
		tmp[1][1] = str(int(tmp[1][1])-1)
		detailArray.append(tmp)

	return detailArray

#view sequential segment
def viewSegmentSideBySide():
	cnt = 0
	#print species order
	print 'Order by (top to bottom):'
	for i in range(0,len(detailArrayAll)):
		print '> ' + getSpecies(i+1)
	print ''

	for i in range(0,len(detailArrayAll)): #get species (0:5)
		for j in range(0,len(detailArrayAll[i])): #get segment
(0:1)
			print detailArrayAll[i][j]#[0]
			print detailArrayAll[i+1][j]#[0]
			print detailArrayAll[i+2][j]#[0]
			print detailArrayAll[i+3][j]#[0]
			print detailArrayAll[i+4][j]#[0]
			print detailArrayAll[i+5][j]#[0]
			print '================================' + str(cnt)
			cnt += 1
		break
```

```
#compare between segments

def compareSegmentFromArray():

	readFastaFromWholeMitoFile()

	cnt = 0


	for j in range(0,len(detailArrayAll[0])): #all have same segment
size - 37

		if detailArrayAll[0][j][0] in detailArrayAll[1][j][0] and
detailArrayAll[0][j][0] in detailArrayAll[2][j][0] and
detailArrayAll[0][j][0] in detailArrayAll[3][j][0] and
detailArrayAll[0][j][0] in detailArrayAll[4][j][0] and
detailArrayAll[0][j][0] in detailArrayAll[5][j][0]:

			print '!S->' + str(cnt) + '..' +
detailArrayAll[0][j][0]

			writeSameSegmentFastaToFile(j)

			writeInfoToFile(detailArrayAll[0][j][0],j)

		else:

			print '@D->' + str(cnt) + '..' +
detailArrayAll[0][j][0]

			checkDetailArrayAll(detailArrayAll[0][j][0])

			writeInfoToFile(detailArrayAll[0][j][0],j)

		cnt += 1


#read from whole mitochondrion fasta file

def readFastaFromWholeMitoFile():

	global fasta

	cnt = 0

	fastaSeq = ''


	f = open(path + '/seq/seq0.fasta','r')


	for line in f:

		if '>' in line:

			if len(fastaSeq) > 0:

				fasta.append(fastaSeq)

			fasta.append(line.replace('\n',''))

			fastaSeq = ''
```

```
            else:
                  fastaSeq += line.replace('\n','')
      fasta.append(fastaSeq)
      for i in range(0,len(fasta)):
            print str(len(fasta[i]))


#write same segment at same index
def writeSameSegmentFastaToFile(j):
      f = open(path + '/seq/seq' + str(j+1) + '.fasta','a')
      fastaIndex = [1,3,5,7,9,11]


      for i in range(0,taxaNumber):
            cnt = fastaIndex[i]
            f.write('>' + getSpecies(i+1))
            f.write('\n')

      f.write(fasta[cnt][int(detailArrayAll[i][j][1][0]):int(detailArr
ayAll[i][j][1][1])])
            f.write('\n\n')
      f.close()


#check different segment at different locus from array
def checkDetailArrayAll(title):
      cnt = 0
      for i in range(0,len(detailArrayAll)):
            for j in range(0,len(detailArrayAll[i])):
                  if title in detailArrayAll[i][j]:
                        writeSegmentFastaToFile(i,j)


#write different segment at different index
def writeSegmentFastaToFile(i,j):
      f = open(path + '/seq/seq' + str(j+1) + '.fasta','a')
      fastaIndex = [1,3,5,7,9,11]
      cnt = fastaIndex[i]
      f.write('>' + getSpecies(i+1))
```

```
	f.write('\n')
	f.write(fasta[cnt][int(detailArrayAll[i][j][1][0]):int(detailArr
ayAll[i][j][1][1])])
	f.write('\n\n')
	f.close()


#write name with sequence number to file
def writeInfoToFile(title,i):
	f = open(path + '/readme.txt','a')
	f.write('seq' + str(i+1) + ': ' + title)
	f.write('\n')
	f.close()
# <END>  SEGMENTS EXTRACTION
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
#<START> ALIGNMENT
#align sequence using MUSCLE
def alignNWriteFasta():
	for i in range(36,37):#0,seqNumber):
		fn = path + '/seq/new-seq' + str(i) + '.fasta'
		afn = path + '/align-new/new-aln' + str(i) + 'gp500.fasta'
		unaligned = LoadSeqs(filename=fn, aligned=False)
		param = {'-gapopen':-500}
		aln = malign(unaligned, DNA, params=param)
		aln.writeToFile(afn)
# <END>  ALIGNMENT
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
#<START> FASTA
#retrieve whole mitochondrion genome from NCBI in fasta
def retrieveWholeMitoFastaFromNcbi():
	gi = '5835820,5835163,5835135,5835149,5834843,251831106'


	ef = EFetch(id=gi, rettype='fasta')
	f = open(path + '/raw/seq0raw.fasta','w')
```

```
	f.write(ef.read())
	f.close()

#rename fasta title to species
def renamingSpeciesInMitoFasta():
	cnt = 0
	f = open(path +'/seq/seq0.fasta','a')
	raw = open(path + '/raw/seq0raw.fasta').readlines()

	for line in raw:
		if '>' in line:
			cnt += 1
			line = '>' + getSpecies(cnt) + '\n'
		f.writelines(line)
	f.close()

# <END>  FASTA
#--------------------------------------------------------------------
#--------------------------------------------------------------------
#<START> MISCELLANEOUS
#return species name
def getSpecies(choice):
	if choice == 1:
		species = 'Hylobates lar' #lar gibbon
	elif choice == 2:
		species = 'Pongo pygmaeus' #bornean orangutan
	elif choice == 3:
		species = 'Pan paniscus' #bonobo/pygmy chimpanzee
	elif choice == 4:
		species = 'Gorilla gorilla' #western gorilla
	elif choice == 5:
		species = 'Gallus gallus' #chicken/red junglefowl
	elif choice == 6:
		species = 'Homo sapiens' #human
```

```
	return species


#print header
def printHeader():
	print '-------------------------------------------------------'
	print 'Research Project(SHGS6280) - So Wei Huo (SGJ10005)'
	print ''
	print '  An empirical assessment of the bootstrap support '
	print '     as an indicator of robustness in phylogenetic trees'


#print choice selection
def printChoice():
	print '-------------------------------------------------------'
	print ' 1. Retrieve gene or rna with its locus from genbank format'
	print ' 2. View sequential segment between taxa (raw format)'
	print ' 3. Compare similar segment & write to separate fasta file'
	print ' 4. Align sequences using Muscle in PyCogent'
	print '10. Retrieve whole mitochondrion genome in FASTA format'
	print '99. <Starter guide>'
	print '-------------------------------------------------------'


#print footer
def printFooter():
	print "\n--Thank you--"
	print '-------------------------------------------------------'


#print time elasped
def printTimeElasped(timeStart):
	m, s = divmod(time.time() - timeStart, 60) #calculate minutes
	h, m = divmod(m, 60) #calculate hours
	print ''
	print '*************************'
```

```
      print '* Time elasped   ' + "%02d:%02d:%02d" % (h,m,s)   + ' *'
#format time

      print '*************************'


#print total time elasped

def printTotalTimeElasped():

      m, s = divmod(time.time() - totalTime, 60) #calculate minutes

      h, m = divmod(m, 60) #calculate hours

      print ''

      print '@+++++++++++++++++++++++++++++++@'

      print '@                               @'

      print '@ TOTAL time elasped ~' + "%02d:%02d:%02d" % (h,m,s)   +
'~ @'

      print '@                               @'

      print '@+++++++++++++++++++++++++++++++@'

# <END>  MISCELLANEOUS

#--------------------------------------------------------------------

printHeader()

switch = True

#Initialize global variables

totalTime = time.time() #total time start

path = '../sequence' #path to sequence directory

taxaNumber = 6 #total taxa used

seqNumber = 38 #total sequences produced

detailArrayAll = [] #keep all segment detail

fasta = [] #all whole mitochondrion sequences

      #(format:                    [species][segment][name|locus(@index
1)][start/stop(only locus)])

while switch:

      timeStart = time.time() #time start

      printChoice()

      methodChoice = input('Choose method: ')

      print ''


      if methodChoice == 1:

            retrieveTypeInGenebank()
```

```
elif methodChoice == 2:
      populateDetailArrayAll()
      viewSegmentSideBySide()
elif methodChoice == 3:
      populateDetailArrayAll()
      compareSegmentFromArray()
elif methodChoice == 4:
      alignNWriteFasta()
elif methodChoice == 10:
      retrieveWholeMitoFastaFromNcbi()
      renamingSpeciesInMitoFasta()
elif methodChoice == 99:
      print '<Guide is here>'
      readFastaFromWholeMitoFile()
else:
      print 'Please choose between 1 and 2 only'

printTimeElasped(timeStart)

#reinitialize array
detailArrayAll = [] #keep all segment detail
fasta = [] #all whole mitochondrion sequences
againInput = input('\nContinue (Yes=1; No=0)? ')

if againInput == 0:
      switch = False
      printTotalTimeElasped()
      printFooter()
      break
else:
      print '------------------------------------------------'
      print '>> Program restarting ...'
```

Appendix G

R codes: Building Tree Using APE and phangorn package

```
#import packages
library(ape)
library(phangorn)

#read the aligned sequence
alnFunc <- function(fileName){
  aln <- read.dna(fileName,format='fasta')
  return(aln)
}

#calculate distance using pairwise
distFunc <- function(aln){
  dist <- dist.dna(aln,as.matrix=TRUE,model='TN93')
  return(dist)
}

#build a nj tree
treeFunc <- function(dist){
  ldist <- log(dist)
  if(any(is.infinite(ldist))){ #test if any infinite value
    tree <- nj(dist)}
  else{
    tree <- nj(ldist)}
  return(tree)
}
```

```
#build a ml tree
mlFunc <- function(tree,aln){
  gtr = pml(tree,data=as.phyDat(aln),model='GTR')
  gtr = optim.pml(gtr,optNni=TRUE)
  return(gtr)
}

#bootstrap support for ml tree
bsFunc <- function(gtr){
  bs <- bootstrap.pml(gtr, bs=1000, optNni=TRUE)
  return(bs)
}

#plot bootstrap tree
plotBSFunc <- function(gtr,bs){
  fnp <- paste('../plot-new/new-plot',i,'.eps',sep="")
  postscript(fnp)

  bsTree <- plotBS(gtr$tree,bs, type="unrooted", cex=0.7,
bs.adj=c(0.5,-0.3))
  add.scale.bar(cex=0.6)

  dev.off()
  print(paste(fn,'written',sep=" "))

  return(bsTree)
}

#write tree in text
writeTreeTxtFunc <- function(bsTree,i){
  fn <- paste('../tree/new-tree',i,'.tre',sep="")
  write.tree(bsTree,file=fn)
  print(paste(fn,'written',sep=" "))
}
```

```
for(i in 0:37){
  fn <- paste('new-aln',i,'gp500.fasta',sep="")
  print(fn)
  aln <- alnFunc(fn)
  tree <- treeFunc(distFunc(aln))
  gtr <- mlFunc(tree,aln)
  bs <- bsFunc(gtr)
  bsTree <- plotBSFunc(gtr,bs)
  writeTreeTxtFunc(bsTree,i)
}
```

R codes: Building graphs using R

```
#building graph for protein
protein.dist <- read.csv(file="dist-protein.csv",sep="|")
protein.length <- read.csv(file="length-protein.csv",sep="|")
protein.dist <- protein.dist[-1]
protein.length <- protein.length[-1]

pa <- cbind(rowMeans(protein.length),protein.dist[1])
pb <- cbind(rowMeans(protein.length),protein.dist[2])
pc <- cbind(rowMeans(protein.length),protein.dist[3])

#buiding division for histogram
par(mfrow=c(2,2))

hist(pa[which(pa[2]>0),-1],main="(a) Bootstrap Value vs Frequency" ,
xlab="Node A (Protein & rRNA)", xlim=c(0,100),ylim=c(0,7))
abline(v=60, lty=2)
hist(pb[which(pb[2]>0),-1],main="(b) Bootstrap Value vs Frequency" ,
xlab="Node B (Protein & rRNA)", xlim=c(0,100),ylim=c(0,7))
abline(v=43, lty=2)
hist(pc[which(pc[2]>0),-1],main="(c) Bootstrap Value vs Frequency" ,
xlab="Node C (Protein & rRNA)", xlim=c(0,100),ylim=c(0,7))
```

```
abline(v=40, lty=2)

#building distribution plot
plot(pa[which(pa[2]>0),-length(pa)],pa[which(pa[2]>0),-1],pch=1,
main="Distribution of Nodes (Protein & rRNA)", ylab="Boostrap Value",
xlab="Sequence Length",ylim=c(0,100),xlim=c(0,2000))
points(pb[which(pb[2]>0),-length(pb)],pb[which(pb[2]>0),-1],pch=15)
points(pc[which(pc[2]>0),-length(pc)],pc[which(pc[2]>0),-1],pch=3)
abline(h=100,lty=4)
abline(h=43,lty=4)

rla <- lm(pa[which(pa[2]>0),-1]~pa[which(pa[2]>0),-length(pa)])
summary(rla)
rlb <- lm(pb[which(pb[2]>0),-1]~pb[which(pb[2]>0),-length(pb)])
summary(rlb)
rlc <- lm(pc[which(pc[2]>0),-1]~pc[which(pc[2]>0),-length(pc)])
summary(rlc)
#########################################################################
#building distribution plot for all non-tRNA
p.all <- cbind(c(pa[,1],pb[,1],pc[,1]),c(pa[,2],pb[,2],pc[,2]))
plot(p.all[,1],p.all[,2],
main="Distribution of Nodes (Protein & rRNA)", ylab="Boostrap Value",
xlab="Sequence Length",ylim=c(0,100),xlim=c(0,2000))
rlap <- lm(p.all[,2]~p.all[,1])
summary(rlap)
#########################################################################
#building graph for tRNA
trna.dist <- read.csv(file="dist-trna.csv",sep="|")
trna.length <- read.csv(file="length-trna.csv",sep="|")
trna.dist <- trna.dist[-1]
trna.length <- trna.length[-1]
```

```
ta <- cbind(rowMeans(trna.length),trna.dist[1])
tb <- cbind(rowMeans(trna.length),trna.dist[2])
tc <- cbind(rowMeans(trna.length),trna.dist[3])

#buiding division for histogram
par(mfrow=c(2,2))

hist(ta[which(ta[2]>0),-1],main="Bootstrap Value vs Frequency" ,
xlab="Node A (tRNA Only)", xlim=c(0,100),ylim=c(0,7))
abline(v=60, lty=2)
hist(tb[which(tb[2]>0),-1],main="Bootstrap Value vs Frequency" ,
xlab="Node B (tRNA Only)", xlim=c(0,100),ylim=c(0,7))
abline(v=43, lty=2)
hist(tc[which(tc[2]>0),-1],main="Bootstrap Value vs Frequency" ,
xlab="Node C (tRNA Only)", xlim=c(0,100),ylim=c(0,7))
abline(v=40, lty=2)

#building distribution plot
plot(ta[which(ta[2]>0),-length(ta)],ta[which(ta[2]>0),-1],pch=1,
main="Distribution of Nodes (tRNA Only)", ylab="Boostrap Value",
xlab="Sequence Length",ylim=c(0,100),xlim=c(0,100))
points(tb[which(tb[2]>0),-length(tb)],tb[which(tb[2]>0),-1],pch=15)
points(tc[which(tc[2]>0),-length(tc)],tc[which(tc[2]>0),-1],pch=3)
abline(h=99,lty=4)
abline(h=35,lty=4)

rlta <- lm(ta[which(ta[2]>0),-1]~ta[which(ta[2]>0),-length(ta)])
summary(rlta)
rltb <- lm(tb[which(tb[2]>0),-1]~tb[which(tb[2]>0),-length(tb)])
summary(rltb)
rltc <- lm(tc[which(tc[2]>0),-1]~tc[which(tc[2]>0),-length(tc)])
summary(rltc)
#####################################################################
```

```
#building histogram for non-tRNA and tRNA for node a,b,c

a.all <- cbind(c(pa[,1],ta[,1]),c(pa[,2],ta[,2]))

b.all <- cbind(c(pb[,1],tb[,1]),c(pb[,2],tb[,2]))

c.all <- cbind(c(pc[,1],tc[,1]),c(pc[,2],tc[,2]))


par(mfrow=c(2,2))


hist(a.all[which(a.all[,2]>0),-1],main="(a)    Bootstrap    Value    vs
Frequency" ,

xlab="Node A", xlim=c(0,100),ylim=c(0,7))

abline(v=60, lty=2)

hist(b.all[which(b.all[,2]>0),-1],main="(b)    Bootstrap    Value    vs
Frequency" ,

xlab="Node B", xlim=c(0,100),ylim=c(0,7))

abline(v=43, lty=2)

hist(c.all[which(c.all[,2]>0),-1],main="(c)    Bootstrap    Value    vs
Frequency" ,

xlab="Node C", xlim=c(0,100),ylim=c(0,7))

abline(v=40, lty=2)

#########################################################################

#calculation of comparison of bootstrap with empirical estimate

sum (a.all[a.all[,2] > 0,2] < 60 ) / length(a.all[a.all[,2] > 0,2])

sum (b.all[b.all[,2] > 0,2] < 43 ) / length(b.all[b.all[,2] > 0,2])

sum (c.all[c.all[,2] > 0,2] < 40 ) / length(c.all[c.all[,2] > 0,2])
```