**Characterization of the complete mitochondrial genomes of *Corynosoma bullosum* (von Linstow, 1892) and *C. evae* Zdzitowiecki, 1984 (Acanthocephala:** **Polymorphida), and the phylogenetic implications**

Y.-Y. Xie1, 2, H.-X. Chen1, 2, T. A. Kuzmina3, 4, O. Lisitsyna3 and L. Li1, 2, \*

**Supplementary file 1: Python script used for calculating base composition, amino acid usage, and relative synonymous codon usage (RSCU).**

From collections import Counter

import matplotlib.pyplot as plt

from pandas import DataFrame

from Bio import SeqIO

import pandas as pd

import numpy as np

import argparse

table5 = { "TTT": "Phe", "TTC": "Phe", "TTA": "Leu", "TTG": "Leu",

 "TCT": "Ser", "TCC": "Ser", "TCA": "Ser", "TCG": "Ser",

 "TAT": "Tyr", "TAC": "Tyr", "TAA": " \* ", "TAG": " \* ",

 "TGT": "Cys", "TGC": "Cys", "TGA": "Trp", "TGG": "Trp",

 "CTT": "Leu", "CTC": "Leu", "CTA": "Leu", "CTG": "Leu",

 "CCT": "Pro", "CCC": "Pro", "CCA": "Pro", "CCG": "Pro",

 "CAT": "His", "CAC": "His", "CAA": "Gln", "CAG": "Gln",

 "CGT": "Arg", "CGC": "Arg", "CGA": "Arg", "CGG": "Arg",

 "ATT": "Ile", "ATC": "Ile", "ATA": "Met", "ATG": "Met",

 "ACT": "Thr", "ACC": "Thr", "ACA": "Thr", "ACG": "Thr",

 "AAT": "Asn", "AAC": "Asn", "AAA": "Lys", "AAG": "Lys",

 "AGT": "Ser", "AGC": "Ser", "AGA": "Ser", "AGG": "Ser",

 "GTT": "Val", "GTC": "Val", "GTA": "Val", "GTG": "Val",

 "GCT": "Ala", "GCC": "Ala", "GCA": "Ala", "GCG": "Ala",

 "GAT": "Asp", "GAC": "Asp", "GAA": "Glu", "GAG": "Glu",

 "GGT": "Gly", "GGC": "Gly", "GGA": "Gly", "GGG": "Gly", }

##RSCU\_count

def get\_synonymous\_codons(genetic\_code\_dict):

 # invert the genetic code dictionary to map each amino acid to its codons

 codons\_for\_amino\_acid = {}

 for codon, amino\_acid in list(genetic\_code\_dict.items()):

 codons\_for\_amino\_acid[amino\_acid] = codons\_for\_amino\_acid.get(amino\_acid, [])

 codons\_for\_amino\_acid[amino\_acid].append(codon)

 #print(genetic\_code\_dict.items())

 # create dictionary of synonymous codons

 # Example: {'CTT': ['CTT', 'CTG', 'CTA', 'CTC', 'TTA', 'TTG'], 'ATG': ['ATG']...}

 return {codon: codons\_for\_amino\_acid[genetic\_code\_dict[codon]] for codon in list(genetic\_code\_dict.keys())}

def RSCU(sequences, genetic\_code\_dict=table5):

 if not isinstance(sequences, (list, tuple)):

 raise ValueError("Be sure to pass a list of sequences, not a single sequence. To find the RSCU of a single sequence, pass it as a one element list.")

 # ensure all input sequences are divisible by three

 for sequence in sequences:

 if len(sequence) % 3 != 0:

 raise ValueError("Input sequence not divisible by three")

 if not sequence:

 raise ValueError("Input sequence cannot be empty")

 # count the number of each codon in the sequences

 #sequences =['ACTGTCGTA','TTAATGCTGAGT']

 sequences = [[sequence[i:i + 3].upper() for i in range(0, len(sequence), 3)] for sequence in sequences]

 # flat list of all codons (to be used for counting)

 codons = []

 for x in sequences:

 codons.extend(x)

 # The number of occurrences of each codon was calculated

 counts = Counter(codons)

 # "if a certain codon is never used in the reference set... assign [its

 # count] a value of 0.5" (page 1285)

 for codon in table5:

 if counts[codon] == 0:

 counts[codon] = 0.5

 # determine the synonymous codons for the genetic code

 synonymous\_codons = get\_synonymous\_codons(genetic\_code\_dict)

 # hold the result as it is being calulated

 result = {}

 # calculate RSCU values

 for codon in genetic\_code\_dict:

 result[codon] = counts[codon] / ((len(synonymous\_codons[codon]) \*\* -1) \* (sum((counts[\_codon] for \_codon in synonymous\_codons[codon]))))

 #print(counts[codon])

 #print(len(synonymous\_codons[codon]))

 #print(sum((counts[\_codon] for \_codon in synonymous\_codons[codon])))

 sum\_codons={}

 for codon in table5:

 sum\_codons[codon]=[counts[codon],result[codon]]

 return sum\_codons

def get\_seq3(remainder, sequence, header):

 r = remainder

 line\_new = sequence[:-r]

 stop\_codon = sequence[-r:]

 len\_seq2 = len(line\_new)

 remainder2 = len\_seq2 % 3

 print(header + ' ' + "incomplete stop codon" + ':' + stop\_codon + ';' + str(r) + '>>' + str(remainder2))

 return line\_new

def read\_fas\_out\_mul3(input):

 name = input.split(".")[0]

 ofile = name + '\_mul3.fas'

 outputfile = str(ofile)

 # print(outputfile)

 out\_fas = open(outputfile, 'a')

 with open(input) as input\_f:

 #fasta = {}

 for line in input\_f:

 line = line.strip()

 if line[0] == '>':

 id = line[1:]

 id2 = id.split(';')

 header = id2[1]

 else:

 sequencelist=[]

 sequencelist.append(line)

 sequence = line

 len\_seq = len(sequence)

 #enter the sequence name

 out\_fas.write('>' + header + '\n')

 #calculates whether the sequence is a multiple of 3

 remainder = len\_seq % 3

 if remainder == 0:

 seq = sequence

 else:

 seq = get\_seq3(remainder, sequence, header)

 out\_fas.write(seq + '\n')

 input\_f.close()

 out\_fas.close()

 return outputfile

##plot\_RSCU\_bar

def make\_stacked\_plot(df\_color):

 # color

 color = {1: '#FFFFAA', 1.5: '#FF9D6F', 2: '#0080FF', 2.5: '#FF2D2D', 3: '#E800E8', 3.5: '#00DB00', 4: '#8600FF',

 4.5: '#FF5809', 5: '#EAC100', 5.5: '#0072E3', 6: '#FF0080'}

 # canvas

 plt.figure(figsize=(15, 10))

 # quadrant 1

 plt.subplot(211)

 # rscu table perspective

 dfrscu = df\_color.pivot\_table(index="AA", columns='color', values='RSCU', aggfunc=sum, fill\_value=0)

 #sets the bottom value of the stack plot and creates a list of color values

 margin\_bottom = np.zeros(len(df\_color['AA'].drop\_duplicates()))

 heights\_list = df\_color['color'].drop\_duplicates()

 # stacking

 for num, h in enumerate(heights\_list):

 values = list(dfrscu[h])

 plt.bar(x=dfrscu.index, height=dfrscu[h].fillna(0), bottom=margin\_bottom, facecolor=color[h], label=h, alpha=1)

 margin\_bottom += values

 # Y-axis heading

 plt.ylabel("RSCU")

 # set the xy axis range

 plt.ylim((-0.5, max(margin\_bottom) + 0.5))

 plt.xlim((-1, len(dfrscu.index)))

 # quadrant 2

 plt.style.use('ggplot')

 ax2 = plt.subplot(212)

 # set the 2nd quadrant background blank

 ax2.patch.set\_facecolor('none')

 # codnon table perspective

 dfcodon = df\_color.pivot\_table(index="AA", columns='color', values='codon', aggfunc=sum, fill\_value='')

 # xy location with codon content

 xs = list(range(0, len(dfcodon.index.values)))

 aas = dfcodon.index.values

 ys = dfcodon.columns.values

 # set the xy axis range and remove the xy axis

 plt.ylim((min(dfcodon.columns.values) - 0.5, max(dfcodon.columns.values)))

 plt.xlim((-1, len(dfcodon.index)))

 plt.xticks([])

 plt.yticks([])

 # codon mapping diagram

 for x in xs:

 for y in ys:

 ax2.text(x, y, dfcodon.iloc[x][y], va="center", ha="center",

 bbox=dict(boxstyle="round", facecolor=color[y], alpha=1))

 # save picture

 plt.savefig(fname="RSCU.svg", format="svg")

 plt.savefig(fname="RSCU.png", format="png", dpi=300)

 plt.show()

def RSCU\_process(input\_fasta):

 mul3\_seq = read\_fas\_out\_mul3(input\_fasta)

 abb = (input\_fasta).split(".")[0]

 seqs = [rec.seq for rec in SeqIO.parse(mul3\_seq, 'fasta')]

 rscu = RSCU(seqs, table5)

 # statistical result

 df\_rscu\_sum = pd.DataFrame(columns=['AA', 'codon', 'count\_num', 'rscu\_val'])

 for codon in rscu.keys():

 info\_list = [table5[codon], codon, rscu[codon][0], round(rscu[codon][1], 3)]

 df\_rscu\_sum.loc[codon] = info\_list

 df\_rscu\_sum.to\_excel('{}\_sum.xlsx'.format(abb))

 # draw a picture

 df\_rscu\_color = pd.DataFrame(columns=['AA', 'codon', 'RSCU', 'color'])

 color = {}

 for codon in rscu.keys():

 if table5[codon] not in color:

 color[table5[codon]] = 6

 else:

 color[table5[codon]] -= 0.5

 info\_list = [table5[codon], codon, round(rscu[codon][1], 3), color[table5[codon]]]

 df\_rscu\_color.loc[codon] = info\_list

 df\_rscu\_color['codon'] = df\_rscu\_color['codon'].str.replace('T', 'U')

 make\_stacked\_plot(df\_rscu\_color)

 df\_rscu\_color.to\_excel('{}\_color.xlsx'.format(abb))

##base\_count\_ratio

def count\_seq(sequence):

 codon1 = []

 codon2 = []

 codon3 = []

 total = len(sequence)

 sequence = [sequence[i:i + 3].upper() for i in range(0, len(sequence), 3)]

 for codon in sequence:

 codon1.append(codon[0])

 try:

 codon2.append(codon[1])

 except:

 continue

 try:

 codon3.append(codon[2])

 except:

 continue

 count\_codon1=Counter(codon1)

 count\_codon2=Counter(codon2)

 count\_codon3=Counter(codon3)

 seqA=count\_codon1['A']+count\_codon2['A']+count\_codon3['A']

 seqT=count\_codon1['T']+count\_codon2['T']+count\_codon3['T']

 seqC=count\_codon1['C']+count\_codon2['C']+count\_codon3['C']

 seqG=count\_codon1['G']+count\_codon2['G']+count\_codon3['G']

 #total=seqA+seqT+seqC+seqG

 #dict\_seq={'A':seqA, 'T':seqT, 'C':seqC, 'G':seqG}

 '''

 dict\_c1=count\_codon1

 dict\_c2=count\_codon2

 dict\_c3=count\_codon3

 '''

 #codon1的ATCG

 cod1A=count\_codon1['A']

 cod1T=count\_codon1['T']

 cod1C=count\_codon1['C']

 cod1G=count\_codon1['G']

 post\_1 = len(codon1)

 #post\_1 = cod1A + cod1T + cod1C + cod1G

 #dict\_c1 = {'A': codon1A, 'T': codon1T, 'C': codon1C, 'G': codon1G}

 #codon2的ATCG

 cod2A=count\_codon2['A']

 cod2T=count\_codon2['T']

 cod2C=count\_codon2['C']

 cod2G=count\_codon2['G']

 post\_2 = len(codon2)

 #post\_2 = cod2A + cod2T + cod2C + cod2G

 #dict\_c2 = {'A': codon2A, 'T': codon2T, 'C': codon2C, 'G': codon2G}

 #codon3的ATCG

 cod3A=count\_codon3['A']

 cod3T=count\_codon3['T']

 cod3C=count\_codon3['C']

 cod3G=count\_codon3['G']

 post\_3 = len(codon3)

 #post\_3 = cod3A + cod3T + cod3C + cod3G

 #dict\_c3 = {'A': codon3A, 'T': codon3T, 'C': codon3C, 'G': codon3G}

 #[dict\_seq,dict\_c1,dict\_c2,dict\_c3]

 return [seqA,seqT,seqC,seqG,total,

 cod1A,cod1T,cod1C,cod1G,post\_1,

 cod2A,cod2T,cod2C,cod2G,post\_2,

 cod3A,cod3T,cod3C,cod3G,post\_3]

def get\_per\_tab(input\_table):

 pertable=input\_table.copy()

 #print(input\_table)

 # pertable.insert(1,column='A%',value=(sum\_table['A']/sum\_table['Total'])\*100)

 pertable["A%"] = (pertable['A'] / pertable['Total']) \* 100

 pertable["T%"] = (pertable['T'] / pertable['Total']) \* 100

 pertable["C%"] = (pertable['C'] / pertable['Total']) \* 100

 pertable["G%"] = (pertable['G'] / pertable['Total']) \* 100

 pertable["AT%"] = ((pertable['A']+pertable['T'])/ pertable['Total']) \* 100

 # 'A-1', 'T-1', 'C-1', 'G-1', 'Post-1',

 pertable["A-1%"] = (pertable['A-1'] / pertable['Post-1']) \* 100

 pertable["T-1%"] = (pertable['T-1'] / pertable['Post-1']) \* 100

 pertable["C-1%"] = (pertable['C-1'] / pertable['Post-1']) \* 100

 pertable["G-1%"] = (pertable['G-1'] / pertable['Post-1']) \* 100

 pertable["AT-1%"] = ((pertable['A-1'] + pertable['T-1']) / pertable['Post-1']) \* 100

 pertable["A-2%"] = (pertable['A-2'] / pertable['Post-2']) \* 100

 pertable["T-2%"] = (pertable['T-2'] / pertable['Post-2']) \* 100

 pertable["C-2%"] = (pertable['C-2'] / pertable['Post-2']) \* 100

 pertable["G-2%"] = (pertable['G-2'] / pertable['Post-2']) \* 100

 pertable["AT-2%"] = ((pertable['A-2'] + pertable['T-2']) / pertable['Post-2']) \* 100

 pertable["A-3%"] = (pertable['A-3'] / pertable['Post-3']) \* 100

 pertable["T-3%"] = (pertable['T-3'] / pertable['Post-3']) \* 100

 pertable["C-3%"] = (pertable['C-3'] / pertable['Post-3']) \* 100

 pertable["G-3%"] = (pertable['G-3'] / pertable['Post-3']) \* 100

 pertable["AT-3%"] = ((pertable['A-3'] + pertable['T-3']) / pertable['Post-3']) \* 100

 return pertable

def get\_sum\_table(input\_table):

 sumtable=input\_table.copy()

 forward\_strand\_table = sumtable[(sumtable['strand'] == '(+)')]

 reverse\_strand\_table = sumtable[(sumtable['strand']=='(-)')]

 sumtable.loc['totoal\_sum'] = sumtable[['A', 'T', 'C', 'G', 'Total', 'A-1', 'T-1', 'C-1', 'G-1',

 'Post-1', 'A-2', 'T-2', 'C-2', 'G-2', 'Post-2', 'A-3', 'T-3', 'C-3',

 'G-3', 'Post-3']].apply(lambda x: x.sum())

 #+

 if len(sumtable[(sumtable['strand']=='(+)')]) > 0 :

 sumtable.loc['forward\_strand'] = forward\_strand\_table[['A', 'T', 'C', 'G', 'Total', 'A-1', 'T-1', 'C-1', 'G-1',

 'Post-1', 'A-2', 'T-2', 'C-2', 'G-2', 'Post-2', 'A-3', 'T-3', 'C-3',

 'G-3', 'Post-3']].apply(lambda x: x.sum()).copy()

 else:

 print("no (+) strand gene")

 #-

 if len(sumtable[(sumtable['strand']=='(-)')]) > 0 :

 sumtable.loc['reverse\_strand'] = reverse\_strand\_table[['A', 'T', 'C', 'G', 'Total', 'A-1', 'T-1', 'C-1', 'G-1',

 'Post-1', 'A-2', 'T-2', 'C-2', 'G-2', 'Post-2', 'A-3', 'T-3', 'C-3',

 'G-3', 'Post-3']].apply(lambda x: x.sum()).copy()

 else:

 print("no (-) strand gene")

 final\_table=get\_per\_tab(sumtable)

 final\_table[['A%', 'T%', 'C%', 'G%', 'AT%', 'A-1%', 'T-1%', 'C-1%','G-1%', 'AT-1%', 'A-2%', 'T-2%', 'C-2%',

 'G-2%', 'AT-2%', 'A-3%','T-3%', 'C-3%', 'G-3%', 'AT-3%']]=final\_table[['A%', 'T%', 'C%', 'G%',

 'AT%', 'A-1%', 'T-1%', 'C-1%','G-1%', 'AT-1%', 'A-2%', 'T-2%', 'C-2%', 'G-2%', 'AT-2%', 'A-3%',

 'T-3%', 'C-3%', 'G-3%', 'AT-3%']].astype('float').round(2)

 final\_table[['A', 'T', 'C', 'G', 'Total', 'A-1', 'T-1', 'C-1', 'G-1','Post-1', 'A-2', 'T-2', 'C-2',

 'G-2', 'Post-2', 'A-3', 'T-3', 'C-3','G-3', 'Post-3']] = final\_table[['A', 'T', 'C',

 'G', 'Total', 'A-1', 'T-1', 'C-1', 'G-1','Post-1', 'A-2', 'T-2', 'C-2', 'G-2', 'Post-2',

 'A-3', 'T-3', 'C-3','G-3', 'Post-3']].astype('int')

 return final\_table

def get\_seqs\_count(input\_fasta):

 base\_count\_dftmp=DataFrame(

 columns=['strand',

 'A', 'T', 'C', 'G', 'Total',

 'A-1', 'T-1', 'C-1', 'G-1', 'Post-1',

 'A-2', 'T-2', 'C-2', 'G-2', 'Post-2',

 'A-3', 'T-3', 'C-3', 'G-3', 'Post-3'])

 #count—seq

 for rec in SeqIO.parse(input\_fasta, 'fasta'):

 sequence=rec.seq

 id=(rec.id).split(';')[1]

 #print(id)

 strand=(rec.id).split(']')[1]

 base\_count\_list=count\_seq(sequence)

 base\_count\_list.insert(0, strand)

 base\_count\_dftmp.loc[id] = base\_count\_list

 #sum\_table.to\_excel("{0}\_{1}\_\_{2}.xlsx".format(prefix, line, data))

 count\_table=base\_count\_dftmp.sort\_values(by=['strand'], ascending=False)

 final\_sum\_table=get\_sum\_table(count\_table)

 abb = str(input\_fasta)

 final\_sum\_table.to\_excel('{}\_base\_count.xlsx'.format(abb))

if \_\_name\_\_ == "\_\_main\_\_":

 parser = argparse.ArgumentParser()

 parser.add\_argument('--input', '-i',

 type=str,

 help='input file in fasta format')

 args = parser.parse\_args()

 RSCU\_process(args.input)

 get\_seqs\_count(args.input)

 print("Be careful! If the total length is not equal to the sum of ATCG,"

 " ambiguous bases exist in the sequence , eg. Y K M ")