Computational Biology
Tufts University Bio 35
Fall 2021

Assignment 6: Homology detection

Homology detection: All of your code for this problem should be in a file called `SexChromosomeEvolution.py`. At the end you will also make a file with comments called `SexChromosomeEvolution.txt`. This assignment is taken (with minor adaptations) from the companion material for CFB. You can find the original at `https://www.cs.hmc.edu/twiki/bin/view/CFB/SexChromosomeEvolution`.

Introduction

Land vertebrates have a wide variety of sex-determination systems. Evolutionary biologists are interested in understanding the origin of this diversity. In this problem you will look at one small question which relates to this larger issue: are mammalian and bird sex chromosomes homologous?

To do this you will compare sequences from a mammal (human) and a bird (chicken). You will identify orthologous genes between these two species using alignment scoring and the best reciprocal hit method. Based on these orthologs you will then create a mapping between the chromosomes of these species. This will allow you to see which mammal chromosome(s) the bird Z is similar to, and which bird chromosome(s) the mammal X is similar to. If Z and X are not homologous to each other, but rather are homologous to other chromosomes, it supports the idea that chromosomal sex determination evolved independently in birds and mammals.

Getting started

Find `humanChickenProteins.py` and texttttblosum62.py in the scripts directory for this assignment. Then create `SexChromosomeEvolution.py` and place it in the same directory as these two files. You can paste the following at the top of `SexChromosomeEvolution.py`:

example
```
import sys
sys.setrecursionlimit(100000)
from humanChickenProteins import *
from blosum62 import *
```

`memoAlignScore(S1, S2, gap, substitutionMatrix,memo)`

We have already developed a function `alignScore` that computes the alignment score. However this function is slow for even moderately sized sequences. To be useful in identifying orthologs between real human and chicken proteins, we will need a function which is considerably faster.

Your first task is to implement a memoized version of `alignScore`, `memoAlignScore(S1, S2, gap,  substitutionMatrix, memo)`, where `memo` is a dictionary. You can start it off as {}. Test your `memoAlignScore` function carefully. Here are some examples of the `memoAlignScore` function in action:

example
```
>>> memoAlignScore('','',-9,blosum62,{})
0
>>> memoAlignScore('','FGTSK',-9,blosum62,{})
-45
>>> memoAlignScore('IVEKGYY','AVEYY',-9,blosum62,{})
4
>>> memoAlignScore('CIEAFGTSKQKRALNSRRMNAVGNDIVSTAVTKAAADVIDAKGVTALIQDVAQD',
    'RDLPIWTSVDWKSLPATEIFNKAFSQGSDEAMYDYMAVYKKSCPQTRR',-9,blosum62,{})
-48
```

Notice that the last example is large enough that your original `alignScore` function (before memoizing it) would be VERY slow!

Getting best reciprocal hits

The data: There are tens of thousands of protein-coding genes in human and chicken. This is enough that even with our fast `memoAlignScore` function it would take quite a while to compare all genes vs all genes. To simplify this assignment, we are providing a small subset of genes from human and chicken. These include some genes with orthologs, and some without.

The data file `humanChickenProteins.py` contains four lists and a dictionary:

- `humanGeneList`
- `chickenGeneList`
- `sampleHumanGeneList`
- `sampleChickenGeneList`
- `geneD`

The lists contain names of genes ('h1' , 'h2', etc.). Your job is simply to figure out which genes are orthologous to which (there are some which do not have a BRH/ortholog in the other species). The two sample gene lists are much smaller, and can be used for testing your code. We have also provided a dictionary called `geneD` which contains information about genes, keyed by name. Each entry in `geneD` is a tuple of the following form: `(chromosome, startPosition, endPosition, proteinSequence)`. `chromosome` and `proteinSequence` are strings, while `startPosition` and `endPosition` are integers giving the coordinates of the start and end of the gene on the chromosome.

An example of how to get information for the chicken gene 'c6':

example
```
>>> chromosome, startPosition, endPosition, proteinSequence=geneD['c6']
>>> chromosome
'chr15'
>>> startPosition
791273
>>> endPosition
791857
>>> proteinSequence[:25]
'MNSGILFLSLLGFLPSVIPTCPLPC'
```

`allScores(geneList1,geneList2)`: Your first step is to write a function to obtain the alignment score between all proteins in two species. `allScores` takes as input two lists of genes. It then takes every protein in the first list and calls `memoAlignScore` on it with every protein in the second list. (Remember that the protein sequence for each gene can be obtained from `geneD`). In `memoAlignScore` you can use the `blosum62` substitution matrix (which you have already imported at the top of your file) and a gap penalty of -9.

`allScores` returns a dictionary as output. The keys in this dictionary are pairs of protein names in the form of a tuple. For example `('h0', 'c3')`. The values in the dictionary are alignment scores as calculated by `memoAlignScore`. In our solution, we follow the convention of putting the gene from `geneList1` first in the key, and the gene from `geneList2` second.

Here is `allScores` at work on the sample datasets:

example
```
>>> allScoresD=allScores(sampleHumanGeneList,sampleChickenGeneList)
>>> len(allScoresD.keys())
20
```

The sample human and chicken gene lists each have five and four genes respectively. So the output of `allScores` here has 5×4 or 20 entries.

example
```
>>> allScoresD[('h4', 'c8')]
-134
```

The human gene 'h4' and the chicken gene 'c8' have an alignment score of -134 between them.

`closestMatch(geneName,allScoresD)`: Given a gene name and a dictionary of alignment scores, `closestMatch` returns the protein from the other species which is most similar (has the highest alignment score).

To do this you will want to look for `geneName` in every key in `allScoresD`. The following syntax will be useful:

example
```
>>> allScoresD.keys()
```

This returns a list of keys (here a list of tuples, since our keys are tuples). You can then search through this list for entries that have `geneName` in them. For this part, the following will likely be useful:

example
```
>>> T = ('spam','pims')
>>> 'spam' in T
True
>>> 'donkey' in T
False
```

Here are some examples of `closestMatch` with the sample data:

example
```
>>> allScoresD=allScores(sampleHumanGeneList,sampleChickenGeneList)
>>> closestMatch('c19',allScoresD)
'h4'
>>> closestMatch('h17',allScoresD)
'c8'
printBRH(geneName,allScoresD)
```

This function takes a gene name, and a dictionary of `memoAlignScore` scores as input. It finds and prints the best reciprocal hit for `geneName`. If there is no best reciprocal hit it returns without printing anything.

A simple way to find the best reciprocal hit for `geneName` is to call `closestMatch` to get its best matching gene in the other species. Then call `closestMatch` on this best match from the other species. If the output after the round trip matches `geneName`, then we've found a best reciprocal hit.

For each best reciprocal hit we will print chromosome, start position, and gene name for the gene in the first species, and then for the gene in the second species. For example:

example
```
>>> allScoresD=allScores(sampleHumanGeneList,sampleChickenGeneList)
>>> printBRH('c8',allScoresD)
chr2 43123243 c8 --- chr3 45016733 h17
>>> printBRH('h7',allScoresD)
>>> printBRH('h17',allScoresD)
chr3 45016733 h17 --- chr2 43123243 c8
```

*Identifying orthologs First test your code on the sample data set, using this wrapper function:

example

```
def runBRHSample():
    '''Print best reciprocal hits for sample data. First in human
    chromosome order, then in chicken chromosome order.'''
    allScoresD=allScores(sampleHumanGeneList,sampleChickenGeneList)
    print 'human --- chicken'
    for geneName in sampleHumanGeneList:
        printBRH(geneName,allScoresD)
    print
    print 'chicken --- human'
    for geneName in sampleChickenGeneList:
        printBRH(geneName,allScoresD)
```

Here's the output when we run it with our solutions for `allScores`, `closestMatch`, and `printBRH`:

example

```
>>> runBRHSample()
human --- chicken
chr11 118415243 h4 --- chr24 5629899 c19
chr11 133938820 h6 --- chr24 2542440 c17
chr14 72399156 h9 --- chr5 28862733 c22
chr3 45016733 h17 --- chr2 43123243 c8

chicken --- human
chr2 43123243 c8 --- chr3 45016733 h17
chr24 2542440 c17 --- chr11 133938820 h6
chr24 5629899 c19 --- chr11 118415243 h4
chr5 28862733 c22 --- chr14 72399156 h9
```

Next, you can run the full data set with the wrapper function below. This may take on the order of 10 minutes.

example

```
def runBRH():
    '''Print best reciprocal hits for full data. First in human
    chromosome order, then in chicken chromosome order.'''
    allScoresD=allScores(humanGeneList,chickenGeneList)
    print 'human --- chicken'
    for geneName in humanGeneList:
        printBRH(geneName,allScoresD)
    print
    print 'chicken --- human'
    for geneName in chickenGeneList:
        printBRH(geneName,allScoresD)
```

Your final task is to interpret the output. If two genes are best reciprocal hits, we take that to mean they are orthologs. So `runBRH` has produced a list of orthologs. If the mammalian X and the bird Z chromosomes are homologous to each other, we would expect them to share orthologs. If they do not share orthologs, it suggests the X and the Z are not homologous.

Create a text file called `SexChromosomeEvolution.txt` with the following:

Based on your output, briefly discuss the answer to this question: does it appear that the human X chromosome is homologous to the chicken Z? Below your answer, paste your output from `runBRH`.