University of Zurich - HS 2013

Computational Science I
Exercise notes: Combinatorics

Tobias Grubenmann

October 3, 2013

Exercise 1

To find all independent solution for the eight-queen problem I first searched for all dependent solution using an integer representation for convenience. After that I converted the solutions to a complex representation, again for convenience. Now, I pick up a solution and delete all rotated and/or reflected solutions. The remaining solutions are all independent solutions of the eight-queen problem.

The function `getIndependentSolutions` returns the independent solutions of the eight-queen problem:

```
from copy import copy

def searchSolutions(checkerboard = [-1, -1, -1, -1, -1, -1,
    -1, -1], n = 0, solutions = []):

    for i in range(8):
        checkerboard[n] = i
        checkSolution(checkerboard, n, solutions)

    return solutions

def checkSolution(checkerboard, n, solutions):

    isValid = True

    for i in range(n):
        # check if queens are in the same row
        if checkerboard[n] == checkerboard[i]:
            isValid = False
        # check if queens are in the same diagonal
        if abs(checkerboard[n] - checkerboard[i]) == abs(n-
           i):
            isValid = False

    if isValid:
        if n == 7:
            # we have found a solution
            solutions += [copy(checkerboard)]
        else:
            # search in next column
            searchSolutions(checkerboard, n+1, solutions)

def convertSolutions(solutions):

    for i in range(len(solutions)):
        for j in range(8):
            solutions[i][j] = 2*j - 7 + (2*solutions[i][j]
               - 7)*1j

def getIndependentSolutions():

    independentSolutions = []

    # search for all solutions
    solutions = searchSolutions()

    # convert the solutions to a complex representation
    convertSolutions(solutions)

    while len(solutions) > 0:
        newSolution = sorted(solutions[0], key=lambda x: x.
           real)
        del solutions[0]

        independentSolutions += [newSolution]

        # remove dependent solutions
        n = 0
        while n < len(solutions):
            dependent = False
            if newSolution == sorted([x * 1j for x in
               solutions[n]], key=lambda x: x.real):
                dependent = True
            elif newSolution == sorted([x * -1 for x in
               solutions[n]], key=lambda x: x.real):
                dependent = True
            elif newSolution == sorted([x * -1j for x in
               solutions[n]], key=lambda x: x.real):
                dependent = True
            elif newSolution == sorted([x.conjugate() for x
                in solutions[n]], key=lambda x: x.real):
                dependent = True
            elif newSolution == sorted([x.conjugate() * -1
               for x in solutions[n]], key=lambda x: x.real
               ):
                dependent = True
            elif newSolution == sorted([x.conjugate() * 1j
               for x in solutions[n]], key=lambda x: x.real
               ):
                dependent = True
            elif newSolution == sorted([x.conjugate() * -1j
                for x in solutions[n]], key=lambda x: x.
               real):
                dependent = True

            if dependent:
                del solutions[n]
            else:
                n += 1

    return independentSolutions
```

From the function `getIndependentSolutions` we will get the following **12** independent solutions:

```
[[(-7-7j), (-5+1j), (-3+7j), (-1+3j), (1-3j), (3+5j), (5-5j
   ), (7-1j)],
 [(-7-7j), (-5+3j), (-3+7j), (-1-3j), (1+5j), (3-1j), (5-5j
   ), (7+1j)],
 [(-7-5j), (-5-1j), (-3+3j), (-1+7j), (1-3j), (3-7j), (5+5j
   ), (7+1j)],
 [(-7-5j), (-5+1j), (-3+5j), (-1-7j), (1-3j), (3+7j), (5+3j
   ), (7-1j)],
 [(-7-5j), (-5+1j), (-3+5j), (-1-1j), (1-7j), (3+7j), (5+3j
   ), (7-3j)],
 [(-7-5j), (-5+3j), (-3-7j), (-1+5j), (1-1j), (3+7j), (5-3j
   ), (7+1j)],
 [(-7-5j), (-5+3j), (-3+7j), (-1-3j), (1-7j), (3-1j), (5+5j
   ), (7+1j)],
 [(-7-5j), (-5+5j), (-3-3j), (-1+3j), (1+7j), (3+1j), (5-7j
   ), (7-1j)],
 [(-7-5j), (-5+5j), (-3+1j), (-1+7j), (1-7j), (3-1j), (5+3j
   ), (7-3j)],
 [(-7-3j), (-5+1j), (-3-5j), (-1+7j), (1-7j), (3+5j), (5-1j
   ), (7+3j)],
 [(-7-3j), (-5+1j), (-3+7j), (-1-1j), (1-7j), (3+5j), (5-5j
   ), (7+3j)],
 [(-7-3j), (-5+3j), (-3-5j), (-1+1j), (1+7j), (3-7j), (5+5j
   ), (7-1j)]]
```

Exercise 2

First, I generate all the 64 different codons with `generateCodons`. The codons are encoded by a triple of numbers where $A = 0$, $C = 1$, $G = 2$, $T = 3$.We can then just convert the codons into diamonds by the formula:

$$diamond = (codon_1, codon_2, 3 - codon_3, 3 - codon_2)$$

Like in the eight-queens problem, I can now select a diamond and delete all diamonds which are reserved or start at the other corner.

```
# A = 0, C = 1, G = 2, T = 3

def generateCodons():

    codons = []

    for i in range(4):
        for j in range(4):
            for k in range(4):
                codons += [[i , j, k]]

    return codons

def convertCodonsToDiamonds(codons):

    diamonds = []
    for i in range(len(codons)):
        diamonds += [[codons[i][0], codons[i][1], 3-codons[
            i][2], 3-codons[i][1]]]

    return diamonds

def getDifferentGamowDiamonds():

    differentDiamonds = []

    codons = generateCodons()

    diamonds = convertCodonsToDiamonds(codons)

    while len(diamonds) > 0:

        newDiamond = diamonds[0]
        del diamonds[0]

        differentDiamonds += [newDiamond]

        # remove equivalent diamonds
        n = 0
        while n < len(diamonds):

            equivalent = False

            if newDiamond == [diamonds[n][0], diamonds[n
                ][3], diamonds[n][2], diamonds[n][1]]:
                equivalent = True
            elif newDiamond == [diamonds[n][2], diamonds[n
                ][3], diamonds[n][0], diamonds[n][1]]:
                equivalent = True
            elif newDiamond == [diamonds[n][2], diamonds[n
                ][1], diamonds[n][0], diamonds[n][3]]:
                equivalent = True

            # remove equivalent diamonds
            if equivalent:
                del diamonds[n]
            else:
                n += 1
    return differentDiamonds
```

The function getDifferentGamowDiamonds returns all 20 different Gamow-diamonds:

```
[[0, 0, 3, 3], [0, 0, 2, 3], [0, 0, 1, 3], [0, 0, 0, 3],
   [0, 1, 3, 2], [0, 1, 2, 2], [0, 1, 1, 2], [0, 1, 0, 2],
   [1, 0, 3, 3], [1, 0, 2, 3], [1, 0, 1, 3], [1, 1, 3, 2],
   [1, 1, 2, 2], [1, 1, 1, 2], [2, 0, 3, 3], [2, 0, 2, 3],
   [2, 1, 3, 2], [2, 1, 2, 2], [3, 0, 3, 3], [3, 1, 3, 2]]
```

Exercise 3

I used the **ELANE** gene to make a comparison between humans, chimpanzees, dogs and cats. First, I downloaded the sequences corresponding to the species using the following code:

```
from Bio import ExPASy, SeqIO

sid = raw_input("Sequence␣id?␣")

try:
    handle = ExPASy.get_sprot_raw(sid)
    seq = SeqIO.read(handle, "swiss")
    SeqIO.write(seq, sid + ".genbank", "genbank")
    print "Sequence␣length", len(seq)
except Exception:
    print "Sequence␣not␣found"
```

Afterward, I made an sequence alignment analysis by putting the downloaded sequence in another code. The following code plots a dot whenever five consecutive characters matches in two different sequences. The coordinates of the plot indicates the position of the five characters in the first and second sequence. In addition, a number indicating how many matches where found for a given pair of sequences is also printed:

```
import pylab

# Gene sequences got from database

chimpanzee = "mtlgrrlaclflacvlpalllggtalaseivggrrarphawpf"\
    "mvslqlrgghfcgatliapnfvmsaahcvanvvravrvgahlsrreptrqvf"\
    "avqrikglngsatinanvqvaqlpaqgrhlgngvqclamgwgllgrnrgias"\
    "vlqelnvtvvtslcrrsnvctlvrgrragvcfgdsgsplvcnglihgiasfv"\
    "rggcasglypdafapvaqfvnwinsiiqrsednpcphprdpdpasrth"
cat = "mtpsrrsagpalapvllamllggpalaseivggrparphawpfmvslqlr"\
    "gghfcggtliapnfvmsaahcvdglnfrsvvavlgahdlrrreptrqmftiq"\
    "rvfengfdpqrllndivilqlngsatinsnvrvarlpaqnqgvgsgvqclam"\
    "gwgqlgttqpppnilqelnvtvvttlcprsnvctlvprrqagicfgdsggpl"\
    "vcngliqgidsfirgscgsgfypdafapvaqfanwidsiirrqddrpsvhpr"\
    "dpasrtl"
human = "mtlgrrlaclflacvlpalllggtalaseivggrrarphawpfmvslq"\
    "lrgghfcgatliapnfvmsaahcvanvnvravrvvlgahnlsrreptrqvfa"\
    "vqrifengydpvnllndivilqlngsatinanvqvaqlpaqgrrlgngvqcl"\
    "amgwgllgrnrgiasvlqelnvtvvtslcrrsnvctlvrgrqagvcfgdsgs"\
    "plvcnglihgiasfvrggcasglypdafapvaqfvnwidsiiqrsednpcph"\
    "prdpdpasrth"
dog = "mtarrvpagpalgpllllatllpgpalaseivggrpaqphawpfmvslqr"\
    "rgghfcggtliapnfvmsaahcvdglnfrsvvvvlgahdlgerestrqlfav"\
    "qrvfengfdpvrlvndivllqlngsatinanvqvarlpaqnqgvgngvqcla"\
    "mgwgqlgtaqppprilqelnvtvvttlcrrsnvctlvprrragicfgdsggp"\
    "lvcngliqgidsfirgscasgffpdafapvaqfvdwinsiirrppalpparp"\
    "gqqdpergaarapppaphrprptq"

species = [human, chimpanzee, dog, cat]

alignment = [];

for n in range(len(species)):
    for m in range(n+1,len(species)):

        alignment += [0]

        pylab.figure()

        for i in range(len(species[n])-5):
            for j in range(len(species[m])-5):
                match = True
                for k in range(5):
                    if species[n][i+k] != species[m][j+k]:
                        match = False

                if match:
                    pylab.plot(i, j, 'bo')
                    alignment[-1] += 1
```

```
print "Human␣-␣Chimpanzee␣alignment␣:␣", alignment[0]
print "Human␣-␣Dog␣alignment␣:␣", alignment[1]
print "Human␣-␣Cat␣alignment␣:␣",alignment[2]
print "Chimpanzee␣-␣Dog␣alignment␣:␣",alignment[3]
print "Chimpanzee␣-␣Cat␣alignment:␣", alignment[4]
print "Dog␣-␣Cat␣alignment␣:␣", alignment[5]
```

The figures below shows the alignment plot for the ELANE gene for the different species:

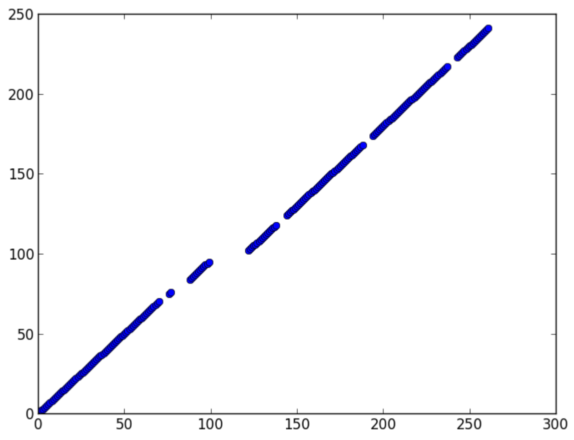

Figure 1: Human - chimpanzee alignment for the ELANE gene. 5 character region match.

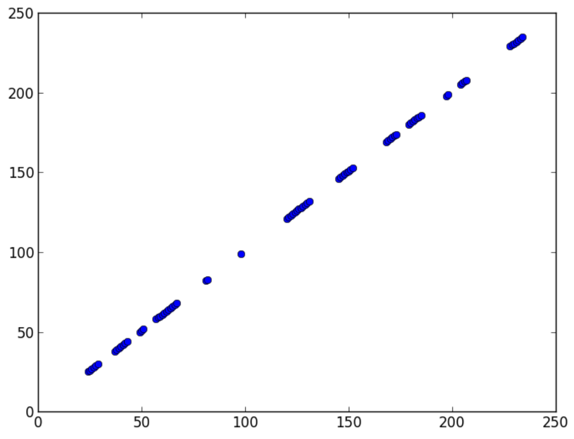

Figure 2: Human - dog alignment for the ELANE gene. 5 character region match.

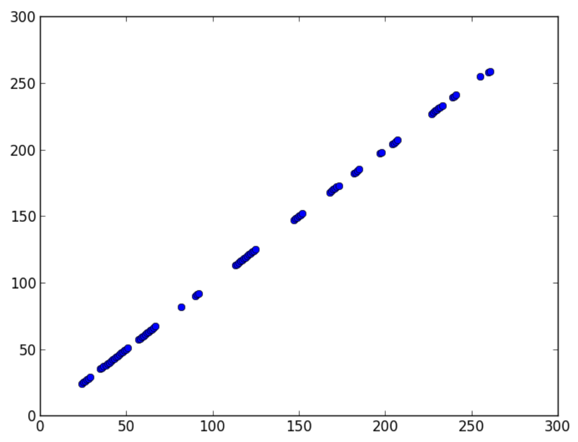

Figure 3: Human - cat alignment for the ELANE gene. 5 character region match.

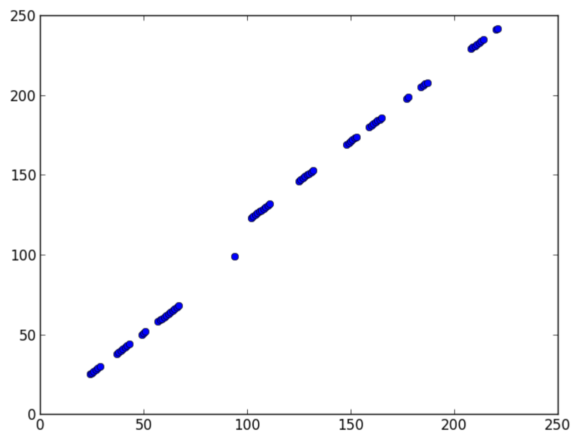

Figure 4: Chimpanzee - dog alignment for the ELANE gene. 5 character region match.

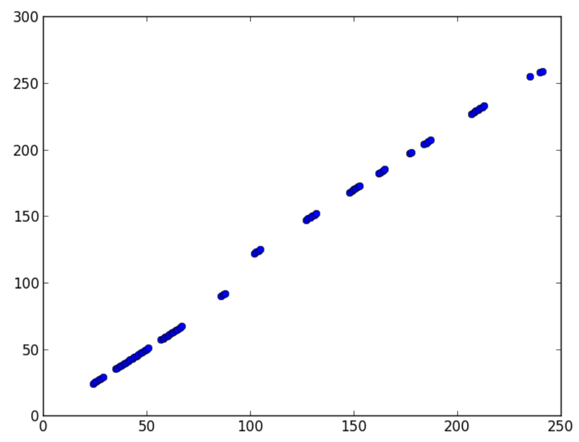

Figure 5: Chimpanzee - cat alignment for the ELANE gene. 5 character region match.

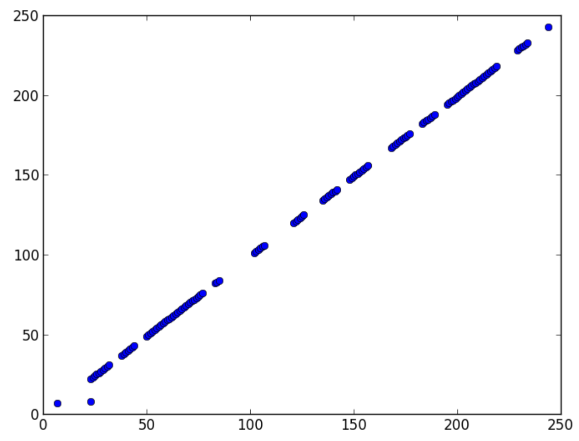

Figure 6: Dog - cat alignment for the ELANE gene. 5 character region match.

From the result of the alignment I eventually plotted a phylogenetic tree for the humaan, the chimpanzee, the dog and the cat:

```
from Bio import Phylo

tree = Phylo.read("tree.txt", "newick")
tree.rooted = True
Phylo.draw(tree)
```

The figure below shows the output of the code above:

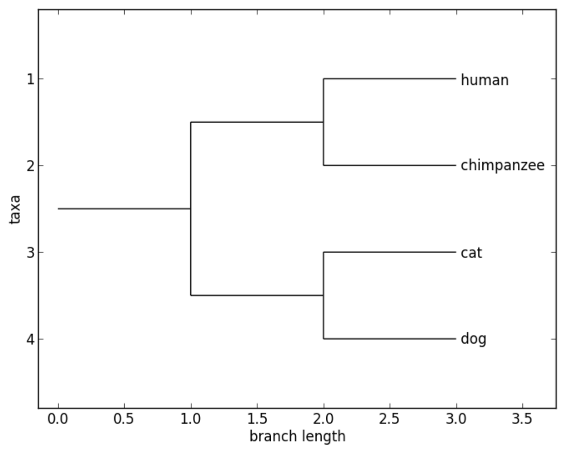

Figure 7: Phylogenetic tree for the humaan, the chimpanzee, the dog and the cat.