Tyrosine Kinase

Sonal Sannigrahi

June 2020

1 Introduction to In and Out Kinases

For each Kinase, there is a conserved activation loop responsible for regulating kinase activity. This activation loop is characterised by DFG and APE motifs. Most kinase inhibitors bind in the region of the ATP binding site, amongst these the Type 1 inhibitors bind to the "Active Conformation" and are associated with the DFG-in conformation of the activation loop. Likewise, the Type 2 inhibitors bind to the "Inactive Conformation" of the same, and are associated to a DFG-out conformation. In the diagram below, the activation loop and the subsequent DFG-in and DFG-out conformations are marked out:

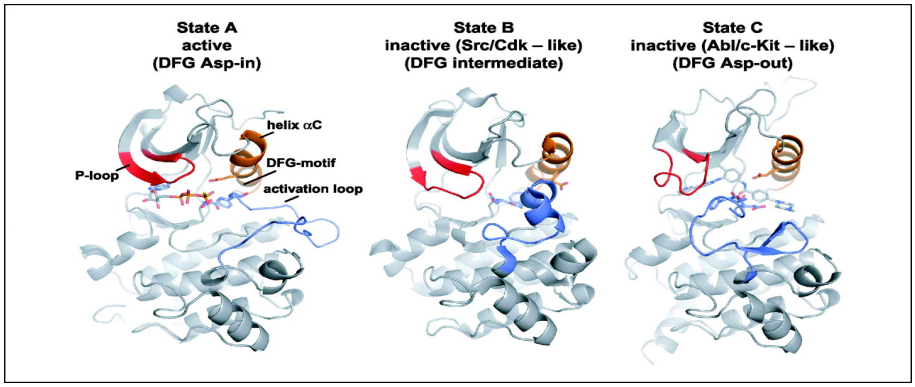

Figure 1: Structurally defined conformational states of the c-Src and Abl kinase domains. Image from Cancer Research

2 Sequence Alignments and Discriminating Positions

In order to compare sequences of proteins, it's necessary to consider the correct alignment of the same. For example, we must consider gaps in the sequences and possibly shifts within recurring motifs. An example can be seen at the end of this document.
To mark out the **IN** versus **OUT** conformations, we shall look at the most closely aligned version of the sequences.

3 Entropy and Scoring Criterion

In a protein sequence, we define the Shannon Entropy as follows:

$$H = -\sum_{i=1}^{M} P_i log_2 P_i$$

Where P_i represents the fraction of residues of amino acid type i, and M is the number of amino acid types (20). In order to interpret how conserved or variable a given sequence alignment is, we look at the value of H which ranges from 0(completely homogeneous) to 4.322(completely variable). Typically $H > 2$ is considered to be variable while $H \leq 1$ is highly conserved. However, it is to be noted that to make such conclusions we must consider over approximately 100 sequence alignments.

With this definition of entropy, we can construct a scoring function which would allow us to determine if a position is a discrimination amino acid position or not. A good discriminating position should split the sequence alignment into two relatively conserved classes. If considering these two classes separately, joining them back together should therefore result in a **high entropy gain**. Thus, we can define our scoring function as:

$$Score(S) = I(S) - H(S_1) - H(S_2)$$

Here, S represents the sequence column we are considering, $I(S)$ is the initial entropy of the whole sequence column, $H(S_1)$ is the entropy of the first class as determined by the discriminating position, and $H(S_2)$ is the entropy of the second resulting class. Along with a high entropy gain, we should also look to minimise the entropy of the two classes themselves.

As the distribution of the two classes of DFG-in and DFG-out conformations is never quite equally split, the score function needs to be updated to account for the individual proportions of the sequence columns occupied by each class.

$$Score(S) = I(S) - w_1 H(S_1) - w_2 H(S_2)$$

Here, the weights are as follows:

$w_1 = \frac{l_1+l_2}{2l_1}$, with l_1 and l_2 being the number of positions in the column occupied by class 1 and 2 respectively.

$w_2 = \frac{l_1+l_2}{2l_2}$, with l_1 and l_2 being the number of positions in the column occupied by class 1 and 2 respectively.

3.1 Defining new score function

Now, we want to calculate I(S), which is essentially going to be the same entropy as before but with additional factors. The goal is to calculate the entropy as if the two classes had the same size. In order to do so, we will redefine probabilities and size of one class, let's say class 1, so that it has the same size as class 2.

Assume we initially had l_1 amino acids in class 1 and l_2 amino acids in class 2. Let's consider one type of amino acid in class 1 and name it A. Let a be the number of this amino acid type in the class 1. Let its probability be p. Then $p = \frac{a}{l_1}$. Since we want to resize class 1 to the size of class 2, we will have new number of amino acids of this type and it will be $a' = a\frac{l_2}{l_1}$ and its new probability in population of two classes of the same size is $p' = \frac{a'}{2l_2} = \frac{p}{2}$. For the amino acid in class two, we also have to assign a new probability and it will be $p' = \frac{a}{2l_2} = \frac{p}{2}$ where p was its probability in class 2. Once we have calculated new probabilities, we can proceed with usual Shannon entropy to calculate I(S).

Let's analyze few simple cases to see if our model for score function gives reasonable results.

case 1:
We have class 1 of l_1 elements of type A and class 2 of l_2 elements of type B.
Since classes are homogeneous, $H(S_1) = H(S_2) = 0$.
$p_A = 1$, $p_B = 1$. Let's calculate new probabilities: $p'_A = \frac{p_A}{2}$ and $p'_B = \frac{p_B}{2}$.

Assume type A and B are the same. When we merge two classes we get one homogeneous class and its entropy is 0, so I(S) = 0. Entropy gain is equal to 0 which is exactly what we wanted to obtain.

Assume type A and B are different. Then $I(s) = -2 * \frac{1}{2} * ln(\frac{1}{2}) = -ln(\frac{1}{2}) = ln(2)$. Again this result is expected since we rescaled the classes to the same size. The entropy gain is score = ln(2) - 0 - 0 = ln(2).

case 2:

Assume that classes 1 and 2 have all different amino acids each. In class 1 there are 20 different amino acids with same probabilities and in class 2 there are 20.

$H(S_1) = -20 * \frac{1}{20} ln(\frac{1}{20}) = ln(20)$
$H(S_2) = -20 * \frac{1}{20} ln(\frac{1}{20}) = ln(20)$
$I(S) = -20 * (\frac{1}{40} + \frac{1}{40}) ln(\frac{1}{40} + \frac{1}{40}) = ln(20)$
$score(S) = \frac{ln(4l_1 l_2)}{2} - ln(l_1) - ln(l_2)$
Note that the entropy of individual classes is the same as I(S) so entropy gain is negative, which means that this configuration is highly undesirable, which is true. We want more or less homogeneous individual classes with a heterogeneous merged class.

3.2 Label Based Score

Another scoring method considered was one based on the DFG-In and DFG-Out labels of the sequences.

Consider a particular alignment column with two known subsets A and B. Let S_A and S_B be the sequence entropies of each subset. Now let us consider the column, not with the amino acid types, but with the known In/Out labels. Now, for each subset A and B, we can compute another entropy, S'_A = the entropy of subset A with its I and O labels:

$$S'_A = -\frac{A_{in}}{A_{total}} ln(\frac{A_{in}}{A_{total}}) - \frac{A_{out}}{A_{total}} ln(\frac{A_{out}}{A_{total}})$$

We calculate a similar entropy for the subset B. Now let us label S_A and S_B to be the total sequence entropies, now a good score function would be:

$$Score = -S'_A - S'_B - S_A - S_B$$

The first two terms will push the subsets to match the true In/Out subsets while the next two terms will push the subsets to have high conservation.

4 Monte Carlo Method

Using the scoring function as described above, we can use a **Monte Carlo Method** to compute the best discriminating position for a given sequence alignment.

```
1
2  from scipy.stats import boltzmann #used for distribution
3  import random
4
5  def entropy(list_input):
6      """Calculate Shannon's Entropy per column of the alignment (H=-\sum_{i=1}^{M} P_i
       \,log_2\,P_i)"""
7
8      import math
9      unique_base = set(list_input)
10     M   =  len(list_input)
11     entropy_list = []
12     # Number of residues in column
13     for base in unique_base:
14         n_i = list_input.count(base) # Number of residues of type i
15         P_i = n_i/float(M) # n_i(Number of residues of type i) / M(Number of residues
        in column)
16         entropy_i = P_i*(math.log(P_i,2))
17         entropy_list.append(entropy_i)
18
19     sh_entropy = -(sum(entropy_list))
20
21     return sh_entropy
22
23 def entropy1(c1, c2):
24     """Computes entropy of two classes combined while also taking into account the
       size of classes"""
25     import math
26     unique_base = set(c1).union(set(c2))
27     M1   =  len(c1)
28     M2   =  len(c2)
29     entropy_list = []
30     # Number of residues in column
31     for base in unique_base:
32         n_1 = c1.count(base) # Number of residues of type i
```

```
        n_2 = c2.count(base)
        if M1 == 0:
          P_i = n_2/float(M2) # n_i(Number of residues of type i) / M(Number of
    residues in column)
        elif M2 == 0:
          P_i = n_2/float(M1)
        else:
          P_i = n_1/float(2*M1) +n_2/float(2*M2)
        entropy_i = P_i*(math.log(P_i,2))
        entropy_list.append(entropy_i)

    sh_entropy = -(sum(entropy_list))

    return sh_entropy

def score(S, c1, c2):
    """
    Computes the Score for a sequence column by the scoring method

    Input: S = sequence entropy, c1 = class 1, c2 = class 2
    Output: score of S

    """

    #Currently working on fixing implementation

    c1_list = list(c1.keys())
    c2_list = list(c2.keys())
    l_1 = len(c1_list)
    l_2 = len(c2_list)
    if c1_list != [] and c2_list != []:
      I_S = entropy1(c1_list,c2_list)
      H_C1 = entropy(c1_list)
      H_C2 = entropy(c2_list)
      Score = I_S - H_C1/2 - H_C2/2  #-log(2,2)
    else:
      Score = -1000
    return Score

def Insertion(S, class_1, class_2):
    """Adds randomly one new X
    returns class 1 and class 2"""

    #choosing a spot where to put X
    if len(class_2) > 1:
      rand_choice = random.sample(set(class_2.keys()),1)[0]
      del class_2[rand_choice]

      class_1[rand_choice] =  S[rand_choice]

    return (class_1,class_2)

def Deletion(S, class_1, class_2):
    """Deletes randomly one X
    returns class 1 and class 2"""

    #choosing a spot where to delete X
    if len(class_1) > 1:
      rand_choice = random.sample(set(class_1.keys()),1)[0]
      del class_1[rand_choice]

      class_2[rand_choice] = S[rand_choice]

    return (class_1,class_2)

def Swap(S, class_1, class_2):
    """swap two randomly chosen X
    returns class 1 and class 2"""

    #choosing spots to swap X
    if len(class_1) != 0 and len(class_2) != 0:
      rand_choice1 = random.sample(set(class_1.keys()),1)[0]
      rand_choice2 = random.sample(set(class_2.keys()),1)[0]


      del class_1[rand_choice1]
```

```
      del class_2[rand_choice2]

      class_1[rand_choice2] = S[rand_choice2]
      class_2[rand_choice1] = S[rand_choice1]
    return (class_1,class_2)

def MonteCarlo(S, max_iter, threshold):
    """
    Computes the discriminating position for a sequence column

    Input: S = sequence column, max_iter = max iterations, threshold = threshold of acceptance for boltzmann distribution (between 0 and 1)
    Output: 2 classes
    """
    n = len(S)
    S_copy = S.copy()
    num_class1 = int(0.30*n) #number in class1
    print(num_class1)
    for _ in range(num_class1): #intialisation
        rand_position = random.randint(0,len(S)-1)
        while S_copy[rand_position] == 'X':
            rand_position = random.randint(0,len(S)-1) #reselect if used position
        S_copy[rand_position] = 'X'

    #Monte Carlo with Insertions and Deletions

    s_entropy = entropy(S)
    n_iter = 0 #counter for iterations
    class_1 = dict() #class 1 determined by X's
    class_2 = dict() #class 2 is rest of S
    for i in range(len(S)):
        if S_copy[i] == 'X':
            class_1[i] =S[i]
        else:
            class_2[i] =S[i]

    print("initial class c1: \n")
    print(class_1)
    print('\n initial class c2: \n')
    print(class_2)

    res = score(S, class_1, class_2)

    while n_iter < max_iter:
        r = random.random()
        if r < 0.25: #25% chance to do an insertion

          (c1,c2) = Insertion(S, class_1, class_2)


        elif r < 0.5: #25% chance to do a deletion

          (c1,c2) = Deletion(S, class_1, class_2)

        else: # 50% chance to do a swap

          (c1,c2) = Swap(S, class_1, class_2)

        res2 = score(S, c1, c2)

        #Boltzmann Distribution coefficient
        delta_S = abs(res2 - res)
        lambda_, N = 0.2, 19
        size = 6000
        ind1 = random.randint(0,size - 1)
        rv = boltzmann.rvs(lambda_, N, size=size)/N
        beta = rv[ind1] #pick float between 0 and 1 with boltzmann distribution
        if  (res2 > res):
            res = res2
            class_1 = c1
            class_2 = c2
        elif (beta*delta_S > threshold): #if better score i.e. higher entropy gain
            res = res2
            class_1 = c1
            class_2 = c2

```

```
        n_iter += 1
        print(res)
        print("\n")
        print(c1)
        print("\n")
        print(c2)

        print("\n\n")


    return class_1, class_2

    "
```

Below is an example of sequence alignment as mentioned earlier:

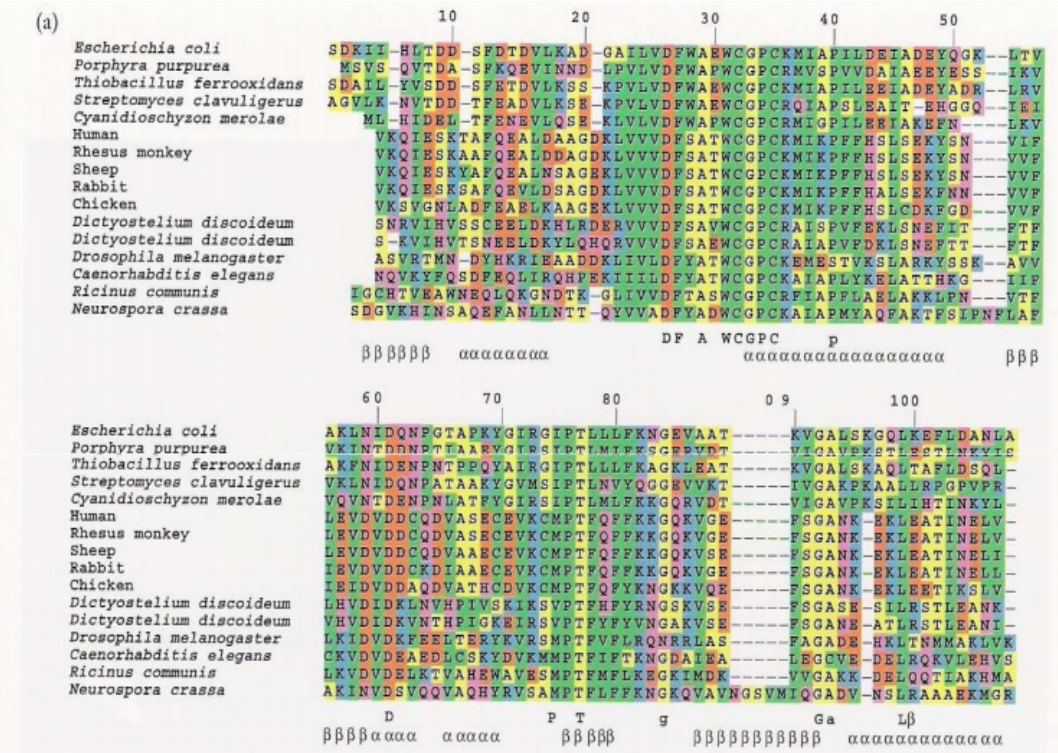

Figure 2: Alignment of amino acid sequences of E.coli thioredoxin and homologues.