Implementation of a MSA-based method for detecting coevolution

Vikas Gupta
Oleg Sitsel

Supervised by: Jotun Hein

Current Research in Bioinformatics 2010
Aarhus University
Denmark

Table of Contents

1. Introduction....3
2. Coevolutionary model....4
3. Multiple sequence alignments....7
4. Inferring phylogeny 8
5. Examined interacting pairs....9
   5.1 Copper pump – copper chaperone....10
   5.2 Elongation factors Ts-Tu.... 10
   5.3 2-oxoisovalerate dehydrogenase....11
6. Experimental procedures....12
   6.1 Dataset.... 12
   6.2 Removing redundancy....12
   6.3 Calculating the phylogenetic parameters....12
   6.4 Calculating independent and correlated scores 13
   6.5 Generating controls.... 13
   6.6 Calculating inter-residual distances....13
   6.7 Calculating the frequency of residues....14
7. Results and discussion.... 14
   7.1 Coevolution score comparision within protein domains....14
   7.2 Coevolution score comparison with inter-residual distances....19
   7.3 Coevolution score comparison with co-conservation....20
8. Conclusions....23
9. Future perspectives.... 24
10. Acknowledgments....24
11. References....24
12. Supplements....25
   12.1 Script 1: coevolution score generation....25
   12.2 Script 2: removing redundancy....29

1. Introduction

Revealing intra-molecular coevolution between amino acid residues has been one of the most important goals of molecular biologists, bioinformaticians and of new emerging areas of research. Many methods have been devised to understand the evolutionary dynamics of proteins through the examination of multiple sequence alignments (MSA's). Although this approach has dramatically improved our understanding of the mutational dynamics of proteins, the complexity of proteins' mutability is beyond methods focusing on the analysis of linear sequences. The last decade has witnessed the emergence of a plethora of mathematical methods and computational tools aimed at drawing the spatial, functional and evolutionary dependencies between amino acid sites within a protein. The coevolutionary relationships between amino acid sites are however swamped in a background of different interacting factors governing the evolutionary dependencies of amino acids. During the last few years many efforts have been devoted to uncover evolutionary relationships between amino acid sites belonging to the same or to different proteins. The importance such studies has been underpinned by many examples where dependencies between amino acid sites have unearthed the functional importance of residues (Fares and Travers 2006; Travers and Fares 2007).

The intrinsic complexity of the evolutionary dependencies between amino acid sites has however hampered the development of sensitive methods to detect functional coevolution. In fact, coevolution between two amino acid sites can be decomposed into stochastic coevolution, functional coevolution and interaction coevolution. Each of these factors has different weights depending on, among other factors, how realistic models are to detect coevolution and quality of the multiple sequence alignment. The sensitivity of most of parametric and non-parametric methods to detect the functional coevolution has been always compromised by the ability of these methods to disentangle the different types of coevolution.

Algorithms for detecting coevolving positions can be classified into two categories: the tree-aware method, which incorporates knowledge of phylogeny, and tree-ignorant, which does not. Tree ignorant methods are frequently orders of magnitude faster, but are widely held to be

insufficiently accurate because of a confounding of shared ancestry with coevolution.

In our studies, we have developed a model for detecting coevolution based on multiple sequence alignment of interacting protein domains and then we have compared results using the tree-aware and tree-ignorant methods. So far we have considered interacting protein datasets such as 2-oxoisovalerate dehydrogenase subunits, elongation factors Ts and Tu, and copper transporting P-type ATPases with copper chaperones. Our tree-ignorant model is based on a single parameter, while in the tree-aware method we consider a constant molecular clock and use distances between any two tree leaves as second parameter of the model.

2. Coevolutionary model

An overview of our model is provided by **Figure 1**, taken from Yeang & Haussler (2007).

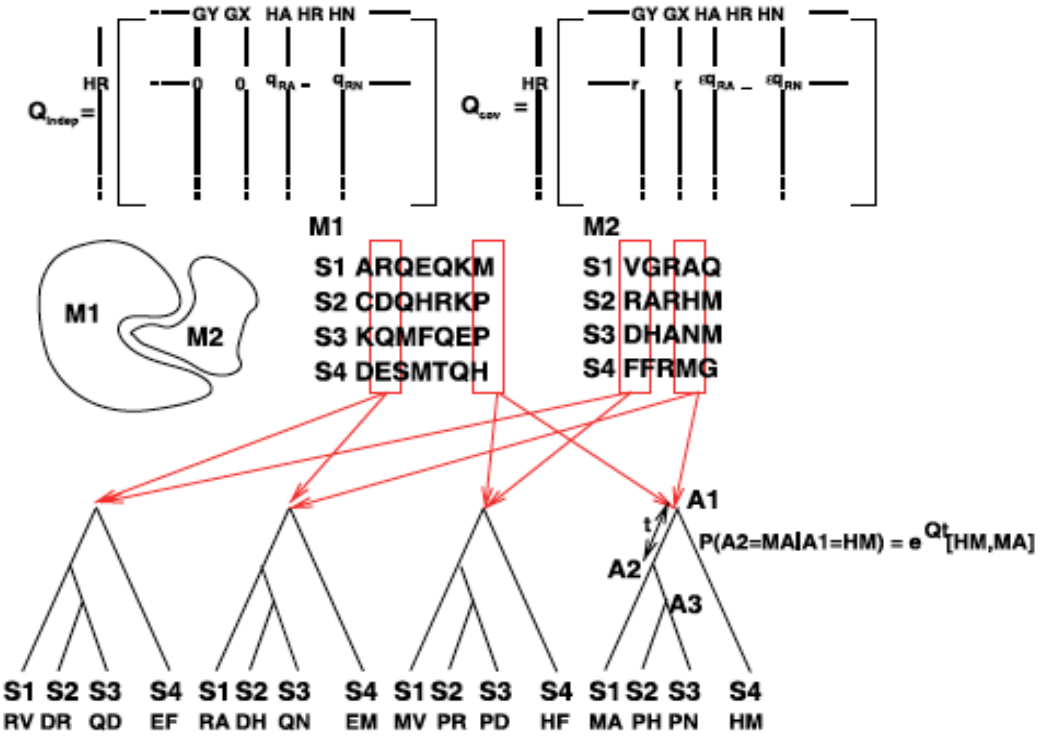


Figure 1

We extend the continuous-time Markov process (CTMP) sequence substitution to model coevolution of amino acid position pairs. The state transitions of a CTMP at an infinitesimal time interval follow a matrix differential equation (**Equation 1**). The instantaneous transition rates are specified by a 24 × 24 substitution rate matrix Q. A CTMP of an amino acid pair is obtained by concatenating the sequence states of two amino acid positions. The substitution rate matrix of two independent amino acid positions can be directly derived from the CTMP of single sites. However, the rate matrix of a general two-component CTMP has much fewer constraints and a larger dimension (576 × 576). We simplify the substitution rate matrix by penalizing all the entries of single changes and rewarding all the entries of double changes with the same weight factors.

The sequence substitution of a single amino acid is modeled by a CTMP. Denote by x(t) the sequence composition. P(x(t)) is 1x24 probability vector of x(t) and follows a Markov process at an infinitesimal time interval:

$$\frac{dP(x(t))}{dt} = P(x(t))Q \qquad \text{Equation 1}$$

where Q is a 24 × 24 substitution rate matrix. Each row of Q must sum to 0 in order to make components of P(x(t)) sum to 1. In this work we used the Dayhoff matrix of amino acid substitution. The transition probability P(x(t)|x(0)) at a finite time interval t is given by the matrix exponential e^{Qt}, which is the solution of Equation 1:

$$P(x(t) = b | x(0) = a) = e^{Qt}[a, b] \qquad \text{Equation 2}$$

Define x(t) = (x1(t),x2(t)) as the joint state of two amino acids. The sequence substitution follows the same equation for the single-site evolution (Equation 1), but the dimensions of the probability vector (1x576) and the rate matrix (576x576) are much larger. If two sites are

independently evolved, then the joint rate matrix Q^i_2 can be derived from the rate matrix of single sites:

$$Q_2^i[(a_1,a_2),(b_1,b_2)] = \begin{cases} Q[a_1,b_1] & \text{if } a_2 = b_2, \\ Q[a_2,b_2] & \text{if } a_1 = b_1, \\ -Q[a_1,b_1] - Q[a_2,b_2] & \text{if } a_1 = b_1, a_2 = b_2, \\ 0 & \text{otherwise.} \end{cases}$$

Equation 3

Q^i_2 [(a1, a2), (b1, b2)] specifies the sequence substitution rate of the independent model from state (a1,a2) to state (b1,b2). In Q_i^2, the rate of a single amino acid change is equal to the corresponding rate in the single site rate matrix Q, and the rates of double amino acid changes are all zero. For example, Q^i_2 [HR, HA] = Q[R, A] and Q^i_2 [HR, GX] =0. This is intuitive since off-diagonal entries of Q^i_2 specify the transition probabilities at an infinitesimal time interval. At an infinitesimal time interval, at most one transition occurs for two independent CTMPs.

Each diagonal entry of Q^i_2 is again -1 and multiplies the sum of other entries in the same row. A true coevolutionary model should reward transitions into the sequence states of selective advantages and penalize the transitions of opposite directions. Due to the difficulty of finding this true model, we constructed a simplified model by reweighing the entries of the independent rate matrix to penalize single transitions and to reward double transitions:

$$Q_2^c[(a_1,a_2),(b_1,b_2)] =$$

$$\begin{cases} \varepsilon Q_2^i[(a_1,a_2),(b_1,b_2)] & \text{if } (a_1 = b_1) \vee (a_2 = b_2), \\ r_{(a_1,a_2)} & \text{if } (a_1 \neq b_1) \wedge (a_2 \neq b_2), \\ -\sum Q_2^c[(a_1,a_2),(b'_1,b'_2)] & \text{if } (a_1 = b_1) \wedge (a_2 = b_2). \end{cases}$$

Equation 4

Single amino acid transitions are penalized by being multiplied by a fixed number ε, where $\varepsilon < 1$. Double amino acid transitions from the same state (α_1, α_2) are rewarded by replacing zeroes with an identical quantity

$$r_{(a_1,a_2)} = \frac{-Q_2^i[(a_1,a_2),(a_1,a_2)] - \sum \varepsilon Q_2^i[(a_1,a_2)(b_1,b_2)]}{479}$$

Equation 5

Its values cause diagonal entries in Q^c_2 to be identical to Q^i_2. Q^c_2 will then favor the sequences that have strong covariation between distinct states.

3. Multiple sequence alignments

Sequences have to be aligned amongst themselves before being checked for coevolution.

Multiple alignments of protein sequences are important in many applications, including phylogenetic tree estimation, secondary structure prediction and critical residue identification. Many multiple sequence alignment (MSA) algorithms have been proposed. Two attributes of MSA programs are of primary importance to the user: biological accuracy and computational complexity. Complexity is of increasing relevance due to the rapid growth of sequence databases, which now contain enough representatives of larger protein families to exceed the capacity of most current programs. Obtaining biologically accurate alignments is also a challenge, as the best methods sometimes fail to align readily apparent conserved motifs.

For aligning our sequences prior to coevolution analysis, we used the multiple alignment software MUSCLE, a MSA program that is considered to provide improvements in accuracy and speed over the more traditional ClustalW (Edgar. R.C. 2004).

The workflow MUSCLE uses is the following:

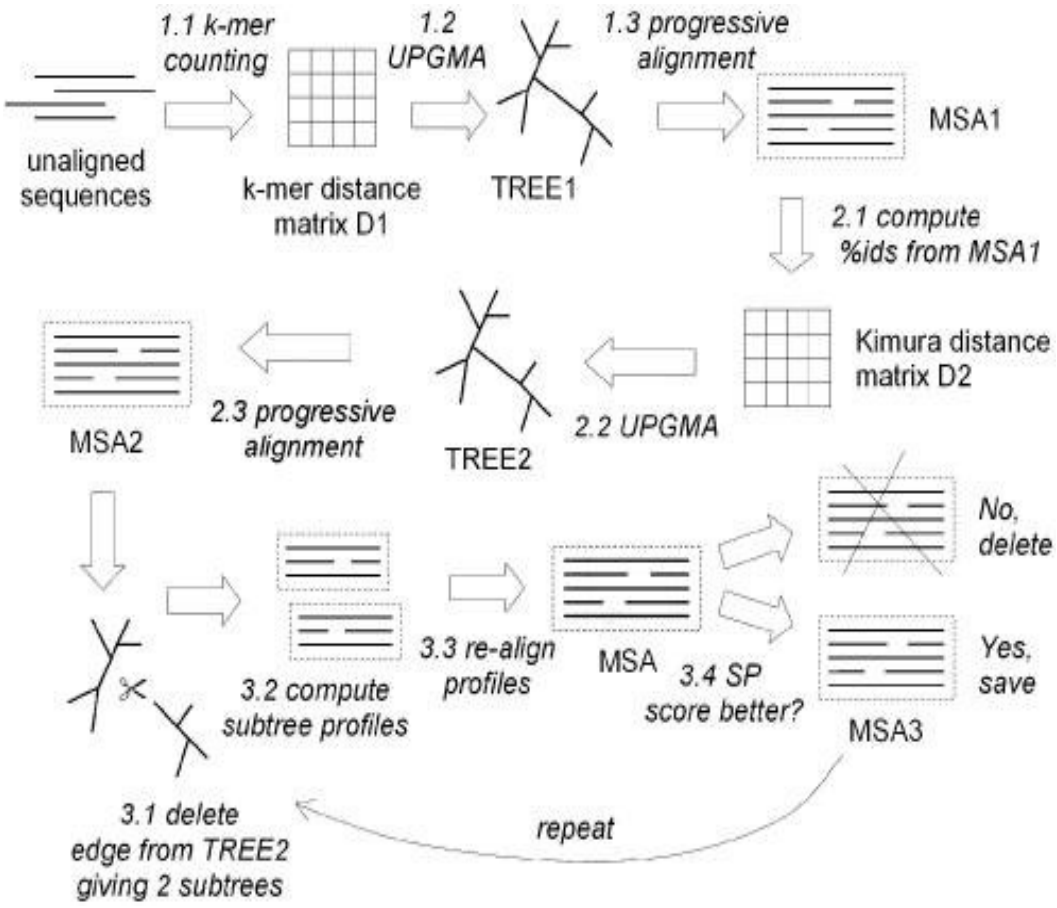


Figure 2

Clarifications:

- A *k*mer is a contiguous subsequence of length *k*, also known as a word or *k*-tuple. Related sequences tend to have more *k*mers in common than expected by chance.
- Distance matrices are clustered using UPGMA (Unweighted Pair Group Method with Arithmetic Mean)

4. Inferring phylogeny

We created tree using ClustalX based on the Neighbour-Joining method. It is a bottom-up clustering method used for the construction of phylogenetic tree. Neighbor-joining is an iterative algorithm. Each iteration consists of

the following steps:

- Based on the distance matrix calculate the matrix Q (defined below).
- Find the pair of taxa in Q with the lowest value. Create a node on the tree that joins these two taxa (i.e., join the closest neighbors, as the algorithm name implies).
- Calculate the distance of each of the taxa in the pair to this new node. Calculate the distance of all taxa outside of this pair to the new node.
- Start the algorithm again, considering the pair of joined neighbors as a single taxon and using the distances calculated in the previous step. Based on a distance matrix relating the *r* taxa, calculate Q as follows:

$$Q(i,j) = (r-2)d(i,j) - \sum_{k=1}^{r} d(i,k) - \sum_{k=1}^{r} d(j,k) \qquad \text{Equation 6}$$

where $d(i,j)$ is the distance between taxa i and j

$$Q_t^c = l \times Q^c \qquad \text{Equation 7}$$

We used calculated distances between each pair of sequences in the tree obtained. Correlated substitution score Q^c for each pair was multiplied by it corresponding distance for a pair of sequence.

where Q_t^c is the correlated score based on the tree-aware method and where l is the distance between two leaves of tree.

5. Examined interacting pairs

Three interacting protein pairs were examined thoroughly during this project. A description of each pair follows.

5.1 Copper pump – copper chaperone

Copper transporting P-type ATPases, more commonly known as copper pumps, are a family of transporters that use a E1-E2 Post-Albers cycle to transport copper ions across membranes either into organelles or into extracellular space. Here, E1 and E2 refer to the two different conformations acquired by the pumps during their functional cycle.

Copper pumps are essential to both eukaryotes and prokaryotes for ensuring correct copper homeostasis. Copper is an essential metal in many enzymes, perhaps the best known of which is Cu-Zn superoxide dismutase, a protein which destroys highly dangerous superoxide radicals. However, copper can be extremely toxic to cells when in excess, not least due to the possibility of hydroxyl radical generation by the Fenton reaction and the potential of copper to bind ectopically to proteins. Therefore, any increase in intracellular copper concentration can be potentially lethal and must be countered by an increased efflux of this ion out if the cytoplasm.

As copper is so toxic, almost no free copper ions are available in cells. Most of the copper is buffered by proteins such as metallothioneins and low molecular weight compounds such as glutathione. This also means that copper cannot be delivered to the pumps in free form. Therefore, copper is delivered to the pump by heavy-metal binding proteins known as copper chaperones. These are relatively small and usually include a conserved CxxC motif for metal binding.

No structure of any copper pumps are as of yet available, although a few chaperone structures have been solved.

5.2 Elongation factors Ts-Tu

Elongation is the stage of protein synthesis during which amino acids are being added to the growing polypeptide chain.

Elongation requires participation of elongation factors EF-Tu (also called EF1A), EF-Ts (EF1B) and EF-G (EF2). Two of these, EF-Tu and EF-G, are small GTP-binding proteins. The sequence of events is as follows: EF-Tu-GTP binds and delivers an aminoacyl-tRNA to the A site on the

ribosome. EF-Tu recognizes and binds all aminoacyl-tRNAs with approximately the same affinity, when each tRNA is bonded to the correct (cognate) amino acid. When the correct anticodon interacts with the mRNA codon, a change in ribosomal conformation occurs which leads to altered positions of active site residues in the bound EF-Tu, with activation of EF-Tu GTPase activity. As GTP on EF-Tu is hydrolyzed to GDP + Pi , EF-Tu undergoes a large conformational change and dissociates from the complex. This allows EF-Ts to interact with EF-Tu in order to reactivate EF-Tu by guanine nucleotide exchange. Interaction with EF-Ts causes EF-Tu to release its bound GDP. Upon dissociation of EF-Ts, EF-Tu binds GTP, which is present in the cytosol at higher concentration than GDP. After binding GTP, EF-Tu is ready for another functional cycle.

The structures of EF-Ts and Ef-Tu are available.

5.3 2-oxoisovalerate dehydrogenase

2-oxoisovalerate dehydrogenase is an enzyme which catalyzes decarboxylation of branched chain 2-oxo acids, such as 2-oxoisovalerate, 2-oxoisocaproate and 2-oxo-3-methylvalerate. An example reaction would be:

$$3\text{-methyl-2-oxobutanoate} + CoA + NAD^{+} \leftrightarrow 2\text{-methylpropanoyl-CoA} + CO_2 + NADH$$

The subunit structure of this enzyme is a tetramer consisting of alpha and beta chains. Mutations in the subunits cause maple syrup urine disease types IA and IB, which is characterized by physical and mental retardation, problems with feeding and a maple syrup odor to the urine.

A 2-oxoisovalerate dehydrogenase structure is available

6. Experimental procedures

6.1 Dataset

Three datasets of interacting proteins were used for this study:

- Elongation factors Ts and Tu (1487 sequences each)
  - Length of aligned sequences:
    - EF-Ts: 862 residues
    - EF-Tu: 640 residues
- 2-oxoisovalerate, subunits A and B (205 sequences each)
  - Length of aligned sequences:
    - Subunit A: 988 residues
    - Subunit B: 1301 residues
- Copper pumps and chaperones (255 sequences each)
  - Length of aligned sequences:
    - Copper pump: 1867 residues
    - Copper chaperone: 362 residues

6.2 Removing redundancy

A number of sequences in the datasets downloaded from Uniprot were found to be redundant. These had to be removed prior to data analysis. This was achieved by using an algorithm, supplemented.

6.3 Calculating the phylogenetic parameters

An algorithm, supplemented, was created to combine the corresponding sequences based on their species of origin into a format that could be used

as an input for constructing a phylogenetic tree.

A Neighbor-joining method was performed on our datasets using ClustalX2.

6.4 Calculating independent and correlated scores

The independent score Q_i, correlated score Q_c and tree-dependent Q_c^t score were calculated. In order to remove false positives that were generating significant background noise, the correlated and tree-dependent score was subtracted by a value of independent score. Graphs generated using these data are supplemented.

6.5 Generating controls

Two types of controls were generated in order to test the statistical significance of our data. Completely random sequences of length comparable to the ones analyzed were generated using the method developed by Stothard (2000), using the software "Random Protein Sequence" from the Sequence Manipulation Suite. In addition, controls that were based on the analyzed sequences but had 20 random mutations introduced into each were generated by the software "Mutate Protein" from the Sequence Manipulation Suite. Graphs of analyzed sequences versus control sequences are provided as a supplement.

6.6 Calculating inter-residual distances

Distances were calculated from the PDB structures which were used in the alignments. Comparison between Cα and scores was done in order to determine whether residues are directly interacting or not. **Figures 15 & 16** show the distribution of the inter-residual distances versus coevolution score.

6.7 Calculating the frequency of residues

The occurrence of residues was calculated in order to distinguish the *coevolving* residues from the *co-conserved* ones. Co-conserved residues generate false positives and must therefore be eliminated. It was observed that few residues with high coevolution score were co-conserved. In **Figures 17 & 18** plot frequency versus score is provided.

7. Results and discussion

7.1 Coevolution score comparision within protein domains

In our study, we have obtained coevolution scores from comparing two interacting domains: A (mostly EF-Ts) and B (mostly EF-Tu). The results are represented in graphical form. The penalizing e-value is shown in the topmost lefthand corner, scores are on the Z-axis, and A with B are on the X and Y axes. Graph data is normalized, which is a compromise between ease of computation and number of peaks / peak sharpness.

In **Figure 3**, it is possible to see the coevolution scores in a variant of EF-Ts and EF-Tu with no gaps removed. The gaps generate large peaks. Tree-ignorant (top layer) and tree-aware (bottom layer) methods show the same overall pattern, implying that the relationship between peaks does not change even when taking the time factor into account.

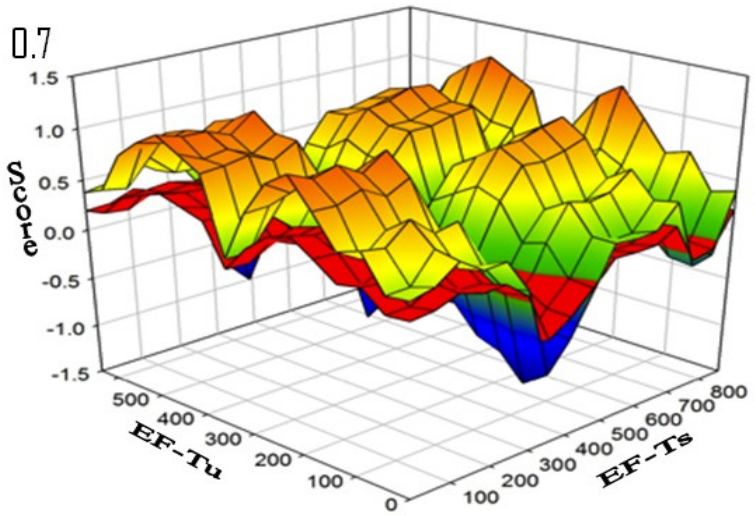


Figure 3

Removing the gaps which generate many false positives changes the picture drastically, as can be seen in the figures that follow.

In **Figure 4**, the Qi-based (coevolution-independent) score is shown. In **Figure 5**, the score is made based on Qc (coevolution-dependent).

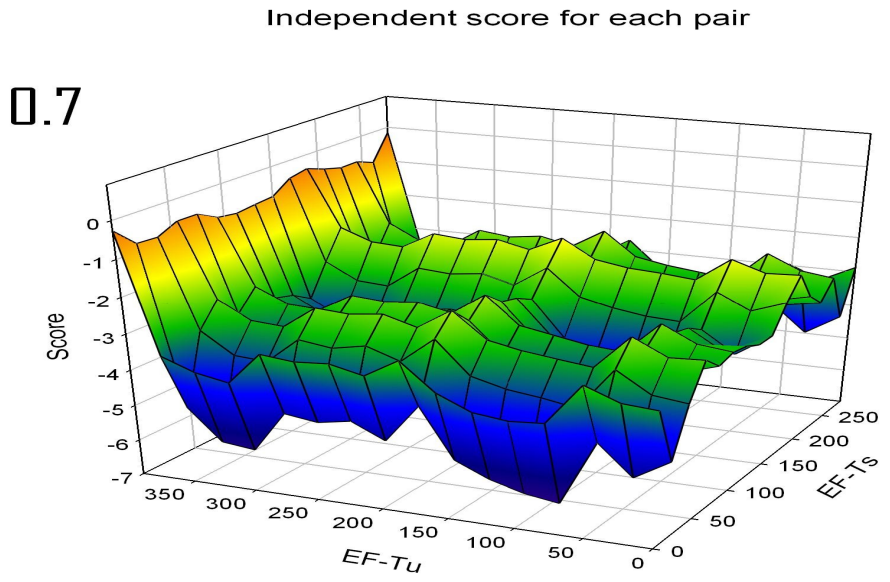


Figure 4

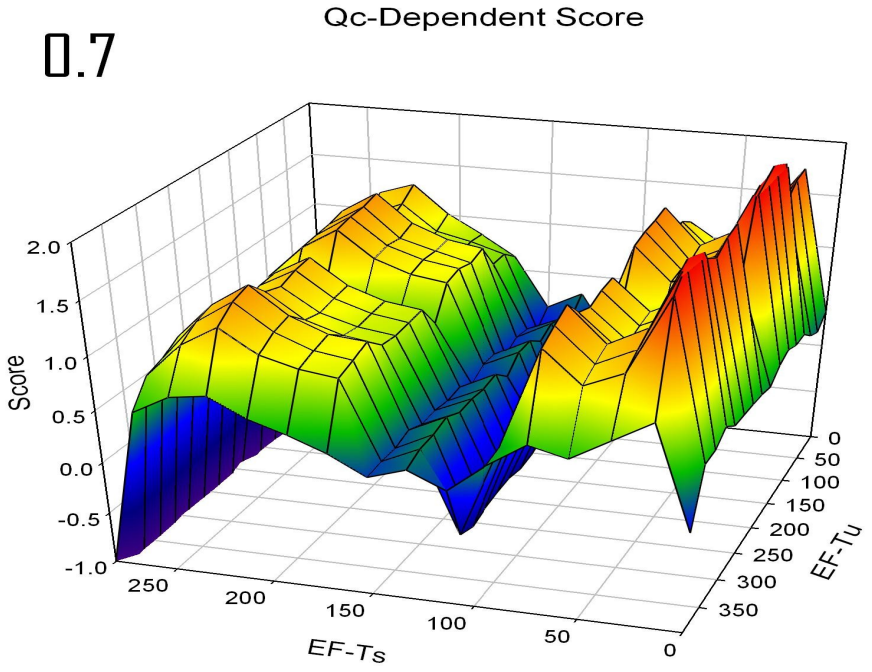


Figure 5

Two different methods were used in order to remove false positives that generated significant background noise. The first, shown in **Figure 6**, involved subtracting the Qi value from Qc. The second, as in **Figure 7**, was based on subtraction of Qi from Q_c^t, the tree-aware score.

As expected, the score values changed slightly, but the overall pattern remained the same.

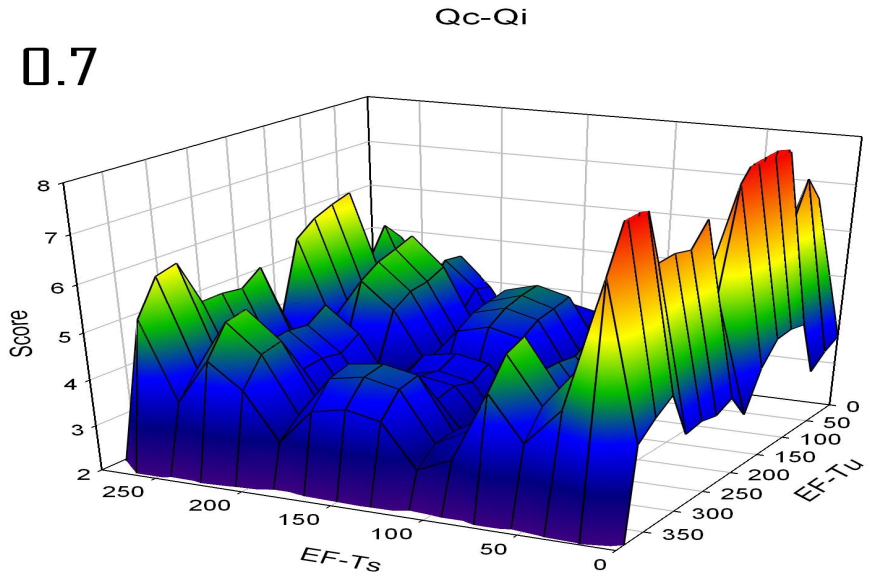


Figure 6

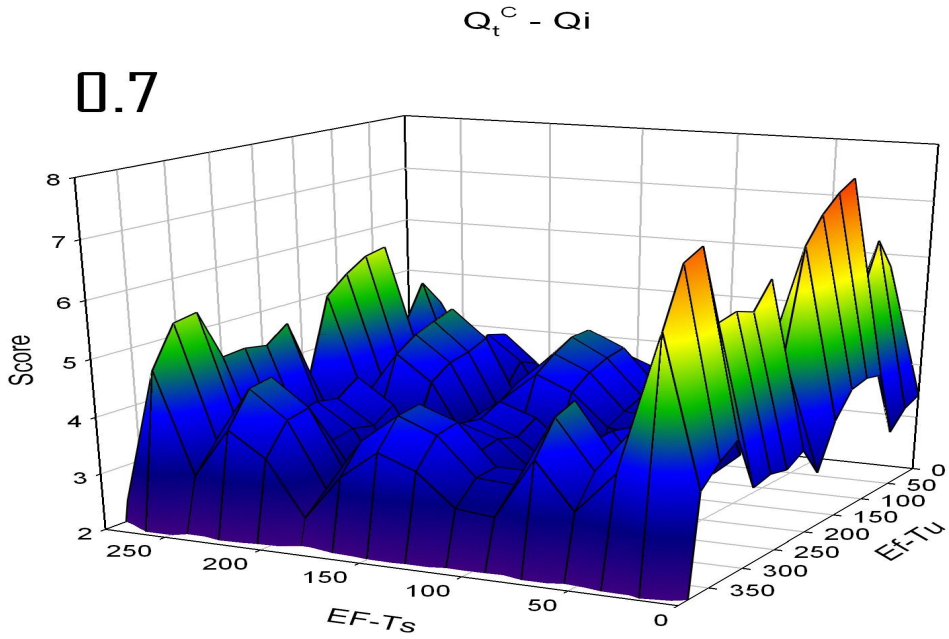


Figure 7

When the penalizing value ε was increased from 0.7 to 0.8, the graphs changed only slightly, with the overall pattern remaining the same (**Figures 8 – 10**). We observed from both sets of results that elongation factor(EF-Ts) has different properties at ends. On the starting side of the sequence alignment we found high score values (normalized), which refers to coevolving sites from residue number 25 to 35 while at the other end of sequence scores are very low as this site is highly variable in the other aligned sequences and in many case, these end residues are missing, leading to a low coevolution score.

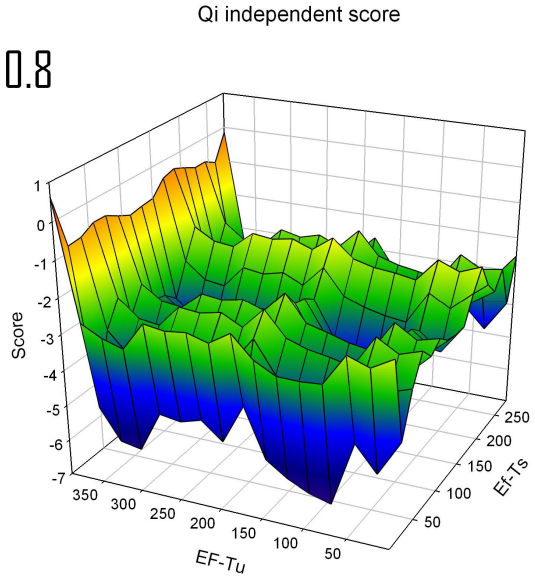


Figure 8

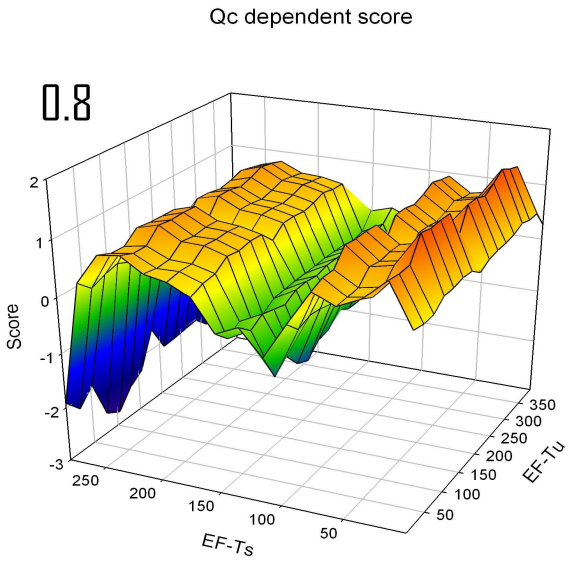


Figure 9

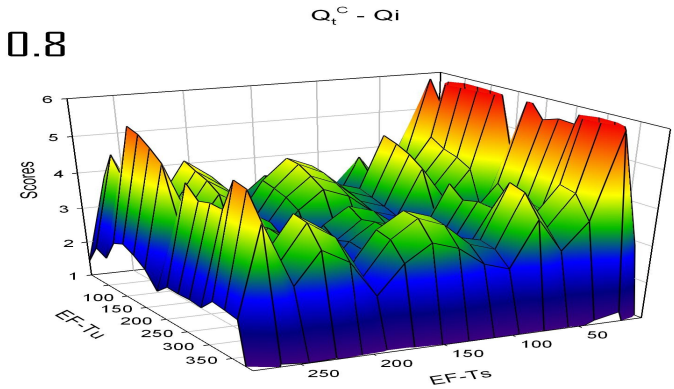


Figure 10

We further analyzed another domain set, obtained from PDB file 1QS0. This data was tested because we have information about its coevolving

residues from the studies performed by David Haussler. **Figure 11** illustrates that we have two sites in subunit B coevolving with three sites with subunit A. Visualization of these residues are shown in later part of report.

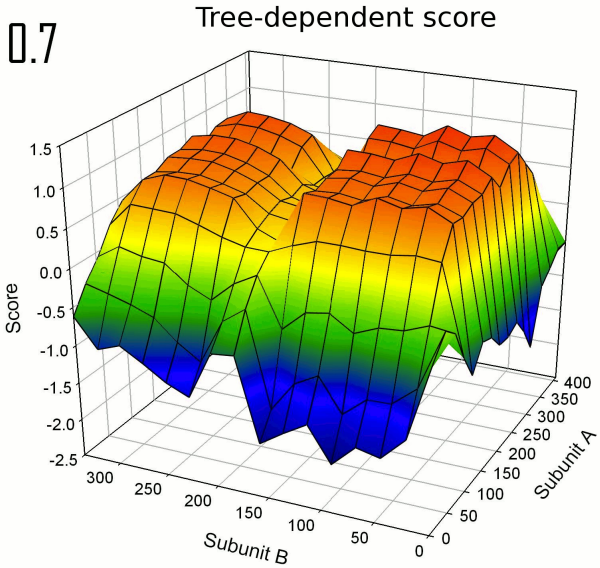


Figure 11

It was also attempted to generate coevolution scores of completely random sequences. The result, shown in **Figure 12**, indicates that no significant coevolution was detected (note the scale). **Figure 13** compares this data with the score of copper pump-chaperone pairs. Additionally, **Figure 14** shows the coevolution scores for Ts-Tu pairs with 20 random mutations in each sequence.

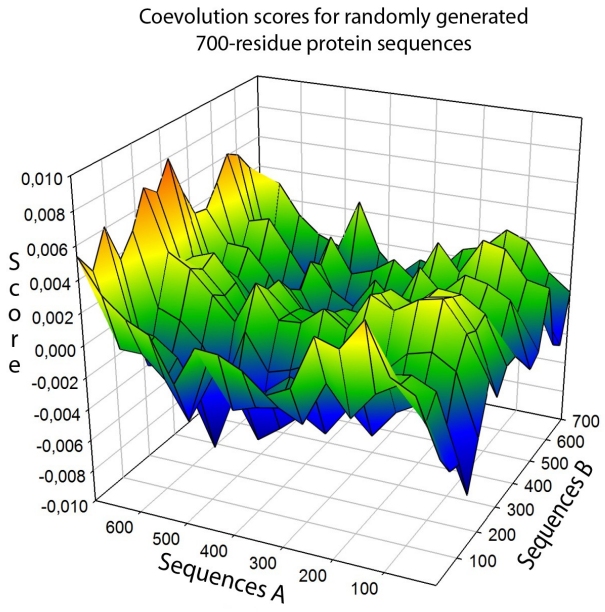


Figure 12

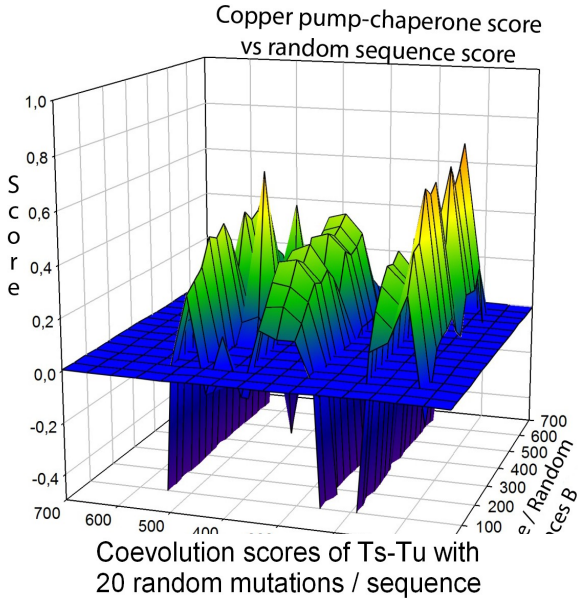


Figure 13

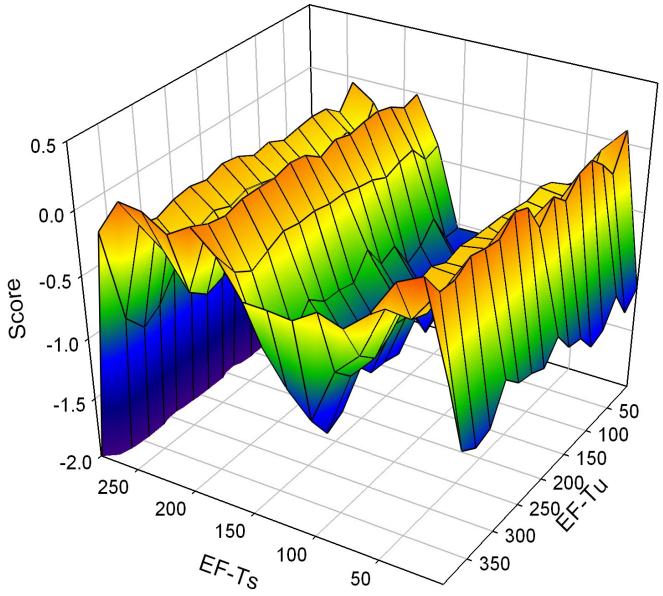
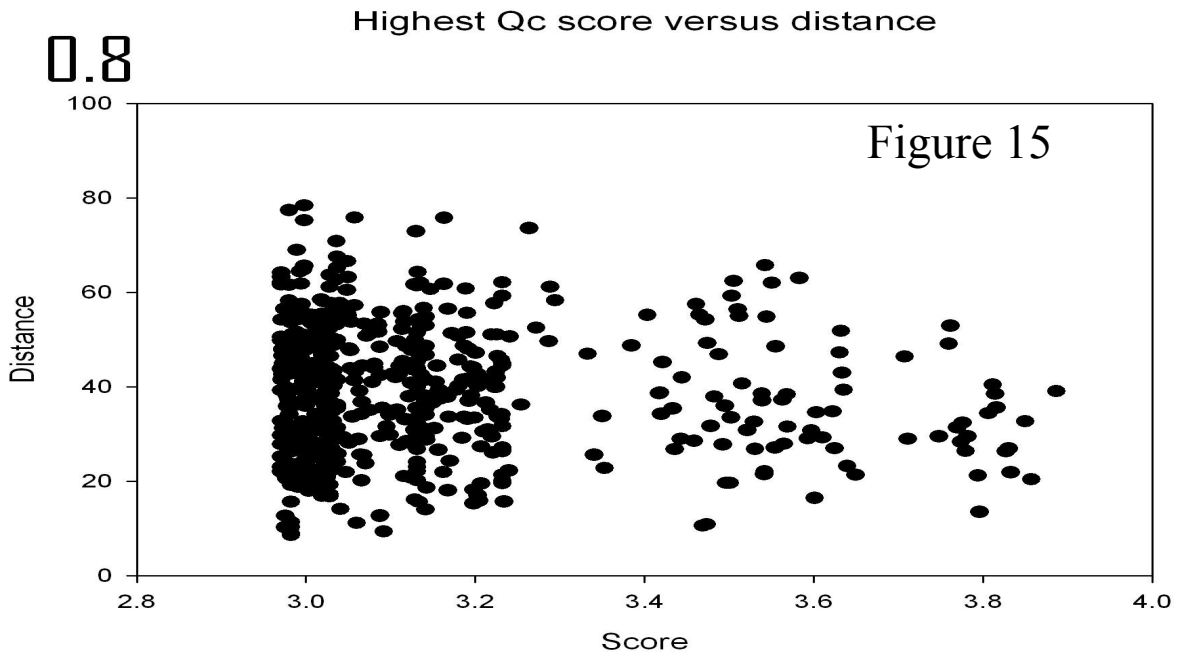
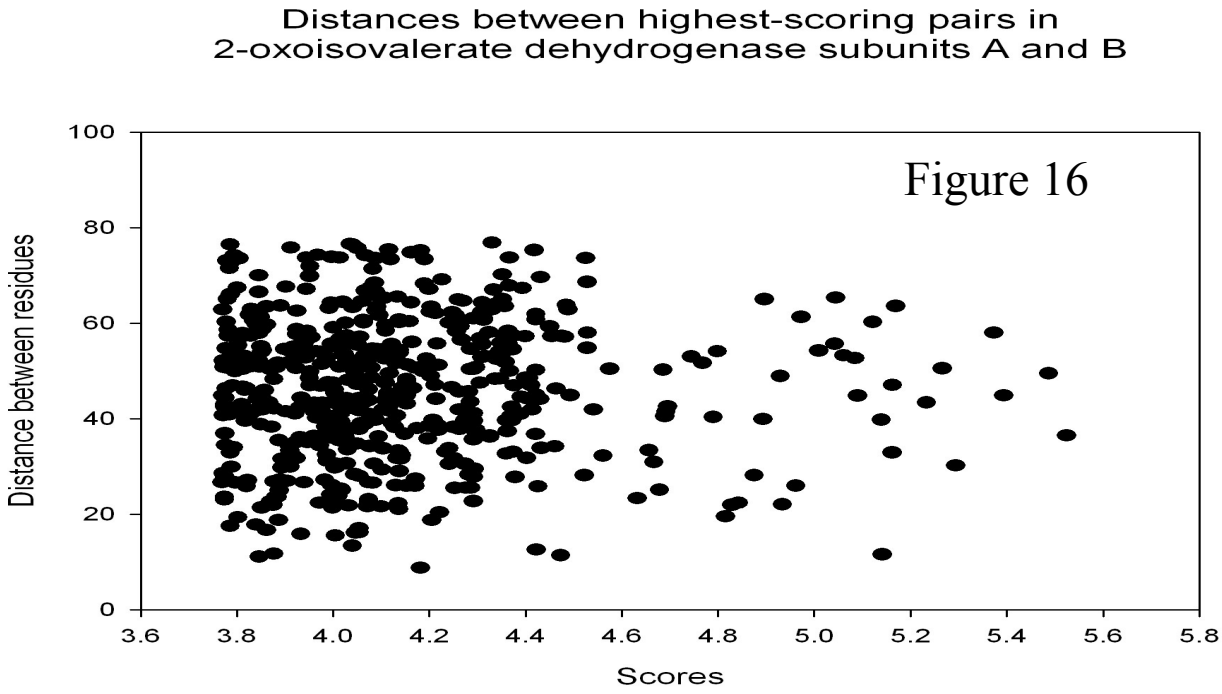


Figure 14

7.2 Coevolution score comparison with inter-residual distances

It was in our interest to calculate a distribution of coevolving scores against inter-residual distances so that we could confirm the fact that coevolving residues are not always in the contact. Cα distances within residues were calculated. We plotted two graphs **(Figures 15 & 16)** for elongation factors Ts-Tu and 2-oxoisovalerate dehydrogenase subunits A and B, respectively. In both plots, we can observe that highly coevolving residues tend to be not very far apart, but still not in close contact.





7.3 Coevolution score comparison with co-conservation

Pairs of positions from two different domains might be fully conserved and could lead to false positive coevolving pairs in some cases. In **Figure 17,** we have demonstrated that highly co-conserved residues in Ts-Tu (>95% identity) are not involved in coevolution (score is Qc-Qi). It can be observed that many highly co-conserved sites have low coevolution scores.

We can also infer that all the positions which have less than 50% identity are generating a very low coevolution score.

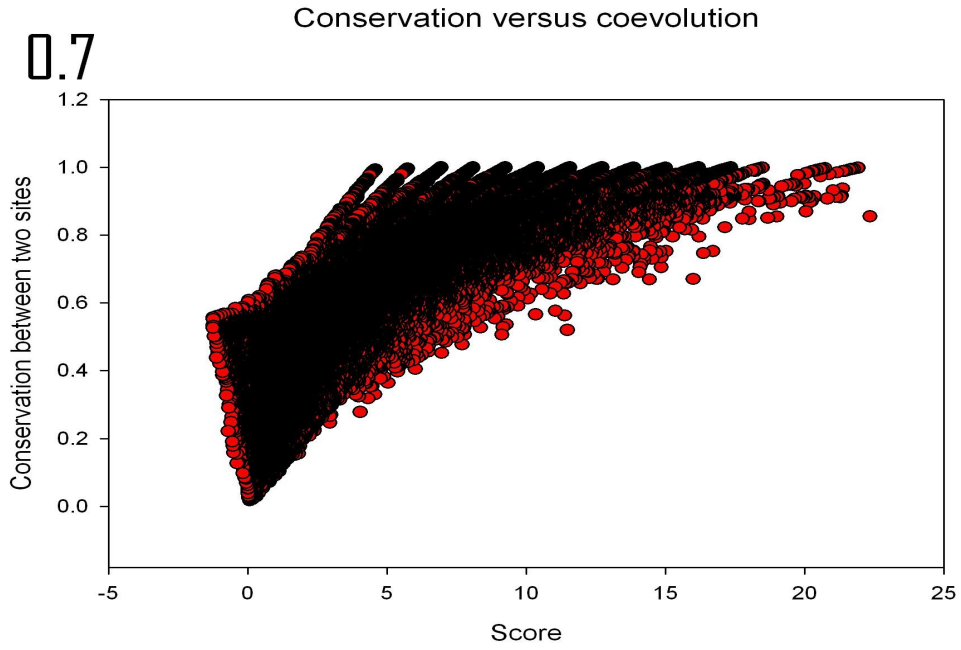


Figure 17

Figure 18 shows the same pair of sequences with a Qc score and a penalizing e-value of 0.8, in which the coevolving residues are highlighted.

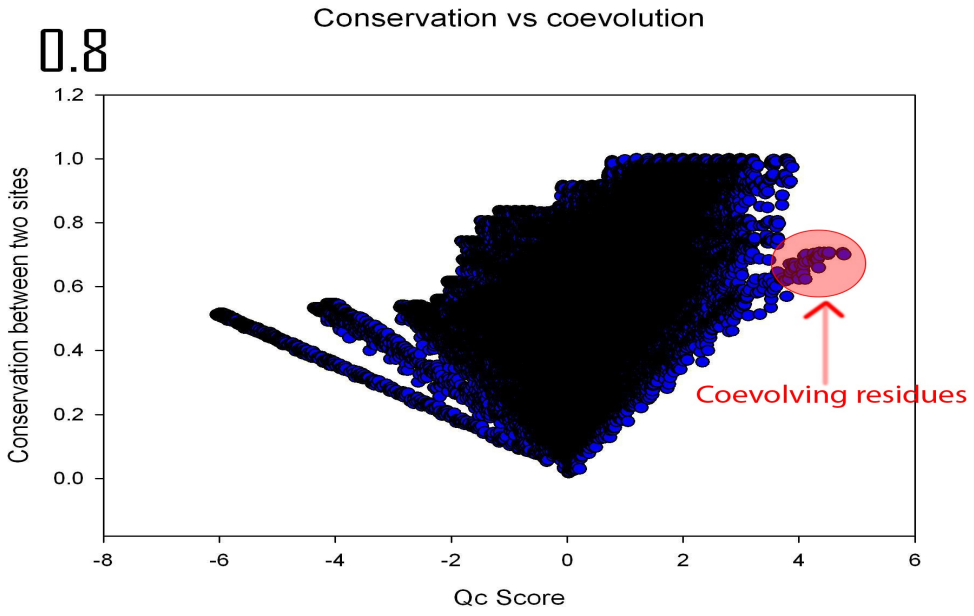


Figure 18

In **Figure 19**, the top 20 coevolving residues from Ts-Tu were highlighted on the PDB structure 1efu and annotated (some were redundant).

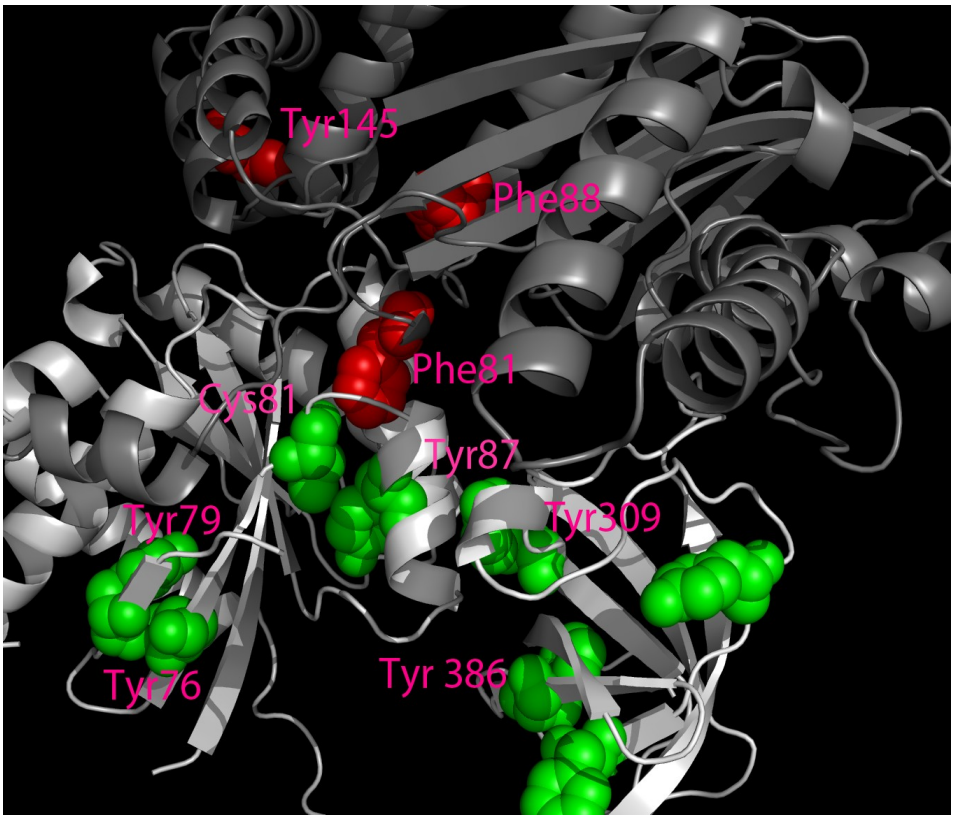


Figure 19

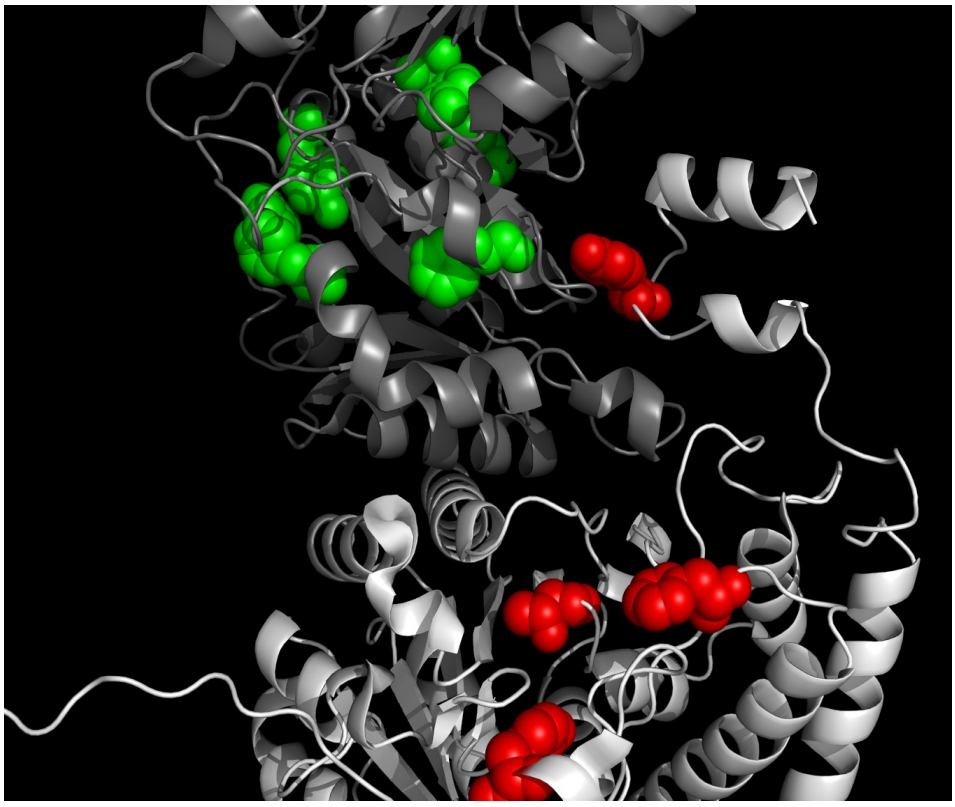


Figure 20

Figure 20, based on the 1qs0 PDB file of 2-oxoisovalerate dehydrogenase shows similar data to **Figure 19. Figure 21** has residues of 2-oxovalerate dehydrogenase that coevolve according to Yeang and Haussler highlighted.

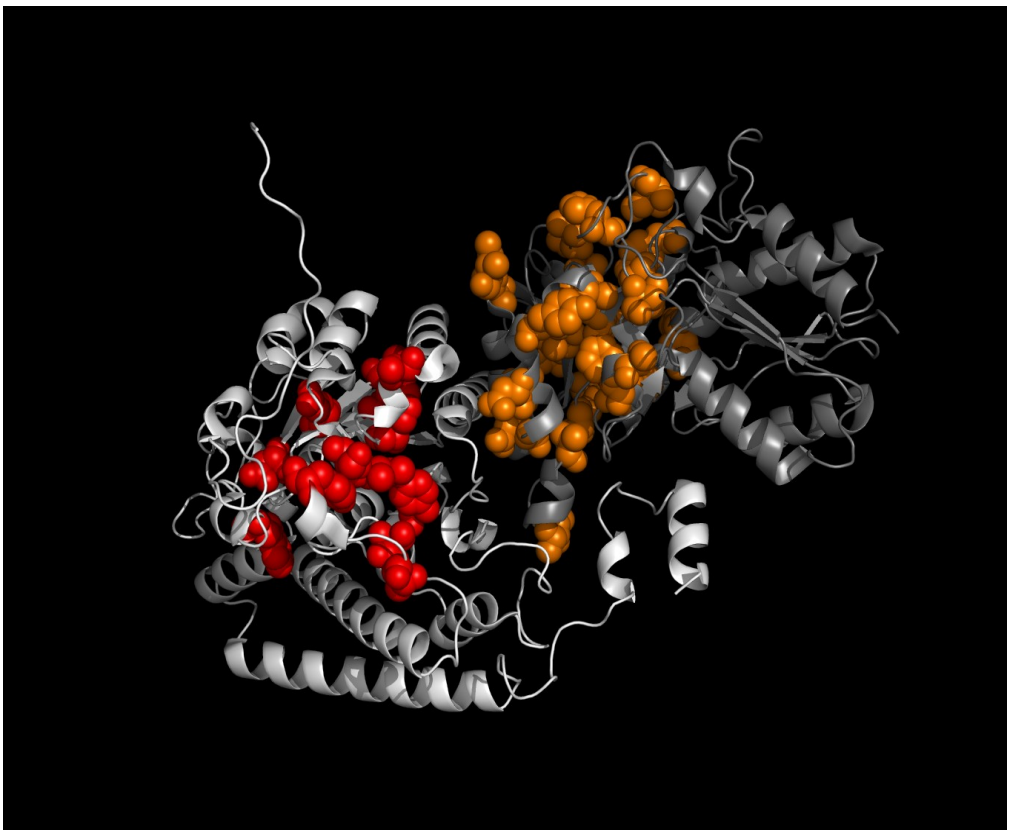


Figure 21

8. Conclusions

- From the results we can conclude that we can predict coevolving residues from the multiple sequence alignment using both tree-aware and tree-ignorant methods, which give a similar coevolution pattern.
- Coevolving residues need not be in direct interaction.
- Coevolving residues have been found in interdomain pairs, many of which belong to the hydrophobic cores.
- Coevolving residues can be detected only after a certain degree of co-conservation. However, high co-conservation does not necessarily imply co-evolution.

• The coevolving residues for an example protein pair (2-oxoisovalerate dehydrogenase subunits A and B) calculated using our

method, correspond to the coevolving residues for that protein according to data provided by Yeang and Haussler (2007).

9. Future perspectives

Considering the importance and applicability of this method, we would like to continue our work and expand this model with use of the maximum likelihood procedure and then also incorporate weighted parsimony methods. We have implemented Sankoff's dynamic-programming algorithms and would like to compare the results obtained using these methods with the model we introduced in our studies.

10. Acknowledgments

The authors would like to thank Jotun Hein for his esteemed guidance. We would also like to thank Dr. Soren Thirup, Dr. Poul Nissen and Dr. Christian Storm Pedersen for their productive feedback. We would also like to thank everyone at BiRC who gave us useful advice related to this project.

11. References

Edgar, R. C. (2004). "MUSCLE: a multiple sequence alignment method with reduced time and space complexity." BMC Bioinformatics 5: 113

Yeang, C. H. and D. Haussler (2007). "Detecting coevolution in and among protein domains." PLoS Comput Biol 3(11): e211

Stothard, P. (2000). "The sequence manipulation suite: JavaScript programs for analyzing and formatting protein and DNA sequences." Biotechniques 28(6): 1102, 1104

Felsenstein, J. (1981). "Evolutionary trees from DNA sequences: a maximum likelihood approach." J Mol Evol 17(6): 368-376

Caporaso, J. G., S. Smit, et al. (2008). "Detecting coevolution without phylogenetic trees? Tree-ignorant metrics of coevolution perform as well as tree-aware metrics." BMC Evol Biol 8: 327

12. Supplements

12.1 Script 1: coevolution score generation

```
########## Script for calculating coevolution score  ###################################

# Auther : Vikas Gupta and Oleg Sitsel
# latest updates on 2010-10-04

#!/usr/bin/env python
import math

#################### Creating a single Amino Acid substitution dictionary ################
#reading file with Single Substitution Matrix
filename='Input substitution file'
f = open(filename,'r')
first_line = True
score = {}
res_dict = {}
for line in f.readlines():
    if line[0] == "#":
                continue
    if first_line:
                line = line.strip()
                letters = line.split()
                for j in range(0,24):
                    res_dict[j] = letters[j]
                first_line = False
    else:
         line = line.strip()
         tokens = line.split()
         row_letter = tokens[0]
         numbers = [int(n) for n in tokens[1:]]
         for col_letter, n in zip(letters, numbers):
             score[(row_letter, col_letter)] = n
##########################################################################################

########## Defining Penalizing factor for single substitution ############################
e = 0.7

########## Defining Independent Double Substitution Rate Matrix ##########################
jrm = {}
r = {}
r_sum = {}
for a1 in range(0,24):
    for a2 in range(0,24):
        r_sum[a1,a2]=0
        count_double = 0
        count_single = 0
        for b1 in range(0,24):
            for b2 in range(0,24):
                if (a2==b2)&(a1!=b1):
                    jrm[(res_dict[a1],res_dict[a2]),(res_dict[b1],res_dict[b2])] = \
                    score[res_dict[a1],res_dict[b1]]
                    count_single += 1
                    #print (a2,b2,a1,b1)
                    #print (jrm[(res_dict[0],res_dict[1]),(res_dict[0],res_dict[1])])
                if (a1==b1)&(a2!=b2):
                    jrm[(res_dict[a1],res_dict[a2]),(res_dict[b1],res_dict[b2])] = \
                    score[res_dict[a2],res_dict[b2]]
                    count_single += 1
                    #print (jrm[(res_dict[0],res_dict[0]),(res_dict[0],res_dict[0])])
                if (a1==b1)&(a2==b2):
                    jrm[(res_dict[a1],res_dict[a2]),(res_dict[b1],res_dict[b2])] = \
```

```
k1 = 0
k2 = 0
flag1 = 1
flag2 = 0
for line in s1.readlines():
    if line[0]=="#":
        continue
    else:
        if (line[0]==">"):
            k1 += 1
            align1[k1] = ''
        elif (line[0]!=">"):
            align1[k1] = align1[k1] + line
            align1[k1] = align1[k1].rstrip()
for line in s2.readlines():
        if(line[0]==">"):
            k2 += 1
            align2[k2] = ''
        elif(line[0]!=">"):
            align2[k2]= align2[k2] + line
            align2[k2] = align2[k2].rstrip()

################## calculating co-conservation ###########################################
match_count = {}
def match(i,j):
    count1 = 0
    count2 = 0
    for i1 in range (1,k1+1):
        if(align1[1][i]==align1[i1][i]):
            count1 +=1
    for j1 in range (1,k2+1):
        if(align2[1][j]==align2[j1][j]):
            count2 += 1
    match_count[i,j] = float(count1/(2*k1) + count2/(2*k2))
    return (match_count[i,j])
##########################################################################################

###################### Relating Alignment residual position with PDB residual position ###
for x in range (1,k1+1):
    l1 = len(align1[x])
    domain1[x,1] = 0
    m1 = 0
    for y in range(0,l1):
        if(align1[x][y]!='-'):
            m1 += 1
            domain1[x,y] = m1
        else:
            domain1[x,y]= 0
for x in range (1,k2+1):
    l2 = len(align2[x])
    domain2[x,1] = 0
    m2 = 0
    for y in range(0,l2):
        if(align2[x][y]!='-'):
            m2 += 1
            domain2[x,y] = m2
        else:
            domain2[x,y]=0

########################### calculating pairwise distances in PDB file ##################
```

```
pdb = open('input.pdb','r')
x_position = {}
y_position = {}
z_position = {}
dis = {}
for i in range(-1,l1):
    for j in range(-1,l2):
        dis[i,j]=0
for line in pdb.readlines():
    line = line.strip()
    tokens = line.split()
    if(len(tokens)>=12):
        if((tokens[0]=='ATOM') & (tokens[2]=='C') & (tokens[4]=='B')):
            x_position[1,int(tokens[5])]=float(tokens[6])
            y_position[1,int(tokens[5])]=float(tokens[7])
            z_position[1,int(tokens[5])]=float(tokens[8])
        if((tokens[0]=='ATOM') & (tokens[2]=='C') & (tokens[4]=='A')):
            x_position[2,int(tokens[5])]=float(tokens[6])
            y_position[2,int(tokens[5])]=float(tokens[7])
            z_position[2,int(tokens[5])]=float(tokens[8])
pdb.close()
pdb1 = open('input','r')
for line in pdb1.readlines():
    line = line.strip()
    tokens1 = line.split()
    if(len(tokens1)>=12):
        if((tokens1[0]=='ATOM') & (tokens1[2]=='C') & (tokens1[4]=='B')):
            pdb2 = open('inout','r')
            for line in pdb2.readlines():
                line = line.strip()
                tokens2 = line.split()
                if(len(tokens2)>=12):
                    if((tokens2[0]=='ATOM') & (tokens2[2]=='C') & (tokens2[4]=='A')):
                            dis[int(tokens1[5]),int(tokens2[5])]=\
                            math.sqrt(((x_position[1,int(tokens1[5])]-x_position[2,int(toke
                            *(x_position[1,int(tokens1[5])]-x_position[2,int(tokens2[5])])))
                            ((y_position[1,int(tokens1[5])]-y_position[2,int(tokens2[5])])*
                            (y_position[1,int(tokens1[5])]-y_position[2,int(tokens2[5])]))+
                            ((z_position[1,int(tokens1[5])]-z_position[2,int(tokens2[5])])*
                            (z_position[1,int(tokens1[5])]-z_position[2,int(tokens2[5])])))))
##############################################################################################

################################## Reading Tree distance file ###############################

phy = open('input_tree.nj','r')
phy_dis = {}
for lines in phy.readlines():
    lines = lines.strip()
    tokens = lines.split()
    if(len(tokens)>5):
        phy_dis[int(tokens[0]),int(tokens[2])] = float(tokens[5])

######################### calculating pair-wise evolution ################################
compare_score_ind = {}
compare_score = {}
compare_score_phy = {}
for i in range (0,l1):
                for j in range (0,l2):
                                compare_score_ind[i,j] = 0
                                compare_score[i,j]=0
```

```
                                compare_score_phy[i,j]=0
                                for k in range (1,2):
                                    for k3 in range (k+1,k2+1):
                                        compare_score_ind[i,j] += jrm[(align1[k][i],align2[
                                        (align1[k3][i],align2[k3][j])]
                                        compare_score[i,j] +=  trm[(align1[k][i],align2[k][
                                        (align1[k3][i],align2[k3][j])]
                                        compare_score_phy[i,j] += (phy_dis[k,k3])*trm[(alig
                                        align2[k][j]),(align1[k3][i],align2[k3][j])]
                                tmp0 = float(compare_score_ind[i,j]/k1)
                                tmp1 = float(compare_score[i,j]/k1)
                                tmp2 = float(compare_score_phy[i,j]/k1)
                                o.write(str(i+1))
                                o.write(str(domain1[1,i]-1))
                                o.write(str(align1[1][i]))
                                o.write(str(j+1))
                                o.write(str(domain2[1,j]-1))
                                o.write(str(align2[1][j]))
                                o.write(str(tmp0))
                                o.write(str(tmp1))
                                o.write(str(tmp2))
                                o.write(str(tmp3))
                                o.write(str(tmp4))
                                o.write(str(dis[(domain1[1,i]-1),(domain2[1,j]-1)]))
                                o.write(str(match(i,j)))
o.close()
############################## Detecting Coevolution :) ###################################
```

12.2 Script 2: removing redundancy

```
############ Script for removing redundancy ####################################


#!/usr/bin/env python

f = open('inout_fasta','r')
o = open('output_fasta','w')
o.close()
line_number = 0
flag = 0
flag2 =0
for line in f.readlines():
    line = line.strip()
    line_number += 1
    if ((line[0]=='>')&(flag==0)):
        n = line_number
        line2 = line
        flag2 = 0
        g = open('reopen.fasta','r')
        for line1 in g.readlines():
            line1 = line1.strip()
            if (line1==line2):
                flag = 1
                flag2 = 1
                n = line_number
        g.close()
        if(flag == 0):
            o = open('check_nr.fasta','a')
            o.write(str(line2))
            o.close()
    if((flag ==0)&(line[0]!='>')&(flag2==0)):
        o = open('check_nr.fasta','a')
        o.write(str(line))
        o.close()
    if((line_number != n)&(line[0]=='>')):
        flag = 0
        flag2 = 1
################################################################################
```