Introducing a New Method for Identifying Protein Structure: Part of the Ongoing Investigation through CASP

Dedeepya Gudipati, Miles Keppler, Sam Carpenter

CSCI 474

**Introduction:**

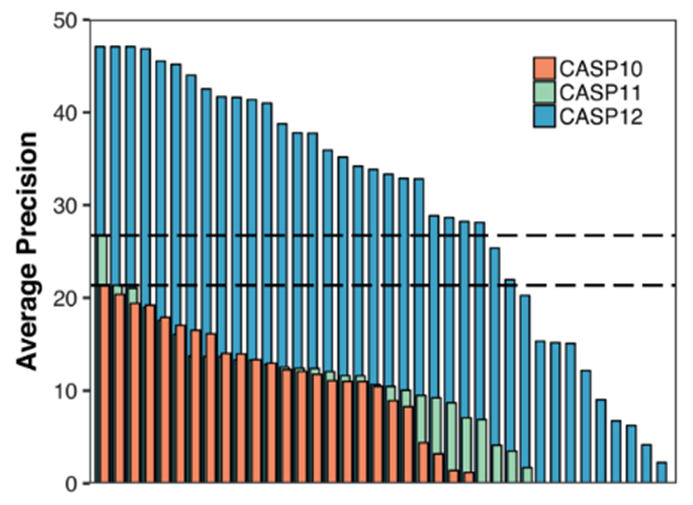
This project provides a new method for identifying a protein coordinate set which is precise to its protein’s actual structure given randomized sets of different coordinate point of each other atom in the amino acids. Our project uses and compares homologs to our query sequences using pyMol producing a score for each coordinate set. The use of homologs to understand the tertiary and quaternary structures of the query sequence has also been used in previous methods to estimate a precious structure of the unknown peptide. We focused on local sequences whilst in search of homologs for each of the peptide sequences to control more variable in our method. This project provides a selection for five provided peptide sequences.

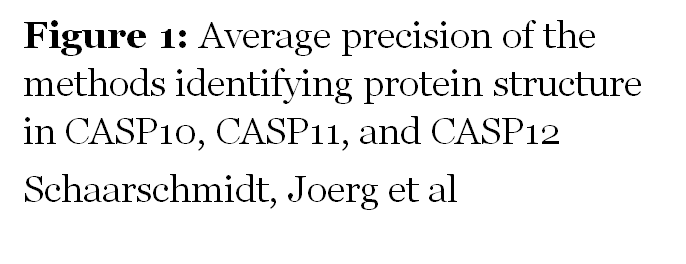
**Summary:**

**Motivation:** Creating a method to model and identify protein structures can help structural biologists and computational biologists decrease time in lab, where they identify protein structure through the crystallography and NMR.

**Goal:** Create a scoring system to predict the correct protein structure of an amino acid sequence given different sets of atom coordinates of each of the amino acids in the sequences.

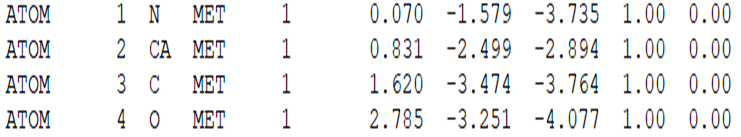
**Approach:** Measure the distance between specific amino acids in their 3D models as a way to evaluate each atom coordinate sets to identify the precise protein structure of the amino acid sequence.

**Background:**

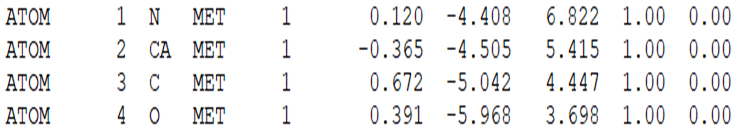
Critical Assessment of protein structure (CASP) is a series of experiments that is conducted by centers around the world with a goal to advance the methods of identifying protein sequence from a peptide sequence. There have been twelve previous CASP experiments in the past and the thirteen experiment, CASP13, is currently in progress. Figure 1 illustrates the progress and advances in precision of methods in previous experiments with CASP12 having the highest precision. This investigation we propose a unique way of the amino acid evaluating and identifying the correct structure of the peptide sequence.

**Data:**

Five folders, each with a different peptide sequence. Each of these folders contain 20 different atom coordinate sets with the same peptide sequence.



|  |
| --- |
| **Figure 2:** First 4 atom coordinates of the first amino acid in set server01\_TS1 in folder T0949 |



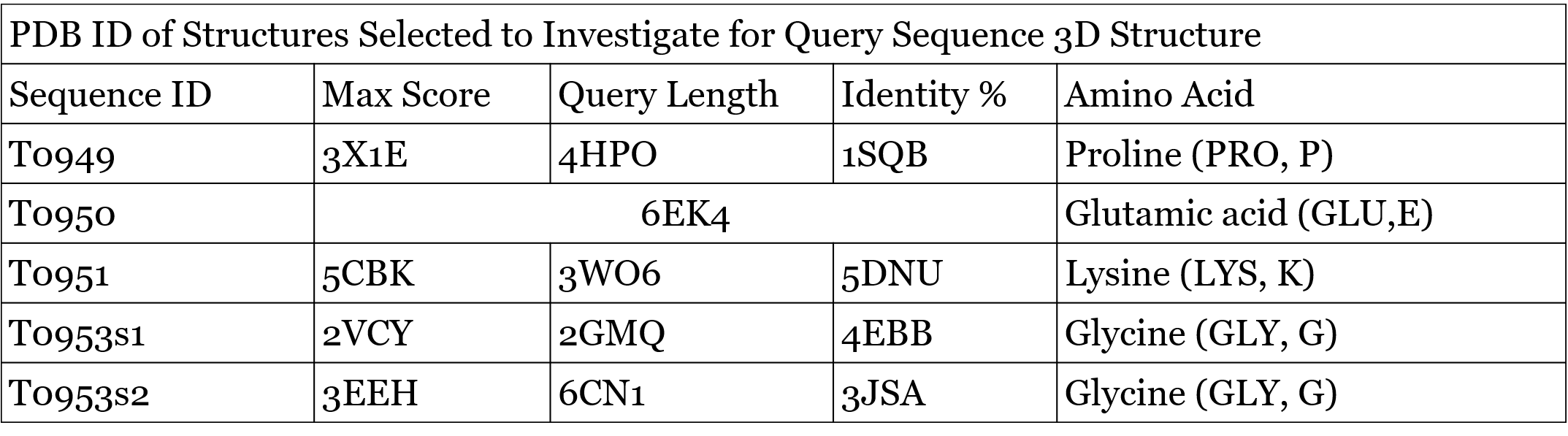
|  |
| --- |
| **Figure 3:** First 4 atom coordinates of the first amino acid in set server02\_TS1 in folder T0949 |

**Methods:**

1. **Basic Local Alignment Search Tool (BLAST)** – script was developed to attain e one-letter code peptide sequence in the T0949 folder files (all of the files in the folder have the same peptide sequence. This sequence was run through an online-tool, BLASTP to find local similarity between known sequences in a database and the query sequence. Parameters used: Protein Data Bank(pdb) database and word size 3.
2. **Choosing homolog pdb files** – the blastp results list multiple proteins and corresponding pdb 4 character codes. We picked three proteins that scores the highest in three different areas: Max Score, Query Length, and the Identity %. These results are listed in **Figure 4**.
3. **Choosing an amino acid –** to focus on a specific amino acid, literature was used to pick a specific amino acid to evaluate. This amino acid must be present in all three sequences and must be a match multiple times to the query sequence. These selections are listed in **Figure 4.**
4. **Measuring the distances between amino acids –** a script was developed to view the chosen the peptide sequences in pyMol and measure the 3-D distance between the same amino acids in different positions. For example, for T0494, the distances between every Proline amino acid were measured in angstroms. This is done to all subject and query sequences separately.
5. **Analyze the distances -**  a script was used to evaluate the similarity of distances, using root mean squared deviation (rMSD) as a metric, and using the sum over all PDB structures found from the BLAST search. We then selected the lowest score, as it showed the least overall deviation. rMSD was used, as opposed to MSD or sum of squared deviation, in order to not ignore a strong result simply because it matched poorly with one of our three metric files.

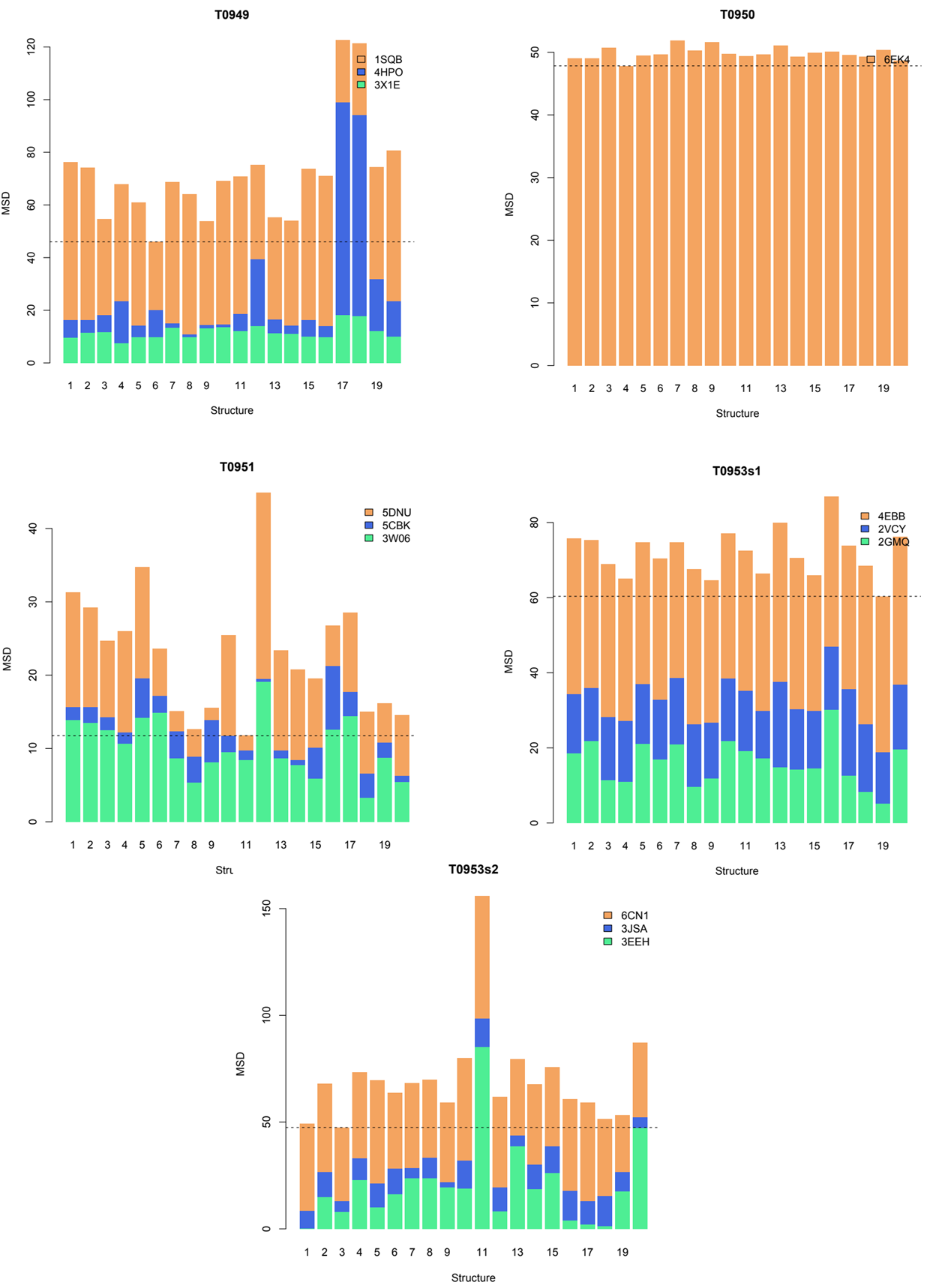
**Results**

**Figure 2:** Selected pdb files and amino acids from the given criteria.

****

These results compare all three pdb files with the 20 sets of coordinates for each of the peptide sequences. The lower the score the more similar the distances are to the pdb files. The structures of the peptide sequences are close to the pdb files that compare with the lowest scores. The line represents the winning coordinate set for the peptide sequence. These results are shown in Figure 4.

We’ve used only one FASTA file for the sequence ID T0950 because the entire query sequence was matched with the FASTA file of 6EK4, so we were expecting that we would find one coordinate file with a very low MSD score.



|  |
| --- |
| **Figure 4:** Thet The MSD scores for all the coordinate sets of all the peptide sequences. Results for the T0950 peptide sequence only contains on MSD scores compared to one pdb file because its FASTA file is an exact match to the query peptide sequence. |

**Discussion:**

Our methods led us to select T0949-9, T0950-4, T0951-11, T0953s1-19, T0953s2-1 as our preferred structures. Interestingly, while T0950 was a perfect match for its BLAST results, that result had a relatively poor rMSD score with all 20 structures. Additionally, none of the 20 was significantly preferred to the others, but because there was still a lowest score, we predict that with little confidence.

It is important to recognize that the selected coordinate sets have MSD scores that are in general the lowest when compared to all three pdb files. For example, the selected coordinate set for the T0953s1 has the lowest MSD scores from 4EBB, 2VCY, and 2GMQ pdb files. This increase our confidence just a little bit in our selections.

None of our results showed vast preference, so it may be a good idea to check more atoms in each structure. It also may be a good idea to use more metric homologous structure files for each sequence. In the future it would be a good idea to examine our precision with past runs of CASP to get an empirical accuracy rate, or simulate data using known structures. Our method could also be improved through strong tools to identify the most important atom positions in a given sequence.

Another interesting outcome from our investigation are the results for the T0950. We only used one FASTA file to get one set of MSD scores. These scores are unexpectedly high for a protein that we have a complete FASTA file for. From this data we assume that none of the provided data sets have close enough structurally close to the selected pdb file. One of the reasons possible for this phenomenon is that there are too many matches that the The results are also very close together, leaving us with little confidence in our results, but also supporting the use of multiple pdb files to produce scores.

**Conclusion:**

Our project goal was to provide a method/algorithm to identify the structure of the protein. This method successfully predicted the quaternary structure of the given peptide sequence. Since all the selected structures show the same patterns in the MSD scores, we are fairly confident that the results are correct. One drawback of the method is that there is manual effort needed to identify important amino acids to use in the algorithm. In contrast, there are several coordinate files that have similar MSD scores, this lowers our confidence level in our selections. Next steps for this project would be evaluating our results by comparing them to the actual 3-D structure of the selected proteins once they are available. We can also edit our algorithm to view the 3D distances between different matched amino acids.

**References:**

Altschul, Stephen F., et al. “Basic Local Alignment Search Tool.” *Journal of Molecular Biology*, vol. 215, no. 3, 1990, pp. 403–410., doi:10.1016/s0022-2836(05)80360-2.

Hille, Russ. “Structure and Function of Xanthine Oxidoreductase.” *European Journal of Inorganic Chemistry*, vol. 2006, no. 10, 2006, pp. 1913–1926., doi:10.1002/ejic.200600087.

Hu, Qingliang, et al. “DWARF14, A Receptor Covalently Linked with the Active Form of Strigolactones, Undergoes Strigolactone-Dependent Degradation in Rice.” *Advances in Pediatrics.*, U.S. National Library of Medicine, 2017, [www.ncbi.nlm.nih.gov/pmc/articles/PMC5684176/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5684176/).

Peter W. Rose, Andreas Prlić, Ali Altunkaya, Chunxiao Bi, Anthony R. Bradley, Cole H. Christie, Luigi Di Costanzo, Jose M. Duarte, Shuchismita Dutta, Zukang Feng, Rachel Kramer Green, David S. Goodsell, Brian Hudson, Tara Kalro, Robert Lowe, Ezra Peisach, Christopher Randle, Alexander S. Rose, Chenghua Shao, Yi-Ping Tao, Yana Valasatava, Maria Voigt, John D. Westbrook, Jesse Woo, Huangwang Yang, Jasmine Y. Young, Christine Zardecki, Helen M. Berman, Stephen K. Burley; The RCSB protein data bank: integrative view of protein, gene and 3D structural information, *Nucleic Acids Research*, Volume 45, Issue D1, 4 January 2017, Pages D271–D281,

Schaarschmidt, Joerg et al. “Assessment of Contact Predictions in CASP12: Co‐evolution and Deep Learning Coming of Age.” *Proteins* 86.Suppl Suppl 1 (2018): 51–66. *PMC*. Web. 5 June 2018.

Stewart, Richard C. “Protein Histidine Kinases: Assembly of Active Sites and Their Regulation in Signaling Pathways.” *Advances in Pediatrics.*, U.S. National Library of Medicine, Apr. 2010, [www.ncbi.nlm.nih.gov/pmc/articles/PMC2847664/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847664/).

Wagner, L., et al. “Unravelling the Immunological Roles of Dipeptidyl Peptidase 4 (DPP4) Activity and/or Structure Homologue (DASH) Proteins.” *Advances in Pediatrics.*, U.S. National Library of Medicine, 6 Mar. 2016, [www.ncbi.nlm.nih.gov/pmc/articles/PMC4872383/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4872383/).

Yuan, Shuguang, et al. “PyMOL and Inkscape Bridge the Data and the Data Visualization.” *Structure*, vol. 24, no. 12, 2016, pp. 2041–2042., doi:10.1016/j.str.2016.11.012.

Appendix

#### FastaFind.py

file = open('server01\_TS1','r')  
FASTA = ''  
i = 0  
for line in file:  
 if line.startswith('ATOM'):  
 x = int(line[23:26])  
 print(x)  
 if i < x:  
 i = i + 1  
 a = line[17:20]  
 print(a)  
 if a == 'ALA':  
 FASTA = FASTA + 'A'  
 elif a == 'ARG':  
 FASTA = FASTA + 'R'  
 elif a == 'ASN':  
 FASTA = FASTA + 'N'  
 elif a == 'ASP':  
 FASTA = FASTA + 'D'  
 elif a == 'ASX':  
 FASTA = FASTA + 'B'  
 elif a == 'CYS':  
 FASTA = FASTA + 'C'  
 elif a == 'GLU':  
 FASTA = FASTA + 'E'  
 elif a == 'GLN':  
 FASTA = FASTA + 'Q'  
 elif a == 'GLX':  
 FASTA = FASTA + 'Z'  
 elif a == 'GLY':  
 FASTA = FASTA + 'G'  
 elif a == 'HIS':  
 FASTA = FASTA + 'H'  
 elif a == 'ILE':  
 FASTA = FASTA + 'I'  
 elif a == 'LEU':  
 FASTA = FASTA + 'L'  
 elif a == 'LYS':  
 FASTA = FASTA + 'K'  
 elif a == 'MET':  
 FASTA = FASTA + 'M'  
 elif a == 'PHE':  
 FASTA = FASTA + 'F'  
 elif a == 'PRO':  
 FASTA = FASTA + 'P'  
 elif a == 'SER':  
 FASTA = FASTA + 'S'  
 elif a == 'THR':  
 FASTA = FASTA + 'T'  
 elif a == 'TRP':  
 FASTA = FASTA + 'W'  
 elif a == 'TYR':  
 FASTA = FASTA + 'Y'  
 elif a == 'VAL':  
 FASTA = FASTA + 'V'  
 else:  
 FASTA = FASTA + 'error'  
  
file.close()  
file2 = open('test.txt','w')  
file2.write(FASTA)  
file2.close()

#### pymolAlign.py

#!/usr/bin/env python  
print("Enter protein: ")  
import \_\_main\_\_  
\_\_main\_\_.pymol\_argv = ['pymol','-qc'] # Pymol: quiet and no GUI  
from time import sleep  
import sys  
import pymol  
pymol.finish\_launching()  
  
protein = raw\_input()  
  
file1 = open('objects.txt', 'r')  
  
objects = []  
for line in file1:  
 objects.append(line.rstrip())  
file1.close()  
  
file2 = open('sequence2.txt', 'r')  
  
query = file2.readline()  
subject = file2.readline()  
x = int(query[:5])  
y = int(subject[:5])  
  
queryList = []  
subjectList = []  
  
j = 5  
Qkey = 0  
Skey = 0  
k = 0  
while(query[j] != ' '):  
 if(query[j] == 'P' and subject[j] == 'P'):  
 queryList.append(int(x) + int(Qkey))  
 subjectList.append(int(y) + int(Skey))  
 k = k + 1  
 Qkey = Qkey + 1  
 Skey = Skey + 1  
 elif(query[j] == '-'):  
 Skey = Skey + 1  
 elif(subject[j] == '-'):  
 Qkey = Qkey + 1  
 else:  
 Qkey = Qkey + 1  
 Skey = Skey + 1  
 j = j + 1  
  
file2.close()  
  
file3 = open('newfile','w')  
  
for index in range(0,20):  
 pymol.cmd.reinitialize()  
  
 pymol.cmd.load('%s.pdb' %(objects[index]))  
  
 file3.write('\n')  
 file3.write(objects[index] + ":\n")  
 file3.write('\n')  
  
 for key in range(0,(k-1)):  
 distance = pymol.cmd.distance("(/" + objects[index] + "///%d/CA)"%(queryList[key]),"(/" + objects[index] + "///%d/CA)"%(queryList[key+1]))  
 file3.write("Distance between %d-%d: %s\n"%(queryList[key], queryList[key+1], distance))  
  
 sleep(0.5) # (in seconds)  
  
pymol.cmd.reinitialize()  
pymol.cmd.fetch('%s' %(protein))  
  
file3.write('\n')  
file3.write(protein + ":\n")  
file3.write('\n')  
  
for key in range(0,(k-1)):  
 distance = pymol.cmd.distance("(/" + protein + "///%d/CA)"%(subjectList[key]) ,"(/" + protein + "///%d/CA)"%(subjectList[key+1]))  
 file3.write("Distance between %d-%d: %s\n"%(subjectList[key], subjectList[key+1], distance))  
  
  
 sleep(0.5) # (in seconds)  
  
file3.close()  
  
pymol.cmd.quit()

#### aminoDist.py

def compareDists(subject:list, \*queries:list) -> float :  
 count = 0  
 for query in queries :  
 count += compareDist(query,subject)  
 return count/len(queries)  
def compareDist(subject:list, query:list) -> float :  
 n = len(subject)  
 if n != len(query) :  
 return -float('inf')  
 return sum([(subject[i]-query[i])\*\*2 for i in range(n)])/n  
def getFromFile(file) :  
 n = 0  
 dists = []  
 this = []  
 for line in file :  
 if line[0:16] == "Distance between" :  
 this.append(float(line.rsplit(':',1)[1]))  
 n += 1  
 elif n > 0 :  
 dists.append(this)  
 this = []  
 n = 0  
 return dists  
files = [  
 open("D:/Bork/Desktop/T0949 3X1E.txt"),  
 open("D:/Bork/Desktop/T0949 4HPO.txt"),  
 open("D:/Bork/Desktop/T0949 1SQB.txt"),  
 open("D:/Bork/Desktop/T0950 6EK4.txt"),  
 open("D:/Bork/Desktop/T0951 3W06.txt"),  
 open("D:/Bork/Desktop/T0951 5CBK.txt"),  
 open("D:/Bork/Desktop/T0951 5DNU.txt"),  
 open("D:/Bork/Desktop/T0953s1 2GMQ.txt"),  
 open("D:/Bork/Desktop/T0953s1 2VCY.txt"),  
 open("D:/Bork/Desktop/T0953s1 4EBB.txt"),  
 open("D:/Bork/Desktop/T0953s2 3EEH.txt"),  
 open("D:/Bork/Desktop/T0953s2 3JSA.txt"),  
 open("D:/Bork/Desktop/T0953s2 6CN1.txt")]  
scores = []  
for file in files :  
 dists = getFromFile(file)  
 scores.append([compareDist(dists[i],dists[20]) for i in range(20)])  
for i in range(len(scores)) :  
 for j in range(len(scores[i])) :  
 scores[i][j] = scores[i][j]\*\*(1/2)  
scoreTotal49 = [scores[0][i]/3 + scores[1][i]/3 +  
 scores[2][i]/3 for i in range(20)]  
scoreTotal50 = [scores[3][i] for i in range(20)]  
scoreTotal51 = [scores[4][i]/3 + scores[5][i]/3 +  
 scores[6][i]/3 for i in range(20)]  
scoreTotal53s1 = [scores[7][i]/3 + scores[8][i]/3 +  
 scores[9][i]/3 for i in range(20)]  
scoreTotal53s2 = [scores[10][i]/3 + scores[11][i]/3 +  
 scores[12][i]/3 for i in range(20)]  
scoreTotals = (scoreTotal49,scoreTotal50,scoreTotal51,  
 scoreTotal53s1,scoreTotal53s2)  
print('49',\*scoreTotal49,'--', sep='\n')  
print('50',\*scoreTotal50,'--', sep='\n')  
print('51',\*scoreTotal51,'--', sep='\n')  
print('53s1',\*scoreTotal53s1,'--', sep='\n')  
print('53s2',\*scoreTotal53s2,'--', sep='\n')  
best = [scores.index(min(scores)) for scores in scoreTotals]  
print(best)  
for file in files :  
 file.close()  
output = open('results.dat','w')  
head = '\t'.join([str(i+1) for i in range(20)])  
output.write(head+'\n')  
for score in scores :  
 output.write('\t'.join([str(num) for num in score])+'\n')  
output.close()

#### CASPplotter.R

data = read.table('results.dat',header=T)

mat = sqrt(as.matrix(data))

colnames(mat) = 1:20

T0949 = mat[1:3,]

T0950 = mat[4,]

T0951 = mat[c(6,5,7),]

T0953s1 = mat[c(9,8,10),]

T0953s2 = mat[c(11,13,12),]

pdf('barplots.pdf')

barplot(T0949, xlab = 'Structure', ylab = 'MSD', main = 'T0949',border = NA,

col = c('seagreen2', 'royalblue', 'sandybrown'),

legend = c('3X1E','4HPO','1SQB'), args.legend = c(bg = NA,bty = 'n'))

abline(min(colSums(T0949)),0,lty = "dashed")

barplot(T0950, xlab = 'Structure', ylab = 'MSD', main = 'T0950',border = NA,

col = c('sandybrown'),

legend = c('6EK4'), args.legend = c(bg = NA,bty = 'n'))

abline(min(T0950),0,lty = "dashed")

barplot(T0951, xlab = 'Structure', ylab = 'MSD', main = 'T0951',border = NA,

col = c('seagreen2', 'royalblue', 'sandybrown'),

legend = c('5CBK','3W06','5DNU'), args.legend = c(bg = NA,bty = 'n'))

abline(min(colSums(T0951)),0,lty = "dashed")

barplot(T0953s1, xlab = 'Structure', ylab = 'MSD', main = 'T0953s1',border = NA,

col = c('seagreen2', 'royalblue', 'sandybrown'),

legend = c('2VCY','2GMQ','4EBB'), args.legend = c(bg = NA,bty = 'n'))

abline(min(colSums(T0953s1)),0,lty = "dashed")

barplot(T0953s2, xlab = 'Structure', ylab = 'MSD', main = 'T0953s2',border = NA,

col = c('seagreen2', 'royalblue', 'sandybrown'),

legend = c('3EEH','6CN1','3JSA'), args.legend = c(bg = NA,bty = 'n'))

abline(min(colSums(T0953s2)),0,lty = "dashed")

dev.off()