# August 24, 2012

Used the scripts in "automate cluster download" to download all the HHOMP clusters. Put them in the folder "automated sequence alignment/clusters". deleted August 28 2012

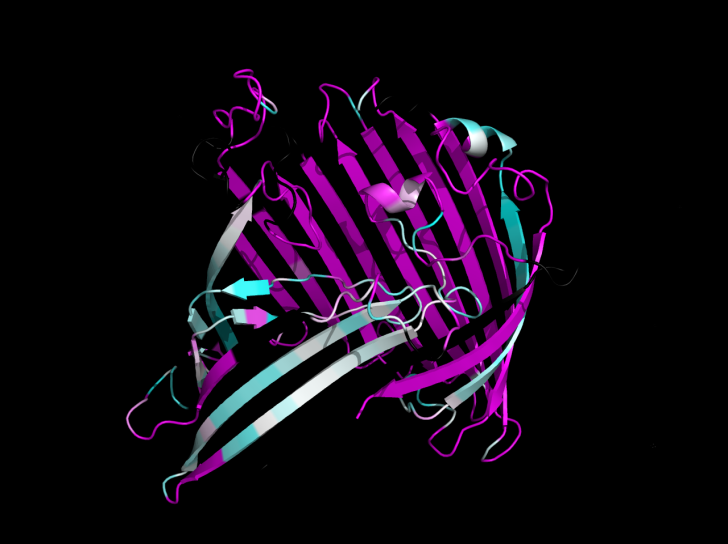
Copied the folder "aligned structures" from the folder "june 2012 lab meeting" renamed "zdiff test structures" August 28 2012

Copied the fodler "TMout" from the folder "bbtm derivation".

Wrote code that does not use ClustalW's "reset all gaps before alignment option". I am not sure if this is necessary but eventually I should ue this option to be safe.

# August 27, 2012

I don't know if we should actually be using BBTM. Maybe the reason the BBTM40 alignment was so off was actually because most of the protein has the substitution rate of soluble protein. This didn't used to concern me, because I don't care about aligning th eloops right. But if Clustal is sacrificing alignment score in the transmembrane region to achieve it in the loops, then we'd see those terrible misalignments. The loops might be aligned decently, though, though they're being aligned with the wrong matrix. Yeah. Oh my god they are. Here's the zdiff for bbtmout colored 0 is cyan 3 is magenta from "june 2012 lab meeting/pictures":



I don't know if I'd say it's aligning that well at the expense of the other parts though... that doesn't even really make sense. Aligning one part correctly *helps* align anothe rpart correctly.

Could it be, though, that better alignment of the less conserved loop regions guides the gonnet alignment?

This is made less likely by the fact that only like 43% of the protein is loop, determined using HHOMP's ProfTMB prediction for one of the two clusters most similar to the proteins that zdiff was calcualted for, cluster 18.1.1:

>>> x = AlignIO.read(r'C:\cygwin\home\alex\beta-barrel-oligomerization\automated\_sequence\_alignment\clusters\OMP.18.1.1.clu', 'clustal')

>>> bbpred = x[2]

>>> bbpred

SeqRecord(seq=Seq('------------------------------------------------------...I--', SingleLetterAlphabet()), id='bb\_pred', name='<unknown name>', description='bb\_pred', dbxrefs=[])

>>> tm = sum(i **in** ('U', 'D') **for** i **in** bbpred)

>>> l = sum(i **in** ('I', 'O') **for** i **in** bbpred)

>>> l/(tm+l)

0.43842364532019706

OmpA is a lot more loopy, though, it's 70% loop:

>>> x = AlignIO.read(r'C:\cygwin\home\alex\beta-barrel-oligomerization\automated\_sequence\_alignment\clusters\OMP.8.1.1.clu', 'clustal')

>>> x[2]

SeqRecord(seq=Seq('-----------------------------IIIIIII--IIII-I----------...---', SingleLetterAlphabet()), id='bb\_pred', name='<unknown name>', description='bb\_pred', dbxrefs=[])

>>> bbpred = x[2]

>>> tm = sum(i **in** ('U', 'D') **for** i **in** bbpred)

>>> l = sum(i **in** ('I', 'O') **for** i **in** bbpred)

>>> l/(tm+l)

0.7

To whatever extent using a more refined substitution matrix can help, to that same extent it hurts to have two halves of the protein with different subsitution rates! Making these BBTM matrices and automating ClustalW seems not worth it, though further testing would make it more clear. Can an HMM method capture the varying ocnditions better than a substitution matrix?

# August 28, 2012

Changed name of "aligned structures" to "zdiff test structures".

Today Daniel sent me a draft of his thesis. It contains the following table as Table 4:

Table - Unique clusters associated with Proteins in our Dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| number | PDB\_ID | Cluster Name | subclusters | # of homol. seqs in MSA |
| 1 | 1A0S | cluster73 | 18.1.1  18.1.2 | 51 |
| 2 | 1AF6 |  | 18.1.1 | n/a |
| 3 | 1E54 | cluster28 | 16.2.1-16.2.5  16.2.8 | 319 |
| 4 | 1FEP | cluster8 | 22.4.2  22.4.4 | 81 |
| 5 | 1I78 |  | 10.1.1 | 15 |
| 6 | 1K24 |  | 10.2.1 | 2 |
| 7 | 1KMO |  | 22.2.4 | 31 |
| 8 | 1P4T | cluster144 | 8.1.5  8.6.3  8.6.2 | 76 |
| 9 | 1QD6 |  | 12.6.1 | 85 |
| 10 | 1QFG |  | 22.1.4 | 224 |
| 11 | 1QJ8 |  | 8.3.1 | 31 |
| 12 | 1QJP | cluster75 | 8.1.1  nn.31.1 | 78 |
| 13 | 1T16 |  | 14.1.1 | 162 |
| 14 | 1THQ |  | 8.5.1 | 14 |
| 15 | 1TLY | cluster108 | 12.5.1  12.5.2 | 36 |
| 16 | 1UYN |  | 12.1.6 | 57 |
| 17 | 2ERV |  | 8.4.1 | 44 |
| 18 | 2F1C |  | 14.2.1 | n/a |
| 19 | 2F1V |  | 8.2.1 | 141 |
| 20 | 2GUF |  | 22.4.5 | 61 |
| 21 | 2J1N | cluster99 | 16.1.1  16.1.2 | 77 |
| 22 | 2O4V |  | 16.4.2 | 75 |
| 23 | 2POR | cluster131 | 16.2.1  16.2.3 | n/a |
| 24 | 2QDZ | cluster53 | nn.5.1  nn.5.2  nn.5.4  cluster43 | 152 |
| 25 | 2VQI |  | nn.2.2 | 206 |
| 26 | 2WJR |  | nn.36.1 | 35 |
| 27 | 3BS0 | cluster71 | 14.1.5  14.1.7  14.1.1  cluster62 | 194 |
| 28 | 3CSL |  | 22.4.6 | 67 |
| 29 | 3DWO |  | 14.1.1 | n/a |
| 30 | 3DZM | cluster165 | 8.1.1 | n/a |
| 31 | 3EFM | cluster18 | cluster6  22.1.7  22.1.4  22.1.5  22.1.3  22.1.6 | 531 |
| 32 | 3EMN |  | nn.54.1 | 25 |
| 33 | 3FHH |  | 22.4.6 | n/a |
| 34 | 3JTY |  | nn.9.1 | 172 |
| 35 | 3PRN | cluster131 | 16.2.1  16.2.3 | n/a |

I copied this table to an Excel document, and removed the red rows: 35, 33, 30, 29, 23, 18, and 2. I saved it as "structure dataset and clusterguide.csv".

I deleted the "clusters" folder - too many clusters in it. Made a new folder "~~ezbeta~~ clusters", which I will now opulate with just the clusters required.

Used this code tomake a file "~~ezbeta~~ clusters/clusters.csv" containing all the above clusters:

**import** csv

pdbid\_clusterid = list()

with open('structure dataset and clusterguide.csv', 'rb') as f:

first = True

**for** row **in** csv.reader(f):

# Skip the first line

**if** first:

first = False

**continue**

# Get the pdbid

**if** row[1] != '':

pdbid = row[1]

**else**:

**continue**

# Get the clustername from the Cluster Name column

**if** row[2] != '':

cluster = row[2]

# But maybe this structure wasn't located to a supercluster

# In that case, get it from the subcluster column.

**else**:

cluster = row[3]

pdbid\_clusterid.append((pdbid, cluster))

with open('ezbeta clusters/clusters.csv', 'wb') as o:

csv.writer(o).writerows([[cluster] \

**for** pdbid, cluster **in** pdbid\_clusterid])

The resulting file has 28 lines - exactly the right number, because the original figure is labeled 1 to 35, and I removed 7 lines.

I then ran this code to download, from HHOMP, every cluster named in "~~ezbeta~~ clusters/clusters.csv"

**import** urllib

**import** re

**import** csv

**class** UnrecognizedClusterName(Exception):

**pass**

url\_base = 'http://toolkit.tuebingen.mpg.de/hhomp\_ali/'

files = list()

**try**:

f = open('clusters.csv', 'r')

**for** line **in** csv.reader(f):

**if** len(line) == 0 **or** line[0] == '':

**continue**

**elif** re.match('cluster\d+', line[0]) **is** **not** None:

files.append(line[0] + '.clu')

**elif** re.match('OMP\..+?\.\d+\.\d+', line[0]) **is** **not** None\

**or** re.match('NodT\.\d', line[0]) **is** **not** None\

**or** re.match('TolC\.\d', line[0]) **is** **not** None:

files.append(line[0] + '.clu')

**elif** re.match('.+?\.\d+\.\d+', line[0]) **is** **not** None:

files.append('OMP.' + line[0] + '.clu')

**else**:

**raise** UnrecognizedClusterName(line[0])

**finally**:

f.close()

**for** filename **in** files:

urllib.urlretrieve(url\_base + filename, filename=filename)

**print**('done')

The code is saved as "~~ezbeta~~ clusters/download.py". However, I often modify files containing python code; the above code, printed in this log, should be referred to for any questions about my methods.

I copied all the .pdb files from "pymol/structures" into a folder called "ezbeta aligned structures"

# August 29, 2012

Created Gonnet matrix alignments from the HHOMP clusters and structures indicated in figure 4 of Daniel's thesis. The alignments were created with ClustalW, accessed through BioPython. The structures are those in "ezbeta aligned structures", which Daniel sent me a long time ago. The clusters are in the "clusters" folder, and were downloaded from the HHOMP website as described above. The following code was used to direct ClustalW:

**from** Bio.Align.Applications **import** ClustalwCommandline

**import** Bio.AlignIO

**import** Bio.SeqIO

**import** Bio.PDB

**from** Bio.Alphabet.IUPAC **import** IUPACProtein

**from** Bio.Seq **import** Seq

**from** Bio.SeqRecord **import** SeqRecord

**import** re

**import** warnings

**import** os

**import** copy

**import** csv

**from** sundries **import** one\_letter

**def** call\_clustalw(seq\_file, mat\_file, output\_file):

'''The function that actually produces the sequence alignment.'''

cline = ClustalwCommandline('clustalw', infile=seq\_file,

matrix=mat\_file, outfile = output\_file)

**return** cline()

**def** cluster\_retriever(cluster\_name):

'''Returns a multiple sequence alignment corresponding to the given

    cluster name.'''

**if** cluster\_name[:3].upper() == 'OMP' **or** 'cluster' **in** cluster\_name:

path = 'clusters/{}.clu'.format(cluster\_name)

**else**:

path = 'clusters/OMP.{}.clu'.format(cluster\_name)

**return** Bio.AlignIO.read(path, 'clustal')

**def** sequence\_retriever(pdb\_paths):

# Match a pathname, with a group that includes the last slash or

# backslash and everything after it

file\_from\_path = re.compile(r'^.\*([/\\].\*?)$')

output = list()

**for** path **in** pdb\_paths:

# Come up with a name to identify the sequence by

match = re.match(file\_from\_path, path)

**if** match **is** None:

name = path

**else**:

name = match.group(0)

# I think ClustalX ignores everything after a space which is

# annoying, so replace the spaces with underscores

name = name.replace(" ", "\_")

# Load the structure. See what happens if you try to load one

# of Daniel's aligned structures without hiding warnings. It's not

# even about anything significant, for this kind of thing,

# just missing b factors and occupancies

with warnings.catch\_warnings():

warnings.simplefilter('ignore')

stru = Bio.PDB.PDBParser().get\_structure(name, path)

# Get the sequence from the structure

seq\_as\_str = str('')

**for** r **in** stru.get\_residues():

# one\_letter is a case-insensitive dictionary mapping

# 3-letter residue names to 1-letter residue names

# Biopython PDB structures use 3-letter residue names

standard\_20 = [n.upper() **for** n **in** one\_letter.keys()]

resn = r.get\_resname().upper()

# Only take stuff that's actually a residue, and not like

# some bound lipid or something

**if** resn **in** standard\_20:

seq\_as\_str += one\_letter[resn]

# In the PDB entry for 1FEP, the FASTA file just has "M"

# where there's selenomethionine in the structure,

# which makes sense because Vik

# says that putting in the selenium is something that's done

# to make structure determination easier, a bacterium's FepA

# is not really going to be full of selenium.

# So, treat selenomethionine as methionine:

**elif** resn == 'MSE':

seq\_as\_str += 'M'

seq\_as\_Seq = Seq(seq\_as\_str, alphabet = IUPACProtein())

# Append a SeqRecord to the output

output.append(SeqRecord(seq\_as\_Seq, name=name, id=name,

description='sequence taken from a structure file'))

**return** output

**def** combine(output\_filename, msa, seqs):

output\_seqs = list(seqs)

**for** record **in** msa:

# An HHOMP alignment has lots of useful stuff besides sequences,

# which I want to ignore here

# I don't know what "gi" means but all the real sequences seem to

# have it at the beginning of their id, so,

# Filter for only those sequences whose id's begin with "gi"

**if** record.id[:2] == 'gi':

# Get rid of gaps

gapless\_as\_str = ''.join(filter(lambda x: x!= '-', record))

gapless\_seq = Seq(gapless\_as\_str)

gapless\_record = copy.deepcopy(record)

gapless\_record.seq = gapless\_seq

output\_seqs.append(gapless\_record)

Bio.SeqIO.write(output\_seqs, output\_filename, 'fasta')

**def** bbtm\_filename(t):

# Make string of t padded with 0's:

t\_string = (3-len(str(t)))\*'0' + str(t)

# Create filename of one of the matrices David Jimenez-Morales

# sent me

filename = os.getcwd() + r'\TMout\bbtmout\bbTMout'+t\_string

**return** filename

**def** write\_series(id\_t\_triplets, output\_filename):

# Information on the series file format can be found here:

# http://bips.u-strasbg.fr/en/Documentation/ClustalX/

# In case the link breaks, the Wayback Machine has an archived page

# from July 27 2011

with open(output\_filename, 'w') as o:

o.write('CLUSTAL\_SERIES\n')

o.write('\n')

**for** id\_bottom, id\_top, t **in** id\_t\_triplets:

o.write(' '.join(["MATRIX", str(id\_bottom), str(id\_top),

bbtm\_filename(t)]))

o.write('\n')

# The API

**class** UnrecognizedMatrix(Exception):

**pass**

**def** align(output\_name, cluster, matrix, \*pdbpaths):

# Match a pathname, with a group that includes everything before the

# last slash or backslash

dir\_from\_path = re.compile(r'^(.\*)[/\\].\*?$')

dir\_match = re.match(dir\_from\_path, output\_name)

**if** dir\_match **is** **not** None:

target\_dir = dir\_match.group(1)

**else**:

target\_dir = ''

# Set infile parameter

infile\_param = target\_dir + '/seqs.fasta'

cluster\_msa = cluster\_retriever(cluster)

pdb\_sequences = sequence\_retriever(pdbpaths)

combine(infile\_param, cluster\_msa, pdb\_sequences)

# Set matrix parameter

# It could be a matrix built into clustal:

**if** matrix.lower() **in** ('blosum', 'pam', 'id', 'gonnet'):

matrix\_param = matrix

# Or it could be BBTM:

**elif** matrix.lower() == 'bbtm':

# These pairings of sequence identities and evolutionary distances

# are taken from a 1994 Clustal paper. They're the same as the

# pairings in that paper

# of sequence identities and PAM numbers for the PAM matrices

series\_path = target\_dir + r'\bbtm\_series.txt'

write\_series([( 0, 40 , 350),

(41, 60 , 120),

(61, 80 , 60 ),

(81, 100, 20 )], series\_path)

matrix\_param = os.getcwd() + '\\' + series\_path

# But otherwise I don't know what it is

**else**:

**raise** UnrecognizedMatrix('Matrix "{0}" not '.format(matrix) \

+ 'recognized. Try BBTM, PAM, '\

+ 'GONNET, BLOSUM or ID.')

**return** call\_clustalw(infile\_param, matrix\_param, output\_name)

**def** align\_all(matrix, output\_dir):

# Retrieve pdbids of structures in the dataset

# and their associated clusters, from the information in Daniel's

# thesis.

pdbid\_clusterid = list()

with open('structure dataset and clusterguide.csv', 'rb') as f:

first = True

**for** row **in** csv.reader(f):

# Skip the first line

**if** first:

first = False

**continue**

# Get the pdbid

**if** row[1] != '':

pdbid = row[1]

**else**:

**continue**

# Get the clustername from the Cluster Name column

**if** row[2] != '':

cluster = row[2]

# But maybe this structure wasn't located to a supercluster

# In that case, get it from the subcluster column.

**else**:

cluster = row[3]

pdbid\_clusterid.append((pdbid, cluster))

# Make the alignments

**for** pdbid, cluster **in** pdbid\_clusterid:

# align(output\_name, cluster, matrix, \*pdbpaths)

align('{}/{} with {}.clu'.format(output\_dir, pdbid, cluster),

cluster, 'gonnet',

'ezbeta aligned structures/aligned\_{}.pdb'.format(pdbid))

The command "align\_all('gonnet', 'gonnet aligned')" was then entered.

The alignments are in the folder "automated\_sequence\_alignment/gonnet aligned". I sent the alignments to Daniel, in a zip file, "gonnet aligned.zip".

# September 4, 2012

Made all of the above code except for "align\_all" into a module called "alignments".

# November 12, 2012

The alignments do not have the template seqquences properly named; the PDB sequence is not in the name. I think I fixed "modules/alignments.py" so that it would name them properly, for the purpose of making zdiffs. So, I'm just going to rerun the script that makes the gonnet alignments and see how it comes out.

I deleted all the current Gonnet alignments, and reran the alignment-generating script with a small inconsequential modification (removing a parameter that was never actually used).