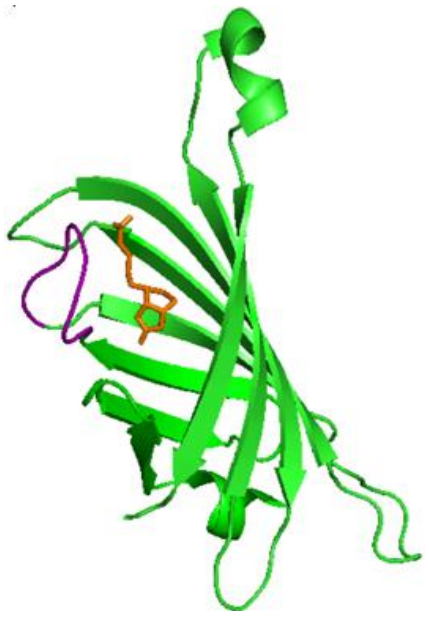
Protein Sequence and Structural Analysis with the R Package Bio3D

Streptavidin is a homotetrameric biotin binding protein first isolated from the bacterium *Streptomyces avidinii*.[1] Of particular interest, its interaction with biotin is one of the strongest non-covalent interactions found in nature.[2] An 8-stranded beta-barrel forms the core structure of a streptavidin monomer and the biotin binding pocket is located within. Several intermolecular interactions between biotin and the pocket are responsible for the tight interaction, as well as a flexible loop that closes upon biotin binding and seals off the pocket.[3]


Streptavidin monomer. Loop 3-4, shown in purple, closes and makes close contact with biotin upon binding. Adapted from 1STP.[3]

The uniquely tight and selective interaction, in combination with streptavidin's high stability, has found it widespread use in applications requiring essentially irreversible binding. Among many such applications include live cell imaging, affinity chromatography and biosensor development.[4] However, the utility of wild-type streptavidin is limited in some applications because the strength of the interaction prevents biotin dissociation except under extreme denaturing conditions, precluding its use in many areas. Because of this, a number of streptavidin mutants have been created to reduce the binding strength and allow reasonable biotin dissociation while still maintaining high specificity.

By design, these reversibly binding mutants obviously lose their capacity for essentially irreversible binding that makes wild-type streptavidin so attractive for immobilisation applications. An ideal, if somewhat paradoxical, mutein would be one that allows reversible binding but also irreversible immobilisation and maintains the high specificity of the biotin-streptavidin interaction. To this end, it has been proposed that a disulfide bond can be introduced into loop 3-4 of a reversibly binding streptavidin mutant to, essentially, lock the loop in place upon biotin binding and thus prevent its dissociation. To aid in the design of such a protein, it would be prudent to determine if any naturally occurring streptavidin homologs already employ this strategy.

The search for homologous proteins sequences is made simple with the Basic Local Alignment Search Tool (BLAST).[5] Of course, manually inspecting the possibly thousands of matches manually is both time consuming and error prone. Fortunately, the R package Bio3D provides a programmatic solution to this problem in addition to facilitating the subsequent sequence and structural analysis.[6,7] Using this package, we can identify potential streptavidin homologs that show a disulfide bond within the loop 3-4 region and perform sequence alignment and structural superposition in preparation for structure guided design of a novel streptavidin mutant. While this is obviously a very specific use case, the techniques used here can be applied to sequence and structural analysis of virtually any group of homologous proteins.

Installing Bio3D and Script Set-Up

Before beginning, it should be noted that this script requires internet access and was tested only on a computer running Windows. A new directory should be created for running the script, since a lot of files are generated.

The Bio3D package is freely available and easy to install from within R. It is also well documented with help files available within RStudio.

```r
install.packages("bio3d") #install bio3d (only required once)
library(bio3d) #attach bio3d package to this script
help(package = "bio3d") #access help files (optional)
```

Bio3D also requires an external program, MUCLE, for multiple sequence alignments. It can be downloaded here: http://www.drive5.com/muscle/downloads.htm. Choose the appropriate download for your computer and save it as muscle.exe in your directory.

The package gplot is used, briefly, for generating heatmaps of the analysis. It can be installed similarly.

```r
install.packages("gplots") #install gplots for heatmaps (only required once)
library(gplots) #and attach gplots
```

All that remains is to clear the workspace and set the working directory to the one created earlier. At this point, the directory should contain the R script and muscle.exe.

```r
rm(list = ls(all = T)) #clear the workspace
setwd("...") #set the working directory
```

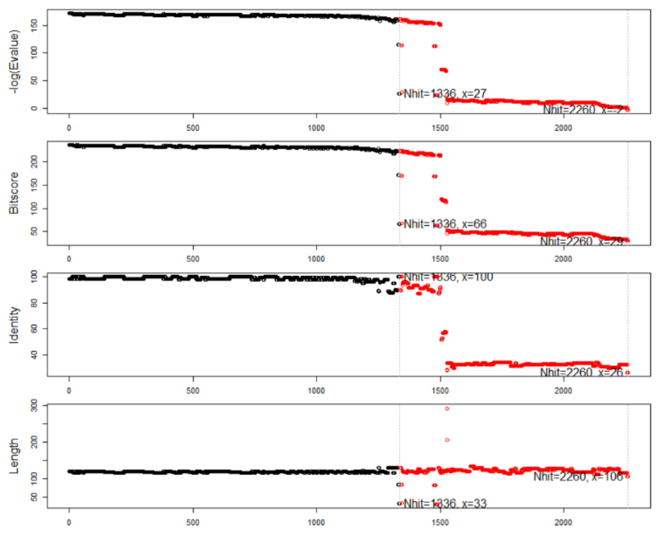
Finding Streptavidin Homologs with BLAST

Bio3D can accept a number of inputs for its blast.pdb function, including PDB files, fasta sequences and raw strings. Since structural analysis is being done here, PDB files will be used as much as is feasible. These files can be downloaded from the Protein Data Bank with Bio3D using the read.pdb function and a 4 or 6 letter PDB identifier. The streptavidin model used for this search will be 1SWE. Once the file has loaded, the sequence can be extracted with Bio3D’s pdbseq function.

```
savID = "1SWE" #PDB ID of a streptavidin model we will use for BLAST
savID_A = "1SWE_A" #streptavidin is a tetramer, this is the PDB ID of a
single monomer (chain A in the model)
sav = read.pdb(savID) #read the PDB into R, this downloads the file from the
Protein Data Bank
savseq = pdbseq(sav) #extract the protein sequence from the PDB file
```

Bio3D can access several databases with blast.pdb, including non-redundant protein sequences, protein data bank and swissprot. Again, since only proteins with resolved 3D structures are of interest here, the protein data bank will be searched (the default). All that must be passed to blast.pdb is a protein sequence. Note that this can take some time depending on the sequence being searched, your internet speed and the speed of the server. This search returns 2260 hits.

```
blast = blast.pdb(savseq) #BLAST search
plot(blast) #quick plot of the BLAST results
```



After the results have been downloaded, some summary statistics are easily plotted, including the E value, identity and length. When running the plot function, a dialog may appear in the console due to the large size of the results; ‘n’ can be entered here for the plot to appear. The E value is used to determine the significance of results. The higher the -log(E) of a result, the more likely that it is significant. The blast.pdb function automatically selects a suitable cut-off based on when there is a large drop in -log(E), which is easily visible in the plot.

Removing Duplicates and Downloading Sequences

Due to the nature of PDB files, the BLAST results return a number of duplicate entries which should be removed for efficient analysis. The first step is simply removing duplicate entries, which brings the candidates from 2260 to 565. The A chain of the search protein, 1SWE, is included as the first entry so that it can be easily referenced later.

```
pdbs = c(savID_A, blast$pdb.id) #get a vector of the PDB IDs from BLAST and include 1SWE_A as the first entry
length(pdbs) #2261 PDBs
pdbs = unique(pdbs) #remove duplicate entries
length(pdbs) #565 PDBs
```

While there is now a vector of unique identifiers, there are still a number of duplicate entries. This is because streptavidin is a naturally multimeric protein and so the list contains each, identical chain for each PDB ID. These can be removed using a custom function, getAllChains, and looping. getAllChains uses a regular expression, explained in the commented code, to match all 6 letter PDB chain IDs for any given letter of the alphabet. Thus, for example, and PDB ID for chain A can be found and, in fact, this is done to create the first population of PDBs (since all A chains are unique by this stage). Next, each subsequent letter of the alphabet is iterated and the corresponding chains are found with getAllChains. Finally, using the %in% operator, it is determined which 4 letter PDB is not already found in the list of unique PDBs and these are added. This pares down the list of PDBs to 222.

```
#given a chain ID and character vector of 6-letter pdb identifiers, returns all pdbs with a specific chain
getAllChains = function(id, pdbs) {
  #using regular expressions to find matches: [A-Z0-9] matches any upppercase letter or digit, {4} indicates it should be matched 4 times, _ matches _
  #the chain ID is concatenated onto the regex
  return(pdbs[grep(paste("[A-Z0-9]{4}_", id, sep = ""), pdbs)])
}

tpdbs = getAllChains("A", pdbs) #get all A chains
for(i in LETTERS[-1]) { #iterate through all letters of the alphabet except A (which we've already accounted for)
  pdbx = getAllChains(i, pdbs) #get all PDB chains that match the current
letter
  tpdbs = c(tpdbs, pdbx[!(substr(pdbx, 1, 4) %in% substr(tpdbs, 1, 4))])
}
pdbs = tpdbs
rm(pdbx, tpdbs, i) #clean-up
length(pdbs) #222 PDBs
```

Finding Potential Disulfide Bond Candidates

To analyze these proteins necessitates downloading their sequences. This is done with Bio3D's get.seq function. To this function is passed the list of PDB sequences to download and the name of the output fasta file to generate containing all of them. Note that this has the potential to be quite time consuming depending on computer and internet speed.

```
seqs = get.seq(pdbs, "./seqs.fasta") #download sequences of all unique PDBs
into a fasta file
length(seqs$id) #simple check to show that all sequences have been found and
downloaded (compare with length of pdbs)
```

To find which proteins contain a cysteine in their loop 3-4 region, this region must be defined in the known structure 1SWE so that other proteins can be aligned to it. Loop 3-4 is aptly named, being the loop between the end of beta-strand 3 and start of beta-strand 4. The PDB file downloaded earlier contains the residue number of these secondary structure elements which can be used to define the residues in the loop.

```
sav_Aseq = savseq[unique(names(savseq))] #sequence of single chain (A)
sav_Aresn = as.numeric(names(sav_Aseq)) #associated residue numbers
loop_34resi = sav_Aseq[sav_Aresn >= sav$sheet$end[3] & sav_Aresn <=
sav$sheet$start[4]] #get the residues in loop 3-4 of sav
```

Next, a multiple sequence alignment is performed with the seqaln function, which makes a call to the external program MUSCLE.

```
aln = seqaln(seqs) #multiple sequence alignment (msa)
```

Finally, the loop 3-4 region of each aligned streptavidin homolog is extracted based on the alignment and the loop 3-4 residues of 1SWE found earlier. A regular expression is used to search the alignment for the loop 3-4 sequence in 1SWE to determine its coordinates in the alignment (since gaps, represented as -, are introduced in the alignment where residues are not conserved). Then, each protein in the alignment is looped through to determine if cysteine (C) is present in these coordinates, and 17 matches are found.

```
match = regexpr(paste(loop_34resi, collapse = "-*?"), paste(aln$ali[1, ],
collapse = "")) #find residue numbers of loop 3-4 in msa

tpdbs = savID_A #create temporary vector for storing PDBS
for(i in 1:length(aln$id)) { #loop through all candidate sequences
  if("C" %in% aln$ali[i, match[1]:(match[1] + attr(match, "match.length"))])
{ #if Cys is present in homologous loop 3-4 region
    tpdbs = c(tpdbs, pdbs[i]) #store the PDB ID
  }
}
pdbs = tpdbs
length(pdbs) #17 PDBs
rm(tpdbs, i) #clean-up
```

Manual Selection of Candidates

Selection of candidates from the remaining PDBs requires some manual inspection. Information from each PDB file can be downloaded with pdb.annotate. Only structures bound to biotin are of interest, so the liganId attribute is searched for such models and the relevant information is extracted into a data frame. From this, four interesting structures are found (rhizavidin, shwanavidin, hoefavidin, bradavdin) and these are selected for further study. The original streptavidin is also selected for comparison.

```
anno = pdb.annotate(pdbs) #download PDB annotations from protein data bank
summ = anno[grep("BTN", anno$ligandId), c("structureId", "structureTitle",
"resolution", "source", "structureMolecularWeight", "chainLength",
"citation")] #get the PDBs that are bound to biotin and return the
"interesting" columns
summ = summ[c(1, 3, 5, 6, 7), ] #manual selection of candidate structures
summ
```

Sequence and Structural Analysis

For the structural alignment, the PDB files must be downloaded. This can be done in batch with get.seq which accepts a vector of 4 or 6 letter PDB IDs and the directory to save the files to. Since the subunits of these proteins are identical, only one from each PDB needs to be aligned and so an individual chain from each PDB (A) is extracted and saved with pdbsplit.

```r
pdbs = paste(summ$structureId, "_A", sep = "") #recreate PDB vector with
candidate chain As
models = get.pdb(pdbs, "./models") #download models from the Protein Data
Bank
length(models) #check the length
models = pdbsplit(models, pdbs, path = "./models/chains") #split the models
to get single chains
```

The PDB sequences are aligned again using pdbaln, which also does MUSCLE based sequence alignment, and written to a fasta file with write.fasta.

```r
mpdbs = pdbaln(models) #seqeunce alignment of PDBs
write.fasta(alignment = mpdbs, file = "aln.fasta") #write the alignment as a
fasta file
```

An iterative structural superposition is performed with core.find to get the region(s) of the proteins that are most similar. From a protein sequence alignment, individual coordinates of each protein are compared and the regions showing most variance are removed at each iteration until a cut-off is reached. At this point, the remaining coordinates are considered to define the core region that is invariant between these proteins. This core region is then used for structural superposition of the entire models so that they may be visually inspected. This is done with Bio3D's pdbfit function, which accepts the PDB alignment, core region and, optionally, the output path for the superimposed models. Once these models are generated and exported, they can be inspected in any PDB viewer, such as PyMOL (image on page 9).

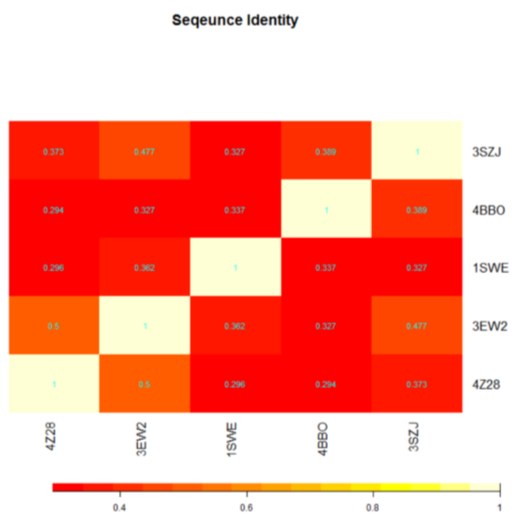
```r
savcore = core.find(mpdbs) #find the most invariant region(s) of the models
xyz = pdbfit(mpdbs, savcore, outpath = "./models") #structural superposition
of the models based on the core region
```

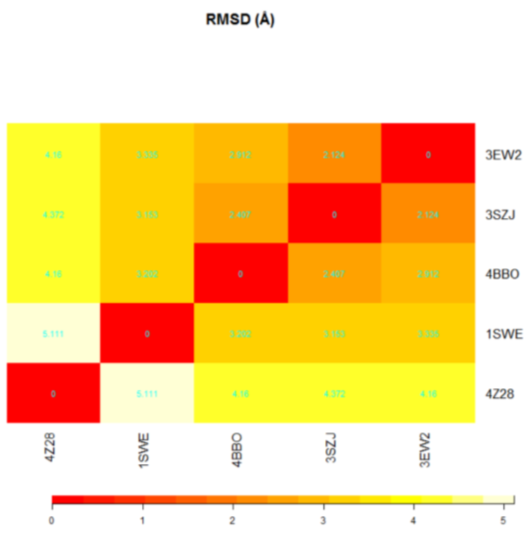
Important information from these sequence and structural alignments is generated by Bio3D and can be easily plotted. Two particularly relevant data are the sequence identity, determined from an alignment with the seqidentity function, and the root mean squared deviation, determined from a structural superposition with the rmsd function. These can be plotted as heatmaps with the gplot package for simple, visual pairwise comparisons. An in-depth explanation of gplot is beyond the scope of this paper but, simply, the heatmap.2 function plots a correlation matrix, such as that generated by seqidentity or rmsd.

```r
seqid = seqidentity(mpdbs) #get the pair-wise seqeunce identity of the
aligned proteins
```

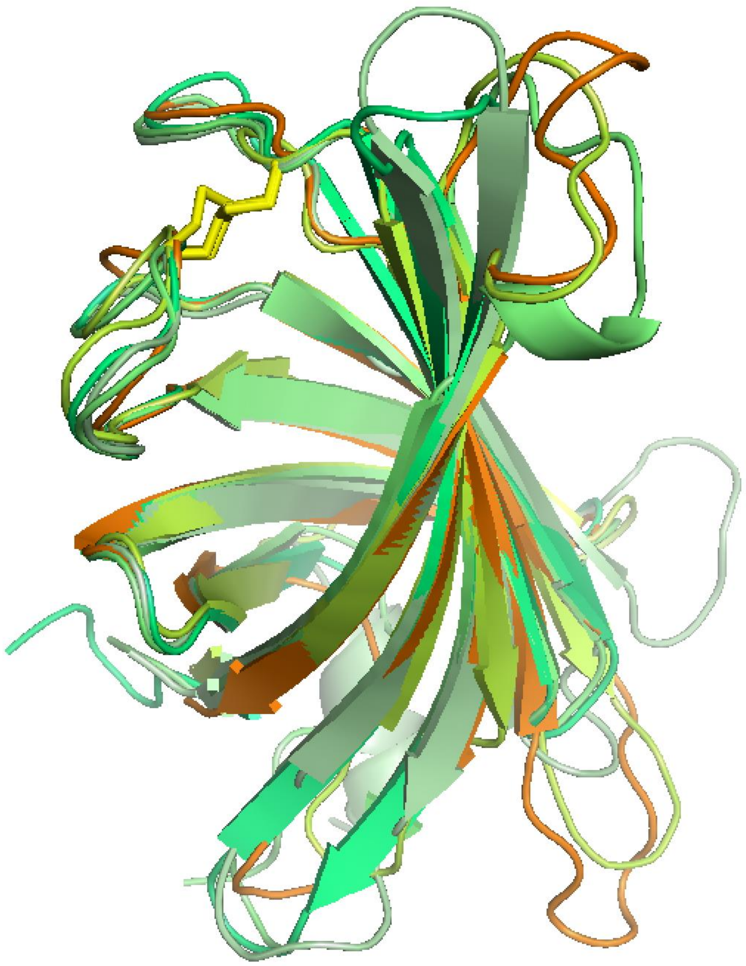
```
heatmap.2(seqid, cellnote = seqid, trace = "none", lmat =
rbind(c(0,3),c(2,1),c(0,4)), lwid = c(1.5,4), lhei = c(1.5,4,1), density.info
= "none", dendrogram = "none", key.title = "", key.xlab = "", main =
"Seqeunce Identity", labRow = summ$structureId, labCol = summ$structureId)
#heatmap with gpolt

alnrmsd = rmsd(xyz) #find pairwise rmsd of the structures
heatmap.2(alnrmsd, cellnote = alnrmsd, trace = "none", lmat =
rbind(c(0,3),c(2,1),c(0,4)), lwid = c(1.5,4), lhei = c(1.5,4,1), density.info
= "none", dendrogram = "none", key.title = "", key.xlab = "", main = "RMSD
(Å)", labRow = summ$structureId, labCol = summ$structureId) #heatmap with
gpolt
```





Bio3D in R brings together many components of protein analysis, such as homology searching, multiple sequence alignment and structural superposition, into one package for seamless use. It is also capable of much more advanced and predicative analyses. For a full overview and many tutorials, visit: http://thegrantlab.org/bio3d/



Structural alignment of streptavidin (orange) and streptavidin homologs (greens). Disulfide bond of loop 3-4 in the homologs is shown in yellow. Striking is the degree of structural alignment, seen both visually and by RMSD, despite the low sequence conservation.

References

1. Chaiet, L. & Wolf, F. J. The Properties of Streptavidin, a Biotin-Binding Protein Produced by Streptomycetes. *Arch. Biochem. Biophys.* **106,** 1–5 (1964).

2. Green, N. M. Avidin and streptavidin. *Methods Enzymol.* **184,** 51–67 (1990).

3. Weber, P. C., Ohlendorf, D. H., Wendoloski, J. J. & Salemme, F. R. Structural Origins of High-Affinity Biotin Binding to Streptavidin. *Science (80-.).* **243,** 85–88 (1989).

4. Dundas, C. M., Demonte, D. & Park, S. Streptavidin-biotin technology: improvements and innovations in chemical and biological applications. *Appl. Microbiol. Biotechnol.* **97,** 9343–53 (2013).

5. Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D. Basic local alignment search tool. *J. Mol. Biol.* **215,** 403–41 (1990).

6. Skjærven, L., Yao, X.-Q., Scarabelli, G. & Grant, B. J. Integrating protein structural dynamics and evolutionary analysis with Bio3D. *BMC Bioinformatics* **15,** 399 (2014).

7. Grant, B. J., Rodrigues, A. P. C., ElSawy, K. M., McCammon, J. A. & Caves, L. S. D. Bio3d: An R package for the comparative analysis of protein structures. *Bioinformatics* **22,** 2695–2696 (2006).

Appendix – Full R Script

```r
install.packages("bio3d") #install bio3d (only required once)
library(bio3d) #attach bio3d package to this script
help(package="bio3d") #access help files (optional)

install.packages("gplots") #install gplots for heatmaps (only required once)
library(gplots) #and attach gplots

rm(list = ls(all = T)) #clear the workspace
setwd("...") #set the working directory


###


savID = "1SWE" #PDB ID of a streptavidin model we will use for BLAST
savID_A = "1SWE_A" #streptavidin is a tetramer, this is the PDB ID of a single monome
r (chain A in the model)
sav = read.pdb(savID) #read the PDB into R, this downloads the file from the Protein
Data Bank
savseq = pdbseq(sav) #extract the protein sequence from the PDB file


blast = blast.pdb(savseq) #BLAST search
plot(blast) #quick plot of the BLAST results


###


pdbs = c(savID_A, blast$pdb.id) #get a vector of the PDB IDs from BLAST and include 1
SWE_A as the first entry
length(pdbs) #2261 PDBs
pdbs = unique(pdbs) #remove duplicate entries
length(pdbs) #565 PDBs

#given a chain ID and character vector of 6-letter pdb identifiers, returns all pdbs
with a specific chain
getAllChains = function(id, pdbs) {
  #using regular expressions to find matches: [A-Z0-9] matches any upppercase letter
or digit, {4} indicates it should be matched 4 times, _ matches _
  #the chain ID is concatenated onto the regex
  return(pdbs[grep(paste("[A-Z0-9]{4}_", id, sep = ""), pdbs)])
}

tpdbs = getAllChains("A", pdbs) #get all A chains
for(i in LETTERS[-1]) { #iterate through all letters of the alphabet except A (which
we've already accounted for)
  pdbx = getAllChains(i, pdbs) #get all PDB chains that match the current letter
  tpdbs = c(tpdbs, pdbx[!(substr(pdbx, 1, 4) %in% substr(tpdbs, 1, 4))])
}
pdbs = tpdbs
rm(pdbx, tpdbs, i) #clean-up
length(pdbs) #222 PDBs
```

```
###


seqs = get.seq(pdbs, "./seqs.fasta") #download sequences of all unique PDBs into a fa
sta file
length(seqs$id) #simple check to show that all sequences have been found and download
ed (compare with length of pdbs)

sav_Aseq = savseq[unique(names(savseq))] #sequenced of single chain (A)
sav_Aresn = as.numeric(names(sav_Aseq)) #associated residue numbers
loop_34resi = sav_Aseq[sav_Aresn >= sav$sheet$end[3] & sav_Aresn <= sav$sheet$start[4
]] #get the residues in loop 3-4 of sav

aln = seqaln(seqs) #multiple sequence alignment (msa)

match = regexpr(paste(loop_34resi, collapse = "-*?"), paste(aln$ali[1, ], collapse =
"")) #find residue numbers of loop 3-4 in msa

tpdbs = savID_A #create temporary vector for storing PDBS
for(i in 1:length(aln$id)) { #loop through all candidate sequences
  if("C" %in% aln$ali[i, match[1]:(match[1] + attr(match, "match.length"))]) { #if Cy
s is present in homologous loop 3-4 region
    tpdbs = c(tpdbs, pdbs[i]) #store the PDB ID
  }
}
pdbs = tpdbs
length(pdbs) #17 PDBs
rm(tpdbs, i) #clean-up


###


anno = pdb.annotate(pdbs) #download PDB annotations from protein data bank
summ = anno[grep("BTN", anno$ligandId), c("structureId", "structureTitle", "resolutio
n", "source", "structureMolecularWeight", "chainLength", "citation")] #get the PDBs t
hat are bound to biotin and return the "interesting" columns
summ = summ[c(1, 3, 5, 6, 7), ] #manual selection of candidate structures
summ


###


pdbs = paste(summ$structureId, "_A", sep = "") #recreate PDB vector with candidate ch
ain As
models = get.pdb(pdbs, "./models") #download models from the Protein Data Bank
length(models) #check the length
models = pdbsplit(models, pdbs, path = "./models/chains") #split the models to get si
ngle chains


mpdbs = pdbaln(models) #sequence alignment of PDBs
```

```r
write.fasta(alignment = mpdbs, file = "aln.fasta") #write the alignment as a fasta fi
le

savcore = core.find(mpdbs) #find the most invariant region(s) of the models
xyz = pdbfit(mpdbs, savcore, outpath = "./models") #structural superposition of the m
odels based on the core region

seqid = seqidentity(mpdbs) #get the pair-wise sequence identity of the aligned protei
ns
heatmap.2(seqid, cellnote = seqid, trace = "none", lmat = rbind(c(0,3),c(2,1),c(0,4))
, lwid = c(1.5,4), lhei = c(1.5,4,1), density.info = "none", dendrogram = "none", key
.title = "", key.xlab = "", main = "Seqeunce Identity", labRow = summ$structureId, la
bCol = summ$structureId) #heatmap with gpolt

alnrmsd = rmsd(xyz) #find pairwise rmsd of the structures
heatmap.2(alnrmsd, cellnote = alnrmsd, trace = "none", lmat = rbind(c(0,3),c(2,1),c(0
,4)), lwid = c(1.5,4), lhei = c(1.5,4,1), density.info = "none", dendrogram = "none",
key.title = "", key.xlab = "", main = "RMSD (Å)", labRow = summ$structureId, labCol =
summ$structureId) #heatmap with gpolt
```