Using Protein Protein Predictions

R. Gentleman

April 25, 2006

? describe methodology that they employed to make predictions of protein-protein interactions in *S. cerevisiae*. They have provided a downloadable set of predicted interactions, from `http://bioinformatics.med.yale.edu/interaction/` and these data are available as the `ppipred` data.

```
> library("y2hStat")

Loading required package: GO
Loading required package: GOstats
Loading required package: graph
Loading required package: Ruuid
Loading required package: annotate
Loading required package: Biobase
Loading required package: RBGL
Loading required package: xtable

Attaching package: 'xtable'


        The following object(s) are masked from package:graph :

         label

Loading required package: Biobase
Loading required package: genefilter
Loading required package: survival
Loading required package: splines
Loading required package: multtest
Loading required package: Category
Loading required package: KEGG
Loading required package: hgu95av2
Loading required package: YEAST
Loading required package: ScISI
Loading required package: apComplex
Loading required package: Rgraphviz
```

```
Loading required package: graph
Loading required package: Rgraphviz

> data(ppipred)
```

The first two columns are yeast gene names, typically they are common names and will need to be translated to systematic names for further processing.

A first decision that should be made is whether to select only those interaction pairs where the prediction is larger than a specified cut-off. For our example we will use a cut-off of 0.5, largely to ensure that the lists are quite short.

```
> ppi2 = ppipred[ppipred$Prob >= 0.5, ]
> dim(ppi2)

[1] 2854    3
```

Now we translate the names. For now we don't worry too much, but one problem is that in the *YEAST* package only verified ORFs are used, and so the uncharacterized ones are not getting translated. We will need some way to do that, in the long run.

```
> library("YEAST")
> trN1 = mget(ppi2[[1]], YEASTCOMMON2SYSTEMATIC, ifnotfound = NA)
> trN1 = sapply(trN1, function(x) x[1])
> ppi2[[1]] = ifelse(is.na(trN1), ppi2[[1]], trN1)
> trN2 = mget(ppi2[[2]], YEASTCOMMON2SYSTEMATIC, ifnotfound = NA)
> trN2 = sapply(trN2, function(x) x[1])
> ppi2[[2]] = ifelse(is.na(trN2), ppi2[[2]], trN2)
```

Now there are 2854 pairs left. For our purposes we want to find interaction pairs that are wholy contained within a predicted protein complex, as that will be used as a basis for helping to determine if particular protein complexes are well defined.

```
> pairsBy = split(ppi2[[2]], ppi2[[1]])
> table(sapply(pairsBy, length))

  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  22
464 198  82  76  50  33  25  12   9  11   6   8   5   1   1   1   1   1   2   1
 23  24  26  29  32  33
  1   1   1   1   1   1
```

The basic idea here is to think of each pair as a bait-prey (although that is not true) since it helps us to reduce the number of protein complexes that need to be examined.

Next we load up the *ScISI* package.

```
> library("ScISI")
> data(ScISI)
```

And now, we find complexes that contain at least one bait.

```
> haveB = row.names(ScISI) %in% names(pairsBy)
> ScSub1 = ScISI[haveB, ]
> havC = colSums(ScSub1) > 0
> ScSub2 = ScISI[, havC]
> byComp = function(cMat, bpL) {
+     rn = row.names(cMat)
+     bNames = names(bpL)
+     ans = rep(0, ncol(cMat))
+     for (i in 1:ncol(cMat)) {
+         nIn = 0
+         protInC = rn[cMat[, i] > 0]
+         wB = bNames[bNames %in% protInC]
+         if (length(wB) == 0)
+             warning("Complex", i, "has a problem")
+         else {
+             for (j in wB) nIn = nIn + length(intersect(bpL[[j]],
+                 protInC))
+         }
+         ans[i] = nIn
+     }
+     ans
+ }
> t1 = byComp(ScSub2, pairsBy)
```