Using the functions within the ppiData package

T Chiang

April 18, 2015

For those interested in analyzing the publicly available protein-protein interaction (PPI) data located within the IntAct repository, `ppiData` is an R-data package with built in functions that can parse through the IntAct repository and gather empirical PPI data such as physical binary interactions between protein pairs via the Yeast 2-Hybrid (Y2H) systems or such as complex co-membership between sets of proteins via the Affinity Purification - Mass Spectrometry (AP-MS) systems. In this vignette, we show how one can collect such PPI data from IntAct, how one can take the collected data and generate useful R-objects (such as the adjacency matrix representation, graph representations, etc), and how to do some simply statistical computations.

For the remainder of the vignette, we shall only be concerned with collecting three particular Y2H data-sets: those corresponding to "awasthi-2001-1", "cagney-2001-1", and "zhao-2005-2" (though we will note which functions are used to create the various data files in this data package). We also use two new terms: viable baits and viable preys. A viable bait for a particular experiment is any protein sampled as a bait that detects at least one other protein as prey within the respective experiment. Viable preys are similarly defined.

The first function that we will call will be the `collectIntactPPIData` function. As the name implies, this function goes into the Intact repository, which exists within this package as a modified XML file downloaded from its primary souce and collects the relevant data given the particular experiments (as given by its IntAct Accession (AC) Identification Code). In our working example ("awasthi-2001-1", "cagney-2001-1", "zhao-2005-2") is identified within IntAct by the AC codes ("EBI-531419", "EBI-698096","EBI-762635"). Before we can gather the information concerning any empirical data, we must first identify its corresponding AC code. (We shall discuss this at the end since this involves searching the internet.)

Let's begin by collecting the specified data-sets via the function `collectIntactPPIData`:

```
> dataList <- collectIntactPPIData(c("EBI-531419", "EBI-698096","EBI-762635"))
> names(dataList)

[1] "allBaits"        "allPreys"        "indexSetAll"     "baitsSystematic"
[5] "preysSystematic" "shortLabel"

> dataList[["shortLabel"]]

[1] "awasthi-2001-1" "cagney-2001-1"  "zhao-2005-2"
```

```
>
> ##NB - intactPPIData = collectIntactPPIData() using the default parameters
>
```

The only argument that `collectIntactPPIData` takes is a character vector of IntAct AC codes which corresponds to particular data-sets (or equivalently, the experiments).

The return value for the function `collectIntactPPIData` is a list of five elements. The "allBaits" and "allPreys" entries of this list contains the IntAct AC Codes of all the unique proteins that proved to be viable baits (respectively viable preys) over all the experiments for which we sought data (in our case, over all three experiments). Though we cannot determine the viable baits from any particular experiment from the "allBaits" entry (similarly for the "allPreys"), but we shall see that this can be done another entry to be discussed. The "baitsSystematic" and "preysSystematic" entries are named lists; the names corrospond to the IntAct AC codes while the entries are the systematic gene name(s) that each AC code can be mapped to.

```
> dataList[["baitsSystematic"]][1:5]

$`EBI-724`
[1] "YIL061C"

$`EBI-15913`
[1] "YHR027C"

$`EBI-13905`
[1] "YDR394W"

$`EBI-13914`
[1] "YGL048C"

$`EBI-15935`
[1] "YDL147W"
```

The "indexSetAll" entry of dataList contains the bait to prey associations for each of the empirical data set. The "indexSetAll" entry contains sub-lists corresponding to each empirical data set. Each sub-list contains a number of length two character vectors: the first entry of this character vector is the bait protein (give by AC code), and the second is the prey protein. Thus, it is not too difficult to obtain the viable baits and viable prey from the "indexSetAll" entry with a little bit of coding. This method is still quite inconvienent since we would still be left with the intact AC codes rather than anything meaningful about the bait and prey proteins.

```
> dataList[["indexSetAll"]][1]

$`awasthi-2001-1`
$`awasthi-2001-1`[[1]]
```

```
 interactor  interactor
  "EBI-724" "EBI-27760"

$`awasthi-2001-1`[[2]]
interactor interactor
 "EBI-724"  "EBI-465"

$`awasthi-2001-1`[[3]]
 interactor  interactor
  "EBI-724" "EBI-21567"

$`awasthi-2001-1`[[4]]
interactor interactor
 "EBI-724" "EBI-4618"
```

As the structure of dataList stands, there is little we can do to manipulate the PPI data for mathematical, graphical, or statistical tests. Because of the undesirability of the data as is, we can generate a sparse matrix representation of the bait to prey affiliation data. For such a representation, we call the `createBPList` function using the entries "indexSetAll", "baitsSystematic", and "preysSystematic" entries of the `dataList` R-object as the arguments for `createBPList`.

```
> bpList <- createBPList(dataList[["indexSetAll"]], dataList[["baitsSystematic"]],
+                        dataList[["preysSystematic"]])
> names(bpList)

[1] "awasthi-2001-1" "cagney-2001-1"  "zhao-2005-2"

> bpList[1]

$`awasthi-2001-1`
$`awasthi-2001-1`$YIL061C
interactor interactor interactor interactor
 "YML029W"  "YHR165C"  "YBR102C"  "YNL192W"

>
> ## NB: y2h = createBPList(intactPPIData[["indexSetAll"]], intactPPIData[["baitsSystematic"]
>
> ## NB: y2hSysGW = Fixme
```

The sparse matrix representation is given by a list of list. The top list contains sub-lists which are named by each experimentor of the data-set (now called the experimental sub-list). Each experimental sub-list itself contains a number of sub-lists (we shall refer to these sub-lists as bait sub-lists). Each bait sub-list is named by a viable bait of the corresponding experiment. The entry for the bait sub-list is a character vector of proteins which were dectected by the corresponding bait. In essence, we can think of this sparse matrix representation as a rooted

tree. The child of the root brings us to a particular experiment (data-set); the child of an experiment brings us to a viable bait of that experiment; and finally, the child of a viable bait (the leaves of this rooted tree) is a viable prey of the viable bait.

How is this a sparse matrix representation? Well if we were to construct an adjacency matrix for each experimental data-set where the viable baits of the experiment indexed the rows and the viable preys of the experiment indexed the columns, we could put a non-negative integer for the number of times a viable bait detected a viable prey. This matrix representation is generally sparse, and so to avoid carring all the zeros, our R-object `bpList` suffices.

It will be useful now to mention the two data files in the R-package *ppiStats*, `y2h` and `y2hSysGW` are of the same structure as `bpList`. Both `y2h` and `y2hSysGW` are collections of sparse matrix representations for various empirical data-sets: `y2h` contains 42 Y2H experimental data-sets while `y2hSysGW` contains 7 Y2H data-sets under the condition that the prey population is genome-wide. Therefore `y2hSysGW` is a subset of `y2h`.

There will be times, however, when we would like to have the adjacency matrix representation of our data. When this is necessary, we can build these adjacency matrices by calling the R-function `bpMatrix` which comes from the *ppiStats*.

```
> bpMats <- lapply(bpList, function(x){
+                  bpMatrix(x, symMat = FALSE, homodimer = FALSE, baitAsPrey = FALSE,
+                      unWeighted = TRUE, onlyRecip = FALSE, baitsOnly = FALSE)})
> bpMats[1]

$`awasthi-2001-1`
        YML029W YHR165C YBR102C YNL192W
YIL061C       0       1       1       1
```

Other than the sparse matrix representation for the empirical data, there are seven other arguments (all logical) that the R-function `bpMatrix` takes. We shall discuss these parameters in turn.

- `homodimer`: If FALSE, all homodimer relationships will be disregarded, otherwise the homodimer relationships will be recorded in the adjacency matrix.
- `unWeighted`: If TRUE, the entries of the adjacency matrix will be binary (0,1) to account for the presence or absence of the bait to prey interaction without regards for multiplicity.
- `onlyRecip`: If TRUE, the matrix will be symmetric with only reciprocated interactions recorded in the adjacency matrix.
- `symMat`: If TRUE, the union of viable baits and viable preys will index both the rows and columns of the adjacency matrix.
- `baitAsPrey`: If TRUE, the union of the viable baits and viable preys will index the columns while only the viable baits will index the rows.
- `baitsOnly`: If TRUE, the matrix will be indexed in the row and column by the viable baits exclusively. NB - if this is set to TRUE, then baitAsPrey must also be set to TRUE.

Based on the mathematical or statistical test or the graphical representation desired, these parameters should be set accordingly.

Once we have generated the adjacency matrix, we can create an instance of the class *graph* by calling the `genBPGraph` function, also found within the *ppiStats*.

```
> bpMats1 <- lapply(bpList, function(x){
+                 bpMatrix(x, symMat = TRUE, homodimer = FALSE, baitAsPrey = FALSE,
+                     unWeighted = TRUE, onlyRecip = FALSE, baitsOnly = FALSE)})
> bpMats1[1]

$`awasthi-2001-1`
        YIL061C YML029W YHR165C YBR102C YNL192W
YIL061C       0       1       1       1       1
YML029W       0       0       0       0       0
YHR165C       0       0       0       0       0
YBR102C       0       0       0       0       0
YNL192W       0       0       0       0       0

> bpGraphs <- lapply(bpMats1, function(x){genBPGraph(x, directed=TRUE, bp=FALSE)})
> bpGraphs[1]

$`awasthi-2001-1`
A graphNEL graph with directed edges
Number of Nodes = 5
Number of Edges = 4

>
> ##NB: Each graph data file is generated similarly using the same function
> ##as above. To see how we generated each of the graph objects, look
> ##in inst/Script/genGraphs.R directory at the script used to generate the
> ##graphNELs
```

The `genBPGraph` is a simple function that takes three arguments:

- `bpMat`: The bait to prey adjacency matrix.
- `directed`: A logical. If TRUE, the function creates a directed graph. If FALSE, the function attempts to create an undirected graph if possible.
- `bp`: A logical. If TRUE, the adjacency matrix is that of a empirical bait to prey interaction matrix where the bait population is not the same as the prey population. Hence, the function will extend the matrix to a larger matrix where the rows and columns are indexed by the union of the baits and preys though the bait to prey interactions are preserved.

Depending on the type of analysis, we can represent the bait to prey interation data by the sparse matrix (list of list), the adjacency matrix representation, and a graphical representation. This vignette only presents enough background on the functions to get a working knowledge of the R-package *ppiStats*. We encourage you to use it in concert with other packages such as *Category* or *coCiteStats* to garner more methods on the data-sets.