HitWalker: Setting up a Database Part 2: Adding Metadata

Daniel Bottomly

November 28, 2012

1 Adding fixed metadata to existing annotation

Here we start with the ens_b57_string_v9_annot.db SQLite database created in Part 1. This database can be found in the HitWalkerData package as is shown below. There are two main types of metadata that be included in the output. The first type is fixed metadata which is consistent for all the samples. An example of this is the set of genes that were actually targeted in a sequence capture experiment. When examining a depiction of a given subnetwork it is informative to know which genes have mutations, which were targeted but did not have mutations and which were not targeted. This can aid in interpretation and potentially help guide future experiments. In this section, we are going to add example metadata to the existing annotation in a way that is persistent. Adding it in a non-persistent manner would involve adding it directly to the `annotatedIGraph` object by merging it in with existing annotation.

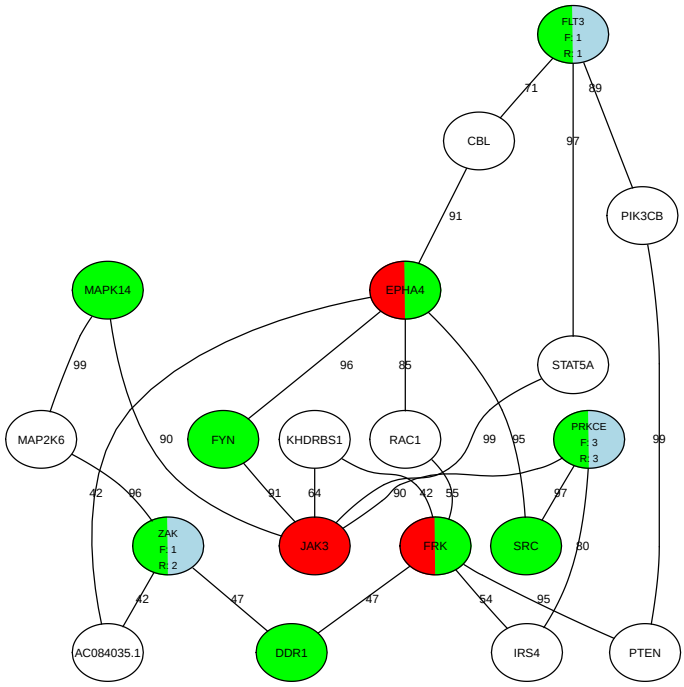
```
> stopifnot(require(HitWalker))
> stopifnot(require(RSQLite))
> stopifnot(require(HitWalkerData))
> #Copy the database to the current directory so it is not overwritten each time.
>
> local.annot.db <- file.path(getwd(), basename(annot.db.path()))
> stopifnot(file.copy(from=annot.db.path(), to=local.annot.db))
> graph.con <- dbConnect("SQLite", local.annot.db)
> #Here we add an additional table which contains two types of metadata.  The column
> #'cap_probes' indicates the presence of sequence capture probes
> #The column 'assay_targets' indicates the genes that are targets of functional assays.
>
> symbol <- c('EPHA4','ZAK','PIK3CB','AC084035.1','CBL','FRK','FYN','MAP2K6','KHDRBS1',
+     'PTEN','PRKCE','MAPK14','IRS4','STAT5A','RAC1','DDR1','SRC','JAK3','FLT3')
> cap_probes <- c(1,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1,1,1,1)
> assay_targets <- c(1,1,1,0,0,1,1,1,0,0,1,1,0,0,0,1,1,1,1)
> fixed.annot <- data.frame(symbol=symbol, cap_probes=cap_probes,
```

```
+     assay_targets=assay_targets)
> dbWriteTable(graph.con, "fixed_annots", fixed.annot, row.names=FALSE, overwrite=TRUE)

[1] TRUE

> #Put together a priorDbParams object from the basic one that is supplied
>
> test.parm <- HitWalker:::basePriorDbParams()
> matrixQuery(test.parm) <- function(sample.id.list = NULL, param.obj)
+ {
+     query <- "SELECT protein1, protein2, combined_score/1000.0 AS weight FROM
+         string_v9_db WHERE combined_score > 400;"
+     return(query)
+ }
> #now we can modify the annotation retrieval query to use this new table
>
> annotQuery(test.parm) <- function(sample.id.list = NULL, param.obj)
+ {
+     query <- "SELECT symbol, gene, transcript, protein, IFNULL(cap_probes,0) AS
+         cap_probes, IFNULL(assay_targets,0) AS assay_targets FROM annotation
+         LEFT OUTER JOIN fixed_annots USING (symbol);"
+ }
> scoreSummaryFunc(test.parm) <- HitWalker:::default.score.summary.func
> variantFilterObj(test.parm) <- HitWalker:::defaultVariantFilter()
> anno.igraph <- get.protein.protein.graph(graph.con, test.parm, get.annotation=TRUE,
+     on.symbol.protein.dup="remove.dup")

Retrieving graph structure
Retreiving gene annotation
Removing 0 nodes due to absense of annotation
Creating graph with 17419 nodes

> head(annotation(anno.igraph))

    symbol            gene      transcript         protein cap_probes assay_targets
9    BET1L ENSG00000177951 ENST00000382762 ENSP00000372210          0             0
10 SCGB1C1 ENSG00000188076 ENST00000342878 ENSP00000344545          0             0
12    ODF3 ENSG00000177947 ENST00000325113 ENSP00000325868          0             0
13   RIC8A ENSG00000177963 ENST00000325207 ENSP00000325941          0             0
14   SIRT3 ENSG00000142082 ENST00000382743 ENSP00000372191          0             0
16  PSMD13 ENSG00000185627 ENST00000431206 ENSP00000396937          0             0

> #Alternatively, this data could be added to the anno.igraph through modification of the
> #data.frame retrieved through the annotation method. It is safer to go through the
> #database at this time though.
>
> dbDisconnect(graph.con)
```

```
[1] TRUE

> stopifnot(file.remove(local.annot.db))
>
```

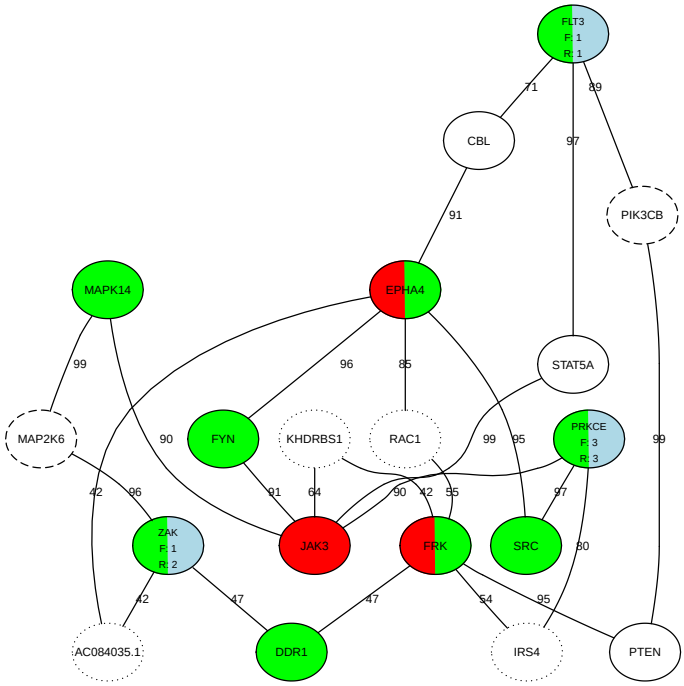
Now that the fixed annotation has been successfully added to the `annotatedIGraph` object, it can be output as part of the `summary` method for a `variantPriorResult` object. Additionally, we can add the annotations to the graph generated through the `plot` method. To do this for the plot, we need to modify the code contained within one or more of the `style.mapping.func`, `shape.mapping.func` or the `node.comb.func` slots. They can be accessed using similarly named methods as is shown below.

```
> #First prioritize the variants as usual
> var.hit.db <- dbConnect("SQLite", hitwalker.db.path())
> test.out <- run.prioritization.patient(var.hit.db, "08-00102", anno.igraph, test.parm)
> dbDisconnect(var.hit.db)

[1] TRUE

> #The metadata is perpetuated to the data.frame resulting from the summary method
>
> summary(test.out)[1:5,c("gene", "symbol", "type.rank", "cap_probes", "assay_targets")]

              gene  symbol type.rank cap_probes assay_targets
10 ENSG00000122025    FLT3         1          1             1
24 ENSG00000091436     ZAK         2          1             1
33 ENSG00000171132   PRKCE         3          1             1
48 ENSG00000071242 RPS6KA2         4          0             0
23 ENSG00000071909   MYO3B         5          0             0

>
> #If we plot it using the default parameters nothing special happens
>

> graph.params <- makeGraphDispParams(file.name = character())
> plot(test.out, graph.params)

Found 3 target nodes and 3 mut nodes
Finding target--mutation distances
Finding target--target distances
Finding mut--mut distances
Making subgraph
Converting to graphNEL

>
```




```
> #However, if we modify the mapping functions, we can change the shape and/or borders
>
> styleFunc(graph.params) <- function(dta)
+ {
+     if (all(c("cap_probes", "assay_targets") %in% colnames(dta)) ==
+         FALSE) {
+         return("solid")
+     }
+     else if (all(dta$cap_probes == 1 & dta$assay_targets == 0))
+     {
+         return("solid")
+     }
+     else if (all((dta$cap_probes %in% 0:1) & (dta$assay_targets == 1)))
+     {
+         return("longdash")
+     }
+     else if (all((dta$cap_probes == 0) & (dta$assay_targets == 0)))
+     {
+         return("dotted")
+     }
+     else
+     {
```

```
+         warning("Unexpected style categories")
+         return("twodash")
+     }
+ }
> plot(test.out, graph.params)

Found 3 target nodes and 3 mut nodes
Finding target--mutation distances
Finding target--target distances
Finding mut--mut distances
Making subgraph
Converting to graphNEL

>
```



The function supplied to the `styleFunc` method takes only a single argument which represents a `data.frame` subset of the actual annotation for a given gene. The `shapeFunc` is similar but changes the node shapes. Also note the use of the `nodeCombinationFunc` method which allows additional logic to be applied after shapes, styles and colors have been assigned. By default it changes the node border back to solid if a given node is assigned a hit color.

2 Adding sample-specific metadata to existing annotation

In some cases, additional data exists for a given sample that does not need to be used as input to the prioritization algorithm. An example of this is gene expression and/or copy number differences. This metadata can be associated with the prioritization results in text summaries and plots similar to the fixed metadata. However, because it can vary between samples it is retrieved using its own query and is added after the creation of the `annotatedIGraph` object. Again we create at least one table and define a query to retrieve the data.

```
> #Note that we add the sample-specific variables into the same database as the rest of the
> #sample data as it will be retrieved through the same connection.
> #Again we will make a local copy of the database
>
> local.hitwalker.db <- file.path(getwd(), basename(hitwalker.db.path()))
> stopifnot(file.copy(from=hitwalker.db.path(), to=local.hitwalker.db))
> var.hit.db <- dbConnect("SQLite", local.hitwalker.db)
> symbol <- c('EPHA4','ZAK','PIK3CB','AC084035.1','CBL','FRK','FYN','MAP2K6','KHDRBS1',
+     'PTEN','PRKCE','MAPK14','IRS4','STAT5A','RAC1','DDR1','SRC','JAK3','FLT3')
> triangle <- c(1,1,1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,1)
> half_tri <- c(1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1)
> samp.annot <- data.frame(recon_name="08-00102", symbol=symbol, triangle=triangle,
+     half_tri=half_tri)
> dbWriteTable(var.hit.db, "samp_overlay", samp.annot, row.names=FALSE, overwrite=TRUE)

[1] TRUE

> #Although not necessary in this case, this would typically be a situation where it would
> #be necessary to use the parameters of this function such as sample.id.list.
> patientOverlay(test.parm) <- function(sample.id.list, param.obj)
+ {
+     query <- paste("SELECT symbol, triangle, half_tri FROM samp_overlay WHERE
+         recon_name = '", sample.id.list$recon, "'", sep="")
+ }
> test.out <- run.prioritization.patient(var.hit.db, "08-00102", anno.igraph, test.parm)
> #The new annotation is present after prioritization
> annotation(getGraph(test.out))[1:5,c("protein", "cap_probes", "assay_targets", "triangle",
+     "half_tri")]

          protein cap_probes assay_targets triangle half_tri
1 ENSP00000264033          1             0        0        0
2 ENSP00000410211          0             0        0        0
3 ENSP00000289153          1             1        1        0
4 ENSP00000397716          1             1        1        0
5 ENSP00000393085          1             1        1        0
```

```
> dbDisconnect(var.hit.db)

[1] TRUE

> stopifnot(file.remove(local.hitwalker.db))
>
```

We then define a function that allows node shapes/styles to be assigned based on the values of the new variables. The `shapeFunc` method takes a function that looks very similar to `styleFunc` but needs to return one of the following values: up_std, up_half, down_std, down_half, both_down, ellipse or rectangle. The HitWalker vignette shows what these look like in practice and demonstrates the low-level functions used for plotting.

```
> shapeFunc(graph.params) <- function(dta)
+ {
+     if (all(c("triangle", "half_tri") %in% colnames(dta)) == FALSE) {
+         return("ellipse")
+     }
+     else if (all(dta$triangle == 1 & dta$half_tri == 0))
+     {
+         return("down_std")
+     }
+     else if (all(dta$triangle == 0 & dta$half_tri == 1))
+     {
+         return("down_half")
+     }
+     else if (all(dta$triangle == 1 & dta$half_tri == 1))
+     {
+         return("both_down")
+     }
+     else if (all(dta$triangle == 0 & dta$half_tri == 0))
+     {
+         return("ellipse")
+     }
+     else
+     {
+         warning("Unexpected gene categories")
+         return("rectangle")
+     }
+ }
> plot(test.out, graph.params)

Found 3 target nodes and 3 mut nodes
Finding target--mutation distances
Finding target--target distances
Finding mut--mut distances
```

```
Making subgraph
Converting to graphNEL

>
```