# HitWalker: Setting up a Database Part 1: Creation of a PPI graph

Daniel Bottomly

November 29, 2012

## 1 Creating an annotation and graph database

In this vignette we will show how to create a database containing both the PPI graph structure and annotation to reliably use HitWalker as well as other approaches. First we will demonstrate how to parse a file from STRING using R. Similar approaches could be pursued using other scripting languages. In addition, if the computer has enough memory the file can be read in completely first. This vignette assumes that the users have installed the Streamer package from Bioconductor as well as RSQLite. Also it may take a while to run. Here we are working with the protein.links.detailed.v9.0.txt.gz file retrieved from http://string-db.org/newstring_cgi/show_download_page.pl. We parse it in chunks of 10,000 lines and only keep the human proteins (organism 9606).

```
> stopifnot(require(Streamer))
> stopifnot(require(RSQLite))
> options(stringsAsFactors=FALSE)
> #we will hold onto the header and use it at each iteration
> tab.head <- strsplit(readLines("protein.links.detailed.v9.0.txt.gz", n=1), "\\s")[[1]]
> rt.prod <- ReadTableProducer("protein.links.detailed.v9.0.txt.gz", nrows=10000,
+     header=FALSE, sep="")
> hu.tab <- data.frame()
> iter <- 0
> while(length(cur.tab <- yield(rt.prod)))
+ {
+     if (iter %% 10000 == 0)
+     {
+         print (iter)
+     }
+
+     #remove the header if its present in the data
+     if (all(as.character(unlist(cur.tab[1,])) == tab.head))
+     {
+         cur.tab <- cur.tab[-1,]
+     }
+
+     names(cur.tab) <- tab.head
+
+     cur.tab.hu <- cur.tab[grepl("^9606\\.", cur.tab$protein1) | grepl("^9606\\.",
+         cur.tab$protein2),]
+
+     if (nrow(cur.tab.hu) > 0)
+     {
+         if (nrow(hu.tab) == 0)
+         {
+             hu.tab <- cur.tab.hu
+         }
+         else
+         {
+             hu.tab <- rbind(hu.tab, cur.tab.hu)
+         }
+     }
+
+     iter <- iter + 1
+ }
> close(rt.prod)
> #check to ensure that all are ensembl human proteins
>
> stopifnot(sum(grepl("^9606\\.ENSP\\d+", hu.tab$protein1) & grepl("^9606\\.ENSP\\d+",
+     hu.tab$protein2)) == nrow(hu.tab))
> #ok, then strip off the organism IDs
>
> hu.tab$protein1 <- sub("9606\\.", "", hu.tab$protein1)
> hu.tab$protein2 <- sub("9606\\.", "", hu.tab$protein2)
> db.con <- dbConnect("SQLite", "ens_b57_string_v9_annot.db")
> dbWriteTable(db.con, "string_v9_db", hu.tab, row.names=FALSE)
>
```

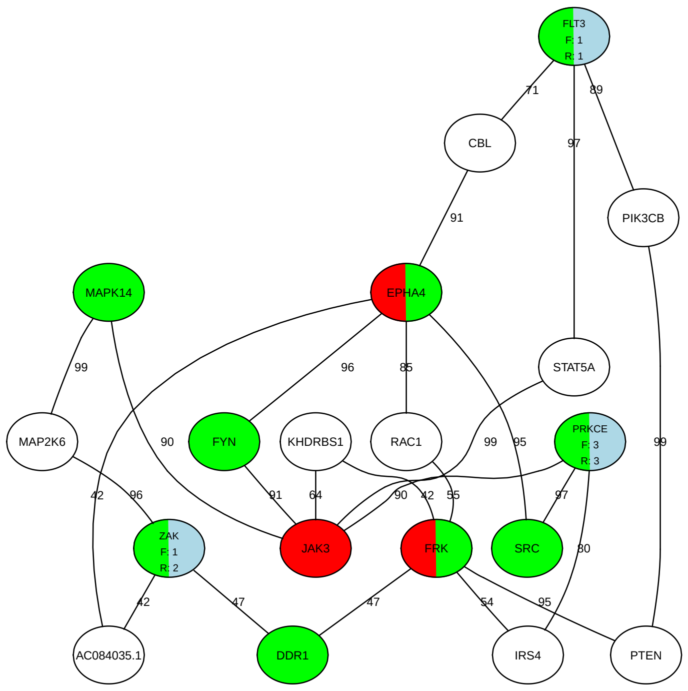
Next we add additional annotation to the database which at a minimum allows linking the Ensembl protein IDs to gene symbols or other summary ID. Additionally, information that is non-sample dependent can be incorporated at this point to potentially be linked to downstream visualization. The simplest way to create the annotation database is through parsing of GTF files from Ensembl. Here we demonstrate an approach for parsing a GTF file from Ensembl 57, though it could also be used for earlier/later versions assuming they have similar file structures. First, the GTF file should be downloaded from ftp://ftp.ensembl.org/pub/release-57/gtf/homo_sapiens/. Then the following steps can be taken:

```
> #we are reading the entire file in, though a Streamer based approach similar to above
> #could be used.
> ens.gtf <- read.delim("Homo_sapiens.GRCh37.57.gtf.gz", sep="\t", header=FALSE)
> #reduce the number of rows to just those potentially of interest
> sub.gtf <- subset(ens.gtf, V3 == "CDS")
> split.annots <- strsplit(sub.gtf$V9, ";")
> #retrieve the annotations that we need
> annot.mat <- sapply(split.annots, function(x)
+        {
+             ens.gene <- grep("gene_id", x)
+             ens.trans <- grep("transcript_id", x)
+             ens.name <- grep("gene_name", x)
+             ens.prot <- grep("protein_id", x)
+
+             stopifnot(length(ens.gene) == 1 && length(ens.trans) == 1 &&
+                 length(ens.name) == 1 && length(ens.prot) == 1)
+
+             return(x[c(ens.name, ens.gene, ens.trans, ens.prot)])
+        })
> annot.mat <- t(annot.mat)
> #strip out all the identifier strings and spaces
> anno.subs <- c("\\s+gene_name\\s+", "\\s+gene_id\\s+", "\\s+transcript_id\\s+",
+     "\\s+protein_id\\s+")
> for (i in 1:length(anno.subs))
+ {
+     annot.mat[,i] <- sub(anno.subs[i], "", annot.mat[,i])
+ }
> #rename and write to the database
> colnames(annot.mat) <- c("symbol", "gene", "transcript", "protein")
> annot.mat <- annot.mat[!duplicated(annot.mat),]
> dbWriteTable(db.con, "annotation",as.data.frame(annot.mat), row.names=FALSE)
> dbDisconnect(db.con)
>
```

Note that the above result could also be reached through the use of the MySQL tables provided by Ensembl. To round out this vignette, we re-create Figure 1 from Bottomly et al. submitted without the node border annotations (dashed and dotted lines).

```
> stopifnot(require(HitWalker))
> stopifnot(require(RSQLite))
> stopifnot(require(HitWalkerData))
> #first we adjust some of the queries to be used in forming the annotatedIGraph object
> test.parm <- HitWalker:::basePriorDbParams()
> #note that in order for get.protein.protein.graph to be used, both the protein1, protein2
> #and weight columns need to be specified.
> 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)
+ }
> annotQuery(test.parm) <- function(sample.id.list = NULL, param.obj)
+ {
+     query <- "SELECT * FROM annotation;"
+ }
> scoreSummaryFunc(test.parm) <- HitWalker:::default.score.summary.func
> variantFilterObj(test.parm) <- HitWalker:::defaultVariantFilter()
> #This is where the ens_b57_string_v9_annot.db lives in
> #HitWalkerData
> db.con <- dbConnect("SQLite", annot.db.path())
> anno.igraph <- get.protein.protein.graph(db.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

> dbDisconnect(db.con)

[1] TRUE

> 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

> 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

>
```


FLT3
F: 1
R: 1
71
89
97
CBL
PIK3CB
91
MAPK14
EPHA4
STAT5A
99
96
85
MAP2K6
FYN
KHDRBS1
RAC1
PRKCE
F: 3
R: 3
90
99
95
99
42
96
91
64
90
42
55
97
ZAK
F: 1
R: 2
JAK3
FRK
SRC
80
42
47
47
54
95
AC084035.1
DDR1
IRS4
PTEN