HitWalker: Case Study 1 – Glioblastoma

Daniel Bottomly

November 29, 2012

1 Introduction

In this vignette we demonstrate the use of HitWalker on data from the cancer genome atlas (TCGA) (http://cancergenome.nih.gov/) as accessed by the CGDS-R package (http://www.cbioportal.org/public-portal/). Although, to our knowledge, there are no publicly available datasets in TCGA with functional assay results it may be instructive to carry out a representative analysis. Here we endeavor to prioritize genes with mutations based on their proximity to genes with dysregulated expression in Glioblastoma. This data has been previously described (TCGA 2008) and is available through the CGDS-R package.

2 Retrieval and preparation of data

As we are also required to define a list of the genes before retrieval of the data, it seems appropriate to use a curated list of genes known to be involved in various types of cancer from the cancer gene census (Futreal et al. 2004). First we download the current list in Excel form (Table_1_full_2012-03-15.xls) from http://www.sanger.ac.uk/genetics/CGP/Census/. This list was converted to CSV through 'File->Save As' drop down menu From Excel. Note that the below code was only executed once and the results saved to `HitWalkerData`. It is accessed through `data(glio)` as in the second section.

```
> stopifnot(require(cgdsr))
> stopifnot(require(biomaRt))
> canc.genes <- read.csv("Table_1_full_2012-03-15.csv", header=TRUE, stringsAsFactors=FALSE)
> #connect to the server
> cgds.con <- CGDS("http://www.cbioportal.org/public-portal/")
> #first retrieve the mutation data. Can retrieve based on Entrez IDs per
> #http://www.cbioportal.org/public-portal/web_api.jsp (which cgdsr queries)
>
> gbm.muts <- getMutationData(cgds.con, caseList="gbm_tcga_pub_all",
+     geneticProfile="gbm_tcga_pub_mutations", genes=canc.genes$GeneID)
> #next we will retrieve data from the gene expression analysis (as an example)
>     #these appear to be Zscores
```

```
>
> gbm.expr <- getProfileData(cgds.con, caseList="gbm_tcga_pub_all",
+     geneticProfile="gbm_tcga_pub_mrna", genes=canc.genes$GeneID)
> #the other piece is to be able to reconcile the Entrez Gene IDs with protein IDs,
> #which in this case are Ensembl protein IDs.  One way of doing this is through
> #the biomaRt package which can provide the appropriate conversions
>
> ensembl = useMart("ensembl",dataset="hsapiens_gene_ensembl")
> entrez.to.ens <- getBM(mart=ensembl, attributes=c("hgnc_symbol", "entrezgene",
+     "ensembl_gene_id", "ensembl_transcript_id", "ensembl_peptide_id"),
+     filters="entrezgene", values=canc.genes$GeneID)
>
```

The first database table we will create lists the patient samples. This is most important for situations where multiple IDs are assigned to a given sample. An example would be separate IDs assigned by both the preparation laboratory and the sequencing or microarray facility. HitWalker has a simple function to allow partial matching of names over all the specified columns of a database. When setting up the `priorDbParams` object set the `id.table` slot to the name of the sample table, `possible.id.cols` slot to the columns to be searched, the `reconciled.id.col` to the name that should be attributed to the sample throughout the analyses and the `sample.id.col` slot to the integer value uniquely indicating the name (i.e. primary key in database jargon). For the TCGA data, something like this would suffice:

```
> stopifnot(require(reshape2))
> stopifnot(require(RSQLite))
> stopifnot(require(HitWalker))
> stopifnot(require(HitWalkerData))
> data(glio)
> common.samps <- intersect(gbm.muts$case_id, gsub("\\.", "-", rownames(gbm.expr)))
> samp.db <- data.frame(sample_id=1:length(common.samps), recon_name=common.samps,
+     alt_name=make.names(common.samps))
> #Note that alt_name is the conversion to R safe names
>
> head(samp.db)

  sample_id   recon_name     alt_name
1         1 TCGA-02-0083 TCGA.02.0083
2         2 TCGA-02-0010 TCGA.02.0010
3         3 TCGA-06-0176 TCGA.06.0176
4         4 TCGA-06-0133 TCGA.06.0133
5         5 TCGA-02-0028 TCGA.02.0028
6         6 TCGA-02-0089 TCGA.02.0089

> db.con <- dbConnect("SQLite", "TCGA_glio_mut_exp.db")
> dbWriteTable(db.con, "sample", samp.db, row.names=FALSE)
```

```
[1] TRUE
```

```
> #If we retrieve a base priorDbParams object we can modify it appropriately as
> #we go using the numerous replacement methods.
>
> base.parm <- HitWalker:::basePriorDbParams()
> idTable(base.parm) <- "sample"
> possibleIdCols(base.parm) <- c("recon_name", "alt_name")
> reconciledIdCol(base.parm) <- "recon_name"
> sampleIdCol(base.parm) <- "sample_id"
>
```

The next step is to put the variant data in the database. First we need to reconcile the GeneIDs with protein IDs. Our example is a suboptimal case as we can retrieve the GeneID and the mutation type, however additional work would be necessary to determine which transcript was mutated. For now, we can associate each mutation with all assigned proteins. This level of annotation is acceptable as STRING stores only a single translated protein per locus (according to their FAQ) so limiting the variant effect to only the transcript in question may be too restrictive. At this point we will also retrieve the bundled `annotatedIGraph` object and filter the proteins to include only the set present in the graph. This filtering step is not essential as it will be performed before the computations regardless.

```
> #Specify how the annotatedIGraph object is to be constructed
> matrixQuery(base.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(base.parm) <- function(sample.id.list = NULL, param.obj)
+ {
+     query <- "SELECT * FROM annotation;"
+ }
> graph.con <- dbConnect("SQLite", annot.db.path())
> graph.obj <- get.protein.protein.graph(graph.con, base.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(graph.con)
```

```
[1] TRUE
```

```
> kept.ens.prots <- subset(entrez.to.ens, ensembl_peptide_id %in%
+     annotation(graph.obj)$protein)
> gbm.muts.merge <- merge(gbm.muts, kept.ens.prots, by.x="entrez_gene_id",
+     by.y="entrezgene", all=FALSE, incomparables=NA, sort=FALSE)
> dbWriteTable(db.con, "variant", gbm.muts.merge, row.names=FALSE)

[1] TRUE

> queryParams(base.parm) <- list()
> variantQuery(base.parm) <- function(sample.id.list, param.obj)
+     {
+         return(paste("SELECT gene_symbol AS symbol, ensembl_peptide_id AS protein,
+             case_id, mutation_type, validation_status, amino_acid_change,
+             functional_impact_score FROM variant WHERE case_id = '",
+             sample.id.list$recon,"'", sep=""))
+     }
> variantFilterObj(base.parm) <- makeVariantFilter()
>
```

Note that we wrote the variant data to the database in a somewhat denormalized (expanded) form which may or may not be how it is stored in a production database. By supplying named elements to the queryParams method and a SQL query that uses them (by the retrieval version of the method applied to param.obj) and the sample information in the sample.id.list, it is relatively easy to specify custom queries to retrieve data per sample. Here we are subsetting the variant information by the value specified in the 'recon' element of the sample.id.list which happens to correspond to the 'recon_name' column of the sample table (See '?reconcile.sample.name'). We are also specifying no variant filter data. The variantFilter object is only useful if hard filters have been applied to the data resulting in labels (e.g. 'PASS' or 'HARD_TO_VALIDATE') and the variants need to be filtered and (re)labeled accordingly.

The last step in database preparation involves adding a table corresponding to the functional assay hits or gene expression values in this example. Again, we will convert the gene identifiers to protein IDs as well as reformat the data so it is more amendable to storage in a database. Additional metadata could also be added as is demonstrated in the Hitwalker_Add_Metadata vignette.

```
> gbm.expr.df <- melt(as.matrix(gbm.expr))
> colnames(gbm.expr.df) <- c("sample", "symbol", "Zscore")
> #Remove NAs and convert them to 0's
> gbm.expr.df$Zscore[is.na(gbm.expr.df$Zscore)] <- 0
> gbm.expr.df$symbol <- as.character(gbm.expr.df$symbol)
> merged.gbm.expr <- merge(gbm.expr.df, kept.ens.prots, by.x="symbol", by.y="hgnc_symbol",
+     all=FALSE, incomparables=NA, sort=FALSE)
> dbWriteTable(db.con, "gene_hit", merged.gbm.expr, row.names=FALSE)
```

```
[1] TRUE
```

```
> #Here we are defining genes with absolute Zscores greater than 2 to be hits
> hitQuery(base.parm) <- function(sample.id.list, param.obj)
+ {
+     return(paste("SELECT symbol, sample, Zscore,ensembl_peptide_id AS protein FROM
+         gene_hit WHERE Sample = '", make.names(sample.id.list$recon) ,
+         "' AND ABS(Zscore) > 2", sep=""))
+ }
> #We also modify this function to ensure that the scores are computed from the right colum
> #Will take the absolute values of the Zscores as directionality is not really taken into
> #account using the RWR framework
> scoreSummaryFunc(base.parm) <- function (dta, summary.col)
+ {
+     dta[, summary.col] <- abs(dta$Zscore)
+     return(dta)
+ }
>
```

Now that the database is setup we can produce summaries and plots for each sample. Here we will choose the top two samples based on having a high Zscore and mutations and will produce summaries and plots for each. In addition, this database is provided in HitWalkerData and the path to it can be found using tcga.glio.path().

```
> int.pats <- dbGetQuery(db.con, "SELECT sample, Zscore FROM gene_hit JOIN sample ON
+     (sample=alt_name) GROUP BY sample HAVING MAX(ABS(Zscore)) > 6")
> int.pats
```

```
        sample   Zscore
1 TCGA.06.0206 6.661944
2 TCGA.06.0213 6.518932
```

```
> samp.res.1 <- run.prioritization.patient(db.con, int.pats$sample[1], graph.obj, base.parm
> samp.res.2 <- run.prioritization.patient(db.con, int.pats$sample[2], graph.obj, base.parm
> dbDisconnect(db.con)
```

```
[1] TRUE
```

```
> summary(samp.res.1)[,c("protein", "symbol", "mutation_type", "amino_acid_change",
+     "type.rank")]
```

```
          protein symbol     mutation_type amino_acid_change type.rank
3 ENSP00000269305   TP53 Missense_Mutation             R273H         1
2 ENSP00000267163    RB1       Splice_Site       R830_splice         2
1 ENSP00000361021   PTEN   Frame_Shift_Del            T319fs         3
```

```
> summary(samp.res.2)[,c("protein", "symbol", "mutation_type", "amino_acid_change",
+     "type.rank")]
```

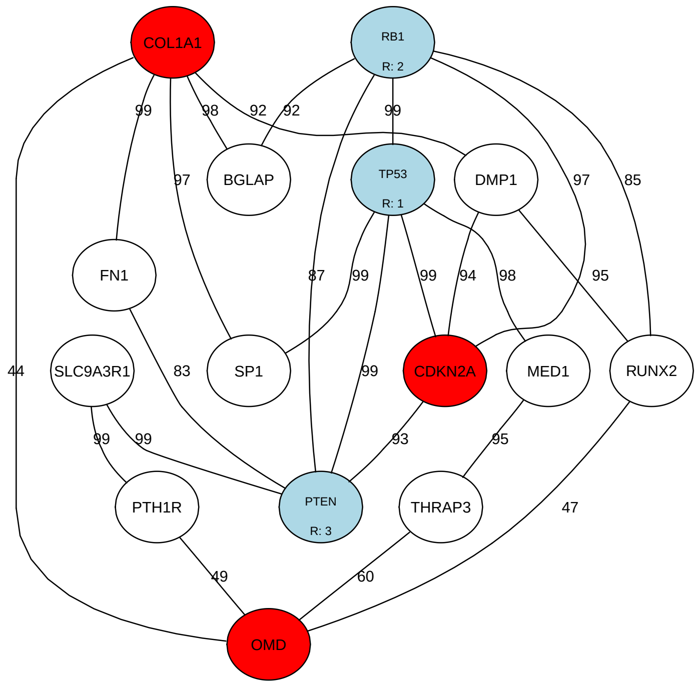
```
          protein symbol     mutation_type amino_acid_change type.rank
2 ENSP00000267163    RB1 Missense_Mutation             V654L         1
1 ENSP00000361021   PTEN Nonsense_Mutation             R335*         2
```

```
> #Make a basic plot but need to change the colors attributed to the gene hits
> #Also, as we just want a node label indicating ranks, we have to adjust the labelFuncList
> #method
>
> graph.params <- makeGraphDispParams(file.name = character())
> hitColors(graph.params) <- "red"
> labelFuncList(graph.params) <- list(HitWalker:::no.label, HitWalker:::no.label,
+     HitWalker:::rank.label)
>
```

```
> plot(samp.res.1, 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
```

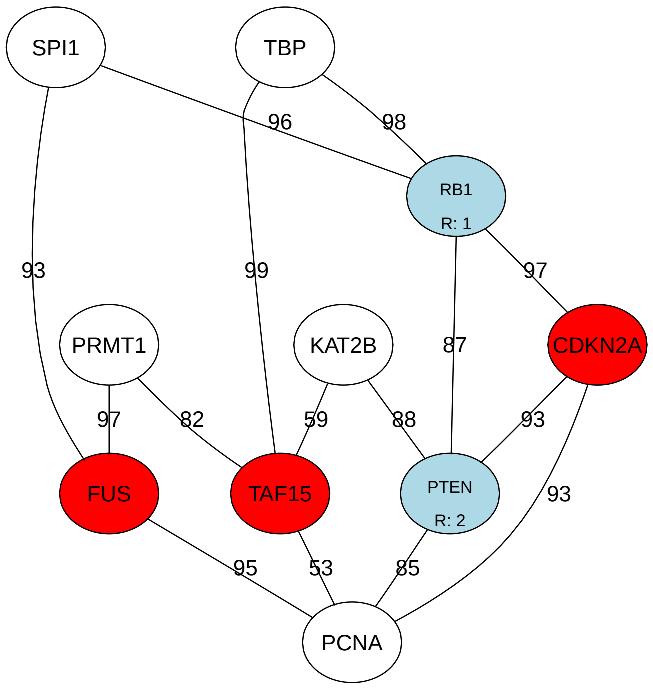
```
>
```



```
> plot(samp.res.2, graph.params)

Found 3 target nodes and 2 mut nodes
Finding target--mutation distances
Finding target--target distances
Finding mut--mut distances
Making subgraph
Converting to graphNEL

>
```



3 References

Futreal,P.A., Coin,L., et al. (2004) A census of human cancer genes. Nat Rev Cancer, 4, 177-183.

TCGA (2008) Comprehensive genomic characterization defines human glioblastoma genes and core pathways. Nature, 455, 1061-1068.