postgwas Vignette

Milan Hiersche, Frank Rühle

July 21, 2014

Contents

1 Introduction

This is a step-by step tutorial on how to use the primary functions in postgwas for visualization and analysis of a GWAS dataset. Initially, the package has to be loaded:

```
> library(postgwas)
```

Beside this vignette, there is additional documentation for the package which can be accessed by stating:

```
> help(package = "postgwas", help_type = "html")
```

Further, examples exist for most functions (here called for the regionalplot function in example):

```
> example(regionalplot)
```

Throughout this vignette, we will use the real-world GWAS datasets on human *height*, *BMI* and waist-hip ratio (*WHR*) provided by the GIANT consortium [2, 6, 1]. For the purpose of this tutorial, the datasests have been truncated, which is described in section 10. To show how postgwas can be run on non-human data, we will further employ the example dataset delivered with the GEMMA software package (mouse_lmm.assoc.txt, [7]).

All code examples given in this vignette can be re-executed by running

```
> require(tools)
> buildVignettes("postgwas")
```

which will effectively also re-generate this pdf document.

Finally, it should be mentioned that most functions in postgwas rely on annotation data that is normally downloaded from the web (e.g. biomarts). There is an alternative option to use local annotation data deposited in data frames. Setting the *use.buffer* argument in the call of a postgwas function to *TRUE* tries to use such local buffer data (or stores downloaded data in the appropriate data frames when not yet existing). The buffer variables are not accessed from the user's workspace but have to exist in the package's environment. Thus, getter and setter functions (*getPostgwasBuffer()*, *setPostgwasBuffer()*) have to be used to grant access to that data.

To avoid unnecessary web access, all examples presented here rely on such preloaded buffer data and have the corresponding *use.buffer* argument set. Here we load the annotation data from a workspace image file (contains a list of buffer variables named bufferHS) and supply it to postgwas :

```
> load("bufferHS.RData")
> setPostgwasBuffer(bufferHS)
```

More details on the buffer data concept are given in section 9.

2 Manhattan plots

The first thing to do with a GWAS dataset usually is creating a manhattan plot, getting an overview of the data. Figure 1 show such a plot for the *WHR* dataset generated with the default settings of the manhattanplot function:

```
> manhattanplot(
+     gwas.dataset = "whrTrunc.txt.remapped.xz",
+     highlight.text = NULL,
+     use.buffer = TRUE,
+     toFile = NULL
+ )
```

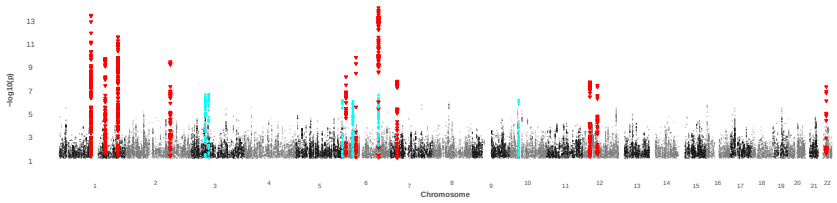

Figure 1: A simple manhattan plot for the *WHR* dataset

As can be seen from the figure, by default all loci beyond a certain p-value threshold are highlighted. There can be multiple such thresholds, and choice of colors can be controlled by the *highlight.color* argument.
It is further possible to add the identifier of the lead SNPs (lowest p-value) for each highlighted locus by setting the argument *highlight.text = "SNP"*. P-value thresholds may be changed by the *highlight.logp* argument, and the *highlight.cex* argument acts as a multiplier for the font and symbol size of the highlighted plot elements:

```
> manhattanplot(
+     "whrTrunc.txt.remapped.xz",
+     highlight.logp = 7.3,
+     highlight.text = "SNP",
+     highlight.cex = 0.8,
+     highlight.col = "cyan",
+     highlight.fontface = "plain",
+     use.buffer = TRUE,
+     toFile = NULL
+ )
```

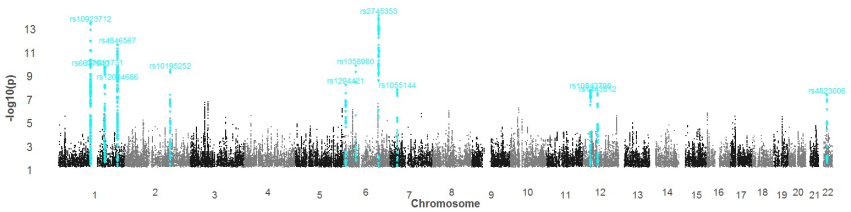

Figure 2: Annotated manhattan plot for the *WHR* dataset

The syntax of all arguments used is described in the help pages which can be browsed as shown in the introduction. As we can see from figure 2, for some loci of chromosome 1, multiple lead

SNPs were identified per locus (e.g. rs10923712 and rs667760 at the leftmost locus). This is because highlighting and identification of the lead SNP occurs within a prespecified window size. Increasing the corresponding *highlight.win* argument will join the two adjacent loci to a single one. This is done in the following code fragment. Additionally, we annotate names of surrounding genes to peak SNPs instead of the SNP identifier itself. The example below demonstrates the usage of all these arguments. It further uses different font sizes and styles for each p-value threshold, with the resulting plot shown in figure 3:

```
> manhattanplot(
+     "whrTrunc.txt.remapped.xz",
+     highlight.logp = c(6, 8, 10),
+     highlight.fontface = c("italic", "italic", "bold"),
+     highlight.cex = c(0.6, 0.6, 1),
+     highlight.text = c("SNP", "genes", "genes"),
+     highlight.win = c(75000, 200000, 200000),
+     ticks.y = TRUE,
+     plot.title = "WHR dataset plus gene annotation",
+     use.buffer = TRUE,
+     reduce.dataset = 5,
+     toFile = NULL
+ )
```

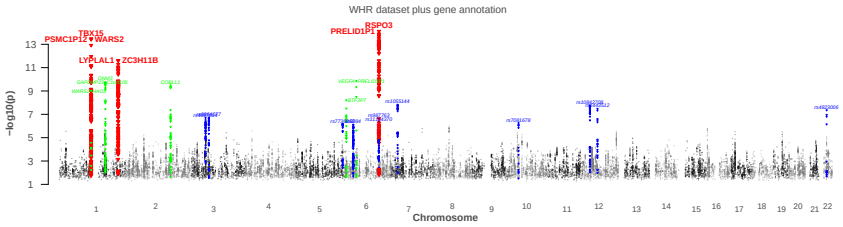

Figure 3: Manhattan plot for the *WHR* dataset, including gene annotation and multiple p-value thresholds

Finally, it might be of interest that changing the aspect ratio of the plot (and width, height, ...) is possible by plotting to a diffferent device using the standard R functions (e.g. using the *png* function, see the documentation there). Doing so, it is important to know that the text size is independent of the width / height ratio, but other graphical elements are not. This means, increasing the width / height of your device will scale up points, triangles etc. in the plot, but not the text. The text size has to be adjusted by setting the pointsize argument to the device functions (there is also a highlight.cex argument to the manhattanplot function, but this affects text and point size of the highlighted regions only). This way, the relation between text and other graphical elements can be preserved. Anyways, normally the default aspect ratio will be well suited (except for a really large number of significant loci which might require a larger width of the plot).

3 Regionalplots

The manhattan plots reveal rs2745353, rs10923712 and rs4846567 as significantly associated SNPs (among others). We take these as exemplary selection to demonstrate how regional association plots can be constructed for these loci. We include the p-value graphs of two datasets in the regional plots, supplying the *height* and *WHR* GWAS result files in the *gwas.datasets* argument. Generally, showing association graphs for multiple studies in a single plot can makes sense for phenotypically related traits or replication studies, for example.

```
> snps <- data.frame(SNP = c("rs2745353", "rs10923712", "rs4846567"))
> regionalplot(
+     snps = snps,
+     gwas.datasets = c("heightTrunc.txt.remapped.xz", "whrTrunc.txt.remapped.xz"),
+     max.logp = 15,
+     ld.options = NULL,
+     out.format = NULL,
+     use.buffer = TRUE
+ )
```

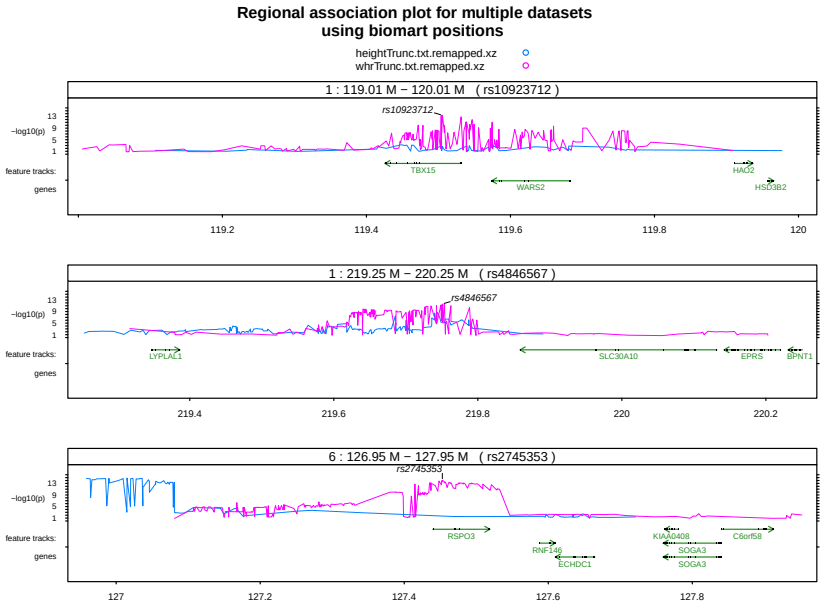

Figure 4: Regional association plot for three SNPs of the *WHR* dataset in comparison with the *height* dataset (color code of p-value graphs is shown in the legend above the plots). Genes are shown as green arrows with their direction according to the strand, and exons shown as black rectangles on the genes.

The *regionalplot* function needs a vector of SNPs (wrapped in a data frame), denoting the regions to plot (one region for each SNP, centered around that SNP - the region width is defined by a window.size argument which is 1 MB by default and omitted here). The second argument is the file name of one or more GWAS result files. Several different file formats are recognized (see help pages). By default, an LD triangle track is also added to the plot, which is decativated here by setting the corresponding argument *ld.options* to NULL. The details of all these arguments are described in the corresponding help pages.

Often, it makes sense to plot many more loci, like the top 50, at once (see section 8 for an example). By default, a fully searchable pdf file is produced that makes it then easy to browse and retrieve certain loci or genes in such a summary file.

It is further possible to create wider or higher plots as shown in the next example, figure 5. This can be done using the *out.format* argument. In the next example we achieve this by just reducing the number of panels (regions plotted) per page, which automatically upscales each region plot to fit the page size. Finer control can be achieved by plotting to a custom device and changing the page dimensions, which is further described in the help page of the regionalplot function. In general, it is recommended to deviate not too much from the default aspect ratio of 4:3, because not all plot elements scale entirely proportional. Additionally, we produce a jpeg file instead of plotting to the screen (*out.format* = *NULL* in the previous example).

Lastly, it is sometimes desireable to define a maximum value for the y-axis of the p-value graph, to prevent a single very low p-value making the difference between moderate p-values undistinguishable. This is not the case here, but anyways, the *max.logp* argument is added in the following example for demonstration purposes:

```
> regionalplot(
+   snps = data.frame(SNP = "rs4846567"),
+   gwas.datasets = "whrTrunc.txt.remapped.xz",
+   ld.options = list(gts.source = 2),
+   out.format = list(file = "pdf", panels.per.page = 3),
+   max.logp = 15,
+   use.buffer = TRUE
+ )
```

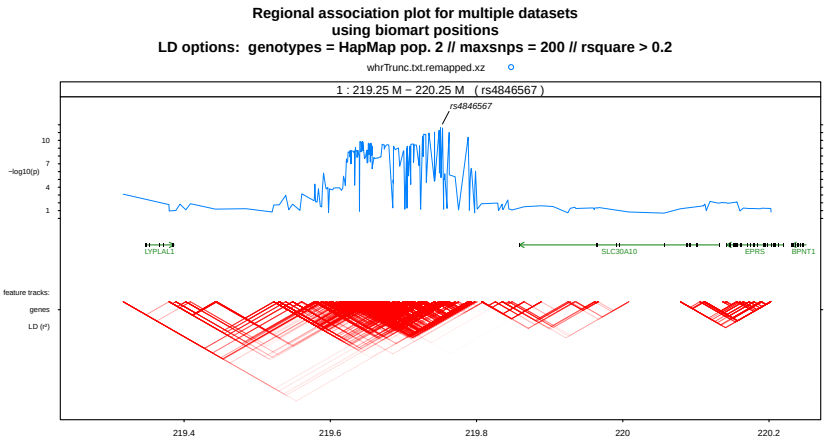

Figure 5: Regional association plot with LD triangles between GWAS SNPs.

Further, in the resulting figure 5 an LD plot has been added using genotype information from the HapMap CEU population (by specifying a population identifier via *gts.source* = *2*). It is alternatively possible (and recommended) to supply custom genotype data to the *ld.options* argument (which can be gwaa/phe or ped/map files, see the *regionalplot* help pages for details). This will extract the required genotypes and usually result in a more accurate LD structure with regard to the present GWAS cohort.

The next and last example dealing with the regionalplot function demonstrates the usage of a resequencing datafile for creation of a rare variant data track. The supplied file has to be in

vcf[1] format, and some additional prerequisites have to be met regarding the file format:

- **Sorted:** The vcf file has to be sorted by chromosome and base position, which can be done using vcftools[2] and the vcf-sort script on the operating system command line.
- **Tabix index:** Because vcf files can be very large, a tabix index has to exist for quick data retrieval. If not yet existent, it can be build in R using the Rsamtools package:

```
> if(library("Rsamtools", logical = TRUE)) {
+   bgzip("reseq.vcf", overwrite = TRUE)
+   indexTabix("reseq.vcf.bgz", format = "vcf4")
+ }
```

- **Region coverage:** The vcf file should contain data for all plotted loci (in fact, there can be regions without data but the corresponding chromosome has to exist in the vcf file - otherwise lookups in the tabix index file will return an error.).
- **INFO column:** The vcf file (which is basically tab-delimited) should contain an INFO column. According to the vcf format definitions, the INFO column can have subcolumns (which are separated by a semicolon and preceded by an identifier, this could look like MYSUBCOL=values;MYSUBCOL2=morevalues;). There has to be such a subcolumn named AF listing the allele frequency of the SNP described in that row. Within the regionalplot function, using regular expressions, data from the INFO column can be further used to filter and display only certain variants. It is recommended to run snpEFF[3] on the vcf file to add functional annotation on all SNPs in an EFF subcolumn of the INFO column which can then be highlighted or filtered using the regular expression syntax. This is also shown in the next example.
- **Base positions:** Lastly, sometimes the base position and chromosome annotation between vcf files and the genes / SNPs of the p-value graph differ. Data from the p-value graph uses base positions and chromosome names from the ENSEMBL biomart by default (when *plot.genes = TRUE* and biomart.config unchanged). It is assumed that data from the vcf file maps to these positions (which is normally the current genome assembly) as well. If this is not the case, there are options to accordingly remap data from the vcf file to biomart positions. For base positions it is sufficient to include an option *remap.positions* which will try to convert base positions from the vcf file to biomart positions. This works for variants with known SNP identifiers (rs IDs). For de novo variants, positions will be imputed by applying the offset of the closest SNP that has been mapped to a biomart position. This best guess will in the most cases be valid (unless there has been an insertion or deletion between the unkown and next mapped variant).

  The chromosome names used have to be handled separately, because extraction of variants from the vcf file occurs before data remapping. When the chromsome names given in the vcf file do not match those downloaded from biomart used, a two-column data frame can be supplied as an argument that maps the biomart chromosome names to those used in the vcf file. Such an example data frame is shown in the following code fragment:

[1] vcf = variant call format
[2] vcftools.sourceforge.net
[3] snpeff.sourceforge.net

```
> chrmap <- data.frame(CHR = 1:23, CHR.VCF = paste("chr", 1:23, sep = ""))
> head(chrmap)

  CHR CHR.VCF
1   1    chr1
2   2    chr2
3   3    chr3
4   4    chr4
5   5    chr5
6   6    chr6
```

Anyways, generally it is recommended to use a vcf file that aligns to the default biomart assembly, which makes these steps dispensable and does not cause data load during remapping.

The following example illustrates the usage of a vcf file that needs chromosome name remapping for variant display. Beside using the corresponding *var.options* argument, in comparison to the previous plot we have further zoomed in by decreasing the *window.size* argument, and again added an LD plot via the *ld.options* argument. Here we restricted the pairwise LD calulcation to 100 SNPs (*max.snps.per.window* subargument), evenly selected over the region, which is usually sufficient to get a proper representation of the LD block structure and saves some calculation time.

The *var.options* argument of the regionalplot function takes a list of argument values, of which one specifies the vcf file to use, which is again a list containing the filename, the chromosome name map and a binary switch whether to remap the base positions to biomart positions (which is not necessary here because the vcf file uses the currenty assembly). Beside the *vcf* subargument, several options are available to filter and highlight specific types of variants. These options are described in detail in the help pages. Here we use the *vcf.info.colorize* argument to extract only variants with SYNONYMOUS and NON_SYNONYMOUS tags in the INFO column. Note that the order in the vcf.info.color argument matters: expressions listed first will be preceded by latter ones, so it is possible to include more specific regular expressions after generic expressions (here, first all variants containing SYNONYMOUS will be colorized, then those matching NON_SYNONYMOUS).

```
> regionalplot(
+   snps = data.frame(SNP = "rs10923712"),
+   gwas.datasets = "whrTrunc.txt.remapped.xz",
+   window.size = 600000,
+   ld.options = list(gts.source = 2, max.snps.per.window = 100),
+   out.format = list(file = "pdf", panels.per.page = 3),
+   max.logp = 15,
+   var.options = list(
+       vcf = list(
+           file = "reseq.vcf.bgz",
+           remap.positions = FALSE,
+           chrom.map = chrmap
+       ),
+       vcf.info.colorize = c(EFF = "SYNONYMOUS", EFF = "NON_SYNONYMOUS")
+   ),
+   use.buffer = TRUE
+ )
```

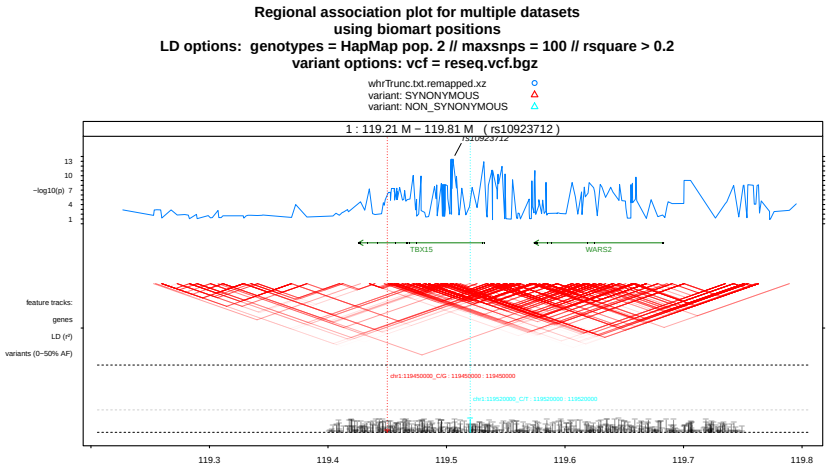

Figure 6: Regional association plot for rs10923712, including hypothetical rare variants (dummy data).

The variant track by default has two components, separated by a light dashed line. The lower part contains a histogram of the allele frequencies of all variants from the vcf file, and the upper part lists the position of all highlighted variants as text. An extended colorized calibration line is drawn through the plot to be able to compare the variant position with genes, LD and associated SNPs. The color code is listed in the legend at the top of all plots showing appropriately colored triangles with the corresponding filter key.

4 SNP to Gene Mapping and Gene-Aggregate p-Values

For associated SNP variants in GWAS, it is not obvious how these map to putatively causal genes. It has been demonstrated that many associated SNPs likely affect the closest gene (e.g. approximately at least 62% = 13 out of 21 loci as roughly estimated in [2]), but still, for certain constellations sophisticated methods for SNP to gene mapping might be desirable. Secondly, in gene-based association analysis like gene set enrichment analysis (GSEA) or similar approches, a correction for the number of SNPs tested per gene should be employed to derive gene-based p-values (depending on the LD structure, there can be many independently tested SNPs per gene). Both steps can be performed within postgwas and are described in the following two paragraphs.

SNP to gene mapping The *snp2gene* function provided with this package allows SNP to be mapped to genes based on proximity (e.g. assign the closest, or n-th closest gene up- or downstream etc) or by linkage disequilibrium of the SNPs in question with SNPs that reside within candidate genes in proximity. We will use the SNPs from the last two regionalplots for our example mapping. This makes it possible to compare the results of the automatic mapping with the gene structure in the regionalplots.

```
> snps <- data.frame(SNP = c("rs10923712", "rs4846567"))
> snp2gene.prox(snps, use.buffer = TRUE)
```

	geneid	genename	start	end	CHR	SNP	BP	direction
15	6913	TBX15	119425669	119532179	1	rs10923712	119505434	cover
5		PSMC1P12	119156955	119158269	1	rs10923712	119505434	up
4	127018	LYPLAL1	219347186	219386207	1	rs4846567	219750717	up
10		RIMKLBP2	219373258	219373909	1	rs4846567	219750717	up
13	10352	WARS2	119573839	119683294	1	rs10923712	119505434	down
7		ZC3H11B	219782859	219786462	1	rs4846567	219750717	down
17		WARS2-IT1	119590028	119607408	1	rs10923712	119505434	down
12		RBMX2P3	119627531	119628577	1	rs10923712	119505434	down
1		RPS3AP12	119669162	119669914	1	rs10923712	119505434	down

Table 1: SNP to gene annotation by proximity. The closest gene in each direction (covering, up- and downstream genes, including overlapping genes) is given for each query SNP.

In comparison to annotation by proximity, the annotation by LD requires much more main memory and computational power. It is highly recommended to use genotype files (preferably gwaa/phe or alternatively ped/map which can also be compressed) to reduce data load. Parallelization can be activated using the *cores* argument, but consumes n-fold memory according to the number of cores used.

```
> snp2gene.LD(snps, use.buffer = TRUE, gts.source = "s2g.ped.xz")
```

In the previous regionalplots, we can see that the genes WARS2 and TBX15 are in the same LD block as rs10923712. Correspondingly, they have high ld.max and ld.mean scores in the snp2gene table, whereas rs4846567 does not have genes in LD, except for LYPLAL1 with very weak LD which is correctly reflected in the snp2gene table. Also note that only established genes with a symbol and entrez ID are contained in the regionalplot, where the snp2gene annotation contains additionally genes without ID by default.

	SNP	CHR	BP	geneid	genename	start	end	ld.max	ld.mean	ld.sdev
2	rs10923712	1	119505434	6913	TBX15	119425669	119532179	0.5715	0.0702	0.1111
3	rs10923712	1	119505434	10352	WARS2	119573839	119683294	0.6832	0.0915	0.1109
7	rs10923712	1	119505434	51179	HAO2	119911402	119936753	0.0175	0.0019	0.0038
9	rs10923712	1	119505434	3284	HSD3B2	119957554	119965658	0.0029	0.0007	0.0009
12	rs4846567	1	219750717	127018	LYPLAL1	219347186	219386207	0.1169	0.0291	0.0346
15	rs4846567	1	219750717	55532	SLC30A10	219858769	220131989	0.0297	0.0058	0.0075
17	rs4846567	1	219750717	2058	EPRS	220141943	220220000	0.0056	0.0013	0.0014
18	rs4846567	1	219750717	10380	BPNT1	220230824	220263804	0.0035	0.0005	0.0010

Table 2: SNP to gene annotation by LD (showing only genes with entrez ID). The column ld.max lists the highest r^2 value between the query SNP and all SNPs within the gene in question, ld.mean the mean of all r^2 values of the query SNP with SNPs in that gene and ld.sdev its standard deviation, respectively.

Gene-based association p-values Genes with varying sizes will often have different number of SNPs assigned. This leads to different number of tests performed for each gene with varying rates of false positive findings with regard to a global significance threshold, thus either the threshold has to be adjusted for each gene/locus, or a method is needed to adjust the p-values. Two such methods, GATES and SpD proposed by Li et. al and D. Nyholt, respectively [3, 4] that estimate the number of independent tests per gene have been made available within postgwas. After determining the number of independent tests, GATES applies the Simes multiple testing correction on the derived number of independent SNPs to obtain a gene-representative p-value, and for SpD, the Sîdàk correction is employed. Both algorithms are wrapped in the gene2p function and add a column of gene-wise p-values to our data frame of snp to gene annotations. Note: Because computations are exhaustive, it is recommended to not use all SNPs from a large chip (say one million SNPs) to compute gene-wise p-values (although it is possible, but will take maybe one or two days and much memory). In section 6 an example is shown that suggests a pre-filtering procedure cutting down the number of SNPs. Otherwise, calculations can be split up per chromosome. Here we set up a small example demonstrating the functionality for all genes on chromsome 22 of the whr dataset:

```
> whr <- read.table("whrTrunc.txt.remapped.xz", header = T)
> whr22 <- whr[whr$CHR == "22", ]
```

Here, we also restrict to intragenic SNPs for aggregate p-value calculation. To do that, we apply *snp2gene* with the option *level = 0* to annotate just the closest gene, and then select intragenic SNPs by extracting only SNPs with the value 'cover' in the *direction* column:

```
> whr22 <- snp2gene.prox(whr22, level = 0, use.buffer = TRUE)
> whr22 <- whr22[whr22$direction == "cover", ]
```

Now we can use the *gene2p* function to obtain gene-wise p-values. We supply genotype data as custom files as usual, although it is again possible to use HapMap downloads (in this case, downloaded genotype data will be dumped to files for occasional re-use):

```
> whr22.gp <- gene2p(
+     whr22,
+     method = GATES,
+     gts.source = "whr22.ped.xz"
+ )
```

Gene-wise p-value calculation works the same way when using the *SpD* method. Comparing the available *GATES* and *SpD* methods, we see that the mean difference between the gene based

	geneid	genename	start	end	CHR	SNP	P	BP	direction	gene.p
1	162	AP1B1	29723669	29819168	22	rs2267134	0.062	29766966	cover	0.07160
2	162	AP1B1	29723669	29819168	22	rs2283858	0.080	29807855	cover	0.07160
3	6527	SLC5A4	32614465	32651328	22	rs2068209	0.180	32626908	cover	0.18000
4	7494	XBP1	29190543	29196585	22	rs2239815	0.021	29192670	cover	0.02100
5	8224	SYN3	32908539	33454358	22	rs16990642	0.059	32925526	cover	0.00835
6	8224	SYN3	32908539	33454358	22	rs4274683	0.061	32958464	cover	0.00835

Table 3: SNP to gene annotation including a column *gene.p* for gene-wise aggregated p-values. Only the first six lines of the original result table are shown. Calculations were based on intragenic SNPs on chromosome 22 of the *WHR* dataset.

p-values of both methods is quite low in our test dataset, where SpD seems to yield overall slightly less conservative p-values than GATES:

```
> whr22.gp.SpD <- gene2p(
+     whr22,
+     method = SpD,
+     gts.source = "whr22.ped.xz"
+ )
> whr22.gp.SpD <- whr22.gp.SpD[order(whr22.gp.SpD$SNP), ]
> whr22.gp  <- whr22.gp[order(whr22.gp.SpD$SNP), ]

> mean(log10(whr22.gp$gene.p) - log10(whr22.gp.SpD$gene.p), na.rm = TRUE)

[1] -0.1748606
```

The major difference between GATES and SpD is that GATES accounts for correlation between p-values of SNPs of a gene, while SpD relies exclusively on LD information to determine the number of independent tests. The p-value correlation has been determined in GATES by fitting a polynomial function to the eigenvalues of the LD matrix that accounts for varying sample sizes of case-control studies. So for GWAS data derived from a case-control study, GATES seems to be well suited, while for other study designs, SpD might be a good alternative.

5 GO enrichment analysis (via topGO)

Having obtained gene-wise p-values, it is possible to calculate a GO term enrichment within the GWAS dataset. For that purpose, the topGO package, which is originally designed for gene expression data, can be used. To ease the data handling, postgwas contains a small wrapper function named *gwasGOenrich*. It will run the GSEA (Gene-Set Enrichment Analysis) - like Kolmogorov-Smirnov statistic of topGO:

```
> enrich.res <- gwasGOenrich(gwas = whr22.gp, ontology = "CC", pruneTermsBySize = 8)
```

	go_id	P	Term
7	GO:0043231	0.11	intracellular membrane-bounded organelle
9	GO:0044464	0.22	cell part
8	GO:0044424	0.90	intracellular part
1	GO:0005575	1.00	cellular_component
2	GO:0005622	1.00	intracellular
3	GO:0005623	1.00	cell
4	GO:0043226	1.00	organelle
5	GO:0043227	1.00	membrane-bounded organelle
6	GO:0043229	1.00	intracellular organelle

Table 4: Result of applying the gene set enrichment analysis functions of the topGO package to the *WHR* chromosome 22 dataset , using the *gwasGOenrich* function.

The function prints some stats on the source data as calculated by topGO (not shown) and returns the enrichment statistics for all GO terms, which are listed in table 4. Further, topGO features a plot of the GO subgraph for the most significant terms, which is stored in a pdf file by *gwasGOenrich* (not shown here but looks like in figure 11).

The function arguments are self-explaining except for *pruneTermsBySize*, which will ignore all terms with equal or less genes annotated. A value between 5 and 10 is recommendable (also suggested by the original authors of the topGO package in their vignette). Further arguments can be looked up in the help pages.

All p-values are given without correction for multiple testing.

6 Network analysis

Basic principles Postgwas contains a *gwas2network* function that helps interpreting GWAS result data under consideration of the functional relationships between associated genes. It needs two basic inputs: First, a network of binary relationships between genes defined as two-column data frame. This could be protein interaction data, co-occurence of genes in a pathway or other user defined gene-gene relationships, with one of two related genes in each column. Appopriate functions for retrieval of such data from public databases are contained (e.g. *getInteractions.path()*). Secondly, a data frame of SNP to gene mappings including a column of p-values is needed, e.g. obtained as outlined in the previous section 4.

Running the *gwas2network* function on these data will produce a graphical view of the data as a network. The edges of the network are weighted by the mutual p-value of the connected vertices (alternative weighting schemes are possible). Graphically, the edge weight and vertex p-value is represented by transparency effects and a corresponding vertex size. Colorization of the vertices can be set to annotated GO terms, either customly selected or by overrepresentation analysis in the network.

For a large network (e.g. including genes beyond genomwide significant hits), it is useful to decompose the network into modules respectively 'gene communities'. This can be done using the *max.communities* argument, which runs the spinglass graph partitioning algorithm [5] of the igraph package on the network. Each vertex will then be assigned to one community, based on the number of edges shared with the community and the edge weights. This will put densely connected genes with preferably low p-values into the same module. It is assumed that modules with a larger number of well associated genes are more important, thus a 'module score' is calculated that measures the deviation of the p-values of the modules vertices from the distribution of p-values in the entire network (wilcoxon rank sum test).

Visualizing functional relationships This paragraph deals with the visualization of genomewide significant loci in a network, given that it is small enough so that a decomposition into gene communities is not necessary.

Again, we use the *WHR* dataset as an example. We obtain the lead SNPs (using the *removeNeighborSNPs* function) of all loci beyond $\alpha = 10^{-6}$ and annotate the closest gene (using *snp2gene*):

```
> snpsW <- removeNeighborSnps(whr[whr$P < 10^-6, ])
> genesW.closest <- snp2gene.prox(snpsW, level = 0, use.buffer = TRUE)
```

Next, we compute the similarity of the GO term architecture for all associated genes (using the *getInteractions.GO* function which relies on the *GOSim* package) as a measure for functional relationship:

```
> network.data <- getInteractions.GO(genesW.closest$geneid, similarity = "hausdorff")
```

Finally, we can visualize our data using the *gwas2network* function:

```
> network.igraph <- gwas2network(
+   gwas.mapped.genes = genesW.closest,
+   network = network.data,
+   max.communities = 0,
+   vertexcolor.GO.regex = list(red = "metaboli"),
+   min.transparency = 0.4,
+   max.transparency = 1,
```

```
+   use.buffer = TRUE
+ )
```

We have used the option (*vertexcolor.GO.regex*) to colorize all genes with specific annotated GO terms. By using this argument, it is possible to define a regex[4] in combination with a color, which will paint a gene in that color when a GO term description matches the regex. As can be seen in the next example, it is also possible to do an automatic colorization by GO term overrepresentation analysis using the argument *vertexcolor.GO.overrep* (or completely deactivate the colorization by setting that argument to NULL), which is recommended when a hypothesis-free representation of the network is desired.

The processed network will be deposited in pdf files in the current directory (as long as the argument *file.verbosity* is greater 0, which has a default of 2). In addition, the function returns the resulting network as an igraph object. This can either be subject to further custom analysis and / or be plotted with the built-in *gwas2network.plot* function, which is shown in the next code fragment. The dedicated plotting allows control over some more parameters like the file format (device used), resolution or point / fontsize if desired. Also, the layout (vertex positions) can be customized in the plotting function (remark: there is also an argument *custom.layout* available in the *gwas2network* function that allows dragging the vertices to a custom position). In the following code fragment, we save the graph as a jpeg image with increased pointsize of the font (which results in larger labels). Finally, it is also possible to store the graph and GO term overrepresentation data (if applied) in text files by setting file.verbosity >= 4, so that any kind of downstream analysis or interpretation should be possible.

```
> gwas2network.plot(
+     network.igraph,
+     filename = "exampleNet.jpeg",
+     device = jpeg, res = 100, pointsize = 16
+ )
```

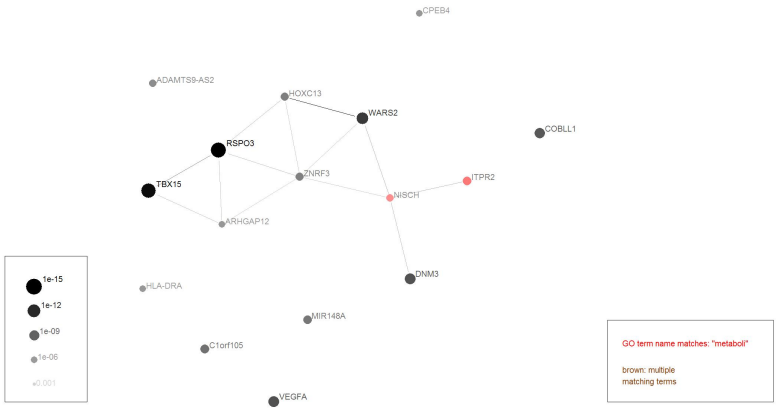

Figure 7: Network of GO term architecture similarity between genes of the *WHR* dataset. Colorization corresponds to GO term annotation and vertex size / transparency corresponds to the assigned p-values (see legend boxes).

[4]regex = regular expression, see for example *help("regex")* in R

The resulting network is shown in figure 7. Genes have by default a size and transparency level (which can be controlled by the *min.transparency* and *max.transparency* arguments) according to their assigned p-value. The source GWAS data *genesW* does not contain a *gene.p* annotation colum, thus the SNP-wise p-values (column *P*) are used to determine these attributes. By default, when multiple p-values are available for one gene, the smallest will be selected as representative.

For interpretation, we see that six genes are (weakly) related by a common GO term architecture. Among them are NISCH and ITPR2, which in addition have a GO term with the keyword 'metaboli' assigned (a manual lookup confirms that NISCH is associated to the GO term 'glucose metabolic process' and ITPR2 to 'energy reserve metabolic process'). Therefore, this module could be of increased interest for further research, especially with regard to a shared functionality or common action on a specific molecular trait.

Nevertheless, it should be kept in mind that such observations just represent a small fragment of information that can be drawn from a GWAs dataset, and are also prone to be biased by the investigator's view on the data. Thus, application of additional tools that perform enrichment and network based analyses[5] are recommended to get a comprehensive view on the data.

So far, we have only considered the gene in closest proximity of each associated SNP as the most likely causal gene for that locus. The next example generates a network considering the closest gene as well as those that are in LD with the peak associated SNP, providing probably a more accurate gene annotation:

```
> genesW.LD <- snp2gene.LD(snpsW, use.buffer = TRUE, gts.source = "whr.ped.xz")
> genesW <- rbind(
+   genesW.closest[, c("SNP", "P", "geneid")],
+   genesW.LD[genesW.LD$ld.max > 0.6 | genesW.LD$ld.mean > 0.1, c("SNP", "P", "geneid")]
+ )

> network.data <- getInteractions.GO(genesW$geneid, similarity = "hausdorff")

> network.igraph <- gwas2network(
+   gwas.mapped.genes = genesW,
+   network = network.data,
+   max.communities = 0,
+   vertexcolor.GO.regex = list(
+                             red = "metaboli",
+                             yellow = "(LDL|HDL|lipoprotein)",
+                             blue = "(insulin|diabet|glucose)"
+                           ),
+   min.transparency = 0.4,
+   max.transparency = 1,
+   use.buffer = TRUE
+ )
```

[5] for example MAGENTA, PANTHER, DAPPLE, GRAIL, etc.

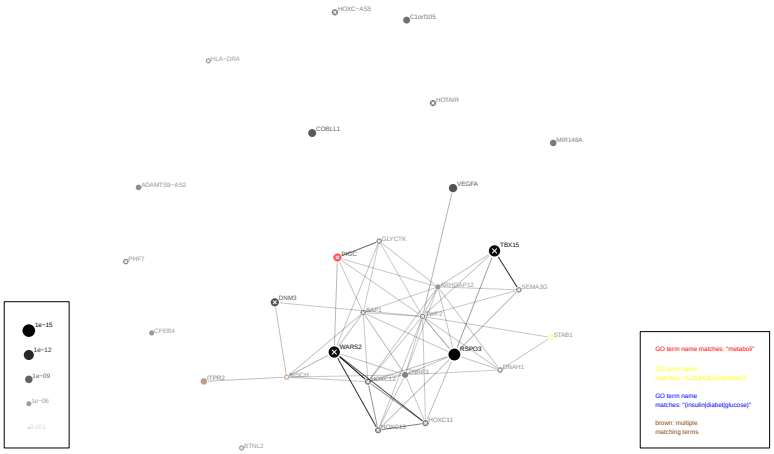

Figure 8: Gene network based on GO term architecture similarity between genes of the *WHR* dataset. SNPs have been assigned to genes by LD. Vertices with crossmarks denote ambiguous annotations (several genes annotated to a single SNP). Colorization corresponds to annotated GO terms (see legend boxes).

Using extended gene annotations, we have to deal with the fact that multiple genes can inherit their p-value from the same SNP. Such cases are denoted with crossmarks in figure 8. For example there could be LD blocks containing many genes participating in the same biological process. A single SNP association that spans the whole block would result in a large number of these genes considered to be associated, although it is more likely that only one or few of them are causal (here, for example, the strongly associated genes TBX15 and WARS2 stem from the same associated locus, as can be seen in the regionalplots). Despite this drawback, we are now able to identify PIGC as another gene that relates to a 'metabolic' GO term (cellular protein metabolic process), which might be quite interesting. Also note that NISCH and ITPR2 are now colorized brown, because they match multiple categories in the *vertexcolor.GO.regex* list.

Finally, it should also be mentioned that it is not mandatory to do an automatic gene mapping: For a small set of loci (e.g. genomewide significant hits), it is as well possible to create regionalplots for all loci of interest and assign genes to the peak SNPs manually under consideration of distance and underlying LD pattern. Also, gene-wise p-values can be calculated and used as explained in section 4.

Decomposing networks into gene communities The next example demonstrates decomposition of a larger graph into gene communities.This is useful when many associated loci beyond genomewide significance are considered like e.g. in the analysis of polygenic traits. The example will be calculated on data with (almost) genomewide coverage of 100'000 SNPs. After mapping of these SNPs to genes and calculation of gene-wise p-values, all genes having $gene.p <= 10^{-4}$ will be considered in the network analysis. Instead of using custom GO term selections for the vertex colorization as in the previous example, we will further run a GO term overrepresentation analysis to automatically determine GO terms for vertex colorization.

Lastly, the following example will also show the feasibility to exert a joint network analysis of two datasets. Genes are then distinguishable by different vertex shapes for each dataset.

We will carry out such a joint analysis using the *BMI* and *height* datasets - for these traits, it is hypothesized that some overlap in the genetic component should exist [6, 2]. For the interpretation of such an analysis, it has to be considered that p-values can not be compared between studies unless both studies exhibit the same study design (i.e. in particular both studies are equally powered). Generally, p-values can be taken as a measure for the probability of a false positive finding, which means the probability of observing the measured genotype data under the null hypothesis. In more powerful studies (e.g. larger sample size), p-values will generally be stronger when a true association is present, but that does not imply that such genes are more 'significant' than genes from less-powered studies. For studies that are incomparable in that sense, either an appropriate correction method has to be used, or p-values could be replaced by an alternative measure like estimation of effect sizes. To do so, it is necessary to modify the *edge.weight.fun* argument of the *gwas2network* function. The following example shows how to integrate two studies technically in a single network, assuming that the studies have an identical underlying design.

```
# cut down height dataset to 100000 SNPs, evenly distributed
# (first sort by CHR / BP, then apply pruneVec)
> height <- read.table("heightRemapped.txt", header = TRUE)
> height <- height[order(height$CHR, height$BP), ]
> height.100k <- height[height$SNP %in% postgwas:::pruneVec(height$SNP, 100000), ]

# annotate closest gene to SNPs
> height.100k.genes <- snp2gene.prox(height.100k, level = 0, use.buffer = TRUE)

# calculate gene-wise p-value and keep only genes with p <= 10^-4 for analysis
> height.100k.genes.p <- gene2p(
+   height.100k.genes,
+   method = GATES,
+   gts.source = "height100k.ped"
+ )
> height.100k.genes.p <- height.100k.genes.p[height.100k.genes.p$gene.p < 10^-4, ]

# same procedure is applied to BMI dataset (not shown)

# merge datasets, cap smallest p-values at 10^-16
# (otherwise vertex size for moderate association at e.g. 10^-7 is too small)
> height.100k.genes.p$pheno <- "height"
> bmi.100k.genes.p$pheno <- "BMI"
> overlap <- rbind(bmi.100k.genes.p, height.100k.genes.p)
> overlap$gene.p[overlap$gene.p < 10^-16] <- 10^-16

> net <- gwas2network(
+     gwas.mapped.genes = overlap,
+     network = getInteractions.path(overlap$geneid),
+ )
```

The resulting figures 9, 10 and 11 of this code fragment have been precalculated (are not re-build when executing the vignette), which took about three hours for the whole pipeline using a standard PC. Figure 9 shows the complete network, and two of the extracted communities (six in total) are exemplary shown in figure 10. Further, figure 11 shows the three best overrepresented GO terms in the first community as a GO subgraph, in detail with its parent terms and stats (e.g. number of genes in the community that are assigned to the term). The figure is produced

by (and explained in) the topGO package[6], which is used by postgwas.

Interpretation of entire network in figure 9 is difficult because of the large number of genes contained, but still possible by zooming into the graphic. The first community listed in figure 10 is easier to interpret. It contains primarily cell division and mitosis-related genes. This might be be very well related to one of the underlying genetic mechanisms of growth-related traits like human height and BMI. It has a low module score assigned, hinting towards an enrichment of well associated genes in that community. The last community obviously contains many genes with weaker association (which is the reason why it is listed last with the highest module score of all six communities). Nevertheless, the subjectively large number of genes associated to the 'small molecule metabolite process' GO term colored in blue seems to be interesting and could also be further investigated. Lastly, the ADCY3 gene in this community is associated with a moderate p-value in both the *height* and *BMI* dataset, which could hint on a pleiotropic function of the gene for these traits.

So far, all networks have been generated by including only genes from the GWAS dataset in the network. As a last note, it is also possible to use networks that exceed the scope of associated genes from the GWAS. The argument *prune* can then be used to define inclusion criteria for the non-GWAS genes in the network (if such genes are included, they get a p-value assigned as defined by the *default.p* argument). This way, it is possible to work on a 'shared interactor network' of protein interactions, for example. The help pages of *gwas2network* contain further information on the *prune* argument.

[6] http://www.bioconductor.org/packages/2.11/bioc/html/topGO.html

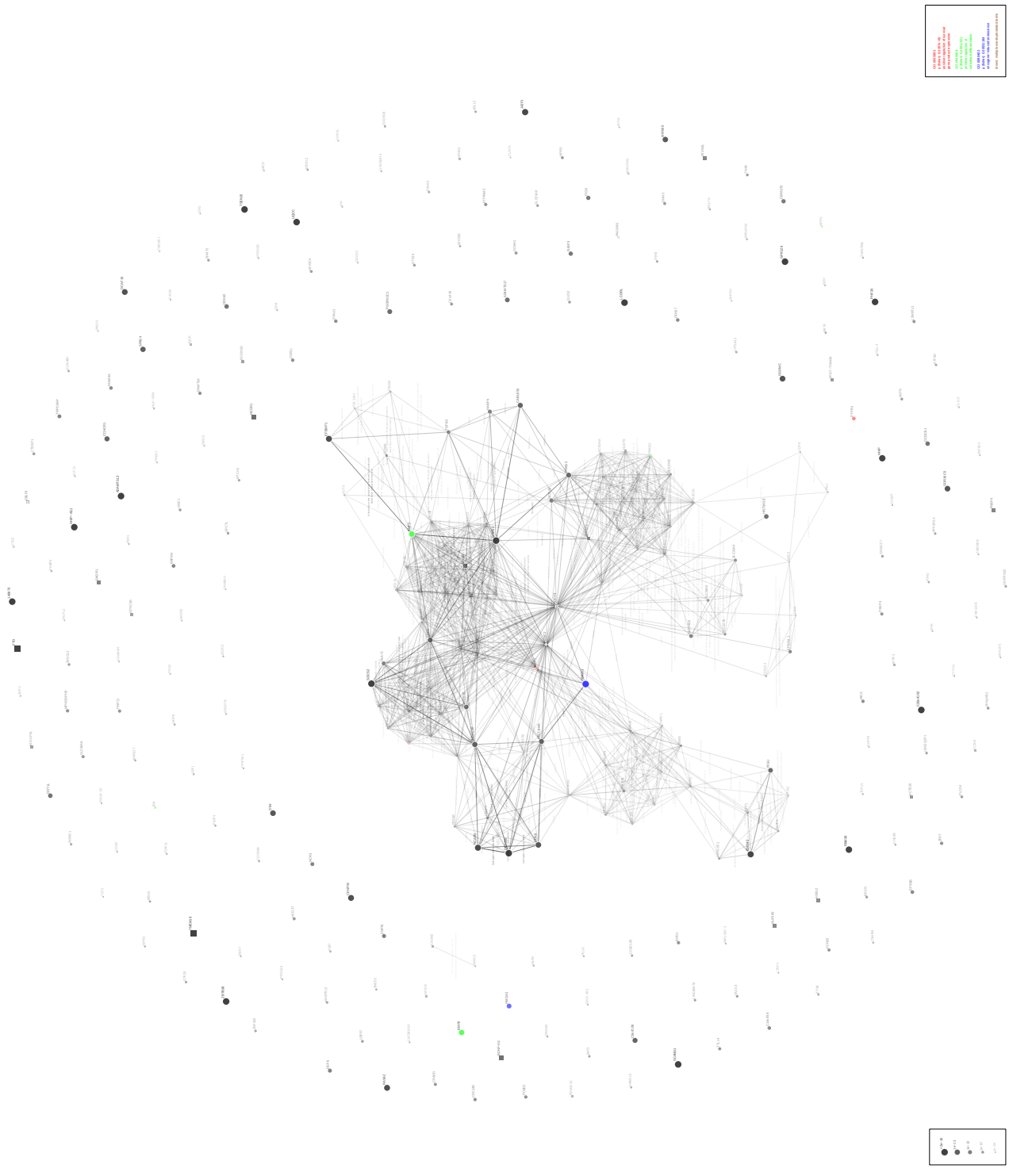

Figure 9: The complete network for both *height* and *BMI* datasets. Vertices are connected by shared pathway membership. Two exemplary subnetworks are singled out in figure 10 for a more detailed view.

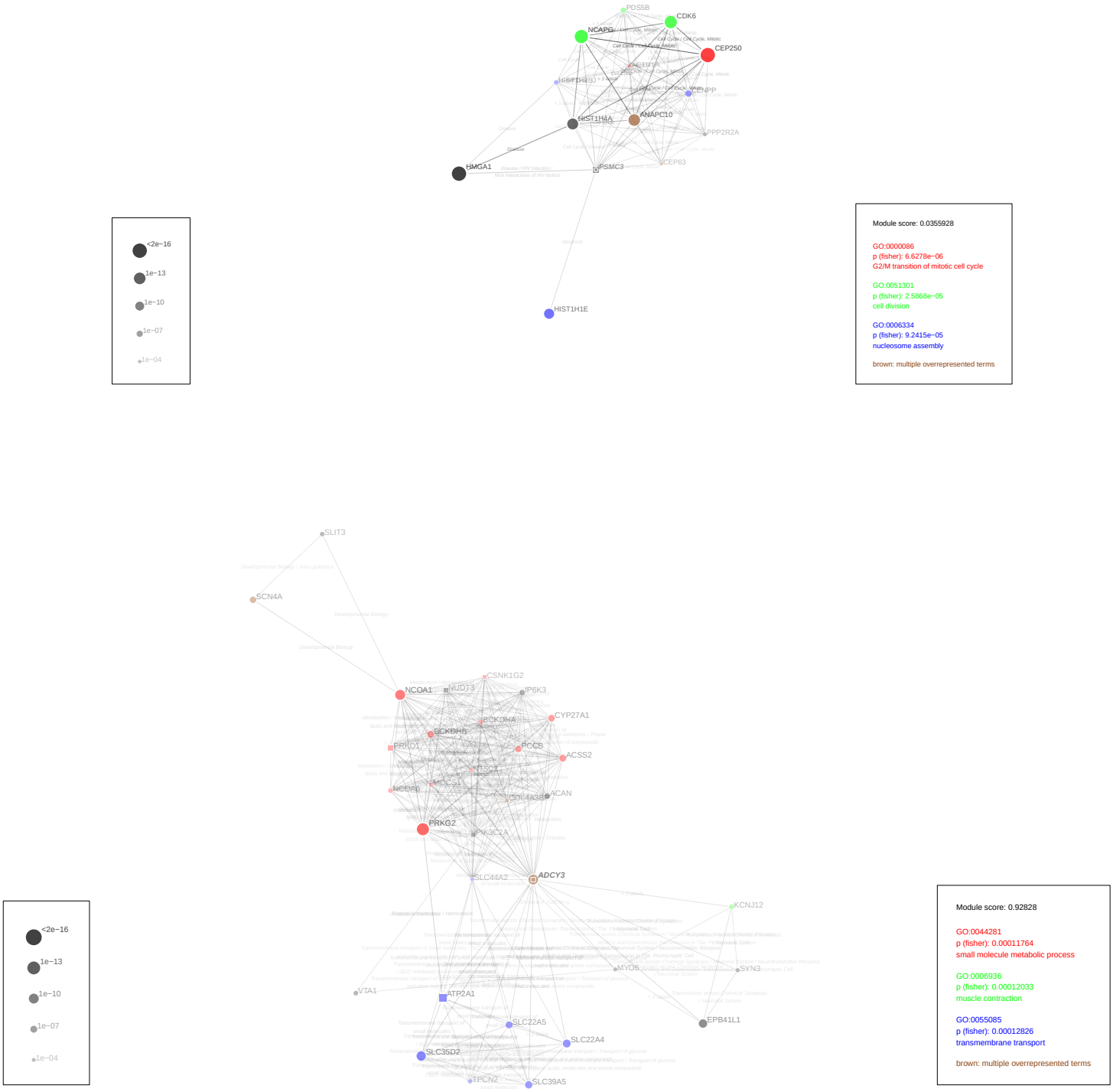

Figure 10: Communities 1 and 6 (of 6) in the network of combined *height* and *BMI* datasets. Vertices are connected by shared pathway membership. Circles denote associated genes from the *height* dataset, and squares for the *BMI* dataset. Genes occuring in both datasets are highlighted with boldface and italic labels.

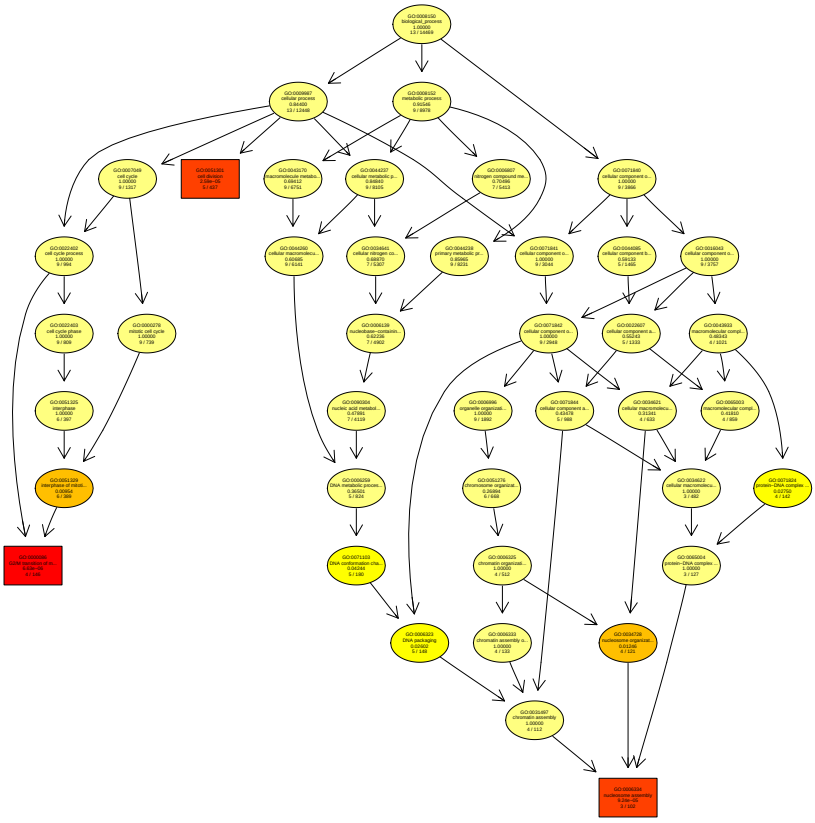

Figure 11: Details for the GO term overrepresentation analysis for the first gene community in figure 10 (produced by the topGO package)

7 GenABEL example

All examples were based on PLINK-like GWAS result files so far. However, all postgwas functions work on GenABEL objects / files as well (and GEMMA and FAsT-LMM formats). We now use the *srdta* dataset that is provided with GenABEL to create a regionalplot. Let us assume that a GWAS analysis has been done in GenABEL:

```
> data(srdta)
> gwas <- ccfast("bt", srdta)
```

The resulting *gwas* object (of class *scan-gwaa*) can be directly passed to *regionalplot* and other functions that require a *gwas.dataset* argument, e.g. manhattanplot. Further, genotype data is available via the *gtdata* slot of the *srdta* object, but could also be provided in GenABEL gwaa/phe files. We use the *srdta@gtdata* object for calculation of an LD track (can similarily be provided to all other postgwas functions that require a *gts.source* argument):

```
> regionalplot(
+   snps = data.frame(SNP = "rs1020"),
+   gwas.datasets = gwas,
+   ld.options = list(gts.source = srdta@gtdata),
+   plot.genes = FALSE
+ )
```

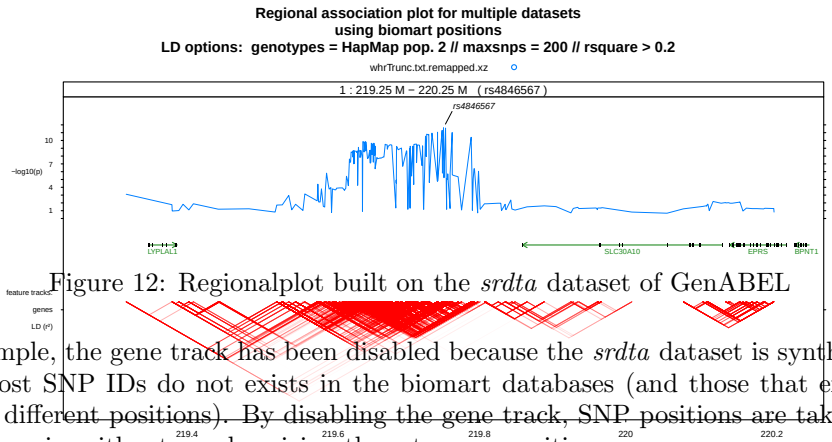

Figure 12: Regionalplot built on the *srdta* dataset of GenABEL

In this example, the gene track has been disabled because the *srdta* dataset is synthetic, meaning that most SNP IDs do not exists in the biomart databases (and those that exist map to completely different positions). By disabling the gene track, SNP positions are taken from the source data as is, without synchronizing them to gene positions.

Lastly, it is of note that the *gwas* object generated by GenABEL contains several slot for p-values. Postgwas functions will always extract the *P1df* slot, and there is no way to directly specify a different one (e.g. *Pc1df* for genomic inflation corrected p-values). However, a data frame (or PLINK-like file) can be constructed and supplied to the *gwas.datasets* argument instead:

```
> gwas.custom <- data.frame(
+     SNP = snpnames(gwas),
+     P = gwas[, "Pc1df"],
+     BP = gwas[, "Position"],
+     CHR = chromosome(gwas)
+ )
```

8 Working with non-human Organisms

The postgwas tools can be principally applied to any kind of species. Depending on the organism, it may nevertheless require some additional effort to get it running. First option (which always works) is to supply buffer data, i.e. manually download all required annotation data and load them into the R workspace as appropriately formatted data frames. These are further described in the help pages, e.g. *help(postgwasBuffer)* and section 9.

A more convenient way is to change the biomart configuration of postgwas, given that your species is annotated via a biomart. Try *help(biomartConfigs)* to get further explanation. Also, the example section of most functions contains an example on non-human data.

Here, the following examples demonstrate how to run the primary postgwas functions with an example dataset for *mus musculus*. It works by supplying the predefined *biomartConfigs$mmusculus* configuration list to the *biomart.configs* argument of each postgwas function, here shown for the manhattanplot:

```
> manhattanplot(
+     "mouse_lmm.assoc.txt.xz",
+     biomart.config = biomartConfigs$mmusculus,
+     reduce.dataset = FALSE,
+     use.buffer = TRUE,
+     toFile = FALSE
+ )
```

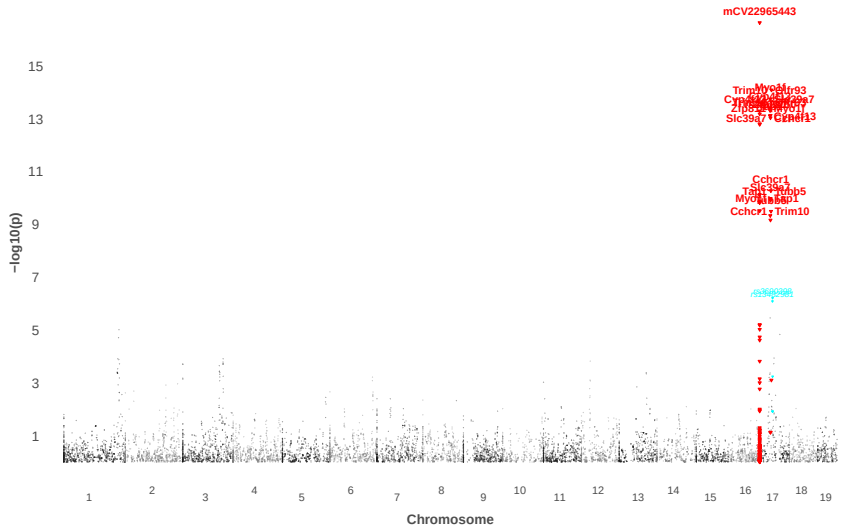

Figure 13: A simple manhattan plot of the *mus musculus* example GWAS dataset from the *GEMMA* package.

Figure 13 obviously contains many (independently?) associated loci within a larger chromosomal region. Here it is more recommendable to generate a manhattanplot without text annotation and instead investigate the loci separately in regionalplots that offer a higher resolution.

Therefore, we extract all loci of interest from the GWAS dataset, defining $\alpha = 5 * 10^{-8}$ as threshold for significance. This is done in the following code fragment:

```
> mm <- postgwas:::readGWASdatasets("mouse_lmm.assoc.txt.xz")
> snps.mm <- removeNeighborSnps(mm[mm$P < 5*10^-8, ], maxdist = 100000)
```

The first line reads the complete GWAS dataset into a data frame (this could be done using the *read.table* function, but here we use the internal postgwas function *readGWASdatasets*, which auto-detects some formats), and on the second line, we extract SNPs meeting our α threshold and prune them down to lead SNPs only (within 500kb windows).

For these loci, we annotate genes as usual, shown in the following code fragment and table 5,

```
genes.mm <- snp2gene.prox(
    snps.mm,
    level = 0,
    use.buffer = TRUE,
    biomart.config = biomartConfigs$mmusculus
)
```

	geneid	genename	start	end	CHR	BP	SNP	P	direction
2	170716	Cyp4f13	32924688	32947402	17	32941145	rs13459151	3.835716e-14	cover
21	240063	Zfp811	32797406	32800938	17	32800415	rs13482952	7.735179e-14	cover
6	14977	Slc39a7	34028267	34031690	17	34030404	rs13482957	1.043051e-10	cover
8	240084	Cchcr1	35517100	35531015	17	35530170	rs13482963	5.340434e-11	cover
NA	19824	Trim10	36869574	36877825	17	36878449	rs13482967	2.262217e-14	up
NA1	258051	Olfr93	37151007	37152051	17	37131683	rs13482968	8.144100e-15	down
7	21354	Tap1	34187553	34197225	17	34196995	rs3023442	9.278717e-14	cover
9	22154	Tubb5	35833926	35838306	17	35834133	rs3682923	3.209200e-10	cover
3	17916	Myo1f	33555719	33607764	17	33563059	rs6249614	1.813323e-14	cover

Table 5: SNP to gene annotation by proximity for mouse data

and generate regionalplots (figure 14):

```
> regionalplot(
+     snps = snps.mm,
+     gwas.datasets = c("mouse_lmm.assoc.txt.xz"),
+     window.size = 350000,
+     biomart.config = biomartConfigs$mmusculus,
+     ld.options = NULL,
+     use.buffer = TRUE
+ )
```

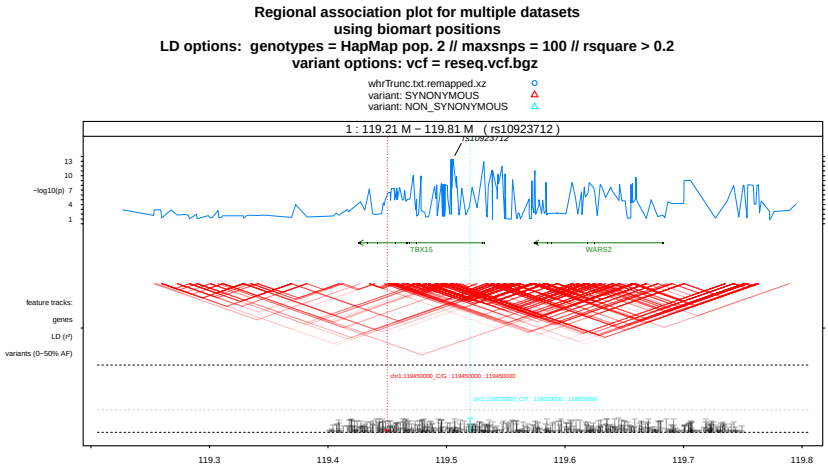

Figure 14: Regional plot of the mmusculus dataset shown for the first 6 significantly associated loci - the example data is very sparse, so the p-value graph only contains few SNPs and appears truncated.

Finally, it is also possible to visualize the GWAS data in a network. We need the *org.Mm.eg.db* package (which can be installed from bioconductor) to be able to deal with Gene Ontology data, and again have to supply the `biomartConfigs$mmusculus` configuration list in the *biomart.configs* argument to obtain the network in figure 15:

```
> if(library(org.Mm.eg.db, logical.return = TRUE)) {
+   network.data <- getInteractions.GO(
+       genes.mm$geneid,
+       GOpackagename = "org.Mm.eg.db",
```

```
+       similarity = "hausdorff"
+   )
+   network.igraph <- gwas2network(
+       genes.mm,
+       network.data,
+       max.communities = 0,
+       vertexcolor.GO.overrep = "org.Mm.eg.db",
+       biomart.config = biomartConfigs$mmusculus,
+       use.buffer = TRUE
+   )
+   gwas2network.plot(network.igraph, file = "exampleNetMouse.pdf")
+ }
```

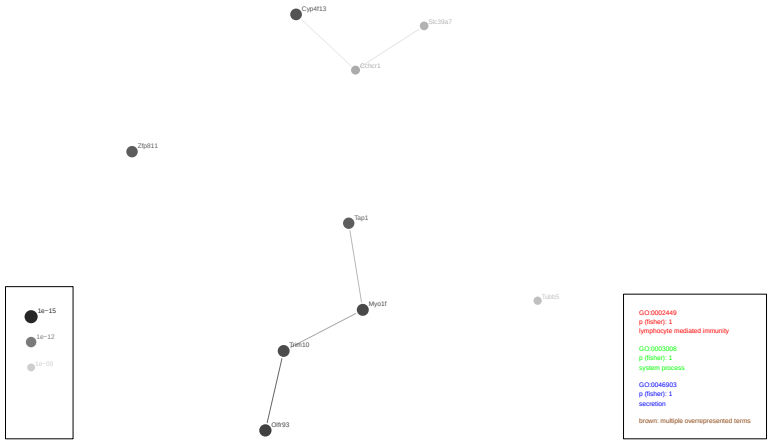

Figure 15: Network of shared GO term architecture between loci of the *mmusculus* dataset

9 Using Buffer Data / Running an Analysis Offline

Most functions in postgwas require specific annotation data: These are by default downloaded from biomarts (which can be configured, see help(biomartConfigs)). Alternatively, all annotation data can also be supplied in a data frame format, with one data frame for each type of annotation (e.g. genes, SNPs, exons, ...). Most functions in postgwas include a *use.buffer* argument which instructs to use annotation from such buffer data frames, if they exist. If all or some of them do not exist and the *use.buffer* argument is set, data for the according buffer type is first downloaded and then stored in the appropriate variables for later (re-)use. However, the data frames are not accessible from the user's workspace, but hidden in the environment of the postgwas package. Access is possible via the *getPostgwasBuffer()* and *setPostgwasBuffer()* functions:

```
> buffers <- getPostgwasBuffer()
```

A list containing all buffer data frames has been returned:

```
> names(buffers)
```

```
[1] "snps"               "genes"              "genes.regionalplot" "exons.regionalplot" "ld.reg
```

Take a look at the data frame for SNP annotation:

```
> snpbuffer <- buffers[["snps"]]
> head(snpbuffer)
```

```
   refsnp_id chr_name chrom_start
1 rs13459151       17    32941145
2 rs13482952       17    32800415
3 rs13482957       17    34030404
4 rs13482963       17    35530170
5 rs13482967       17    36878449
6 rs13482968       17    37131683
```

It is possible to modify the annotation data frame and resubmit it to the package:

```
> snpbuffer[1, "refsnp_id"] <- "alternativeID"
> setPostgwasBuffer(snps = snpbuffer)
```

Also, the complete list of buffer data frames can be submitted, allowing to restore the state for a certain analysis at once:

```
> setPostgwasBuffer(uselist = buffers)
```

Buffer variables that have the value *NULL* are considered as not set and will be filled with downloaded annotation data when *use.buffer = TRUE*. The function *clearPostgwasBuffer* can be used to reset all variables to NULL.

Further information can be found in the package documentation by running:

```
> help("postgwasBuffer")
> example(postgwasBuffer)
```

10 Example Data

A vignette comes as a single pdf document, but in fact it is compiled during the package installation process on your local machine by running all examples. The postgwas package includes some source data files so that the code in this vignette can be run. The folder on your local machine where the data files are located can be identified by executing:

```
system.file("doc", package = "postgwas")
```

The GWAS data files found there are truncated versions of the original GIANT consortium GWAS data files[7]. The original datafiles contain about 2.5 million SNPs, which is far too much for our examples. The datasets have been preprocessed as follows:

```
> gwas.whr <- read.table(
+                 "GIANT_WHRadjBMI_Heid2010_publicrelease_HapMapCeuFreq.txt",
+                 header = T
+             )
> gwas.whr <- gwas.whr[, c("MarkerName", "p")]
> colnames(gwas.whr) <- c("SNP", "P")
> gwas.whr.trunc <- gwas.whr[
+                     gwas.whr$P < (runif(nrow(gwas.whr))*0.7)^4 |
+                     gwas.whr$P < quantile(gwas.whr$P, 0.001),
+                   ]
> write.table(gwas.whr.trunc, "whrTrunc.txt", row.names = F, sep = "\t")
```

This prunes SNPs randomly, with increased probability to retain low p-value SNPs. The top 1% of SNPs (lowest p-values) are always kept. The resulting file *whrTrunc.txt* contains 127'750 SNPs after this truncation step.

Because most postgwas functions require a CHR and BP column to run, these had also to be added to the file. This can be done using biomart queries with a builtin function

```
> bm.remapSnps("whrTrunc.txt")
```

which would produce a whrTrunc.txt.remapped file containing CHR and BP columns. But, as the dataset is quite large, using biomart should be avoided. It is comparably simple to download and appropriate SNP annotation file from a web resource, and merge it with the GWAS result file using the *merge* function of R.Both approaces yield the final example data file with the following heading lines:

```
> head(read.table("whrTrunc.txt.remapped.xz", header = TRUE))
```

```
         SNP     P CHR        BP
1 rs10000036 0.066   4 139219262
2  rs1000005 0.030  21  34433051
3 rs10000101 0.063   4 171716087
4  rs1000014 0.094  16  24417536
5 rs10000140 0.050   4  89948054
6  rs1000016 0.038   2 235690982
```

Preprocessing of the *BMI* and *height* datasets, which are used only in the network analysis is described in the according section 6.

[7] downloadable from http://www.broadinstitute.org/collaboration/giant/index.php/GIANT_consortium_data_files

References

[1] Iris M Heid, Anne U Jackson, Joshua C Randall, Thomas W Winkler, Lu Qi, Valgerdur Steinthorsdottir, Gudmar Thorleifsson, M Carola Zillikens, Elizabeth K Speliotes, Reedik Mägi, and et.al. Meta-analysis identifies 13 new loci associated with waist-hip ratio and reveals sexual dimorphism in the genetic basis of fat distribution. *Nature genetics*, 42(11):949–60, November 2010.

[2] Hana Lango Allen, Karol Estrada, Guillaume Lettre, Sonja I Berndt, Michael N Weedon, Fernando Rivadeneira, Cristen J Willer, Anne U Jackson, Sailaja Vedantam, Soumya Raychaudhuri, and et.al. Hundreds of variants clustered in genomic loci and biological pathways affect human height. *Nature*, 467(7317):832–8, October 2010.

[3] Miao-Xin Li, Hong-Sheng Gui, Johnny S H Kwan, and Pak C Sham. GATES: a rapid and powerful gene-based association test using extended Simes procedure. *American journal of human genetics*, 88(3):283–93, March 2011.

[4] Dale R Nyholt. A simple correction for multiple testing for single-nucleotide polymorphisms in linkage disequilibrium with each other. *American journal of human genetics*, 74(4):765–9, May 2004.

[5] Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1), July 2006.

[6] Elizabeth K Speliotes, Cristen J Willer, Sonja I Berndt, Keri L Monda, Gudmar Thorleifsson, Anne U Jackson, Hana Lango Allen, Cecilia M Lindgren, Jian'an Luan, Reedik Mägi, and et.al. Association analyses of 249,796 individuals reveal 18 new loci associated with body mass index. *Nature genetics*, 42(11):937–48, November 2010.

[7] Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature genetics*, 44(7):821–4, July 2012.