Facultad de Ciencias

Universidad Zaragoza

Appendix: Application of Mendelian Randomization in the inference of gene regulatory networks

Anexo: Aplicación de la Aleatorización Mendeliana en inferencia de redes de regulación genética

Trabajo Fin de Grado

Autor:

Claudia Llop Moreno

Tutor:

Dr. Joaquín Sanz Remón

Grado de Física.
2021-2022

Índice

1. Pipeline ingredients (i): data types under analysis **2**

1.1. Measuring gene expression through RNA-seq 2

1.2. Genotyping chips 3

1.3. Genomic feature data and samples meta data 4

2. Pipeline ingredients (ii): Analytic modules **4**

2.1. Phenotypes correlations: Co-expression networks analysis. 4

2.2. Control variables: eQTL-mapping 5

3. Trimmed M Means algorithm (TMM) **6**

4. Benjamini-Hochberg against Storey-Tibshirani as the method for FDR estimation **7**

5. Implementation of balanced-SNPs algorithm **9**

6. Link direction inference under different genetic variable selection criteria **9**

7. Codes **11**

7.1. Prepare input tables 11

7.2. WCGNA 15

7.3. Identify eQTLs 19

7.4. Mendelian Randomization for Top-SNPs method 28

7.5. Mendelian Randomization for Balanced-SNPs method 36

7.6. Implementation of empirical null models to quantify directional bias in link prediction for Top-SNPs method 44

7.7. Implementation of empirical null models to quantify directional bias in link prediction for Balanced-SNPs method 46

1. Pipeline ingredients (i): data types under analysis

1.1. Measuring gene expression through RNA-seq

The main goal of this TFG is the analysis of the co-variation patterns of pairs of genes in a given experimental cohort, and the implementation of a statistical method to infer causal directions between the genes conforming the different pairs in a genome-wide gene co-expression network. Gene Co-expression Networks (GCN) are concerned with describing the correlation patterns of gene expression across sets of samples in which the expression of a large number of genes has been measured simultaneously.

Nowadays, the technologies of choice to characterize genome-wide expression levels in arbitrarily large cohorts of experimental samples are the so-called Next Generation sequencing technologies, or NGS, whose output is the sequencing of all the messenger RNA present in a given sample (RNA-seq). Using NGS technologies, it is nowadays possible to identify which of the genes encoded in our DNA are expressed in a given tissue, and to what extent, unlocking the interrogation of tens of thousands of genes in thousands of samples at once (1).

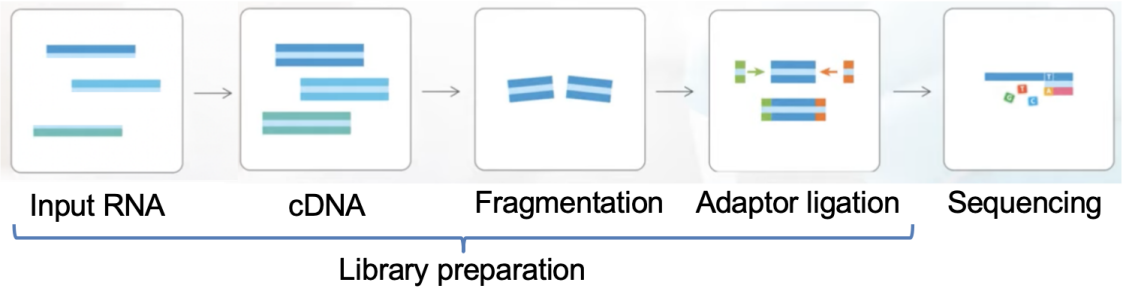


Figura 1: RNA-sequencing process.

RNA sequencing follows 5 main steps. From a sample of single-stranded RNA molecules we obtain cDNA (complementary DNA). The cDNA strand is synthesized to have double-stranded cDNA molecules whose nucleotide sequence is complementary to the original RNA. It is then chopped, adapters are placed at the ends of each fragment to facilitate sequencing and the resulting fragments, which constitute the so called sequencing library, are finally sequenced. In the following, we will look further into the process described.

The first 4 steps are part of what is called library preparation. From a given biological sample, cells are broken and their total RNA is extracted. This RNA is of many types: rRNA (ribosomal), tRNA (transfer RNA; small RNAs that carry amino acids to ribosomes for translation), mRNA (messenger) and other types. Since RNA-seq is typically bound to the sequencing of messenger RNA alone, we need a method to separate them so that we are left with only the mRNA that we are looking to sequence. This is done by an enrichment process. In our case we start with eukaryotic cells, whose mRNA molecules have a poly-A tail, which is leveraged using a Thymine-type bait to precipitate the mRNA we need and isolate.

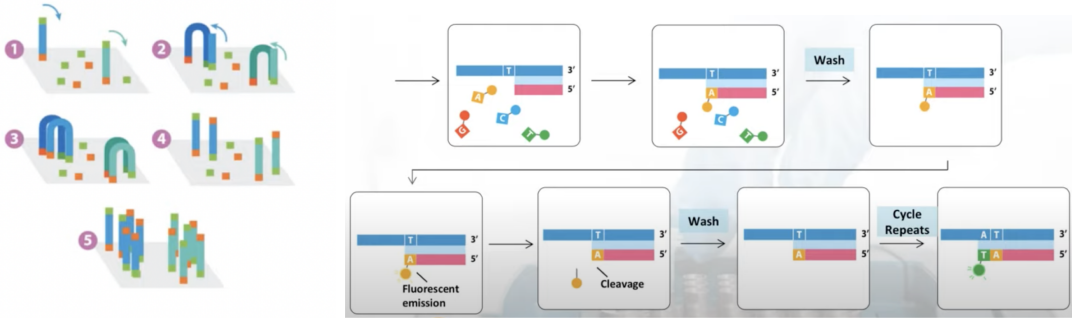
Next, the sample, already strongly enriched in mRNA content, is incubated with reverse transcriptase to synthesize a cDNA strand. When it is completed, the RNA strand is hydrolyzed and a second (complementary) cDNA strand is generated. Then cDNA molecules are fragmented, generating short reads of homogeneous size. At this point, adapters are added to the fragments. These are small oligonucleotides that have different functions, including ensuring the hybridization of the fragments to complementary sequences that have been seeded in the surface of the

sequencing flowcell. Others are primer sequences that serve to start a complementary synthesis reaction. As sequencing denatures the strands (separates them), primers are needed to synthesize the complementary strands.

Once the library is prepared we move on to sequencing. In sequencing by synthesis -which is the method used in Illumina sequencing, the technology used in this study-, we start from a flow cell, that is, a surface seeded with DNA sequences complementary to the adapters placed at the ends of the fragments. The adapters adhere to these sequences. The number of strands in the sample needs to be increased for the signal to have sufficient amplitude to be detected. To do this, the amplification process begins. The parallel strand that remains attached to the floor of the flow cell is sequenced. This is folded and hybridizes with the nearby complementary adapter and generates the chain again as can be seen in Figure 2a. Then, the corresponding sequences are denatured which translates into two strands with the same sequence covalently attached to the flow cell. Finally, this process is repeated until enough copies of the original fragment are synthesized. This process is named bridge amplification.

For sequencing we pull nucleotides (Adenine, Guanine, Cytosine, Tinine) which are labeled with different colored fluorescent labels. They also have a radical that prevents further nucleotides from binding to the sequencing chain, therefore, when the corresponding nucleotide is anchored, it prevents the binding of the next nucleotide in the chain. The flow cell is then stimulated with a laser in such a way that the nucleotide returns the color of its fluorescent label and is recognized, and sequenced (Figure 2b). Then, the radical stopping further reactions is removed, and the process is iterated to continuing reading nucleotides. In this way the sequence of nucleotides in each cluster of the flow cell, is registered.

Once all the fragments have been sequenced, a final step prior to statistical analysis is the mapping to a reference genome. This step briefly consists of the identification of the most likely genomic location, or locations, each fragment maps to, counting the number of fragments mapping into each gene or transcript. The result of that process is the matrix of expression data that we use as input for our analyses.



(a) Amplification process.

(b) Sequencing by laser method.

Figura 2: Parts of the RNA-sequencing process.

1.2. Genotyping chips

A single nucleotide polymorphism (SNP) is a type of single nucleotide variant (that is: a genomic position whose genotype is variable in a population), whose less frequent alelle, is present in more than 1 % of the population. SNP chips are DNA microarrays that test genetic variation at thousands, or millions of specific locations across the genome(2), which makes them an excellent tool for studying common genetic variation, which can be used to assess ancestry as well as predisposition to many complex multifactorial diseases. (3)

A SNP chip briefly consists of an array of dwells containing a matrix of beads, on top of which, pairs of specific DNA sequences are seeded. These sequences are different in each dwell, and correspond to the complementary sequences immediately preceding the SNP you want to detect in each dwell (Figure 3).

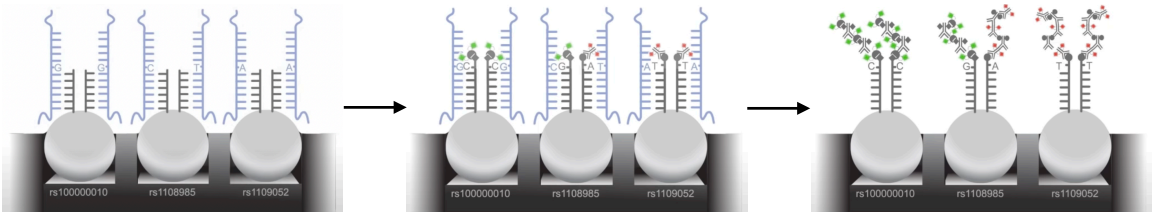


Figura 3: Genotyping chips for SNP sequencing process.

After loading the chip with a solution containing fragments of DNA of a given individual, the fragments that will hybridize to each bead will have the SNP of interest as the first free nucleotide that is not hybridized. Once you have hybridized the molecules, you pour a soup of labelled nucleotides of the 4 types onto the chip and each one is added to the strands that were originally linked to the SNP. Guanines and Cytosines are labeled with one color marker and Adenines and Thymines with another color. The original molecule hybridizing to the chip is washed and the fluorescent signal is amplified by the addition of further molecules to each bead. Finally, the distinction between nucleotides of each color allows the identification of the genotype of each SNP, either homozygous or heterozygous, what is commonly referred to as SNP-calling.

1.3. Genomic feature data and samples meta data

In addition to raw expression tables and SNP genotypes, in order to complete our analytical pipeline, we need to know certain attributes of the genomic features (genes and SNPs), as well as certain key information of our samples (sample metadata).

The only genomic features needed for completing these study are three: chromosome and genomic locations for genes and SNPs, (in order to distinguish between putatively cis- and -trans tests from the gene-SNO distances, and to exclude co-expression pairs involving nearby genes), and gene biotypes as well, since we will focus only on protein coding genes in our study.

Concerning sample meta.data, our samples come from an homogeneous panel of blood derived macrophages from male donors with balanced biological determinants, such as age and sex (all male). The only relevant metadata for our study in the sequencing batch each sample corresponds to among n=9 different batch levels.

2. Pipeline ingredients (ii): Analytic modules

2.1. Phenotypes correlations: Co-expression networks analysis.

A gene co-expression network (GCN) is an undirected graph, where each node corresponds to a gene, and a pair of nodes is connected with an edge if there is a significant co-expression relationship between them. (4) Having gene expression profiles of a number of genes for several samples or experimental conditions, a gene co-expression network can be constructed by looking

for pairs of genes which show a similar expression pattern across samples, since the transcript levels of two co-expressed genes often show significant co-variation patterns across samples, even when these are biological replicates. Gene co-expression networks are usually constructed using datasets generated by high-throughput gene expression profiling technologies such as RNA-Seq. (5). One of the most relevant approaches to the analysis of co-expression networks is based on modularity analysis, which, broadly speaking, aim at identifying groups of proteins whose interaction patterns in the network are stronger and more frequent than with the remaining of the system. Their characterization and functional characterization through ontology enrichment analyses constitutes a valuable tool to understand gene variation patterns in a given cohort.

In spite of its wide utility, co-expression networks are, per se, barely informative of causality relations across genes. To that end, more sophisticated inference approaches are needed to complete the transition from the (undirected) co-expression networks, -capturing correlations- to the (directed) regulatory networks (GRNs) -capturing causality-. In a GRN, a directed edge connects two genes, representing a biochemical process such as a reaction, transformation, interaction, activation or inhibition, or a set of them, that translates into a causal effect bounding the activity of the link sender to that of the link receiver. Compared to a GRN, a GCN does not attempt to infer the causality relationships between genes and in a GCN the edges represent only a correlation or dependency relationship among genes.

In this work, through Mendelian Randomization we seek to infer causality in a gene co-expression network (GCN).

2.2. Control variables: eQTL-mapping

It is known that SNPs located in regulatory regions, e.g. transcription factor (TF) binding sites, are often eQTLs (Expression quantitative trait loci), as they modulate gene expression.(6) An eQTL is a locus that explains a fraction of the genetic variance of a gene expression phenotype. Standard eQTL analysis involves a direct association test between markers of genetic variation with gene expression levels typically measured in tens or hundreds of individuals. This association analysis can be performed proximally (*cis*) or distally (*trans*) to the gene. (7)

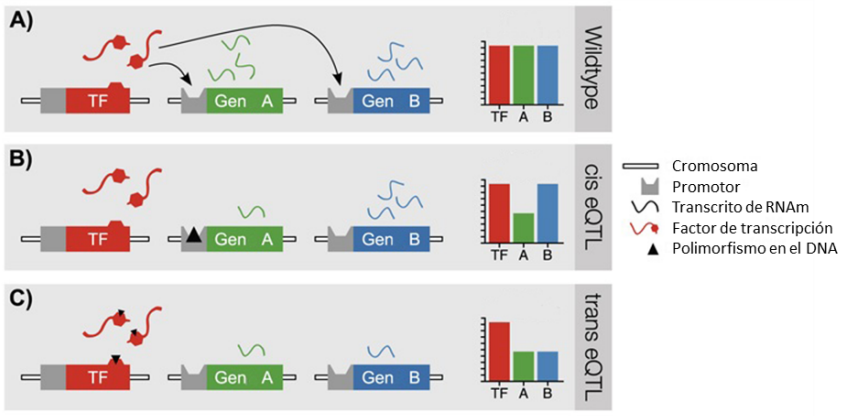


Figura 4: Illustration of an cis- and trans-eQTL concept (8).

Starting point at Figure 4 shows a transcription factor that regulates two copies, A and B, of the same gene: one inherited from the mother and the other from the father. In the first case we see what happens when you have a SNP (triangle) attached at the place where a TF has to bind nearby the target gene. The top row represents a case where regulation of both copies works

equally. In the bottom row, the presence of a cis-genetic variant nearby the gene inherited from the mother, translates into a difference in the expression levels between both copies (cis-EQTL). This situation represents what we know as cis-eQTL: it is a genetic variant that affects the expression level of a gene only in the same molecule in which the variant is found and not in another.

Finally, in the last row, we see a case when a variant is found in the coding part of the regulator, which, specially if this changes an amino acid of the TF which, in turn, modifies its affinity for the binding site of the target gene, resulting in less expression in both cases. Here the variant works in trans because it lies on one molecule but generates effects on other molecules. In this case the expression of both genes would be the same, if interrogated in a single individual, and we would only see differences when comparing with the levels across individuals. Typically cis-eQTLs are related as genetic variants close to a gene and trans as distant. Although not correct by definition, in practice this is true in the vast majority of cases.

3. Trimmed M Means algorithm (TMM)

The weighted and trimmed mean of M values, or TMM(9), is an algorithm implemented in the *edgeR*(10) software package that use scale factors per sample to normalize gene expression. These factors allow to transform the raw reads data to a relative measurement such as read counts per million reads sequenced, enabling comparisons of gene expression across samples of different complexity. To divide the observed number of counts ($Y_{g,k}$) of gene g in library k summarized from the raw reads by the total number of reads in the $k-th$ library, N_k, is not enough to produce reliable comparisons across samples. Instead, we can define $\mu_{g,k}$ as the true and unknown expression level of the gene of interest g in sample k; then, if L_k is the length of gene g, we can model the expected value of as follows:

$$E(Y_{gk}) = N_k \mu_{g,k} L_g / S_k \tag{1}$$

Where

$$S_k = \sum_{g'} \mu_{g',k} L_{g'} \tag{2}$$

Although the total RNA production $S_{k'}$ cannot be directly estimated, it is possible to estimate the relative production of two samples, $S_k/S_{k'} = \rho(k, k')$. Clearing the true expression level of gene g in library k from 1 as below:

$$\mu_{g,k} \approx \frac{Y_{gk} \cdot S_k}{N_k L_g} \tag{3}$$

the ratio of expression of that gene between two libraries k and k' remains:

$$\frac{\mu_{g,k}}{\mu_{g,k'}} = \frac{Y_{gk}}{N_k} / \frac{Y_{gk'}}{N'_k} \rho_g(k, k') \tag{4}$$

Now we use the approximation $\mu_{g,k} = \mu_{g,k'}$ assuming that all genes in our dataset have the same true expression between samples k and k', i.e. the genes are not differentially expressed, and we can rewrite an independent estimation for the true ratio:

$$\frac{Y_{gk'}}{N'_k} / \frac{Y_{gk}}{N_k} = \rho_g(k, k') \tag{5}$$

This approximation is not quite correct, as it is not valid for all genes, for some may be strongly differentially expressed between samples in many cases. To overcome this limitation,

the extreme values of the distribution of $\rho_g(k,k')$ values are trimmed (30 % of them, by default), and the rest are averaged. The statistic that will be trimmed is:

$$log(\rho_g(k,k')) = log(\frac{Y_{gk'}}{N'_k}) - log(\frac{Y_{gk}}{N_k}) \tag{6}$$

In this way, the most extreme estimates across all genes that are sufficiently expressed are excluded. The trimmed values will be averaged, and the weights of each observation will be extracted from the expected statistical uncertainty, expressed as the expected variance, associated to each of the observations:

$$Var(log(\rho_g(k,k'))) = (\frac{1}{E(\rho_g(k,k'))})^2 \cdot Var(\rho_g(k,k')) = \frac{N_{k'} - Y_{gk'}}{N_{k'} \cdot Y_{gk'}} + \frac{N_k - Y_{gk}}{N_k \cdot Y_{gk}} \tag{7}$$

From this equation, the weights will be obtained as the inverse values of those variances:

$$w_{g,k'} = Var(log(\rho_g(k,k')))^{-1} \tag{8}$$

Now, once the weights, and the set G^* of genes that survive the trimming are defined, we defined the global trimmed-M-means, which is the final estimator of :

$$TMM_{k'} = \frac{\sum_{g \in G^*} w_{g,k'} \rho_g(k,k')}{\sum_{g \in G^*} w_{g,k'}} \tag{9}$$

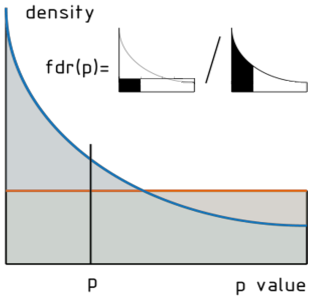
Finally, the coefficients of all samples k' are calculated with respect to the common reference sample k using these $TMM_{k'}$ values as relative normalization coefficients (in the sense that $TMM_k = 1$). Once multiplied by the library depths (number of fragments per sample) they constitute suitable denominators for normalization of gene expression for comparisons across samples in each library. These normalization factors, in our case, are used to build log_2 counts per million (cpm) expression estimates as:

$$log_2(cpm)_{gk} = log_2(1E6 \cdot \frac{Y_{gk} + 0{,}5}{N_k TMM_k + 1}) \tag{10}$$

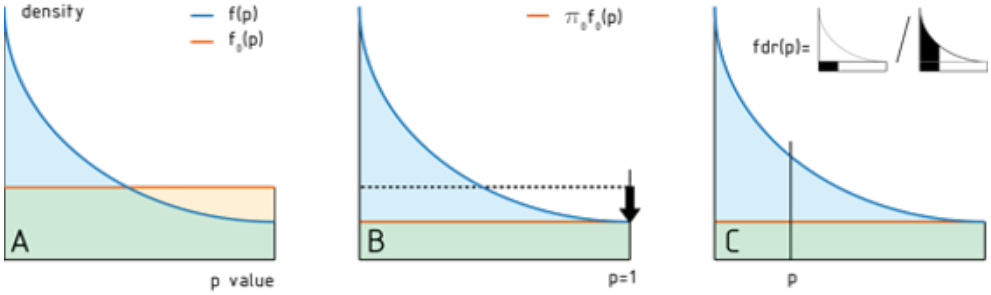
Where the "$+0{,}5$" and "$\backslash + 1$" are introduced to avoid computational singularities when processing big datasets. In the context of our pipeline; coefficients are estimated using the function *calcnormfactors*, in the bioinformatic package *edgeR*(10), while the $log_2(cpm)$ transformation of eq. 10 is done using *voom*, in the *limma* (11) package.

4. Benjamini-Hochberg against Storey-Tibshirani as the method for FDR estimation

Now comes the first alteration we make to the work proposed by Regina Santesteban(12). In her case, she uses the Benjamini-Hochberg (BH)(13) correction on the vector of p-values. We will use the Storey-Tibshirani (ST)(14) for multiple testing correction method for greater precision. What are the differences between these two methods?
The BH method [5a] is equally useful for analyzing uniform null distributions and empiric nulls coming from permutations. However, this method is too conservative. The approach implicity assumes that the true fraction of null hypothesis in your data (π_0) is 1. This is $f(p) = \pi_0 f_0$. On the other hand, the ST method [5b] drops the assumption of $\pi_0 = 1$ and move towards a two component model: $f(p) = \pi_0 f_0(p) + (1 - \pi_0) f_A(p)$.



(a) Benjamini-Hochberg



(b) Storey-Tibshirani

Figura 5: Differences between the two methods: BH & ST

Both methods offer their own definition of the FDR. For BH: $fdr(p) = F_0(p)/F(p)$ and for ST: $fdr(p) = \pi_0(p)F_0(p)/F(p)$.
The ST method provides an often considerably larger amount of statistical power with respect to BH. Contrary to BH, it also allows a straightforward definition for the False Non Discovery Rate: $FnDR(p) = 1-\pi_0/\hat{\pi}_0(p)$ where $\hat{\pi_0}(p) = \frac{1-F(p)}{1-F_0(p)}$. On the downside, ST is only implemented against a uniform null distribution: a problem in some cases, when flat null distributions are not a good idea.
But what exactly do FDR and FnDR mean? If you look at the table below, the test correctly deemed significant are given by a. The ones correctly deemed not-significant are given by d. Type I errors (not-significant test called significant) in this case are given by b and type II (significant test called not-significant) by c. The value of the π_0 used before in this terms would be $\pi_0 = b + d$. Finally, the FDR is defined by the fraction of type I errors one makes after calling significant everything under a give p-value threshold, i.e. $fdr = b/(a + b)$. The FnDR is the fraction of type II errors one makes after calling non-significant everything above a given p-value threshold: $fndr = c/(d + c)$.

	True alternative hypothesis H_A	**True null hypothesis H_0**
Called significant	a	b
Called not-significant	c	d

Tabla 1: Significance elements through true null and alternative hypothesis.

False non discovery rates should be used for defining condition-specific tests: e.g. a gene is condition specific if we can confidently assure we can accept it as an alternative hypothesis under condition A (i.e. it is under FDR threshold at A), and, at the same time, we can confidently reject it at condition B (It is under FnDR threshold at B). This will be used later when the

definition of directionality is imposed.

5. Implementation of balanced-SNPs algorithm

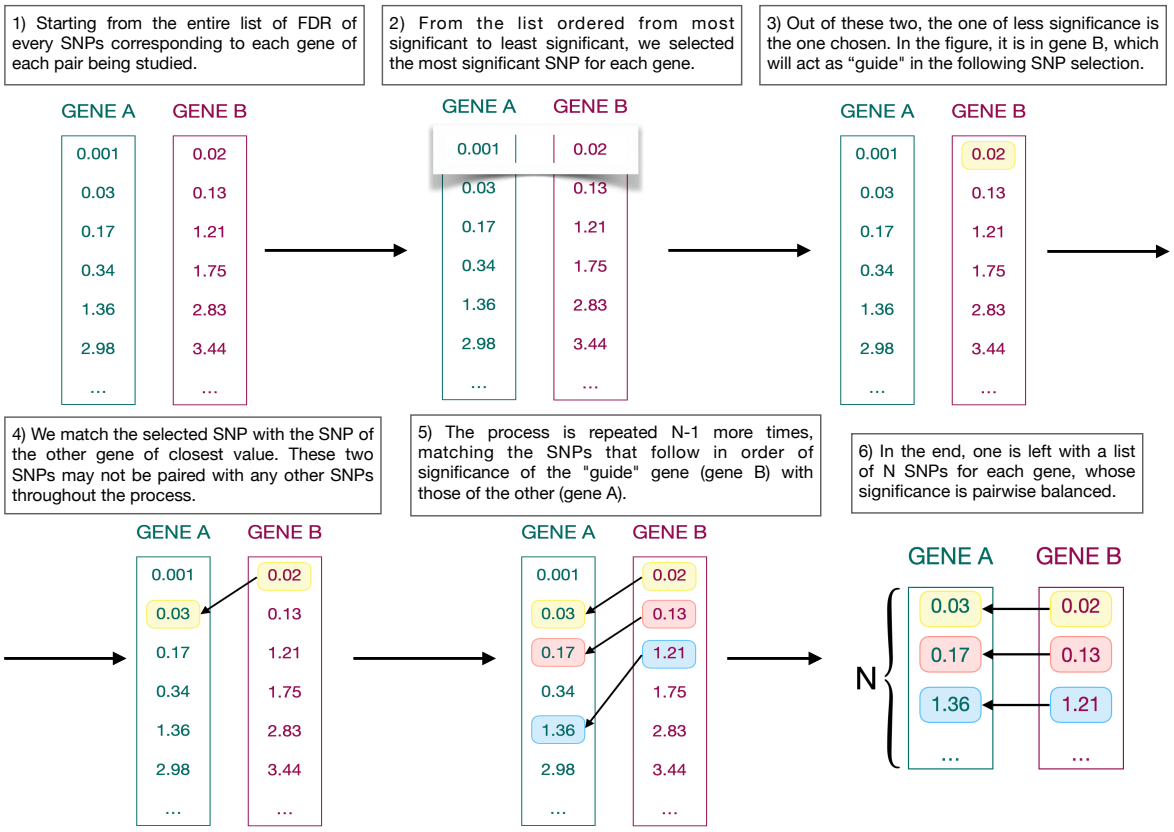
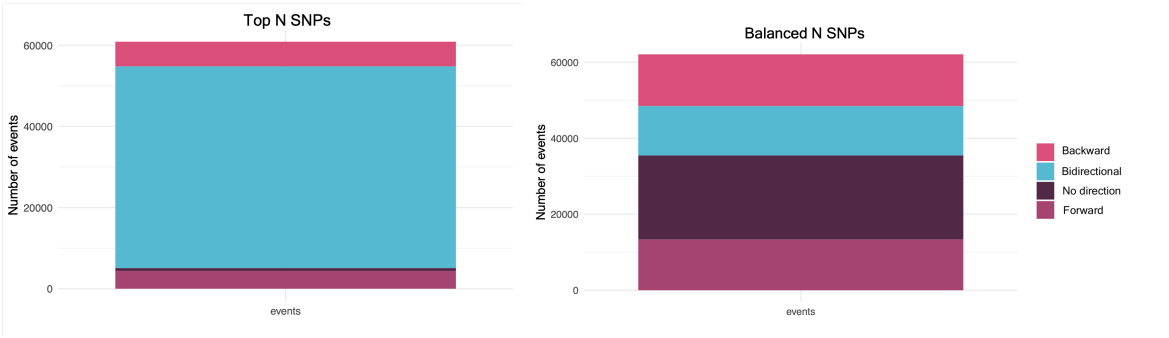


Figura 6: Process of balancing SNPs FDR for each pair of genes.

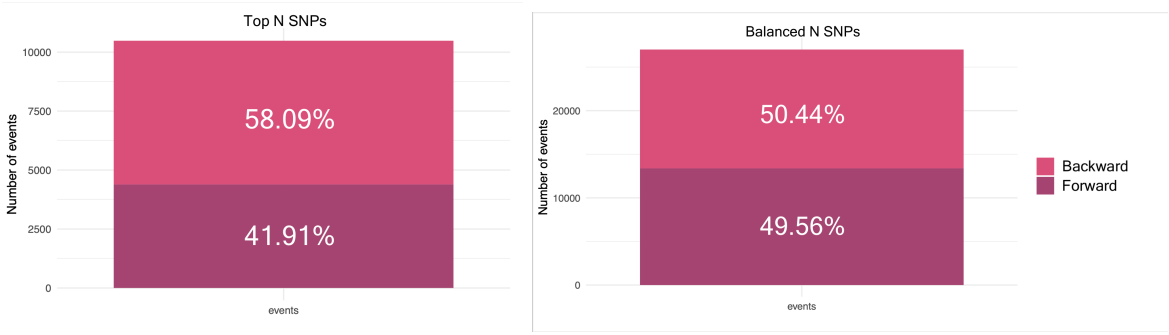
6. Link direction inference under different genetic variable selection criteria

The results presented in the report are reproduced by changing the forward/backward linkage definition strategy. In this case, instead of comparing the two most significant SNPs of each pair, we took the logarithmic sum of the significance of the N SNPs of each gene. These results are similar in magnitude to those presented in the main paper, noting again that the bias given by the balanced-SNPs algorithm is much smaller.



(a) Top SNPs (b) Balanced SNPs

Figura 7: Fraction of links of defined direction results using the mean of each N SNP to discern hypothesis null or alternative.



(a) Top SNPs (b) Balanced SNPs

Figura 8: Fraction of links of defined forward/backward using the mean of each N SNP to discern hypothesis null or alternative.

7. Codes

7.1. Prepare input tables

```
#################################################################
#### 1. Load dependencies ######################################
#################################################################

first_run=0
{
    ## bioMart is needed to get genes biotypes (identify protein coding genes)
    ## SNPRelate is needed to calcullate genotype Principal components.
    library(edgeR)
    library(limma)
    library(biomaRt)
    library(SNPRelate)

    ul=function(tab,n=5)
       {
           rs=min(nrow(tab),n) #Coge valor mínimo entre 5 y el número de filas que hay
           cs=min(ncol(tab),n) #Coge valor mínimio entre 5 y el número de columnas que hay
           return(tab[1:rs,1:cs]) #Devuelve la matriz desde 1 hasta el numero minimo anterior para
               columnas y filas
       }
    ur=function(tab,n=5)
       {
           rs=min(nrow(tab),n)
           cs=min(ncol(tab),n)
           return(tab[1:rs,(ncol(tab)-cs):ncol(tab)]) #¿Para qué sirve? ¿Por qué 5?
       }
       dcols=function(tab){data.frame(colnames(tab))} #Guarda los nombres de las columnas
       drows=function(tab){data.frame(rownames(tab))} #Guarda los nombres de las filas
}
```

```
#################################################################################################################
#### 1. Load reads and metadata, subset relevant samples for our needs & make sample orders concurrent ####
#################################################################################################################
    ######

{
    ## Load the set of individuals that passed initial QC steps.
    item=load("inputs/raw/individuals_sets.Rdata")
    individuals=individuals_set$no_dup
    ##94 guys.


    ## Load the metadata; and subset only the samples corresponding to those guys, at time=2h and
        condition="NI".
    ## As you see we will be looking to only a subset of this rather big dataset.
    metadata = read.table("inputs/raw/samples_data.txt")
    metadata=metadata[which(metadata$TimePoint=="02h" & metadata$Treatment=="NI"),]
    rownames(metadata)=paste0(metadata$Individual,"_",metadata$Treatment,"_",metadata$TimePoint)
    ## In this condition/time, only 90 out of the 94 individuals were represented.

    ## Load the raw expression matrix; then subset the individuals;
    reads = read.table("inputs/raw/reads_Regina.txt",header=TRUE,check.names=FALSE)
    reads=reads[,which(colnames(reads) %in% rownames(metadata))]

    dim(reads)
    dim(metadata)

    reads=reads[,order(colnames(reads))]
    metadata=metadata[order(rownames(metadata)),]

    reads=reads[,order(colnames(reads))]
    metadata=metadata[order(rownames(metadata)),]
```

```
######################################################################
#### 2. Load genes dictionary; filter protein coding,autosomal ####
####    genes   &  translate  unfriendly  gene  names to HUGOs ####
######################################################################

{
    genes = read.table("inputs/raw/genes_data.txt",header = TRUE, stringsAsFactors = FALSE)
    if(first_run==0)
    {
    human <- useMart("ENSEMBL_MART_ENSEMBL", dataset="hsapiens_gene_ensembl");
    genesV2 = getLDS(attributes = c("ensembl_gene_id"), filters = "ensembl_gene_id", values =
        as.character(genes$gene_id) , mart = human, attributesL = c("gene_biotype"), martL = human,
        uniqueRows=T)

    from_biomart=list(mart=human,LDS=genesV2)
    system("mkdir -p inputs/processed")
    save(from_biomart, file = "inputs/processed/biomart_data.RData")
    }else{
        load("inputs/processed/biomart_data.RData")
        human=from_biomart$mart
        genesV2=from_biomart$LDS
    }



    protein_coding_genes=genesV2$Gene.stable.ID[which(genesV2$Gene.type=="protein_coding")]

    genes=genes[which(genes$gene_id %in% protein_coding_genes),]
    reads=reads[which(rownames(reads) %in% protein_coding_genes),]

    genes=genes[order(genes$gene_id),]
    reads=reads[order(rownames(reads)),]
    length(which(rownames(reads)!=genes$gene_id))




    length(unique(genes$gene_name))
    ## There is some non-unique gene_names; this often happens, let us define a label ID that will
        be the gene name for the first gene_name appearance, and, for the repeated ones, the ENSEMBL
        IDs.
    genes$label=genes$gene_name
    genes$label[which(duplicated(genes$gene_name))]=genes$gene_id[which(duplicated(genes$gene_name))]
    rownames(reads)=genes$label
    rownames(genes)=genes$label

    ## And order alphabetically both so it looks nice.
    reads=reads[order(rownames(reads)),]
    genes=genes[order(genes$label),]
    length(which(rownames(reads)!=genes$label))

    ## In this dictionary file I also have the positions of the genes, which will be needed to do
        the EQTL mapping step.

    ## Filter out non autosomal genes, and modify the chromosome notations:
    colnames(genes)[3]="chr"
    genes$chr=as.character(genes$chr)
    genes=genes[which(genes$chr %in% as.character(c(1:22))),]
    reads=reads[which(rownames(reads) %in% genes$label),]
    length(which(rownames(reads)!=genes$label))
    genes$chr=paste0("chr",genes$chr)
    ## Filter only the columns needed by matrixEQTL, in the order they are needed too.
    genes=genes[,c(7,3,4,5)]

}
```

```
################################################################################################
#### 3. Remove lowly expressed genes (and subset the genes list file accordingly) ####
################################################################################################

{
    {
    design=model.matrix(~1,data=metadata)
    dge <- DGEList(counts=reads)
    dge <- calcNormFactors(dge)
    v <- voom(dge,design,plot=FALSE)
    exp=v$E

    voom_x=apply(exp,1,mean)
    length(which(voom_x>1))
    # 10586
    }
    reads=reads[which(voom_x>1),]
    genes=genes[which(voom_x>1),]

    length(which(rownames(reads)!=genes$label))
}
```

```
################################################################################################
#### 4. Load genotype data and genotype positions files. Subset gentype data. ####
################################################################################################
{
  {
    snpspos = read.table(paste0("inputs/raw/SNP_positions.txt"),header = TRUE, stringsAsFactors =
        FALSE)
    rownames(snpspos)=snpspos$snp
    unique(snpspos$chr)
    ## Just to see that the file only contains snps in autosomes.
    gtypes = read.table(paste0("inputs/raw/SNP_genotypes_data.txt"),header = TRUE, stringsAsFactors
        = FALSE)

    gtypes=gtypes[,which(colnames(gtypes) %in% metadata$Individual)]
    gtypes=gtypes[,order(colnames(gtypes))]
    length(which(colnames(gtypes)!=metadata$Individual))
    ## We will have to change the order of the samples:
    gtypes=gtypes[,colnames(gtypes)]
    reads=reads[,order(as.character(metadata$Individual))]
    metadata=metadata[order(as.character(metadata$Individual)),]

    length(which(rownames(metadata)!=colnames(reads)))
    length(which(as.character(metadata$Individual)!=colnames(gtypes)))
    ## Everything is ok, so let us simplify sample notation, given that we will not analyze samples
        from other times or treatments here.
    ## We could filter out SNPs out from the cis-windows of the genes that we wantt o analyze, but
        it is easier doing that within the matrixEQTL run.
    ## Now, filter for minimum allele frequency:
    mafs=apply(gtypes,1,function(x){length(which(x>0))/length(x)})
    gtypes=gtypes[which(mafs>0.05),]
    snpspos=snpspos[which(mafs>0.05),]
    length(which(rownames(gtypes)!=rownames(snpspos)))
  }
```

```
##############################################################################################
#### 5. Get the genotype principal components and configure the useful metadata file ####
##############################################################################################
{
    samples=colnames(gtypes) #Individuos I_xx
    gtypes_pca=data.frame(snp_id=rownames(gtypes),gtypes) #Se crea una fila con el primer elemento
        snp_id y los siguientes I_xx
    system("mkdir -p inputs/aux") #Crea un directorio
    snpgdsCreateGeno("inputs/aux/GDS_genotypes.gds", genmat = as.matrix(gtypes_pca[, samples]),
        sample.id = unique(samples), snp.id = gtypes_pca$snp_id, snpfirstdim=TRUE)
    #Create a SNP genotype dataset from a matrix
    snpgdsSummary("inputs/aux/GDS_genotypes.gds")
    genofile <- snpgdsOpen("inputs/aux/GDS_genotypes.gds")
    pca <- snpgdsPCA(genofile) #Principal Component Analysis (PCA) on SNP genotype data
    tab <- data.frame(sample.id = pca$sample.id,
        PC1 = pca$eigenvect[,1],    # the first eigenvector
        PC2 = pca$eigenvect[,2], PC3 = pca$eigenvect[,3], PC4 = pca$eigenvect[,4], PC5 =
            pca$eigenvect[,5],
        stringsAsFactors = FALSE)
    metadata=cbind(metadata,tab)
    pca$varprop[1]*90
    # 1.568026 first PC does only explain 1.7% of total variance, 57% more variance than a random
        direction: there is no population structure.
    length(which(metadata$Individual!=tab$sample.id))
    metadata=cbind(Individual=tab$sample.id,Batch=metadata$Batch,tab[,c(2:6)])
    rownames(metadata)=tab$sample.id
    {
        colnames(reads)=metadata$Individual
        colnames(gtypes)=metadata$Individual
        rownames(metadata)=metadata$Individual
    }
}
```

```
#####################################################################
#### 6. Check the congruence & Write the procees input tables. ####
#####################################################################

{
    dim(reads)
    #[1] 10586    90
    dim(metadata)
    #[1] 90  7
    dim(gtypes)
    #[1] 6159725      90
    dim(genes)
    #[1] 10586     4
    dim(snpspos)
    #[1] 6159725       3

    ## Samples congruence
    length(which(colnames(reads)!=rownames(metadata)))
    length(which(colnames(reads)!=colnames(gtypes)))
    ## Genes congruence
    length(which(rownames(reads)!=rownames(genes)))
    ## SNPs congruence.
    length(which(rownames(gtypes)!=rownames(snpspos)))


    system("mkdir -p inputs/processed")
    write.table(reads,"inputs/processed/reads_ok.txt")
    write.table(metadata,"inputs/processed/metadata_ok.txt")
    write.table(genes,"inputs/processed/genes_ok.txt")
    write.table(snpspos,"inputs/processed/snpspos_ok.txt")
    write.table(gtypes,"inputs/processed/gtypes_ok.txt")

}
```

7.2. WCGNA

```
################################################
#### 1. Load dependencies ######################
################################################
{
    ## gdata is needed for the function of upper & lower triangles.
    ## cowplot aligns ggplot panels nicely.
    ## ggrepel has agorithm to add labels in data in ggplot plots a way that they repel each other.

    library(edgeR)
    library(limma)
    library(ggplot2)
    library(cowplot)
    library(gdata)
    library(ggrepel)

    ul=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,1:cs])
    }
    ur=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,(ncol(tab)-cs):ncol(tab)])
    }
    dcols=function(tab){data.frame(colnames(tab))}
    drows=function(tab){data.frame(rownames(tab))}

}



################################################
#### 2. Load reads and metadata, processed ####
################################################

{
    reads=read.table("inputs/processed/reads_ok.txt")
    metadata=read.table("inputs/processed/metadata_ok.txt")
}

#############################################################################
#### 3. Do log transformation, batch removal & variance stabilization ####
#############################################################################

{

    ##Get voom tranformed data
    design=model.matrix(~Batch,data=metadata)
    dge <- DGEList(counts=reads)
    dge <- calcNormFactors(dge)
    v <- voom(dge,design,plot=FALSE)
    write.table(design,paste0("outputs/2_coexpression/design.txt"))

    ## Do the fit of the y values (do not use v as is as input of lmfit: we just want to run least
        squares on the y matrix of log(cpm):

    exp=v$E
    write.table(exp,paste0("outputs/2_coexpression/exp.txt"))
    fit2=lmFit(exp,design)
    write.table(fit2,paste0("outputs/2_coexpression/fit2.txt"))

```

```
    ## Define the voom plot axis:
    voom_x=apply(log2(reads + 0.5),1,mean)
    voom_y=(fit2$sigma)^0.25
    voom_table=data.frame(x=voom_x,y=voom_y)


    ## Get the table with the lowess fit.
    span=0.5
    l <- data.frame(lowess(voom_table$x, voom_table$y, f = span))
    ## In some parts, you get two or three points identical until the last decimal figure printed:
    l=l[!duplicated(l$x),]
    write.table(l,paste0("outputs/2_coexpression/lowess_fit.txt"))

    ## This defines a function-like object that represents the lowess fit.
    tech_var_func=approxfun(l$x, l$y,rule=2)
    voom_table$tech_var=tech_var_func(voom_table$x)
    write.table(voom_table,paste0("outputs/2_coexpression/voom_table.txt"))
    pl_voom_raw=ggplot(voom_table)+geom_point(aes(x=x,y=y))+geom_line(aes
        (x=x,y=tech_var),color="red")+theme_minimal()+xlab("Mean expression")+ylab("")

    ## Now, we get the scaled residuals:
    aux=as.matrix(log2(reads + 0.5))
    scalings=aux
    for(i in 1:ncol(scalings))scalings[,i]=tech_var_func(aux[,i])^4
    res<-residuals(object=fit2, exp)
    write.table(res,paste0("outputs/2_coexpression/res.txt"))
    scaled_res=res/scalings
    write.table(scaled_res,paste0("outputs/2_coexpression/scaled_res.txt"))
```

```
    ## And reconstruct the expresion where real residuals are substituted with scaled ones:
    exp_prediction=exp-res
    exp_stabilized=exp_prediction+scaled_res

    ## Then, run the fit again, and retrieve the sigmas:
    fit3=lmFit(exp_stabilized,design)
    write.table(fit3,paste0("outputs/2_coexpression/fit3.txt"))
    voom_y_stabilized=sqrt(fit3$sigma)

    ## This is the same as before with the stabilized data
    voom_table_stabilized=data.frame(x=voom_x,y=voom_y_stabilized)
    #span=1
    l_stabilized <- data.frame(lowess(voom_table_stabilized$x, voom_table_stabilized$y, f = span))
    l_stabilized=l_stabilized[!duplicated(l_stabilized$x),]
    write.table(l_stabilized,paste0("outputs/2_coexpression/lowess_fit_estabilized.txt"))
    tech_var_func_stabilized=approxfun(l_stabilized$x, l_stabilized$y,rule=2)
    voom_table_stabilized$tech_var=tech_var_func_stabilized(voom_table_stabilized$x)
    write.table(voom_table_stabilized,paste0("outputs/2_coexpression/voom_table_stabilized.txt"))
    pl_voom_stabilized=ggplot(voom_table_stabilized)+geom_point(aes(x=x,y=y))+geom_line(aes
        (x=x,y=tech_var),color="red")+xlim(6,16)+theme_minimal()+xlab("Mean expression")+ylab("")
    voom_plots=plot_grid(pl_voom_raw,pl_voom_stabilized,ncol=2,align="h")

    ## And here we have the expression matrix once Batch effects have been removed and the variance
        has been stabilized: optimal setup for coexpression network reconstruction.

    exp=t(scaled_res) #Traspuesta
    write.table(exp,paste0("outputs/2_coexpression/exp.txt"))

}
```

```
#################################################################################################
#### 4. Get pairwise co-expression tests & network object, storing only the significant ones ####
#################################################################################################

{
    threshold_DC=0.01
    dofs=nrow(exp)-2-(ncol(design)-1)

    correlations_matrix <- cor(exp,method="pearson")
    write.table(correlations_matrix,paste0("outputs/2_coexpression/correlations_matrix.txt"))
    t_matrix=sqrt(dofs)*correlations_matrix/(sqrt(1-correlations_matrix*correlations_matrix))
    write.table(t_matrix,paste0("outputs/2_coexpression/t_matrix.txt"))
    p_value_matrix=2*pt(-abs(t_matrix),dofs)
    write.table(p_value_matrix,paste0("outputs/2_coexpression/p_value_matrix.txt"))
    p_value_vector=upperTriangle(p_value_matrix)
    write.table(p_value_vector,paste0("outputs/2_coexpression/p_value_vector.txt"))

    # From BH to ST
    library(qvalue)
    fdrs_vector=qvalue(p_value_vector)$qvalues
    #fdrs_vector=p.adjust(p_value_vector,method="BH")
    ## Benjamini-Hochberg

    fdrs_matrix=p_value_matrix
    upperTriangle(fdrs_matrix)=fdrs_vector
    fdrs_matrix=t(fdrs_matrix)
    upperTriangle(fdrs_matrix)=fdrs_vector
    fdrs_matrix=t(fdrs_matrix)

    rownames(fdrs_matrix)=rownames(p_value_matrix)
    colnames(fdrs_matrix)=colnames(p_value_matrix)



    network=data.frame(which(fdrs_matrix<threshold_DC,arr.ind=TRUE))
    network=network[which(network$row<network$col),]

    network$gen_row=rownames(fdrs_matrix)[network$row]
    network$gen_col=rownames(fdrs_matrix)[network$col]

    load_data=function(x,y){correlations_matrix[x,y]}
    network$r=mapply(load_data,x=network$row,y=network$col)

    load_data=function(x,y){t_matrix[x,y]}
    network$t=mapply(load_data,x=network$row,y=network$col)

    load_data=function(x,y){p_value_matrix[x,y]}
    network$p=mapply(load_data,x=network$row,y=network$col)

    load_data=function(x,y){fdrs_matrix[x,y]}
    network$fdr=mapply(load_data,x=network$row,y=network$col)

    network=network[order(network$p),]
    rownames(network)=paste0(network$gen_row,"_",network$gen_col)
    involved=unique(c(network$gen_row,network$gen_col))

    length(involved)
    # 10033

    exp_sub=exp[,which(colnames(exp) %in% involved)]
    scaled_res_sub=scaled_res[which(rownames(scaled_res) %in% involved),]

    pairwise_matrixes=list(correlations_matrix=correlations_matrix,
         t_matrix=t_matrix, p_value_matrix=p_value_matrix,
         fdrs_matrix=fdrs_matrix,dofs=dofs)
}

```

```
##############################
#### 5. Get WGCNA modules ####
##############################

if(FALSE){
    # Call the network topology analysis function
    sft = pickSoftThreshold(exp_sub, powerVector = seq(2,30,by=2), blockSize = ncol(exp_sub),verbose = 5)
    pow=25
    # Plot the results:
    power_tab=sft$fitIndices
    power_tab$label=""
    power_tab$label[which(power_tab$SFT.R.sq>0.85)[1]]=as.character(power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]])


    pl1=ggplot(power_tab)+geom_point(aes(x=Power,y=SFT.R.sq))+geom_line(aes(x=Power,y=SFT.R.sq))+
        geom_hline(yintercept=0.85)+geom_vline(xintercept=power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]])+
        geom_text_repel(aes(x=Power,y=SFT.R.sq,label=label))+
        ggtitle(paste0("Scale freedom (pow.est=",power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]],")"))

    power_tab$label[which(power_tab$SFT.R.sq>0.85)[1]]=paste0("Power=",as.character(power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]]),", <k>=",
        as.character(round(power_tab$mean.k.[which(power_tab$SFT.R.sq>0.85)[1]])))



    pl2=ggplot(power_tab)+geom_point(aes(x=Power,y=mean.k.))+geom_line(aes(x=Power,y=mean.k.))+
        geom_hline(yintercept=0.85)+geom_vline(xintercept=power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]])+
        geom_text_repel(aes(x=Power,y=mean.k.,label=label),vjust=1,hjust=1)+
        ggtitle(paste0("Mean degree (pow.est=",power_tab$Power[which(power_tab$SFT.R.sq>0.85)[1]],")"))
    pl_WGCNA=plot_grid(pl1,pl2,ncol=2,align="h")


    ## This, with saveTOMs=TRUE will save the TOM file into the desired folder, which is this one:
    system("mkdir -p outputs/2_coexpression/")
    net = blockwiseModules(exp_sub, power = pow, TOMType = "unsigned", minModuleSize = 30,
        reassignThreshold = 0, mergeCutHeight = 0.25, numericLabels = TRUE, pamRespectsDendro = FALSE,
        maxBlockSize=length(involved)+1, saveTOMs = TRUE, saveTOMFileBase = "outputs/2_coexpression/",
            verbose = 3)

    ## Let´s put exp_sub back to the more normal position:
    exp_sub=data.frame(t(exp_sub))
}
```

```
##########################
#### 6. Save results ####
##########################

{
    ### Variance stabilized and batch-free expression matrix.
    write.table(scaled_res_sub,paste0("outputs/2_coexpression/stabilized_expression.txt"))

    ## Pairwise objects and network table:
    save(pairwise_matrixes,file="outputs/2_coexpression/pairwise_matrixes.Rdata")
    write.table(network,paste0("outputs/2_coexpression/network",format(threshold_DC,digits=2),".txt"))

    ## WGCNA net object containing the modules & some plots from the WGCNA analyses
    # JQ lo pongo dentro del condicional
    if(modules)
    {
        save(net,file="outputs/2_coexpression/WGCNA_net.Rdata")

        pdf("outputs/2_coexpression/WGCNA_plots.pdf",width=7,height=4)
        print(pl_WGCNA)
        dev.off()
    }

    genes=read.table("inputs/processed/genes_ok.txt")
    ### Get only the genes that appeared in the coexp. network:
    genes=genes[which(rownames(genes) %in% colnames(exp_sub)),]
    length(which(rownames(genes)!=colnames(exp_sub)))
    length(which(rownames(genes)==colnames(exp_sub)))

    write.table(genes,paste0("outputs/2_coexpression/coexpressed_genes.txt"))

}
```

7.3. Identify eQTLs

```
################################################################################
#### 1. Load dependencies ######################################################
################################################################################

{
    library(MatrixEQTL)
    library(edgeR)
    library(limma)
    library(gdsfmt)
    library(SNPRelate)
    library(qvalue)
    library(stats)
    library(stringr)
    library(data.table)

    ul=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,1:cs])
    }
    ur=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,(ncol(tab)-cs):ncol(tab)])
    }
    dcols=function(tab){data.frame(colnames(tab))}
    drows=function(tab){data.frame(rownames(tab))}

}

```

```
################################################################################
#### 2. Load items needed for EQTL mapping #####################################
################################################################################
{
    SNPs_positions_file_name="inputs/processed/snpspos_ok.txt"
    expression_file_name="outputs/2_coexpression/stabilized_expression.txt"
    genes_file_name="inputs/processed/genes_ok.txt"
    Genotypes_file_name="inputs/processed/gtypes_ok.txt"
}

################################################################################
#### 3. Check input items congruences ##########################################
################################################################################
{
    snpspos=read.table(SNPs_positions_file_name)
    gtypes=read.table(Genotypes_file_name)
    genes=read.table(genes_file_name)
    expression=read.table(expression_file_name)
    dim(expression)
    #[1] 10033    90
    dim(gtypes)
    #[1] 6159725      90
    dim(genes)
    #[1] 10033     4
    dim(snpspos)
    #[1] 6159725       3
    ## Some checks:
    ## Samples wise
    length(which(colnames(expression)==colnames(gtypes)))/length(colnames(expression))
    ## Genes wise:
    length(which(rownames(expression)==rownames(genes)))/length(rownames(expression))
    ## SNPs-wise (we could filter these out using the positions, but we will leave matrixEQTL do it)
    length(which(rownames(snpspos)==rownames(gtypes)))/length(rownames(snpspos))
}



################################################################################
### 4. Build matrixEQTL inputs #################################################
################################################################################

{
    ## Matrix EQTL needs its inputs in .txt input tables saved in a particular format:
    write.table(expression,expression_file_name, quote=F, sep="\t", row.names=TRUE)
    write.table(genes,genes_file_name, quote=F, sep="\t", row.names=TRUE)
    write.table(snpspos,SNPs_positions_file_name, quote=F, sep="\t", row.names=TRUE)
    write.table(gtypes,Genotypes_file_name, quote=F, sep="\t", row.names=TRUE)
}

################################################################################
#### 5. Configure & Run Matrix for cisEQTL #####################################
################################################################################

{
    gene = SlicedData$new();
    gene$fileDelimiter = "\t";     # the TAB character
    gene$fileOmitCharacters = "NA"; # denote missing values;
    gene$fileSkipRows = 1;          # one row of column labels
    gene$fileSkipColumns = 1;       # one column of row labels
    gene$fileSliceSize = 2000;      # read file in slices of 2,000 rows
    gene$LoadFile(expression_file_name);

    ## Load covariates: there is none, so we declare an empty cov. file:

```

```
covariates_file_name=character()
cvrt = SlicedData$new();
cvrt$fileDelimiter = "\t";      # the TAB character
cvrt$fileOmitCharacters = "NA"; # denote missing values;
cvrt$fileSkipRows = 1;          # one row of column labels
cvrt$fileSkipColumns = 1;       # one column of row labels
if(length(covariates_file_name)>0) {
    cvrt$LoadFile(covariates_file_name);
}

## Load genotype data

snps = SlicedData$new();
snps$fileDelimiter = "\t";      # the TAB character
snps$fileOmitCharacters = "NA";
snps$fileOmitCharacters = "-9" # denote missing values;
snps$fileSkipRows = 1;          # one row of column labels
snps$fileSkipColumns = 1;       # one column of row labels
snps$fileSliceSize = 2000;      # read file in slices of 2,000 rows
snps$LoadFile(Genotypes_file_name)

## Set up further program parameters
useModel = modelLINEAR
output_file_name_cis = tempfile()
pvOutputThreshold_cis = 1
pvOutputThreshold = 0;
errorCovariance = numeric()
cisDist = 1e5
output_file_name = tempfile()
output_file_name_cis = tempfile()


me = Matrix_eQTL_main(snps = snps, gene = gene, cvrt = cvrt, output_file_name = output_file_name,
    useModel = useModel,
    errorCovariance = errorCovariance, verbose = TRUE, output_file_name.cis = output_file_name_cis,
    pvOutputThreshold = pvOutputThreshold, pvOutputThreshold.cis = pvOutputThreshold_cis, snpspos
        = snpspos,
    genepos = genes, cisDist = cisDist, pvalue.hist = "qqplot", min.pv.by.genesnp = TRUE,
        noFDRsaveMemory = FALSE);
cis_eqtls=me$cis$eqtls
# 5766136       6

## Filter out tests corresponding to SNPs in perfect linkage disequilibrium.

elemental=function(x){
  paste0(x,collapse="")
}

gtypes_collapsed=apply(gtypes,1,elemental)
snpspos$gtypes_collapsed=gtypes_collapsed

revert=function(word){
word=str_replace_all(word,"0","3")
word=str_replace_all(word,"2","0")
word=str_replace_all(word,"3","2")
return(word)
}

snpspos$gtypes_collapsed_reversed=revert(snpspos$gtypes_collapsed)

aux=snpspos[,c(1,4,5)]
colnames(aux)[1]="snps"
result=merge(cis_eqtls, aux, by="snps")
result$test_ID=paste0(result$gtypes_collapsed,"_",result$gene)
result$test_ID_reversed=paste0(result$gtypes_collapsed_reversed,"_",result$gene)
# 5766136      10
```

```
161
162     result=result[which(!duplicated(result$test_ID)),]
163     # 2477579      10
164
165     result=result[which(!result$test_ID %in% result$test_ID_reversed),]
166     # 2409953      10
167
168     result$FDR_ok=qvalue(result$pvalue)$qvalues
169     cis_eqtls=result
170     # 2409953      11
171
172 }
173
```

```
174 ###############################################################################
175 #### 6. Run again in trans for conjugated EQTLs: algorithm top ##############
176 ###############################################################################
177
178 {
179     ## First: select genes with at least one EQTL at 5% FDR
180
181     threshold=0.05
182     hits=cis_eqtls[which(cis_eqtls$FDR_ok<threshold),]
183     genes_with_at_least_one_eqtl=unique(hits$gene)
184     length(genes_with_at_least_one_eqtl)
185     # 815
186     cis_useful=cis_eqtls[which(cis_eqtls$gene %in% genes_with_at_least_one_eqtl),]
187     cis_useful=cis_useful[order(cis_useful$gene,cis_useful$FDR_ok),]
188     cis_useful=cis_useful[,c(1,2,3,4,11,6)]
189     # 280078  6
190     # cis_useful contains all cis-tests pointing to genes for which at least one EQTL at FDR=5% is
          found (815 of such genes)
191
192     ## Let's pick up the top-3 most significant EQTLs per gene.
193     N=3
194
195     first_instances=cis_useful[which(!(duplicated(cis_useful$gene))),]
196     resto=cis_useful[which((duplicated(cis_useful$gene))),]
197
198     for(iter in 2:N){
199         appendix=resto[which(!(duplicated(resto$gene))),]
200         if(nrow(appendix)<length(genes_with_at_least_one_eqtl))
201         {
202             print(paste("In iter=",iter," some genes lack SNPs"))
203         }
204         first_instances=rbind(first_instances,appendix)
205         resto=resto[which((duplicated(resto$gene))),]
206     }
```

```r
top_cis_snps=first_instances[order(first_instances$gene,first_instances$FDR_ok),]
dim(top_cis_snps)
# [1] 2445    6

length(unique(top_cis_snps$snps))
## 2387: less bc. some SNPs are EQTL of more than 1 nearby gene

snpspos_trans_subset_top=snpspos[which(snpspos$snp %in% top_cis_snps$snps),c(1:3)]
dim(snpspos_trans_subset_top)
#[1] 2387    3

# Let us subset the genotype tables to contain only those 2387 SNPs
gtypes_trans_subset_top=gtypes[which(rownames(gtypes) %in% top_cis_snps$snps),]
#[1] 2387   90

# And the expression and gene-data tables to contain only those 815 genes.
genes_trans_subset_top=genes[which(rownames(genes) %in% top_cis_snps$gene),]
#[1] 815   4
expression_trans_subset_top=expression[which(rownames(expression) %in% top_cis_snps$gene),]
#[1] 815   90

## Check the congruence of these new arguments:
## Sample-wise
length(which(colnames(expression_trans_subset_top)==colnames(gtypes_trans_subset_top)))/length
    (colnames(expression_trans_subset_top))
## Genes wise:
length(which(rownames(expression_trans_subset_top)==rownames(genes_trans_subset_top)))/length
    (rownames(expression_trans_subset_top))
## SNPs-wise (we could filter these out using the positions, but we will leave matrixEQTL do it)
length(which(rownames(snpspos_trans_subset_top)==rownames(gtypes_trans_subset_top)))/length
    (rownames(snpspos_trans_subset_top))
```

```r
## Write them in matrix-EQTL-friendly format
system("mkdir -p outputs/3_EQTLs/trans_subset_inputs/top")

trans_subset_top_SNPs_positions_file_name="outputs/3_EQTLs/trans_subset_inputs/top/snpspos_ok.txt"
trans_subset_top_expression_file_name="outputs/3_EQTLs/trans_subset_inputs/top/stabilized_expressio
    n.txt"
trans_subset_top_genes_file_name="outputs/3_EQTLs/trans_subset_inputs/top/coexpressed_genes.txt"
trans_subset_top_Genotypes_file_name="outputs/3_EQTLs/trans_subset_inputs/top/gtypes_ok.txt"


write.table(expression_trans_subset_top,trans_subset_top_expression_file_name, quote=F, sep="\t",
    row.names=TRUE)
write.table(genes_trans_subset_top,trans_subset_top_genes_file_name, quote=F, sep="\t",
    row.names=TRUE)
write.table(snpspos_trans_subset_top,trans_subset_top_SNPs_positions_file_name, quote=F, sep="\t",
    row.names=TRUE)
write.table(gtypes_trans_subset_top,trans_subset_top_Genotypes_file_name, quote=F, sep="\t",
    row.names=TRUE)

#### Now, Configure & Run Matrix EQTL

gene = SlicedData$new();
gene$fileDelimiter = "\t";     # the TAB character
gene$fileOmitCharacters = "NA"; # denote missing values;
gene$fileSkipRows = 1;          # one row of column labels
gene$fileSkipColumns = 1;       # one column of row labels
gene$fileSliceSize = 2000;      # read file in slices of 2,000 rows
gene$LoadFile(trans_subset_top_expression_file_name);

## Load covariates: there is none, so we declare an empty cov. file:
```

```
    trans_subset_top_covariates_file_name=character()
    cvrt = SlicedData$new();
    cvrt$fileDelimiter = "\t";      # the TAB character
    cvrt$fileOmitCharacters = "NA"; # denote missing values;
    cvrt$fileSkipRows = 1;          # one row of column labels
    cvrt$fileSkipColumns = 1;       # one column of row labels
    if(length(covariates_file_name)>0) {
      cvrt$LoadFile(trans_subset_top_covariates_file_name);
    }

    ## Load genotype data

    snps = SlicedData$new();
    snps$fileDelimiter = "\t";      # the TAB character
    snps$fileOmitCharacters = "NA";
    snps$fileOmitCharacters = "-9" # denote missing values;
    snps$fileSkipRows = 1;          # one row of column labels
    snps$fileSkipColumns = 1;       # one column of row labels
    snps$fileSliceSize = 2000;      # read file in slices of 2,000 rows
    snps$LoadFile(trans_subset_top_Genotypes_file_name)

    useModel = modelLINEAR
    output_file_name_cis = tempfile()
    pvOutputThreshold_cis = 1
    pvOutputThreshold = 1;
    errorCovariance = numeric()
    cisDist = 1e5
    output_file_name = tempfile()
    output_file_name_cis = tempfile()


    conjugated = Matrix_eQTL_main(snps = snps,gene = gene, cvrt = cvrt, output_file_name =
        output_file_name, useModel = useModel, errorCovariance = errorCovariance, verbose = TRUE,
        output_file_name.cis = output_file_name_cis, pvOutputThreshold = pvOutputThreshold,
        pvOutputThreshold.cis = pvOutputThreshold_cis, snpspos = snpspos_trans_subset_top, genepos =
        genes_trans_subset_top, cisDist = cisDist, pvalue.hist = "qqplot", min.pv.by.genesnp = TRUE,
        noFDRsaveMemory = FALSE);

    trans_eqtls_top=conjugated$trans$eqtls
    #1942163       6
    ## JQ por claridad
    cis_eqtls_conjugated_top=conjugated$cis$eqtls
    #[1] 3242    6
    #so, we now have that 1942163+3242 (conjugated EQTLS labelled as trans plus cis (a minority,
        obviously: cases here the SNP lies, by chance, nearby the conjugated gene)) = 815 * 2387
        (genes times snps tested)

}
```

```
##############################################################################
#### 7. Run again in trans for conjugated EQTLs: algorithm balanced #########
##############################################################################

{

  ## Load the table of cis_EQTLs involcing genes with at least one hit (termed genes under genetic
      control).
  cis=cis_eqtls[which(cis_eqtls$gene %in% top_cis_snps$gene),]
  cis=cis[,c(1,2,11)]
  colnames(cis)[3]="FDR"
  cis=cis[order(cis$gene,cis$FDR),]

  ## Load network
  network=fread(paste0("outputs/2_coexpression/network0.01.txt"))
  rownames(network)=network$V1
  colnames(network)[1]="link"
  red=network
  red=red[,c("gen_row","gen_col")]
  N=3

  ## Subset network links connecting pairs of genes under genetic control.
  red=red[which(red$gen_row %in% top_cis_snps$gene),]
  red=red[which(red$gen_col %in% top_cis_snps$gene),]


  ## For each link, we will use 3 SNPs per gene that will be, in general link-specific (i.e. we will
      allow that  the same gene uses different cis-EQTLs in different links if that helps to reach
      better significance-balance sets of SNPS). Here, I declare container table to store the
      identities and stats of these EQTLs.



  final=data.frame(
  gene_one=rep(NA,nrow(red)*N),gene_two=rep(NA,nrow(red)*N),
  snp_one=rep(NA,nrow(red)*N),snp_two=rep(NA,nrow(red)*N),
  FDR_one=rep(0,nrow(red)*N),FDR_two=rep(0,nrow(red)*N)
  )

  ## This loops runs the balancing algorithm

  for(j in 1:nrow(red))
  {
      if(j%%1000==0) print(j)
      aux_row=cis[which(cis$gene == red$gen_row[j]),]
      aux_row=aux_row[order(aux_row$FDR),]
      aux_col=cis[which(cis$gene == red$gen_col[j]),]
      aux_col=aux_col[order(aux_col$FDR),]
      aux_row$top=0
      aux_col$top=0

      posicion_row=1
      posicion_col=1
      for(i in 1:N){
          if(aux_row$FDR[posicion_row]<=aux_col$FDR[posicion_col])
          {
              guide_posicion=posicion_col
              match_posicion=which.min(abs(-log10(aux_row$FDR[(posicion_row+1):nrow(aux_row)]) +
                  log10(aux_col$FDR[posicion_col])))+posicion_row
              aux_col$top[guide_posicion]=i
              aux_row$top[match_posicion]=i
              posicion_col=guide_posicion+1
              posicion_row=match_posicion+1
          }else{
```

```
            guide_posicion=posicion_row
            match_posicion=which.min(abs(-log10(aux_col$FDR[(posicion_col+1):nrow(aux_col)]) +
                log10(aux_row$FDR[posicion_row])))+posicion_col
            aux_row$top[guide_posicion]=i
            aux_col$top[match_posicion]=i
            posicion_row=guide_posicion+1
            posicion_col=match_posicion+1
          }
        }

    aux_row=aux_row[which(aux_row$top>0),]
    aux_col=aux_col[which(aux_col$top>0),]
    appendix=cbind(aux_row,aux_col)[,c(2,6,1,5,3,7)]
    colnames(appendix)=c("gene_one","gene_two","snp_one","snp_two","FDR_one","FDR_two")

    final[(N*j-N+1):(N*j),]<-appendix

    }
system("mkdir -p outputs/3_EQTLs/trans_subset_inputs/match")
write.table(final,"outputs/3_EQTLs/trans_subset_inputs/match/SNPs_pairings.txt")

# Now, we gather all the SNPs appearing in the result table of the SNPs_balancing alhgorithm to then
    run trans-EQTL mapping.

snpspos_trans_subset_match=snpspos[which(snpspos$snp %in%
    unique(c(final$snp_one,final$snp_two))),c(1:3)]
gtypes_trans_subset_match=gtypes[which(rownames(gtypes) %in%
    unique(c(final$snp_one,final$snp_two))),]
genes_trans_subset_match=genes[which(rownames(genes) %in% unique(c(final$gene_one,final$gene_two))),]
expression_trans_subset_match=expression[which(rownames(expression) %in%
    unique(c(final$gene_one,final$gene_two))),]
## 31571 unique SNPS involved (many appear several times), on 815 genes, 90 muestras



## Check the congruence of these new arguments:
## Sample-wise
length(which(colnames(expression_trans_subset_match)==colnames(gtypes_trans_subset_match)))/length
    (colnames(expression_trans_subset_match))
## Genes wise:
length(which(rownames(expression_trans_subset_match)==rownames(genes_trans_subset_match)))/length
    (rownames(expression_trans_subset_match))
## SNPs-wise (we could filter these out using the positions, but we will leave matrixEQTL do it)
length(which(rownames(snpspos_trans_subset_match)==rownames(gtypes_trans_subset_match)))/length
    (rownames(snpspos_trans_subset_match))

system("mkdir -p outputs/3_EQTLs/trans_subset_inputs/match")

trans_subset_match_SNPs_positions_file_name="outputs/3_EQTLs/trans_subset_inputs/match/snpspos_ok
    .txt"
trans_subset_match_expression_file_name="outputs/3_EQTLs/trans_subset_inputs/match/stabilized_express
    ion.txt"
trans_subset_match_genes_file_name="outputs/3_EQTLs/trans_subset_inputs/match/coexpressed_genes.txt"
trans_subset_match_Genotypes_file_name="outputs/3_EQTLs/trans_subset_inputs/match/gtypes_ok.txt"


write.table(expression_trans_subset_match,trans_subset_match_expression_file_name, quote=F,
    sep="\t", row.names=TRUE)
write.table(genes_trans_subset_match,trans_subset_match_genes_file_name, quote=F, sep="\t",
    row.names=TRUE)
write.table(snpspos_trans_subset_match,trans_subset_match_SNPs_positions_file_name, quote=F,
    sep="\t", row.names=TRUE)
write.table(gtypes_trans_subset_match,trans_subset_match_Genotypes_file_name, quote=F, sep="\t",
    row.names=TRUE)

```

```
  #### Configure & Run Matrix EQTL ############################

  gene = SlicedData$new();
  gene$fileDelimiter = "\t";     # the TAB character
  gene$fileOmitCharacters = "NA"; # denote missing values;
  gene$fileSkipRows = 1;          # one row of column labels
  gene$fileSkipColumns = 1;       # one column of row labels
  gene$fileSliceSize = 2000;      # read file in slices of 2,000 rows
  gene$LoadFile(trans_subset_match_expression_file_name);

  ## Load covariates: there is none, so we declare an empty cov. file:

  trans_subset_match_covariates_file_name=character()
  cvrt = SlicedData$new();
  cvrt$fileDelimiter = "\t";      # the TAB character
  cvrt$fileOmitCharacters = "NA"; # denote missing values;
  cvrt$fileSkipRows = 1;          # one row of column labels
  cvrt$fileSkipColumns = 1;       # one column of row labels
  if(length(covariates_file_name)>0) {
    cvrt$LoadFile(trans_subset_match_covariates_file_name);
  }

  ## Load genotype data

  snps = SlicedData$new();
  snps$fileDelimiter = "\t";      # the TAB character
  snps$fileOmitCharacters = "NA";
  snps$fileOmitCharacters = "-9" # denote missing values;
  snps$fileSkipRows = 1;          # one row of column labels
  snps$fileSkipColumns = 1;       # one column of row labels
  snps$fileSliceSize = 2000;      # read file in slices of 2,000 rows
  snps$LoadFile(trans_subset_match_Genotypes_file_name)



  useModel = modelLINEAR
  output_file_name_cis = tempfile()
  pvOutputThreshold_cis = 1
  pvOutputThreshold = 1;
  errorCovariance = numeric()
  cisDist = 1e5
  output_file_name = tempfile()
  output_file_name_cis = tempfile()

  conjugated_match = Matrix_eQTL_main(snps = snps,gene = gene, cvrt = cvrt, output_file_name =
      output_file_name, useModel = useModel, errorCovariance = errorCovariance, verbose = TRUE,
      output_file_name.cis = output_file_name_cis, pvOutputThreshold = pvOutputThreshold,
      pvOutputThreshold.cis = pvOutputThreshold_cis, snpspos = snpspos_trans_subset_match, genepos =
      genes_trans_subset_match, cisDist = cisDist, pvalue.hist = "qqplot", min.pv.by.genesnp = TRUE,
      noFDRsaveMemory = FALSE);

  trans_eqtls_match=conjugated_match$trans$eqtls
  # 25685872        6
  cis_eqtls_conjugated_match=conjugated_match$cis$eqtls
  # 44493     6

  #so, we now have that 25685872 + 44493 (conjugated EQTLS labelled as trans plus cis (a minority,
      obviously: cases here the SNP lies, by chance, nearby the conjugated gene)) = 815 * 31571 (genes
      times snps tested)

}
```

```
################################################################################
#### 8. Write output files ####################################################
################################################################################

{
    system("mkdir -p outputs/3_EQTLs/results/match/")
    system("mkdir -p outputs/3_EQTLs/results/top/")

    write.table(cis_eqtls, file = "outputs/3_EQTLs/results/cis.txt")

    write.table(top_cis_snps, file = "outputs/3_EQTLs/results/top/cis_top_N.txt")
    write.table(trans_eqtls_top, file = "outputs/3_EQTLs/results/top/conjugated_trans_top.txt")
    write.table(cis_eqtls_conjugated_top, file = "outputs/3_EQTLs/results/top/conjugated_cis_top.txt")


    write.table(final, file = "outputs/3_EQTLs/results/match/cis_balanced_N.txt")
    write.table(trans_eqtls_match, file = "outputs/3_EQTLs/results/match/conjugated_trans_match.txt")
    write.table(cis_eqtls_conjugated_match, file =
        "outputs/3_EQTLs/results/match/conjugated_cis_match.txt")

}
```

7.4. Mendelian Randomization for Top-SNPs method

```
################################################################
#### 1. Load dependencies ######################################
################################################################
{
    library(edgeR)
    library(limma)
    library(gdsfmt)
    library(SNPRelate)
    library(qvalue)
    library(stats)
    library(MendelianRandomization)
    library(data.table)
    library(cobs)
    library(ggplot2)
    library(cowplot)

    ul=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,1:cs])
    }
    ur=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,(ncol(tab)-cs):ncol(tab)])
    }
    dcols=function(tab){data.frame(colnames(tab))}
    drows=function(tab){data.frame(rownames(tab))}
}
```

```
################################################################
#### 2. Load and pretty up input tables ########################
################################################################

{
    ## Input # 1: Network (here I just load it; the formatting requires an entire step, which also
        needs the genes info table loaded here as well).
    network=fread("outputs/2_coexpression/network0.01.txt")
    colnames(network)[1]="link"
    nrow(network)
    # 25897510 pairs.
    genes=read.table(paste0("outputs/2_coexpression/coexpressed_genes.txt"))


    ## Input # 2: cis-EQTLs data. Load, rename FDR column, declare sd column

    cis_hits_top=fread("outputs/3_EQTLs/results/top/cis_top_N.txt")
    cis_hits_top=cis_hits_top[,c(2:7)]
    cis_hits_top$sd=cis_hits_top$beta/cis_hits_top$statistic
    colnames(cis_hits_top)[5]="FDR"
    dim(cis_hits_top)
    # 2445       6

    ## Input # 3: conjugated-EQTLs data. Load cis and trans, rename FDR column, declare sd column,
        type, and rbind

    trans_hits_top=fread("outputs/3_EQTLs/results/top/conjugated_trans_top.txt")
    trans_hits_top=trans_hits_top[,c(2:7)]
    trans_hits_top=trans_hits_top[order(trans_hits_top$gene),]
    dim(trans_hits_top)
    # 1942163       6




    trans_hits_top$type="trans"

    conj_cis_hits_top=fread("outputs/3_EQTLs/results/top/conjugated_cis_top.txt")
    conj_cis_hits_top=conj_cis_hits_top[,c(2:7)]
    conj_cis_hits_top=conj_cis_hits_top[order(conj_cis_hits_top$gene),]
    dim(conj_cis_hits_top)
    # 3242    6
    conj_cis_hits_top$type="cis"

    conj_hits_top=rbind(trans_hits_top,conj_cis_hits_top)
    conj_hits_top$sd=conj_hits_top$beta/conj_hits_top$statistic
    colnames(conj_hits_top)[5]="FDR"
    dim(conj_hits_top)

    ## Input # 4: gtypes correlations.

    gtypes=fread(paste0("inputs/processed/gtypes_ok.txt"))
    gtypes=data.frame(gtypes)
    rownames(gtypes)=gtypes$V1
    gtypes=gtypes[,2:ncol(gtypes)]
    gtypes_top=gtypes[which(rownames(gtypes) %in% unique(cis_hits_top$snps)),]
    correlations_top=cor(t(gtypes_top))



}
```

```
##################################################################################################
#### 3. Format network: subset only pairs under genetic control that are far enough from each other ###
##################################################################################################

{
    net_use_top=network[which(network$gen_row %in% cis_hits_top$gene & network$gen_col %in%
        cis_hits_top$gene),]
    dim(net_use_top)
    ## [1] 131644      JQ: 9
    rm(network)

    vecinity_test=function(i,threshold=1000000){
    gen_A=net_use_top$gen_row[i]
    gen_B=net_use_top$gen_col[i]
    chunk=genes[c(gen_A,gen_B),]
    if(chunk$chr[1]!=chunk$chr[2])
    {
        return(0)
    }else{
        first_gene=which.min(chunk$start)
        last_gene=which.max(chunk$start)
        if(first_gene==last_gene){
            return(1)
        }else if(chunk$end[last_gene]<chunk$end[first_gene]){
            return(1)
        }else if(chunk$start[last_gene]-chunk$end[first_gene]<threshold){
            return(1)
        }else{
            return(0)
        }




    }
    }

    net_use_top$vecinity=NA
    net_use_top$vecinity=sapply(c(1:nrow(net_use_top)),vecinity_test)
    dim(net_use_top)
    #131644     10
    net_use_top=net_use_top[which(net_use_top$vecinity!=1),]
    dim(net_use_top)
    #131230     10

    ## Check to verify that the 815 genes survived:
    length(unique(c(net_use_top$gen_row,net_use_top$gen_col)))
    # 815

}

##############################################
#### 4. Checkpoint: save MR input objects ###
##############################################

{
    system("mkdir -p outputs/4_MR/top")
    MR_inputs=list(net_use_top=net_use_top,
        cis_hits_top=cis_hits_top,conj_hits_top=conj_hits_top,correlations_top=correlations_top)
    save(MR_inputs,file="outputs/4_MR/top/MR_inputs_checkpoint.Rdata")
}
```

```
##############################################
#### 5. Run MR ##############################
##############################################

{
    net_infer_Mendel_Rand=function(coexp_top,cis_tests_top,conjugated_top,correl_top,i,N=3){

        #if(i %% 200==0)
        #print(i)
        gen_a=coexp_top$gen_row[i]
        gen_b=coexp_top$gen_col[i]

        cis_chunk_fw=cis_tests_top[which(cis_tests_top$gene %in% gen_a),]
        cis_chunk_bw=cis_tests_top[which(cis_tests_top$gene %in% gen_b),]

        if(nrow(cis_chunk_fw)!=N)print(paste("Issue: cis_chunk_fw has", nrow(cis_chunk_fw)," tests"))
        if(nrow(cis_chunk_bw)!=N)print(paste("Issue: cis_chunk_bw has", nrow(cis_chunk_bw)," tests"))

        trans_chunk_fw=conjugated_top[which(conjugated_top$gene==gen_b & conjugated_top$snps %in%
            cis_chunk_fw$snps),]
        trans_chunk_bw=conjugated_top[which(conjugated_top$gene==gen_a & conjugated_top$snps %in%
            cis_chunk_bw$snps),]

        if(nrow(trans_chunk_fw)!=N)print(paste("Issue: trans_chunk_fw has", nrow(trans_chunk_fw),"
            tests"))
        if(nrow(trans_chunk_bw)!=N)print(paste("Issue: trans_chunk_bw has", nrow(trans_chunk_bw),"
            tests"))

        if(length(which(trans_chunk_fw$type=="cis")>0))print(paste("Issue: trans_chunk_fw includes
            some cis tests"))




        if(length(which(trans_chunk_bw$type=="cis")>0))print(paste("Issue: trans_chunk_bw includes
            some cis tests"))

        set_fw=which(colnames(correl_top) %in% cis_chunk_fw$snps)
        set_bw=which(colnames(correl_top) %in% cis_chunk_bw$snps)

        if(length(set_bw)!=N)print(paste("Issue:", length(set_fw)," SNPS in fw test in the corr.
            matrix"))
        if(length(set_bw)!=N)print(paste("Issue:", length(set_bw)," SNPS in bw test in the corr.
            matrix"))

        cors_fw=as.matrix(correl_top[set_fw,set_fw])
        cors_bw=as.matrix(correl_top[set_bw,set_bw])

        valid_cor_fw=length(cors_fw)>1
        valid_cor_bw=length(cors_bw)>1

        if(valid_cor_fw){
            valid_cor_fw=abs(det(cors_fw))>1E-15
        }
        if(valid_cor_bw){
            valid_cor_bw=abs(det(cors_bw))>1E-15
        }

        if(valid_cor_fw){
            MR_input_fw <- mr_input(bx = cis_chunk_fw$beta,bxse = cis_chunk_fw$sd,by =
                trans_chunk_fw$beta,byse = trans_chunk_fw$sd,corr=cors_fw)
            MR_fw <- mr_ivw(MR_input_fw,correl = TRUE)
        }else{
        MR_input_fw <- mr_input(bx = cis_chunk_fw$beta,bxse = cis_chunk_fw$sd,by =
            trans_chunk_fw$beta,byse = trans_chunk_fw$sd)
```

```
        MR_fw <- mr_ivw(MR_input_fw,correl = FALSE)
        }

        if(valid_cor_bw){
        MR_input_bw <- mr_input(bx = cis_chunk_bw$beta,bxse = cis_chunk_bw$sd,by =
            trans_chunk_bw$beta,byse = trans_chunk_bw$sd,corr=cors_bw)
        MR_bw <- mr_ivw(MR_input_bw,correl = TRUE)
        }else{
        MR_input_bw <- mr_input(bx = cis_chunk_bw$beta,bxse = cis_chunk_bw$sd,by =
            trans_chunk_bw$beta,byse = trans_chunk_bw$sd)
        MR_bw <- mr_ivw(MR_input_bw,correl = FALSE)
        }

        output_fw=c(MR_fw@Estimate,MR_fw@StdError,MR_fw@CILower,MR_fw@CIUpper,MR_fw@Pvalue)
        output_bw=c(MR_bw@Estimate,MR_bw@StdError,MR_bw@CILower,MR_bw@CIUpper,MR_bw@Pvalue)
        return(c(output_fw,output_bw))
    }

    #thing=load("outputs/4_MR/top/MR_inputs_checkpoint.Rdata")
    #net_use_top=MR_inputs[["net_use_top"]]
    #cis_hits_top=MR_inputs[["cis_hits_top"]]
    #conj_hits_top=MR_inputs[["conj_hits_top"]]
    #correlations_top=MR_inputs[["correlations_top"]]
    #rm(MR_inputs)



    system("mkdir -p outputs/4_MR/top/Result_chunks/")
    for(j in 1:130){
        print(paste("CHUNK: ",j,"\n"))
        name=paste0("outputs/4_MR/top/Result_chunks/chunk_",j,".txt")
        down=1+(j-1)*1000
        up=j*1000
        res=sapply(c(down:up),
        net_infer_Mendel_Rand,
        coexp_top=net_use_top,
        cis_tests_top=cis_hits_top,
        conjugated_top=conj_hits_top,
        correl_top=correlations_top)

        res=data.frame(t(res))
        write.table(res,name)
    }

    name=paste0("outputs/4_MR/top/Result_chunks/chunk_131.txt")
    down=130001
    up=nrow(net_use_top)
    res=sapply(c
        (down:up),net_infer_Mendel_Rand,coexp_top=net_use_top,cis_tests_top=cis_hits_top
        ,conjugated_top=conj_hits_top,correl_top=correlations_top)
    res=data.frame(t(res))
    write.table(res,name)

    result_top=read.table(paste0("outputs/4_MR/top/Result_chunks/chunk_1.txt"))
```

```
243
244     for(j in 2:131)
245     {
246         print(paste("CHUNK: ",j,"\n"))
247         name=paste0("outputs/4_MR/top/Result_chunks/chunk_",j,".txt")
248         appendix_top=read.table(name)
249         result_top=rbind(result_top,appendix_top)
250     }
251
252     colnames(result_top)=c(
253     "Theta_FW","Sd_FW","LowerCI_FW","UpperCI_FW","P_FW",
254     "Theta_BW","Sd_BW","LowerCI_BW","UpperCI_BW","P_BW")
255
256     result_top$gen_one=net_use_top$gen_row
257     result_top$gen_two=net_use_top$gen_col
258     dim(result_top)
259     #131230     10
260
261     result_top$index=c(1:nrow(result_top))
262     result_top=result_top[order(-result_top$P_FW),]
263     result_top$n_larger=c(1:nrow(result_top))
264     result_top$pi_0_gorro_FW=result_top$n_larger/((nrow(result_top))*(1-result_top$P_FW))
265     result_top=result_top[order(result_top$P_FW),]
266     constraint_matrix=as.matrix(data.frame(c(0,2),c(0,1),c(1,0)))
267     result_top$pi_0_gorro_smooth_FW=cobs(result_top$P_FW,
268     result_top$pi_0_gorro_FW,
269     constraint="decrease",
270     nknots=14,method="quantile",
271     pointwise=constraint_matrix,maxiter=500,print.warn=TRUE,print.mesg=TRUE)$fitted
272
273     pi_0_FW=min(result_top$pi_0_gorro_smooth_FW)
274
275     result_top$Fdr_FW=p.adjust(result_top$P_FW,method="BH")*pi_0_FW
276     result_top$Fndr_FW=1-(pi_0_FW/result_top$pi_0_gorro_smooth_FW)
277
278     pl_FW=ggplot(result_top)+
279     geom_point(aes(x=P_FW,y=pi_0_gorro_FW),size=0.1,color="blue",alpha=0.3)+
280     geom_line(aes(x=P_FW,y=pi_0_gorro_smooth_FW),color="red")+
281     geom_line(aes(x=P_FW,y=Fndr_FW),color="green")+
282     geom_line(aes(x=P_FW,y=Fdr_FW),color="black")+
283     geom_hline(yintercept=pi_0_FW)+ylim(0,1)
284
285     result_top=result_top[order(-result_top$P_BW),]
286     result_top$n_larger=c(1:nrow(result_top))
287     result_top$pi_0_gorro_BW=result_top$n_larger/((nrow(result_top))*(1-result_top$P_BW))
288     result_top=result_top[order(result_top$P_BW),]
289     constraint_matrix=as.matrix(data.frame(c(0,2),c(0,1),c(1,0)))
290     result_top$pi_0_gorro_smooth_BW=cobs
            (result_top$P_BW,result_top$pi_0_gorro_BW,constraint="decrease",nknots=14,method="quantile"
            ,pointwise=constraint_matrix,maxiter=1000,print.warn=FALSE,print.mesg=FALSE)$fitted
291     pi_0_BW=min(result_top$pi_0_gorro_smooth_BW)
292
293     result_top$Fdr_BW=p.adjust(result_top$P_BW,method="BH")*pi_0_BW
294     result_top$Fndr_BW=1-(pi_0_BW/result_top$pi_0_gorro_smooth_BW)
295
296     pl_BW=ggplot(result_top)+ geom_point(aes(x=P_BW,y=pi_0_gorro_BW),size=0.1,color="blue",alpha=0.3)
            + geom_line(aes(x=P_BW,y=pi_0_gorro_smooth_BW),color="red") +
            geom_line(aes(x=P_BW,y=Fndr_BW),color="green") + geom_line(aes(x=P_BW,y=Fdr_BW),color="black")
            + geom_hline(yintercept=pi_0_BW)+ylim(0,1)
297
298     pl_fdrs_fndrs=plot_grid(pl_FW,pl_BW,ncol=2)
```

```
    pdf("outputs/4_MR/top/pl_fdrs_fndrs.pdf",height=4,width=8)
    print(pl_fdrs_fndrs)
    dev.off()

    result_top=result_top[order(result_top$index),]
    result_top=result_top[,c(11,12,1:5,15:18,6:10,19:22)]
    result_top$link=paste0(result_top$gen_one,"_",result_top$gen_two)

}

##############################################
#### 6. Clasify link directions #############
##############################################

{
    get_mean_log10Fdr=function(i){
        if(i%%1000==0)print(i)
        fdrs_one=cis_hits_top$FDR[which(cis_hits_top$gene %in% result_top$gen_one[i])]
        fdrs_two=cis_hits_top$FDR[which(cis_hits_top$gene %in% result_top$gen_two[i])]
        output=c(mean(-log10(fdrs_one)),mean(-log10(fdrs_two)))
        return(output)
    }

    means_log10=t(sapply(1:nrow(result_top),get_mean_log10Fdr))

    result_top$mean_log_fdr_one=means_log10[,1]
    result_top$mean_log_fdr_two=means_log10[,2]




    get_min_Fdr=function(i)
    {
        if(i%%1000==0)print(i)
        fdrs_one=cis_hits_top$FDR[which(cis_hits_top$gene %in% result_top$gen_one[i])]
        fdrs_two=cis_hits_top$FDR[which(cis_hits_top$gene %in% result_top$gen_two[i])]
        output=c(max(-log10(fdrs_one)),max(-log10(fdrs_two)))
        return(output)
    }

    maxs_log10=t(sapply(1:nrow(result_top),get_min_Fdr))

    result_top$max_log_fdr_one=maxs_log10[,1]
    result_top$max_log_fdr_two=maxs_log10[,2]

    ## One_more_than_two (Two_more_than_one) means gene one (two) has more significant cis EQTLs than
        gene two (one)
    result_top$flag_direction_mean="One_more_than_two"
    result_top$flag_direction_mean
        [which(result_top$mean_log_fdr_one<result_top$mean_log_fdr_two)]="Two_more_than_one"

    ## One_more_than_two (Two_more_than_one) means gene one (two) more significant cis EQTLs than gene
        two (one)
    result_top$flag_direction_max="One_more_than_two"
    result_top$flag_direction_max
        [which(result_top$max_log_fdr_one<result_top$max_log_fdr_two)]="Two_more_than_one"

    th_fndr=0.1
    th_fdr=0.1
    ## Count FW links from one to two
    FW_1=length(which(result_top$flag_direction_mean=="One_more_than_two" & result_top$Fdr_FW<th_fdr &
        result_top$Fndr_BW<th_fndr))
```

```
    ## Count FW links from two to one
    FW_2=length(which(result_top$flag_direction_mean=="Two_more_than_one" & result_top$Fdr_BW<th_fdr &
        result_top$Fndr_FW<th_fndr))

    ## Count BW links from one to two
    BW_1=length(which(result_top$flag_direction_mean=="One_more_than_two" & result_top$Fdr_BW<th_fdr &
        result_top$Fndr_FW<th_fndr))
    ## Count BW links from two to one
    BW_2=length(which(result_top$flag_direction_mean=="Two_more_than_one" & result_top$Fdr_FW<th_fdr &
        result_top$Fndr_BW<th_fndr))

    FW_links=FW_1+FW_2
    BW_links=BW_1+BW_2

    FW_links/(FW_links+BW_links)
    # 0.4190195

    ## Ahora, a ver el criterio max:

    ## Count FW links from one to two
    FW_1=length(which(result_top$flag_direction_max=="One_more_than_two" & result_top$Fdr_FW<th_fdr &
        result_top$Fndr_BW<th_fndr))
    ## Count FW links from two to one
    FW_2=length(which(result_top$flag_direction_max=="Two_more_than_one" & result_top$Fdr_BW<th_fdr &
        result_top$Fndr_FW<th_fndr))

    ## Count BW links from one to two
    BW_1=length(which(result_top$flag_direction_max=="One_more_than_two" & result_top$Fdr_BW<th_fdr &
        result_top$Fndr_FW<th_fndr))
    ## Count BW links from two to one
    BW_2=length(which(result_top$flag_direction_max=="Two_more_than_one" & result_top$Fdr_FW<th_fdr &
        result_top$Fndr_BW<th_fndr))

    FW_links=FW_1+FW_2
    BW_links=BW_1+BW_2

    FW_links/(FW_links+BW_links)
    # 0.4566005


    write.table(result_top,"outputs/4_MR/top/result.txt")

}
```

7.5. Mendelian Randomization for Balanced-SNPs method

```
################################################################
#### 1. Load dependencies ######################################
################################################################
{
    library(edgeR)
    library(limma)
    library(gdsfmt)
    library(SNPRelate)
    library(qvalue)
    library(stats)
    library(MendelianRandomization)
    library(data.table)
    library(cobs)
    library(ggplot2)
    library(cowplot)

    ul=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,1:cs])
    }
    ur=function(tab,n=5)
    {
        rs=min(nrow(tab),n)
        cs=min(ncol(tab),n)
        return(tab[1:rs,(ncol(tab)-cs):ncol(tab)])
    }
    dcols=function(tab){data.frame(colnames(tab))}
    drows=function(tab){data.frame(rownames(tab))}
}
```

```
################################################################
#### 2. Load and pretty up input tables ########################
################################################################

{
    ## Input # 1: Network (I just load it; the formatting requires an entire step, which also needs
        the genes info table loaded here as well).
    network=fread("outputs/2_coexpression/network0.01.txt")
    colnames(network)[1]="link"
    nrow(network)
    # 25897510 pairs.
    genes=read.table(paste0("outputs/2_coexpression/coexpressed_genes.txt"))


    ## Input # 2: cis-EQTLs data. Here, in cis_hits_balanced we only stored
    ## the tests of the IDs that will be paired.
    ## But the tests stats are in cis.txt.
    ## I merge them to build the sd columns, needed for MR
    cis_hits_balanced=fread("outputs/3_EQTLs/results/match/cis_balanced_N.txt")
    cis_hits_balanced=cis_hits_balanced[,c(2:7)]

    cis_detailed_stats=fread("outputs/3_EQTLs/results/cis.txt")
    cis_detailed_stats=cis_detailed_stats[,c(2:5,7)]
    colnames(cis_detailed_stats)[1]="snp"

    cis_detailed_stats$test=paste0(cis_detailed_stats$snp,"_",cis_detailed_stats$gene)

    cis_hits_balanced$test=paste0(cis_hits_balanced$snp_one,"_",cis_hits_balanced$gene_one)
    cis_hits_balanced_output=merge(cis_hits_balanced,cis_detailed_stats,by="test")
    cis_hits_balanced_output=cis_hits_balanced_output[,c(2:7,10:12)]
    colnames(cis_hits_balanced_output)[c(7:9)]=paste0(colnames(cis_hits_balanced_output)
        [c(7:9)],"_one")

```

```r

    cis_hits_balanced_output$test=paste0(cis_hits_balanced_output$snp_two,"_",cis_hits_balanced_output$gene_two)
    cis_hits_balanced_output=merge(cis_hits_balanced_output,cis_detailed_stats,by="test")
    cis_hits_balanced_output=cis_hits_balanced_output[,c(4,2,10,8,9,6,5,3,15,13,14,7)]

    colnames(cis_hits_balanced_output)[c(9:11)]=paste0(colnames(cis_hits_balanced_output)[c(9:11)],"_two")
    cis_hits_balanced_output$link=paste0(cis_hits_balanced_output$gene_one,"_",cis_hits_balanced_output$gene_two)
    dim(cis_hits_balanced_output)
    #  394932      7
    cis_hits_balanced=cis_hits_balanced_output
    rm(cis_hits_balanced_output)
    cis_hits_balanced$sd_one=cis_hits_balanced$beta_one/cis_hits_balanced$statistic_one
    cis_hits_balanced$sd_two=cis_hits_balanced$beta_two/cis_hits_balanced$statistic_two

    ## Input # 3: conjugated-EQTLs data. Load cis and trans, rename FDR column, declare sd column, type, and rbind
    trans_hits_balanced=fread("outputs/3_EQTLs/results/match/conjugated_trans_match.txt")
    trans_hits_balanced=trans_hits_balanced[,c(2:7)]
    trans_hits_balanced=trans_hits_balanced[order(trans_hits_balanced$gene),]
    dim(trans_hits_balanced)
    # 25685872        6
    trans_hits_balanced$type="trans"

    conj_cis_hits_balanced=fread("outputs/3_EQTLs/results/match/conjugated_cis_match.txt")
    conj_cis_hits_balanced=conj_cis_hits_balanced[,c(2:7)]
    conj_cis_hits_balanced=conj_cis_hits_balanced[order(conj_cis_hits_balanced$gene),]
    dim(conj_cis_hits_balanced)
    # 44493     6
    conj_cis_hits_balanced$type="cis"



    conj_hits_balanced=rbind(trans_hits_balanced,conj_cis_hits_balanced)
    conj_hits_balanced$sd=conj_hits_balanced$beta/conj_hits_balanced$statistic
    colnames(conj_hits_balanced)[5]="FDR"
    dim(conj_hits_balanced)
    # 25730365        8

    ## Input # 4: gtypes correlations.
    gtypes=fread(paste0("inputs/processed/gtypes_ok.txt"))
    gtypes=data.frame(gtypes)
    rownames(gtypes)=gtypes$V1
    gtypes=gtypes[,2:ncol(gtypes)]
    gtypes_balanced=gtypes[which(rownames(gtypes) %in% unique(c(cis_hits_balanced$snp_one,cis_hits_balanced$snp_two))),]
    correlations_balanced=cor(t(gtypes_balanced))
}
```

```
##################################################################################################
    3. Format network: subset only pairs under genetic control that are far enough from each other ###
##################################################################################################
{
    ## This command to subset the network is different from the balanced case: different info in
        dataframes

    net_use_balanced=network[which(network$link %in% cis_hits_balanced$link),]
    dim(net_use_balanced)
    ## [1] 131644      JQ: 9
    rm(network)

    vecinity_test=function(i,threshold=1000000){
    gen_A=net_use_balanced$gen_row[i]
    gen_B=net_use_balanced$gen_col[i]
    chunk=genes[c(gen_A,gen_B),]
    if(chunk$chr[1]!=chunk$chr[2])
    {
        return(0)
    }else{
        first_gene=which.min(chunk$start)
        last_gene=which.max(chunk$start)
        if(first_gene==last_gene){
            return(1)
        }else if(chunk$end[last_gene]<chunk$end[first_gene]){
            return(1)
        }else if(chunk$start[last_gene]-chunk$end[first_gene]<threshold){
            return(1)
        }else{
            return(0)
        }
    }
    }




    net_use_balanced$vecinity=NA
    net_use_balanced$vecinity=sapply(c(1:nrow(net_use_balanced)),vecinity_test)
    dim(net_use_balanced)
    #131644     10
    net_use_balanced=net_use_balanced[which(net_use_balanced$vecinity!=1),]
    dim(net_use_balanced)
    #131230     10

    ## Check to verify that the 815 genes survived:
    length(unique(c(net_use_balanced$gen_row,net_use_balanced$gen_col)))
    # 815
}

```

```
##################################################################################################
#### 4. Filter elements from cis_hits_balanced and conj_hits_balanced that will not be used. ###
##################################################################################################
{
    cis_hits_balanced=cis_hits_balanced[which(cis_hits_balanced$link %in% net_use_balanced$link),]
    nrow(cis_hits_balanced)
    #393690

    cis_hits_balanced$trans_test_fw=paste0(cis_hits_balanced$snp_one,"_",cis_hits_balanced$gene_two)
    cis_hits_balanced$trans_test_bw=paste0(cis_hits_balanced$snp_two,"_",cis_hits_balanced$gene_one)

    conj_hits_balanced$test=paste0(conj_hits_balanced$snps,"_",conj_hits_balanced$gene)
    nrow(conj_hits_balanced)
    # 25730365
    conj_hits_balanced=conj_hits_balanced[which(conj_hits_balanced$test %in%
        unique(c(cis_hits_balanced$trans_test_fw,cis_hits_balanced$trans_test_bw))),]
    dim(conj_hits_balanced)
    #[1] 776489      9
    summary(factor(conj_hits_balanced$type))
    ## trans
    ## 776489
    ## All remaining trans tests are trans (we erased the cis ones during the vicinity filtering)

}



##############################################
#### 5. Checkpoint: save MR input objects ###
##############################################

system("mkdir -p outputs/4_MR/match/")
MR_inputs=list(net_use_balanced=net_use_balanced,
    cis_hits_balanced=cis_hits_balanced,conj_hits_balanced=conj_hits_balanced
    ,correlations_balanced=correlations_balanced)
save(MR_inputs,file="outputs/4_MR/match/MR_inputs_checkpoint_match_fv.Rdata")

##############################################
#### 6. Run MR ###############################
##############################################

net_infer_Mendel_Rand=function
    (coexp_balanced,cis_tests_balanced,conjugated_balanced,correl_balanced,i,N=3)
{

    if(i%%100==0) print(i)
    gen_a=coexp_balanced$gen_row[i]
    gen_b=coexp_balanced$gen_col[i]
    link_ref=paste0(gen_a,"_",gen_b)

    cis_chunk=cis_tests_balanced[which(cis_tests_balanced$link == link_ref),]
    if(nrow(cis_chunk_fw)!=N)print(paste("Issue: cis_chunk_fw has", nrow(cis_chunk_fw)," tests"))

    cis_chunk_fw=cis_chunk[,c("snp_one","gene_one","beta_one","sd_one")]
    cis_chunk_bw=cis_chunk[,c("snp_two","gene_two","beta_two","sd_two")]

    colnames(cis_chunk_fw)=c("snps","gene","beta","sd")
    colnames(cis_chunk_bw)=c("snps","gene","beta","sd")

```

```
    trans_chunk_fw=conjugated_balanced[which(conjugated_balanced$gene==gen_b &
        conjugated_balanced$snps %in% cis_chunk_fw$snps),]
    trans_chunk_bw=conjugated_balanced[which(conjugated_balanced$gene==gen_a &
        conjugated_balanced$snps %in% cis_chunk_bw$snps),]

    if(nrow(trans_chunk_fw)!=N)print(paste("Issue: trans_chunk_fw has", nrow(trans_chunk_fw),"
        tests"))
    if(nrow(trans_chunk_bw)!=N)print(paste("Issue: trans_chunk_bw has", nrow(trans_chunk_bw),"
        tests"))

    set_fw=which(colnames(correl_balanced) %in% cis_chunk_fw$snps)
    set_bw=which(colnames(correl_balanced) %in% cis_chunk_bw$snps)

    if(length(set_fw)!=N)print(paste("Issue:", length(set_fw)," SNPS in fw test in the corr. matrix.
        Iter",i))
    if(length(set_bw)!=N)print(paste("Issue:", length(set_bw)," SNPS in bw test in the corr. matrix.
        Iter",i))

    cors_fw=as.matrix(correl_balanced[set_fw,set_fw])
    cors_bw=as.matrix(correl_balanced[set_bw,set_bw])

    valid_cor_fw=length(cors_fw)>1
    valid_cor_bw=length(cors_bw)>1

    if(valid_cor_fw){
        valid_cor_fw=abs(det(cors_fw))>1E-15
    }
    if(valid_cor_bw){
        valid_cor_bw=abs(det(cors_bw))>1E-15
    }
```

```
    if(valid_cor_fw){
    MR_input_fw <- mr_input(bx = cis_chunk_fw$beta,bxse = cis_chunk_fw$sd,by =
        trans_chunk_fw$beta,byse = trans_chunk_fw$sd,corr=cors_fw)
    MR_fw <- mr_ivw(MR_input_fw,correl = TRUE)
    }else{
    MR_input_fw <- mr_input(bx = cis_chunk_fw$beta,bxse = cis_chunk_fw$sd,by =
        trans_chunk_fw$beta,byse = trans_chunk_fw$sd)
    MR_fw <- mr_ivw(MR_input_fw,correl = FALSE)
    }

    if(valid_cor_bw){
    MR_input_bw <- mr_input(bx = cis_chunk_bw$beta,bxse = cis_chunk_bw$sd,by =
        trans_chunk_bw$beta,byse = trans_chunk_bw$sd,corr=cors_bw)
    MR_bw <- mr_ivw(MR_input_bw,correl = TRUE)
    }else{
    MR_input_bw <- mr_input(bx = cis_chunk_bw$beta,bxse = cis_chunk_bw$sd,by =
        trans_chunk_bw$beta,byse = trans_chunk_bw$sd)
    MR_bw <- mr_ivw(MR_input_bw,correl = FALSE)
    }

    output_fw=c(MR_fw@Estimate,MR_fw@StdError,MR_fw@CILower,MR_fw@CIUpper,MR_fw@Pvalue)
    output_bw=c(MR_bw@Estimate,MR_bw@StdError,MR_bw@CILower,MR_bw@CIUpper,MR_bw@Pvalue)
    return(c(output_fw,output_bw))

}

##
#thing=load("outputs/4_MR/match/MR_inputs_checkpoint_match_fv.Rdata")
#net_use_balanced=MR_inputs[["net_use_balanced"]]
#cis_hits_balanced=MR_inputs[["cis_hits_balanced"]]
#conj_hits_balanced=MR_inputs[["conj_hits_balanced"]]
#correlations_balanced=MR_inputs[["correlations_balanced"]]
#rm(MR_inputs)
```

```
start_point=1
end_point=130

system("mkdir -p outputs/4_MR/match/Result_chunks_fv/")
for(j in start_point:end_point){
    print(paste("CHUNK: ",j,"\n"))
    name=paste0("outputs/4_MR/match/Result_chunks_fv/chunk_",j,".txt")
    down=1+(j-1)*1000
    up=j*1000
    res=sapply(c(down:up),
    net_infer_Mendel_Rand,
    coexp_balanced=net_use_balanced,
    cis_tests_balanced=cis_hits_balanced,
    conjugated_balanced=conj_hits_balanced,
    correl_balanced=correlations_balanced)

    res=data.frame(t(res))
    write.table(res,name)
}

name=paste0("outputs/4_MR/match/Result_chunks_fv/chunk_131.txt")
down=130001
up=nrow(net_use_balanced)
res=sapply(c
    (down:up),net_infer_Mendel_Rand,coexp_balanced=net_use_balanced
    ,cis_tests_balanced=cis_hits_balanced,conjugated_balanced=conj_hits_balanced
    ,correl_balanced=correlations_balanced)
res=data.frame(t(res))
write.table(res,name)

result_balanced=read.table(paste0("outputs/4_MR/match/Result_chunks_fv/chunk_1.txt"))
```

```
for(j in 2:131)
{
    print(paste("CHUNK: ",j,"\n"))
    name=paste0("outputs/4_MR/match/Result_chunks_fv/chunk_",j,".txt")
    appendix_balanced=read.table(name)
    result_balanced=rbind(result_balanced,appendix_balanced)
}

colnames(result_balanced)=c(
"Theta_FW","Sd_FW","LowerCI_FW","UpperCI_FW","P_FW",
"Theta_BW","Sd_BW","LowerCI_BW","UpperCI_BW","P_BW")

result_balanced$gen_one=net_use_balanced$gen_row
result_balanced$gen_two=net_use_balanced$gen_col
dim(result_balanced)
#131230     12

##############################################
#### 7. Clasify link directions #############
##############################################

result_balanced$index=c(1:nrow(result_balanced))
result_balanced=result_balanced[order(-result_balanced$P_FW),]
result_balanced$n_larger=c(1:nrow(result_balanced))
result_balanced$pi_0_gorro_FW=result_balanced$n_larger/((nrow(result_balanced))*
    (1-result_balanced$P_FW))
result_balanced=result_balanced[order(result_balanced$P_FW),]
constraint_matrix=as.matrix(data.frame(c(0,2),c(0,1),c(1,0)))
result_balanced$pi_0_gorro_smooth_FW=cobs
    (result_balanced$P_FW,result_balanced$pi_0_gorro_FW,constraint="decrease",nknots=14
    ,method="quantile",pointwise=constraint_matrix,maxiter=1000,print.warn=FALSE,print
    .mesg=FALSE)$fitted
```

```r
pi_0_FW=min(result_balanced$pi_0_gorro_smooth_FW)

result_balanced$Fdr_FW=p.adjust(result_balanced$P_FW,method="BH")*pi_0_FW
result_balanced$Fndr_FW=1-(pi_0_FW/result_balanced$pi_0_gorro_smooth_FW)

pl_FW=ggplot(result_balanced)+
    geom_point(aes(x=P_FW,y=pi_0_gorro_FW),size=0.1,color="blue",alpha=0.3) +
    geom_line(aes(x=P_FW,y=pi_0_gorro_smooth_FW),color="red") +
    geom_line(aes(x=P_FW,y=Fndr_FW),color="green")+ geom_line(aes(x=P_FW,y=Fdr_FW),color="black") +
    geom_hline(yintercept=pi_0_FW)+geom_hline(yintercept=0.1,color="green")+ylim(0,1)

result_balanced=result_balanced[order(-result_balanced$P_BW),]
result_balanced$n_larger=c(1:nrow(result_balanced))
result_balanced$pi_0_gorro_BW=result_balanced$n_larger/((nrow(result_balanced))*
    (1-result_balanced$P_BW))
result_balanced=result_balanced[order(result_balanced$P_BW),]
constraint_matrix=as.matrix(data.frame(c(0,2),c(0,1),c(1,0)))
result_balanced$pi_0_gorro_smooth_BW=cobs
    (result_balanced$P_BW,result_balanced$pi_0_gorro_BW,constraint="decrease",nknots=14
    ,method="quantile",pointwise=constraint_matrix,maxiter=1000,print.warn=FALSE,print
    .mesg=FALSE)$fitted

pi_0_BW=min(result_balanced$pi_0_gorro_smooth_BW)

result_balanced$Fdr_BW=p.adjust(result_balanced$P_BW,method="BH")*pi_0_BW
result_balanced$Fndr_BW=1-(pi_0_BW/result_balanced$pi_0_gorro_smooth_BW)

pl_BW=ggplot(result_balanced)+
    geom_point(aes(x=P_BW,y=pi_0_gorro_BW),size=0.1,color="blue",alpha=0.3)+
    geom_line(aes(x=P_BW,y=pi_0_gorro_smooth_BW),color="red")+
    geom_line(aes(x=P_BW,y=Fndr_BW),color="green")+ geom_line(aes(x=P_BW,y=Fdr_BW),color="black") +
    geom_hline(yintercept=pi_0_BW)+geom_hline(yintercept=0.1,color="green")+ylim(0,1)
```

```r
pl_fdrs_fndrs=plot_grid(pl_FW,pl_BW,ncol=2)

pdf("outputs/4_MR/match/pl_fdrs_fndrs_fv.pdf",height=4,width=8)
print(pl_fdrs_fndrs)
dev.off()

pl_corners=ggplot(result_balanced)+geom_point(aes(x=-log10(Fdr_BW),y=-log10(Fdr_FW)))
## Este plot mola tanto que casi hace desconfiar :DD
result_balanced=result_balanced[order(result_balanced$index),]
result_balanced=result_balanced[,c(11,12,1:5,15:18,6:10,19:22)]
result_balanced$link=paste0(result_balanced$gen_one,"_",result_balanced$gen_two)


get_mean_log10Fdr=function(i)
{
    if(i%%1000==0)print(i)
    set=which(cis_hits_balanced$link %in% result_balanced$link[i])
    fdrs_one=cis_hits_balanced$FDR_one[set]
    fdrs_two=cis_hits_balanced$FDR_two[set]
    output=c(mean(-log10(fdrs_one)),mean(-log10(fdrs_two)))
    return(output)
}

means_log10=t(sapply(1:nrow(result_balanced),get_mean_log10Fdr))

result_balanced$mean_log_fdr_one=means_log10[,1]
result_balanced$mean_log_fdr_two=means_log10[,2]

logfdr1=-log10(cis_hits_balanced$FDR_one)
logfdr2=-log10(cis_hits_balanced$FDR_two)
```

```r
get_min_Fdr=function(i)
{
    if(i%%1000==0)print(i)
    set=which(cis_hits_balanced$link %in% result_balanced$link[i])
    output=c(max(logfdr1[set]),max(logfdr2[set]))
    return(output)
}

maxs_log10=t(sapply(1:nrow(result_balanced),get_min_Fdr))

result_balanced$max_log_fdr_one=maxs_log10[,1]
result_balanced$max_log_fdr_two=maxs_log10[,2]


## One_more_than_two (Two_more_than_one) means gene one (two) has more significant cis EQTLs than gene two (one)
result_balanced$flag_direction_mean="One_more_than_two"
result_balanced$flag_direction_mean
    [which(result_balanced$mean_log_fdr_one<result_balanced$mean_log_fdr_two)]="Two_more_than_one"

## One_more_than_two (Two_more_than_one) means gene one (two) more significant cis EQTLs than gene two (one)
result_balanced$flag_direction_max="One_more_than_two"
result_balanced$flag_direction_max
    [which(result_balanced$max_log_fdr_one<result_balanced$max_log_fdr_two)]="Two_more_than_one"

th_fndr=0.1
th_fdr=0.1
## Count FW links from one to two
FW_1=length(which(result_balanced$flag_direction_mean=="One_more_than_two" &
    result_balanced$Fdr_FW<th_fdr & result_balanced$Fndr_BW<th_fndr))
## Count FW links from two to one




FW_2=length(which(result_balanced$flag_direction_mean=="Two_more_than_one" &
    result_balanced$Fdr_BW<th_fdr & result_balanced$Fndr_FW<th_fndr))

## Count BW links from one to two
BW_1=length(which(result_balanced$flag_direction_mean=="One_more_than_two" &
    result_balanced$Fdr_BW<th_fdr & result_balanced$Fndr_FW<th_fndr))
## Count BW links from two to one
BW_2=length(which(result_balanced$flag_direction_mean=="Two_more_than_one" &
    result_balanced$Fdr_FW<th_fdr & result_balanced$Fndr_BW<th_fndr))

FW_links=FW_1+FW_2
BW_links=BW_1+BW_2

FW_links/(FW_links+BW_links)
# 0.4955952

## Ahora, a ver el criterio max:

## Count FW links from one to two
FW_1=length(which(result_balanced$flag_direction_max=="One_more_than_two" &
    result_balanced$Fdr_FW<th_fdr & result_balanced$Fndr_BW<th_fndr))
## Count FW links from two to one
FW_2=length(which(result_balanced$flag_direction_max=="Two_more_than_one" &
    result_balanced$Fdr_BW<th_fdr & result_balanced$Fndr_FW<th_fndr))

## Count BW links from one to two
BW_1=length(which(result_balanced$flag_direction_max=="One_more_than_two" &
    result_balanced$Fdr_BW<th_fdr & result_balanced$Fndr_FW<th_fndr))
## Count BW links from two to one
BW_2=length(which(result_balanced$flag_direction_max=="Two_more_than_one" &
    result_balanced$Fdr_FW<th_fdr & result_balanced$Fndr_BW<th_fndr))

```

```
FW_links=FW_1+FW_2
BW_links=BW_1+BW_2

FW_links/(FW_links+BW_links)
# 0.5002591

write.table(result_balanced,"outputs/4_MR/match/result_fv.txt")
```

7.6. Implementation of empirical null models to quantify directional bias in link prediction for Top-SNPs method

```
library(qvalue)

result_top=read.table("outputs/4_MR/top/result.txt")
forward_top=result_top[which(result_top$Fdr_FW<0.1 & result_top$Fndr_BW<0.1),]
backward_top=result_top[which(result_top$Fndr_FW<0.1 & result_top$Fdr_BW<0.1),]

iter=100000
N=nrow(result_top)
Fdr=0.1
Fndr=0.1
N1=nrow(forward_top)
N2=nrow(backward_top)

n_media=(N1+N2)/2
N1=n_media
N2=n_media

pi_0_FW=qvalue(result_top$P_FW)$pi0
pi_0_BW=qvalue(result_top$P_BW)$pi0

pi_medio=(pi_0_FW+pi_0_BW)/2
pi_0_FW=pi_medio
pi_0_BW=pi_medio

p1_FW=(N1*(1-Fdr))/(N*pi_0_FW*(2-Fdr))
p2_FW=(N2*Fndr)/(N*pi_0_FW)
p3_FW=(N2*(1-Fndr))/(N*(1-pi_0_FW))
p4_FW=(N1*Fdr)/(N*(1-pi_0_FW)*(2-Fdr))

```

```
p1_BW=(N1*(1-Fdr))/(N*pi_0_BW*(2-Fdr))
p2_BW=(N2*Fndr)/(N*pi_0_BW)
p3_BW=(N2*(1-Fndr))/(N*(1-pi_0_BW))
p4_BW=(N1*Fdr)/(N*(1-pi_0_BW)*(2-Fdr))

percentages=rep(NA,iter)

for(i in 1:iter){

  ## Trato de declarar solo un dataframe con todo lo asignable para hacer la alocación de memoria de todo de una vez:
  tab=data.frame(
    platonic_FW=rep("FALSE",N),empiric_FW=rep("UNKNOWN",N),
    platonic_BW=rep("FALSE",N),empiric_BW=rep("UNKNOWN",N),
    r_platonic_FW=runif(N),r_empiric_FW=runif(N),
    r_platonic_BW=runif(N),r_empiric_BW=runif(N))

  ## Declarar platonic labels:
  tab$platonic_FW[which(tab$r_platonic_FW<pi_0_FW)]="TRUE"
  tab$platonic_BW[which(tab$r_platonic_BW<pi_0_BW)]="TRUE"

  ## Declarar empiric labels:
  tab$empiric_FW[which(tab$platonic_FW=="TRUE" &  tab$r_empiric_FW< p1_FW)]="TRUE"
  tab$empiric_FW[which(tab$platonic_FW=="TRUE" &  tab$r_empiric_FW>=p1_FW & tab$r_empiric_FW<(p1_FW+p2_FW))]="FALSE"
  tab$empiric_FW[which(tab$platonic_FW=="FALSE" & tab$r_empiric_FW< p4_FW)]="TRUE"
  tab$empiric_FW[which(tab$platonic_FW=="FALSE" & tab$r_empiric_FW>=p4_FW & tab$r_empiric_FW<(p4_FW+p3_FW))]="FALSE"



  tab$empiric_BW[which(tab$platonic_BW=="TRUE" &  tab$r_empiric_BW< p1_BW)]="TRUE"
  tab$empiric_BW[which(tab$platonic_BW=="TRUE" &  tab$r_empiric_BW>=p1_BW & tab$r_empiric_BW<(p1_BW+p2_BW))]="FALSE"
  tab$empiric_BW[which(tab$platonic_BW=="FALSE" & tab$r_empiric_BW< p4_BW)]="TRUE"
  tab$empiric_BW[which(tab$platonic_BW=="FALSE" & tab$r_empiric_BW>=p4_BW & tab$r_empiric_BW<(p4_BW+p3_BW))]="FALSE"

  # Ahora calculo los porcentages
  number_links_only_FW=length(which(tab$empiric_FW=="TRUE" &  tab$empiric_BW=="FALSE"))
  number_links_only_BW=length(which(tab$empiric_FW=="FALSE" & tab$empiric_BW=="TRUE"))

  percentages[i]=number_links_only_FW/(number_links_only_FW+number_links_only_BW)

  if(i%%100==0) print(i)
}

summary(percentages)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.3591  0.4775  0.5000  0.5000  0.5227  0.6667
sd(percentages)
#0.0.03365883

save(percentages,file="outputs/4_MR/top/percentages_top_100K.Rdata")
```

7.7. Implementation of empirical null models to quantify directional bias in link prediction for Balanced-SNPs method

```
library(qvalue)

result_balanced=read.table("outputs/4_MR/match/result_fv.txt")
forward_balanced=result_balanced[which(result_balanced$Fdr_FW<0.1 & result_balanced$Fndr_BW<0.1),]
backward_balanced=result_balanced[which(result_balanced$Fndr_FW<0.1 & result_balanced$Fdr_BW<0.1),]

iter=100000
N=nrow(result_balanced)
Fdr=0.1
Fndr=0.1
N1=nrow(forward_balanced)
N2=nrow(backward_balanced)

n_media=(N1+N2)/2
N1=n_media
N2=n_media

pi_0_FW=qvalue(result_balanced$P_FW)$pi0
pi_0_BW=qvalue(result_balanced$P_BW)$pi0

pi_medio=(pi_0_FW+pi_0_BW)/2
pi_0_FW=pi_medio
pi_0_BW=pi_medio

p1_FW=(N1*(1-Fdr))/(N*pi_0_FW*(2-Fdr))
p2_FW=(N2*Fndr)/(N*pi_0_FW)
p3_FW=(N2*(1-Fndr))/(N*(1-pi_0_FW))
p4_FW=(N1*Fdr)/(N*(1-pi_0_FW)*(2-Fdr))



p1_BW=(N1*(1-Fdr))/(N*pi_0_BW*(2-Fdr))
p2_BW=(N2*Fndr)/(N*pi_0_BW)
p3_BW=(N2*(1-Fndr))/(N*(1-pi_0_BW))
p4_BW=(N1*Fdr)/(N*(1-pi_0_BW)*(2-Fdr))

percentages=rep(NA,iter)

for(i in 1:iter){

    ## Trato de declarar solo un dataframe con todo lo asignable para hacer la alocación de memoria de
        todo de una vez:
    tab=data.frame(
    platonic_FW=rep("FALSE",N),empiric_FW=rep("UNKNOWN",N),
    platonic_BW=rep("FALSE",N),empiric_BW=rep("UNKNOWN",N),
    r_platonic_FW=runif(N),r_empiric_FW=runif(N),
    r_platonic_BW=runif(N),r_empiric_BW=runif(N))

    ## Declarar platonic labels:
    tab$platonic_FW[which(tab$r_platonic_FW<pi_0_FW)]="TRUE"
    tab$platonic_BW[which(tab$r_platonic_BW<pi_0_BW)]="TRUE"

    ## Declarar empiric labels:
    tab$empiric_FW[which(tab$platonic_FW=="TRUE" &  tab$r_empiric_FW< p1_FW)]="TRUE"
    tab$empiric_FW[which(tab$platonic_FW=="TRUE" &  tab$r_empiric_FW>=p1_FW &
        tab$r_empiric_FW<(p1_FW+p2_FW))]="FALSE"
    tab$empiric_FW[which(tab$platonic_FW=="FALSE" & tab$r_empiric_FW< p4_FW)]="TRUE"
    tab$empiric_FW[which(tab$platonic_FW=="FALSE" & tab$r_empiric_FW>=p4_FW &
        tab$r_empiric_FW<(p4_FW+p3_FW))]="FALSE"

```

```
    tab$empiric_BW[which(tab$platonic_BW=="TRUE" &  tab$r_empiric_BW< p1_BW)]="TRUE"
    tab$empiric_BW[which(tab$platonic_BW=="TRUE" &  tab$r_empiric_BW>=p1_BW &
        tab$r_empiric_BW<(p1_BW+p2_BW))]="FALSE"
    tab$empiric_BW[which(tab$platonic_BW=="FALSE" & tab$r_empiric_BW< p4_BW)]="TRUE"
    tab$empiric_BW[which(tab$platonic_BW=="FALSE" & tab$r_empiric_BW>=p4_BW &
        tab$r_empiric_BW<(p4_BW+p3_BW))]="FALSE"

    # Ahora calculo los porcentages
    number_links_only_FW=length(which(tab$empiric_FW=="TRUE" &  tab$empiric_BW=="FALSE"))
    number_links_only_BW=length(which(tab$empiric_FW=="FALSE" & tab$empiric_BW=="TRUE"))

    percentages[i]=number_links_only_FW/(number_links_only_FW+number_links_only_BW)

    if(i%%100==0) print(i)
}

summary(percentages)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.4404  0.4912  0.5000  0.5000  0.5088  0.5567
sd(percentages)
#0.01306218

save(percentages,file="outputs/4_MR/match/percentages_match_100K.Rdata")
```

Referencias

[1] Getting started with RNA-sequencing. Wang Z., Gerstein M. & Snyder M., "*RNA-seq: a revolutionary tool for transcriptomics*". Nat. Rev. Genet, 2009;10(1), 57–63. Retrieved in August 2022 from https://www.technologynetworks.com/genomics/articles/rna-seq-basics-applications-and-protocol-299461

[2] Getting started with SNP-chips. "*Use of SNP chips to detect rare pathogenic variants: retrospective, population based diagnostic evaluation*". https://www.bmj.com/content/372/bmj.n214

[3] Thomas LaFramboise, "*Single nucleotide polymorphism arrays: a decade of biological, computational and technological advances, Nucleic Acids Research*"., Volume 37, Issue 13, 1 July 2009, Pages 4181–4193, https://doi.org/10.1093/nar/gkp552

[4] Stuart, Joshua M; Segal, Eran; Koller, Daphne; Kim, Stuart K. (2003) "*A gene-coexpression network for global discovery of conserved genetic modules*".

[5] Weirauch, Matthew T (2011). "*Gene coexpression networks for the analysis of DNA microarray data*". Applied Statistics for Network Biology: Methods in Systems Biology. pp. 215–250.

[6] Gorlov, I., Xiao, X., Mayes, M. et al."*SNP eQTL status and eQTL density in the adjacent region of the SNP are associated with its statistical significance in GWA studies*"., https://bmcgenomdata.biomedcentral.com/articles/10.1186/s12863-019-0786-0

[7] Alexandra C. Nica Emmanouil T. Dermitzakis ,"*Expression quantitative trait loci: present and future*".

[8] Consulting about eQTLs in https://es.wikipedia.org/wiki/eQTL

[9] Jéssica Moreira Batista da Silva (2020) "*Characterization of molecular heterogeneity in celiac disease patients*" p. 8-10.

[10] Robinson MD, McCarthy DJ, Smyth GK "*edgeR: A Bioconductor package for differential expression analysis of digital gene expression data*". Bioinformatics. 2009; 26(1):139-40

[11] Ritchie, M. E., Phipson, B., Wu, D. I., Hu, Y., Law, C.W., Shi, W., Smyth, G. K. (2015) "*limma powers differential expression analyses for RNA-sequencing and microarray studies*". Nucleic acids research, 43(7),e47-e47.

[12] Regina Santesteban Azanza (2020) "*Métodos computacionales para la caracterización de relaciones causales entre genotipo y fenotipo: Heredabilidad de la expresión y coexpresión genética*".

[13] Benjamini, Y., Hochberg, Y (1995) "*Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing*". Journal of the Royal statistical society: series B (Methodological), 57(1), 289-300

[14] Storey JD, Tibshirani R (2003) "*Statistical significance for genomewide studies*". PNAS 100:9440–9445