REBET (subREgion-based BurdEn Test)

June 13, 2026

Contents

Introduction

There is an increasing focus to investigate the association between rare variants and common complex diseases in the hope of explaining the missing heritability that remains unexplained from genome-wide association studies (GWAS) of common variants. Recent studies have reported that rare variants contribute to the genetic susceptibility for a number of complex traits or diseases, including human adult height, lipid levels, autism, ischemic stroke, prostate cancer and breast cancer. Detecting rare variant associations is statistically challenging, stemming from two characteristics of rare variants: low frequency and heterogeneous risk effects. The power of detecting a single rare variant would be very low unless the sample size and/or effect size is extremely large. Thus it has been proposed to aggregating rare variants in a gene or genomic region to boost the power, but this can also negatively affect power due to the problem of heterogeneous effects. To take these heterogeneous effects into account, burden tests have been considered which reweight the effects of rare variants based on their frequencies. While these tests have robust power for detecting a susceptibility region containing clusters of causal variants, they do not readily identify which variants, or class of them, in a gene contribute most to the association.

The subREgion-based BurdEn Test (REBET) simultaneously detects the rare variant association of a gene and identifies the most susceptible sub-regions that drive the gene-level significant association. In order to apply REBET, biologically meaningful sub-regions within a gene need to be specified. The rare variants within each sub-region may share common biologic characteristics, such as functional domain or functional impact. REBET then searches all possible combinations of sub-regions, identifies the one with the strongest association signal through linear burden test, and assesses its statistical significance while adjusting for multiple tests involved in the sub-region search. For detecting overall association for a gene, REBET has robust power when risk effects are relatively homogeneous within sub-regions, but potentially heterogeneous across sub-regions.

1 Installation

Installing the REBET package from Bioconductor or Github.

1.1 Install from Bioconductor

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("REBET")
```

1.2 Install the developmental version from Github

```
devtools::install_github("wheelerb/REBET")
```

2 Loading the package

Before using the REBET package, it must be loaded into an R session.

```
> library(REBET)
```

3 Example

The REBET package requires the user to have the genotype data in memory in the form of a matrix. However, much imputed genotype data exists in large compressed files. In this example, we will have all of the data in external files and create the data objects we need for the rebet function. The genotype data is stored in an output file created from the IMPUTE2 software, and the phenotype data is stored is a tab delimted text file. Get the paths to the data files.

```
> genofile  <- system.file("sampleData", "geno_impute.txt.gz", package="REBET")
> subfile   <- system.file("sampleData", "subjects.txt.gz", package="REBET")
> phenofile <- system.file("sampleData", "pheno.txt.gz", package="REBET")
```

3.1 Phenotype data

The phenotype data file contains the response, covariates, and subject ids that we need for the analysis. In this file, the outcome variable is a column called "Response", the subject id variable is called "Subject", and we will adjust for variables "Age" and "Gender". First, we will read in the data and look at the first five rows.

```
> data <- read.table(phenofile, header=1, sep="\t")
> data[1:5, ]
```

```
  Subject  Response Age Gender
1       1 11.911188  66 FEMALE
2       2  9.540571  59 FEMALE
3       3  9.346940  50 FEMALE
4       4 12.164063  71 FEMALE
5       5 10.495913  51   MALE
```

Notice that the gender variable "Gender" is a character variable, so we need to create a dummy variable for gender.

```
> data[, "MALE"] <- as.numeric(data[, "Gender"] %in% "MALE")
```

3.2 Sub-regions

For our analysis, we will only consider the four sub-regions of chromosome 7 defined below. These regions of interest are protein binding regions, and we will name them "SR1", "SR2", "SR3", and "SR4".

```
> subRegions <- rbind(c(87654800, 87661050),
+                     c(87661051, 87668870),
+                     c(87668871, 87671945),
+                     c(87671946, 87673200))
> rownames(subRegions) <- paste("SR", 1:4, sep="")
> subRegions
```

```
        [,1]     [,2]
SR1 87654800 87661050
SR2 87661051 87668870
SR3 87668871 87671945
SR4 87671946 87673200
```

Since we are only looking at these four sub regions, we will use the minimum and maximum positions of these sub regions when the genotype data is read.

```
> min.loc <- min(subRegions)
> max.loc <- max(subRegions)
```

3.3 Genotype subject ids

The genotype data does not contain subject ids - the subject ids are stored in a separate file that gives the order of the subjects in the genotype data. We will read in this file of genotype subject ids so that we can match the genotype data with the phenotype data.

```
> geno.subs  <- scan(subfile, what="character")
```

The set and order of subjects may not be the same in the phenotype and genotype data. We need the common set of subjects and the correct order.

```
> tmp   <- data[, "Subject"] %in% geno.subs
> data  <- data[tmp, ]
> order <- match(data[, "Subject"], geno.subs)
```

3.4 Genotype data

The genotype data is in a file created from the IMPUTE2 software. Each row of this file has the form: Snpid RSid Position A1 A2 $P1_1P1_2P1_3P2_1P2_2P2_3$... where A1, A2 are the alleles and $Pj_1 = P(a1/a1), Pj_2 = P(a1/a2), Pj_3 = P(a2/a2)$ for the jth subject. We do not know how many variants are in the file and do not know how many variants are in the sub-regions defined above, but we know it should not be more than 100. So we will read in the file row by row instead of attempting to read in the entire file at once. We will initialize some objects to store the necessary information we need from the genotype file. The matrix G will store the expected dosages for the variants we want. The vectors snps and locs will store the variant names and positions.

```
> upper.n <- 100
> G       <- matrix(data=NA, nrow=nrow(data), ncol=upper.n)
> snps    <- rep("", upper.n)
> locs    <- rep(NA, upper.n)
```

Before the genotype file is read we need some vectors that will pick off the probability of each genotype for each subject.

```
> id1 <- seq(from=1, to=3*length(geno.subs), by=3)
> id2 <- id1 + 1
> id3 <- id1 + 2
```

3.5 Reading the genotype data

Now we are ready to open the genotype file and read it row by row. In the code below, we are only going to store the variants that are between the min.loc and max.loc defined above. For such variants, we compute the expected dosage for each subject as $Pj_2 + 2 * Pj_3$, which make allele a2 the effect allele. Note that we must check for missing genotypes - if all three probabilities are 0, then the expected dosage is NA (not 0!).

```
> index <- 0
> fid   <- gzfile(genofile, "r")
> while(1) {
+   vec <- scan(fid, what="character", sep=" ", quiet=TRUE, nlines=1)
+   if (!length(vec)) break
+   snp <- vec[2]
+   loc <- as.numeric(vec[3])
+   if ((loc >= min.loc) & (loc <= max.loc)) {
+     geno.probs  <- as.numeric(vec[-(1:5)])
+     probs1      <- geno.probs[id1]
+     probs2      <- geno.probs[id2]
+     probs3      <- geno.probs[id3]
+     dosage      <- probs2 + 2*probs3
+
+     # Check for missing genotypes
+     tmp <- (probs1 == 0) & (probs2 == 0) & (probs3 == 0)
+     tmp[is.na(tmp)] <- TRUE
+     if (any(tmp)) dosage[tmp] <- NA
+
+     index       <- index + 1
+     G[, index]  <- dosage[order]
+     snps[index] <- snp
+     locs[index] <- loc
+   }
+ }
> close(fid)
```

Subset the objects G, snps, and locs by the number of variants we stored, which is the number index.

```
> G    <- G[, 1:index, drop=FALSE]
> snps <- snps[1:index]
> locs <- locs[1:index]
> colnames(G) <- snps
```

3.6 Input arguments to rebet

The rebet function requires a vector for the response, a matrix of genotypes, a vector of sub-region names for the variants, and optionally a matrix of adjusted covariates. The matrix of genotypes is G, which was created above. Create the response vector Y and matrix of covariates X.

```
> Y   <- as.numeric(data[, "Response"])
> X   <- as.matrix(data[, c("Age", "MALE")])
```

Now create the vector E of sub-region names for each variant in the genotype matrix G. Recall that each row of the matrix subRegions created above defins a sub-region, and that the rownames of this matrix give the sub-region name.

```
> E <- rep("", index)
> for (i in 1:nrow(subRegions)) {
+   tmp <- (locs >= subRegions[i, 1]) & (locs <= subRegions[i, 2])
+   tmp[is.na(tmp)] <- FALSE
+   if (any(tmp)) E[tmp] <- rownames(subRegions)[i]
+ }
```

3.7 Calling rebet and summarizing results

With all of the input arguments being defined, the rebet function can be called.

```
> ret <- rebet(Y, G, E, covariates=X)
```

The returned object from rebet is summarized using the h.summary function in the ASSET package. The resulting summary shows that sub-region SR3 is highly significant.

```
> print(h.summary(ret))
```

```
$Meta
   SNP       Pvalue     OR CI.low CI.high
1 Gene 0.0001554158 43.915  6.186 311.755

$Subset.1sided
   SNP       Pvalue       OR   CI.low  CI.high      Pheno
1 Gene 5.434698e-08 532447.2 4590.188 61762179 Region_SR3

$Subset.2sided
   SNP       Pvalue     Pvalue.1  Pvalue.2     OR.1 CI.low.1 CI.high.1  OR.2
1 Gene 4.336478e-07 4.224093e-08 0.5527489 532447.2 4771.827  59411208 0.401
  CI.low.2 CI.high.2    Pheno.1    Pheno.2
1     0.02     8.199 Region_SR3 Region_SR2
```

4 Session Information

```
> sessionInfo()
```

```
R version 4.6.0 (2026-04-24)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so;  LAPACK versio

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
```

```
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

time zone: Etc/UTC
tzcode source: system (glibc)

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] REBET_1.30.0 ASSET_2.30.0

loaded via a namespace (and not attached):
 [1] xfun_0.58        Matrix_1.7-5     lattice_0.22-9   magrittr_2.0.5
 [5] splines_4.6.0    maketools_1.3.2  glue_1.8.1       tibble_3.3.1
 [9] knitr_1.51       pkgconfig_2.0.3  generics_0.1.4   buildtools_1.0.0
[13] lifecycle_1.0.5  mvtnorm_1.4-1    cli_3.6.6        vctrs_0.7.3
[17] grid_4.6.0       compiler_4.6.0   sys_3.4.3        tools_4.6.0
[21] rmeta_3.0        pillar_1.11.1    evaluate_1.0.5   survival_3.8-6
[25] rlang_1.2.0      expm_1.0-0       msm_1.8.2        MASS_7.3-65
```