GLIDE (GPU-based LInear Detection of Epistasis): How To

Chloé-Agathe Azencott
chloe-agathe.azencott@mines-paristech.fr
MLCB Research Group, Max Planck Institutes Tübingen (Germany)
and
Mines ParisTech–CBIO, Institut Curie, INSERM U900 (France)

January 16, 2014

1 Introduction

The goal of this document is to provide insights (and scripts) to facilitate the use of `GLIDE` [1] for epistatsis detection in GWAS data.

`GLIDE` can be downloaded from https://github.com/BorgwardtLab/GLIDE and the companion scripts as well as this documentation from http://github.com/chagaz/glide-scripts.

Disclaimer: Those are scripts I have been using in the context of a specific project, on specific machines. I have tried to keep them clean and documented, but there were not written to be very generally applicable. In particular there may be hard-coded paths and constants in them. They are provided more as examples of what can be done and how than as off-the-shelf software.

`GLIDE` computes the following linear regression:

$$\text{Pheno} = \alpha + \beta\,\text{SNP}_1 + \gamma\,\text{SNP}_2 + \delta\,\text{SNP}_1\text{SNP}_2$$

and computes a t-test to evaluate whether δ is significantly different from 0.

The following document illustrates how to use `GLIDE` for epistasis detection on a fictional data set, called `mydata_final`, and containing 3314 samples and 464776 SNPs. The following assumes that the genotype data is provided as `PLINK`[1] binary files:

```
mydata_final.bed
mydata_final.bim
mydata_final.fam
```

2 Converting `PLINK` files to `GLIDE` input

Note: This overrides the `plink2glide.py` script provided in `glide`.

2.1 `GLIDE` input format

`GLIDE` takes as input two genotype files (and will compute the linear regression for all pairs (SNP1, SNP2) with SNP1 belonging to the first file and SNP2 belonging to the second file), that have as many lines as SNPs

[1] http://pngu.mgh.harvard.edu/~purcell/plink/

and as many space-separated columns as samples, and one phenotype file, with as many lines as samples.

Files `Test1kind_first1ksnp.txt` and `Test1kind_second1ksnp.txt` in the `glide` repository are examples of such input genotype files. `Test1kind_pheno.txt` is an example of such input phenotype file.

Here I provide one way to convert `PLINK` binary files to `GLIDE` input files.

2.2 Step1: Converting binary `PLINK` to raw `PLINK` format

```
# filter out for MAF, HWE and missing genotypes
plink --noweb --bfile mydata_final --maf 0.01 --hwe 1e-6 --geno 0.1 --make-bed
--out mydata_final_clean

# convert from binary PLINK to raw PLINK
plink --noweb --bfile mydata_final_clean --recodeA --out mydata_final_clean
```

2.3 Step2: Converting raw `PLINK` files to `GLIDE` input files

In a Python console:

```
proot = "mydata_final_clean"

# Extract phenotypes (encode binary as 0/1)
with open("%s.raw" % proot) as f, open("%s.pheno" % proot, 'w') as g:
    hdr = f.readline()
    for line in f:
        # for continuous phenotype uncomment the following line:
        # g.write("%d\n" % (float(line.split()[5])))
        # for binary phenotype encoded as 0/1 uncomment the following line:
        # g.write("%d\n" % (input(line.split()[5])))
        # for binary phenotype encoded as 1/2 uncomment the following line:
        # g.write("%d\n" % (int(line.split()[5]) - 1))
f.close()
g.close()
```

`GLIDE` input genotypes (referred to as `.glideIn`) are the transposed of raw `PLINK` files. Raw `PLINK` and `glideIn` formats are both essentially CSV formats (space-separated). One can use packages like `PyTables` or `pandas` to efficiently process it (see examples in Section 2.6). However I have been using a (rather) naive implementation to transpose raw `PLINK` files (you may want to grab coffee or lunch).

```
# Transpose genotypes
python transpose.py mydata_final_clean
```

2.4 Missing values

`GLIDE` cannot deal with missing values, as it relies on matrix operations that would fail if some of the matrix entries are missing. I recommend imputing the missing values naively: replace the missing values with the most common SNP value, or in the case of binary data, most common SNP value in the same class (cases or control). Then, for interesting pairs, you can remove the individuals with missing values for either SNP, and recompute the linear regression cleanly; see Section 4.3. If the rate of missing values is low enough, this will only make a small difference. Then again, if the rate of missing values is high, you might want to reconsider analyzing this data to start with.

```
# Impute missing SNP values naively (majority)
python naive_impute.py mydata_final_clean
```

2.5 Computing single-locus associations and genomic control with PLINK

Generally you will also be interested in single-locus associations, which you can compute with PLINK. You may also want to check for inflation of the genomic control, to know whether correcting for population structure will be needed (see Section 4.3).

```
# Single locus association and genomic control inflation factor
plink --bfile mydata_final --linear --adjust --gc --out mydata_final_singleLocus
```

You can use qqplot.py to visualize the QQ-plot between the expected and observed distributions of p-values.

2.6 Saving data in HDF5 format

HDF5 is a file format that makes it possible to store and efficiently process large data sets such as the GLIDE input files we have just generated. This step is not required (I always provide a non-HDF5 alternative to my scripts), but some of the post-processing will be smoother with HDF5 files.

```
python plink2h5.py mydata_final_clean mydata_final_clean.h5
```

This script uses the plinkio package[2] to read PLINK data and PyTables to write h5 data efficiently.

3 Running GLIDE on a GPU machine

3.1 Compiling GLIDE

This section appplies to an sm_20 GPU architecture. For a different architecture, you may need different flags and block sizes.

Make sure that the following NVCCCFLAGS are in use in the Makefile:

```
NVCCFLAGS= -arch=sm_20  --compiler-bindir=/usr/bin/gcc-4.4
```

Set the proper BLOCK_SIZE in GLIDE.h:

```
//sm_20
#define BSx 16
#define BSy 16
#define BSz 1
//sm_20

/*//begin: sm_13
#define BSx 10
#define BSy 10
#define BSz 1
//end: sm_13 */
```

Compile:

```
make all
```

3.2 Preparing input data

GLIDE takes **two** genotype files as input, and computes linear regression for all pairs of SNP1 belonging to the first file and SNP2 belonging to the second file. You can give it twice the full .glideIn as input, but in

[2] https://github.com/fadern/libplinkio

practice it works faster if you pre-chop the input file yourself and then run GLIDE sequentially on all possible pairs of the smaller files (which I try to refer to as *tiles*).

You can run some timing experiments if you want to determine the best size of those tiles. Actually, one should take the time to sit down and think about how to determine optimal tile sizes rather than shooting in the dark. For the data and GPUs I worked with when creating this documentation, I found that 24576 SNPs was a good choice. For optimal load balancing, this number should be a multiple of BSx $\times$ BSy.

```
# Split input GLIDE file into tiles of size 24576 SNPs
split -l 24576 mydata_final_clean.glideInImputed mydata_final_clean.glide_
```

3.3 Running GLIDE

```
# Create the bash file containing the GLIDE commands
# python split_glide.py <root name of genotype files> <phenotype file>
#  <number of individuals> <t-test threshold> <root of bash file>
python split_glide.py mydata_final_clean mydata_final_clean.pheno 3314 6 runGlideMyData
```

The GPU to use, identified by its number (see the output of `lsgpu`), is specified by the `-g` flag (also see `GPULIST` in `split_glide.py`), and the bash script generated by `split_glide.py` takes the name `<root of bash file>_<gpu number>.sh`. If the GPU is not free, `GLIDE` will segfault:

```
*****************
Linear Regression
*****************
GPU Number 0 in use
Reading in Matrices into Host Success
Copying matrices to GPU success
Segmentation fault
```

Always start by testing a single command, the output of

```
head -1 <root of bash file>_<gpu number>.sh
```

The output produced is `<root name of genotype files>_<tile1>_<tile2>.output`. Only the pairs with a t-score larger than `<t-test threshold>` are written out to the output. If no t-score was larger than `<t-test threshold>`, this file will be empty. The fewer pairs to write out, the faster the code, so this does not necessarily mean you have a problem (other tiles might give you significant pairs). On the other hand, if your threshold is too large, you'll only get empty files and won't be able to relax things a bit (nor to convince yourself everything ran fine).

For testing a single command, I recommand setting `<t-test threshold>` to a low value (e.g. 3.5) so as to get an output. You can then check that everything is in order by recomputing on CPU the p-values for the top 10 pairs of the output (`mydata_final_clean_af_af.output` for our example):

```
# convert t-scores into p-values and recover SNPs IDs from block indices
python compute_pvalues.py mydata_final_clean_af_af.output af_af
mydata_final_clean.bim 3314 mydata_final_clean_af_af.pvals

# get the first 10 p-values
head mydata_final_clean_af_af.pvals > mydata_final_clean_af_af.pvals.10

# recompute p-values on CPU
py rerun_h5.py mydata_final_clean.h5 mydata_final_clean.bim
mydata_final_clean.phenoGlide mydata_final_clean_af_af.pvals.10
mydata_final_clean_af_af.pvals.10.rerun

# compare outputs (visually)
more mydata_final_clean_af_af.pvals.10.rerun | cut -f 11,15
```

Tip: there are a few hardcoded block/chunk sizes values in `CONST.py` and `compute_pvalues.py`, make sure they are set properly (also: see Section 4).

When you are satisfied with the result, go ahead and run the full script. You might want to time the first command and count how many of them you have to get an estimate of the total runtime.

```
chmod +x runGlideMyData_0.sh
./runGlideMyData_0.sh
```

4 Post-processing

4.1 Computing p-values

Here's how to transform the t-scores in p-values, retrieve SNP IDs from their tile/grid/block coordinates, and gather everything in a single file.

`blockToIndex` in `CONST.py` must have been computed using for TILESIZE the tile size used when splitting the input file of `GLIDE`, which must then be set accordingly in `compute_block_to_index.py`.

```
# Compute blockToIndex dictionary in CONST.py
python compute_block_to_index.py
```

```
# Compute all p values
# TILESIZE = 24576
python compute_all_pvalues.py mydata_final_clean mydata_final_clean.bim 3314
```

```
# cat and sort the p values files
tail -n +2 mydata_final_clean_*.pvals | sort -k 7 -s -g > mydata_final_clean.spvals

# clean
# remove lines starting with "==>"
# (! sed -i takes an additional option on MacOS)
sed -i '/^==>/ d' mydata_final_clean.spvals
# remove blank lines
# (! sed -i takes an additional option on MacOS)
sed -i '/^$/d' mydata_final_clean.spvals
# add header
echo `head -1 mydata_final_clean_aa_aa.pvals` |
  cat - mydata_final_clean.spvals > mydata_final_clean_0
mv mydata_final_clean_0 mydata_final_clean.spvals
```

4.2 Keeping significant pairs of SNPs

The multiple-hypothesis testing correction for epistasis suggested by [2] is $n \times (n-1)/8$ where n is the number of SNPs.

In a Python console:

```
n = 464776
corr = n * (n-1) / 8
sig = 0.05 / corr

pairs = set([]) # some pairs computed twice, keep track to keep only one copy
with open("mydata_final_clean.spvals") as f,
  open("mydata_final_clean.spvals.sig", 'w') as g:
    for line in f:
      pval = float(line.split()[10])
      if pval < sig:
        pair = [line.split()[0], line.split()[3]]
        pair.sort()
        pair = "_".join(pair)
        if not pair in pairs:
            g.write(line)
            pairs.add(pair)
    else:
        break
f.close()
g.close()
```

4.3 Re-running linear regression without imputation

This is how to run the same linear regression as `GLIDE`, but removing the individual with missing values (on a pair-by-pair basis, ie. two different groups of individuals might be used for two different pairs) instead of using imputed missing values.

There's one (slower) version that directly reads `glideIn` files:

```
python rerun.py mydata_final_clean.glideIn mydata_final_clean.snpNames \
  mydata_final_clean.pheno mydata_final_clean.spvals.sig \
  mydata_final_clean.spvals.rerun
```

and its counterpart that uses HDF5 files (see Section 2.6) to create them.

```
python rerun_h5.py mydata_final_clean.h5 mydata_final_clean.bim \
  mydata_final_clean.phenoGlide mydata_final_clean.spvals.sig \
  mydata_final_clean.spvals.rerun
```

You can also use these scripts to run a *logistic regression* (using `test_logistic` instead of `test_linear`) if the phenotype is binary.

4.4 Re-running linear regression with covariates

Use `rerun_confounders.py` or (more efficient) `rerun_confounders_h5.py` to include confounders in the regression model.

To include sex as a confounder (particularly relevant if your hits are on chromosome X):

```
# get sex information
cut -d " " -f 5 mydata_final.fam > mydata_final.sex

# run again, with sex as additional explanatory variable
python rerun_confounder.py mydata_final_clean.glideIn mydata_final_clean.snpNames \
  mydata_final.sex mydata_final_clean.pheno mydata_final_clean.spvals.sig \
  mydata_final_clean.spvals.rerun_sex

# h5 version
python rerun_confounder_h5.py mydata_final_clean.h5 mydata_final_clean.bim \
  mydata_final.sex mydata_final_clean.phenoGlide mydata_final_clean.spvals.sig \
  mydata_final_clean.spvals.rerun_sex
```

To compute principal components to use as covariates, you can use `GCTA`[3]:

```
~/gcta/gcta64 --bfile mydata_final_clean --make-grm --out mydata_final_clean
--thread-num 1 --autosome
~/gcta/gcta64 --grm Dmydata_final_clean --pca 20 --out mydata_final_clean
```

4.5 Meta-analysis

If your study comprises several populations (cohorts), you probably want to run separate analyses on each population, then combine the results into one meta-study (to avoid population stratification). `meta_analysis.py` is an example of how to run such a meta-analysis, using the inverse-variance based approach of METAL [3].

References

[1] T. Kam-Thong, C.-A. Azencott, L. Cayton, B. Ptz, A. Altmann, N. Karbalai, P. G. Smann, B. Schlkopf, B. Mller-Myhsok, K. M. Borgwardt K.M. GLIDE: GPU-Based Linear Regression for the Detection of Epistasis *Human Heredity*, 73:220–235, 2012.

[2] T. Becker, C. Herold, C. Meesters, M. Mattheisen, M. P. Baur. Significance levels in genome- wide interaction analysis (GWIA). *Ann Hum Genet*, 75:29-35, 2011.

[3] C. J. Willer, Y. Lim, G. R. Abecasis. METAL: fast and efficient meta-analysis of genomewide association scans. *Bioinformatics*, 26(17): 21902191, 2010.

[3] http://www.complextraitgenomics.com/software/gcta/index.html